

INSTITUTE OF COMPUTER
SCIENCES
Master in Artificial Intelligence and Data
Science

Universitätsstr. 1 D–40225 Düsseldorf



Heinrich Heine
Universität
Düsseldorf

AI-based fluorescent labeling for cell line development

Hanna Pankova

Master thesis

Date of issue: 01. April 2022
Date of submission: 29. August 2022
Reviewers: Prof. Dr. Markus Kollmann
Dr. Wolfgang Halter

Erklärung

Hiermit versichere ich, dass ich diese Master thesis selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 29. August 2022

Hanna Pankova

Abstract

Cell line development is an expensive and time-consuming process, however that is the most modern approach for producing the proteins needed in various pharmaceuticals.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Notation	2
2	Domain knowledge	3
2.1	Biology	4
2.1.1	Cell line development process	4
2.1.1.1	CLD steps	5
2.1.2	Project specifications of cell line development for Merck KGaA	6
2.2	Deep learning and machine learning basics	7
2.2.1	Neural networks	7
2.2.2	Dimensionality reduction methods	12
2.2.2.1	UMAP	12
2.2.2.2	PaCMAPI	13
2.2.2.3	PCA	13
2.2.3	Clustering	14
2.2.3.1	DBSCAN	14
2.3	Imaging	15
2.3.1	Digital imaging	15
2.3.2	Microscopy imaging	15
2.3.2.1	Image acquisition peculiarities	15
2.3.2.2	Crops combination technique	16
3	Implementation and experiments	18
3.1	Model training	18
3.1.1	Neural network architecture	18
3.1.2	Available data	19
3.1.3	Training costs estimation	21
3.1.4	Augmentations	21
3.1.4.1	Special augmentations for rotation and scaling	22
3.1.5	Model setup	23

3.1.5.1	Weight Initialization	23
3.1.5.2	Regularization	24
3.1.5.3	Optimizers	26
3.2	Nuclei	27
3.2.1	Preprocessing	27
3.2.2	Training and predictions	27
3.2.2.1	Convergence	27
3.2.2.2	Predictions quality	30
3.2.3	Postprocessing for nuclei segmentation	30
3.2.3.1	Thresholding algorithms	32
3.2.4	Influence of scaling on predictions quality	33
3.3	Endoplasmic Reticulum	34
3.3.1	Preprocessing	34
3.3.2	Training and predictions	34
3.3.3	Combination of nuclei and actin predictions	35
3.3.4	Generalizability across phenotypes	35
3.4	Golgi	36
3.4.1	Preprocessing	36
3.4.1.1	Background removal algorithms	36
3.4.2	Training and predictions	37
3.4.3	Alternative ways to improve predictions	39
3.4.3.1	Asymmetrical losses	39
3.4.3.2	Use of gradient in loss	39
3.4.3.3	Noise reduction methods	39
3.5	GFP	40
3.5.1	Preprocessing	40
3.5.2	Predictions	40
3.5.3	Downstream metrics	41
3.5.4	Combination of GFP, nuclei and ER	41
3.6	Model evaluation	42
3.6.1	Metrics for downstream tasks	42
3.6.2	Influence of different loss functions on metrics for downstream tasks	42
4	Stability study	43

4.1	Stability study	43
4.1.1	Artificial corruptions	43
4.1.2	Real corruptions	44
4.1.2.1	Not fixed cells imaging as corrupted input	44
4.1.2.2	Real-world examples of corruptions	44
4.1.3	Influence of corruptions on metrics for downstream tasks	44
4.1.4	Improving predictions with additional corruption augmentations .	44
4.2	UNET embeddings study	45
4.2.1	Application of various dimentionality reduction methods	45
4.2.2	Autoencoder embeddings as an alternative	45
4.2.3	Clustering of PacMAP embeddings	48
4.2.3.1	Clustering on UNet embeddings	48
4.3	Drift detection	49
4.3.1	A need to detect drift	49
4.3.2	Maximum mean discrepancy for drift detection	49
4.3.3	Online version of MMD algorithm	49
5	Software Tools	51
5.1	Foundry, Palantir	51
5.2	AWS	51
5.3	Streamlit	51
6	Future research	52
7	Summary	53
References		54
List of Figures		55
List of Tables		56

1 Introduction

1.1 Motivation

Nowadays recombinant proteins are widely used in biomedical research and production of medicines that are used in the variety of therapeutic needs like vaccines and antibodies [TODO add references]. Therefore there is currently a great need for high-volume and high-quality recombinant protein production. That is why the optimization and improvement of cell line development (CLD) as a process in use for the production of recombinant proteins is extremely important.

Clone screening is a step of the CLD process in which cells are analyzed for further selection of the most stable and productive clones. Fluorescence microscopy provides data about the cell structure that enables better clone selection, however it is not only expensive and time-consuming, but also toxic for the cells. Automating fluorescence microscopy for clone selection via convolutional neural networks *in silico* significantly simplifies the existing procedure of clone selection, reducing phototoxicity, time and expenses needed for the analysis.

The goal of this thesis is to provide a proof of concept on whether an *in silico* approach to fluorescent labeling can substitute manual cell staining and provide all the needed information that would be used for further clone screening and selection. That is particularly why the research at hand is aimed towards the specific needs, pipelines and data used at Merck KGaA. In this research four UNet models (for four target proteins highlighting different cell organelles) were developed for automating fluorescence cell staining based on DIC microscopy imaging of CHO cells: nuclei, endoplasmic reticulum, [[green fluorescent protein]] and Golgi apparatus. Another important goal of this research that differentiates it from the similar studies like [TODO cite LaChance 2020 and cite Christiansen 2018] is to not only provide deep learning models for the fluorescence predictions but also study their reliability and be able to detect drift during image acquisition that can happen quite easily due to the sensitivity of the microscope settings as well as the cell phenotypes, scaling and fixation procedures.

This thesis is laid out as follows: Section 2 reviews the biological concepts needed to understand the application of this research, it also reviews machine and deep learning concepts used for data analysis; Section 3 provides an overview of the implementation and the results of the experimental *in silico* fluorescence predictions; Section 4 shows stability of the deep learning models developed in the previous section and provides valuable insights on the information from their embeddings; Section 5 details the practical tools used for the development at Merck KGaA and Section 6 explores possible future research questions that arose from the current analysis and provides concluding remarks and succinct recommendations.

1.2 Notation

$x^{(i)}$ The i-th input image (sample) from a dataset

X_{train} A set of training examples

$y^{(i)}$ The target image associated with the i-th input sample from a dataset

\mathbb{R} A set of real numbers

$A_{i,j}$ An element on the i-th row and j-th column of a matrix A

p_{data} Data generating distribution

2 Domain knowledge

The *in silico* fluorescence labeling approach has proven to be very promising as a substitute to the manual cell staining processes [TODO cite all the relevant references]. For example, the research of [TODO cite Christiansen 2018] did not only prove successful prediction of different cell stains with a variety of modalities and cell types, but it had also successfully determined cell viability. Nevertheless, the study is limited mainly to transmitted light (TL) z-stack imaging. This refers to the networks input being comprised of 3D images, which is not the case in this work. [cite Ounkomol 2018] too shows successful predictions of several organelles in bright-field TL 3D images using 3D convolutional neural networks. However, switching to 2D data did not yield adequate results for them. Other, newer studies like [cite Ugawa 2021] provide an application of label-free fluorescence predicting already at the sorting stage, when a high-throughput system sorts cells individually. However, only a single-pixel detector is used by this study, meaning that it captures a wave rather than an image. Nonetheless one can recover an image with heavy computations if needed [cite Sadao Ota 2018].

There are two very promising studies by [cite Cheng 2021] and [cite LaChance 2020]. Even though the former manages to reach a state-of-the art performance on label-free fluorescence reconstruction, it uses reflectance images from oblique dark-field illumination as the input, which is a more specific cell imaging approach. Still, this input provides higher structural contrast in comparison to any transmission technique [cite Boustany 2010]. The latter study uses an easier imaging technique (DIC imaging) as an input, which shows great results even with low-resolution data. Both of these studies provide results based not only on training metrics, but also on performance of the models for metrics used in the downstream tasks. This is very important in the label-free fluorescence labeling research and was not present in papers before LaChance. In the thesis at hand, many methods from the LaChance paper were used as both the data and the processes in the project pipeline of Merck KGaA align very well with the study conducted in that paper.

All of the studies mentioned above, as well as this work rely on the premise that the input imaging type (here DIC) contains enough information to predict the fluorescence signal from it. This is a reasonable assumption because DIC, as well as bright-field and phase contrast imaging, are very often used for determining cell morphology [TODO cite Kasprowicz 2017].

This chapter provides a brief overview of the biological background needed to understand the process of cell line development (CLD) and the role of fluorescent *in silico* labeling of DIC cell images within. It also covers the fundamentals of deep and machine learning techniques used here including clustering and dimensionality reduction approaches. At the end of the chapter, a brief summary of the microscopy image acquisition process used in the research is given.

2.1 Biology

2.1.1 Cell line development process

Cell line development (CLD) is a process of generating single cell-derived clones that produce high and consistent levels of target therapeutic protein [TODO cite pharma.lonza.com/offering/mammalian/cell-line-development]. Therapeutic proteins in this case are so-called recombinant proteins and they are widely used in biomedical research, the production of medication and for various therapeutic needs such as, for example, vaccines and monoclonal antibodies (mAbs) [TODO cite Ohtake 2013, Jefferis 2021, Funaro 1996]. A recombinant protein, as defined by [cite Barbeau, J], is a modified or manipulated protein encoded by a recombinant DNA. Recombinant DNA in turn consists of a plasmid, where the genes of the target protein of interest are cloned downstream of a promoter region. As soon as this plasmid is transfected to a host cell (for example some mammalian cells that are able to produce the protein), the host will start to express this protein of interest. Today there is a great need for the production of high volumes of good quality recombinant proteins, both in industrial as well as research contexts [TODO cite Tihanyi 2020]. This is the reason why the goal of many research projects in recombinant protein production is to improve expression efficiency and create high-throughput systems to improve the CLD processes [cite Tihanyi 2020].

One of the most popular host cells used in CLD and in this thesis specifically are chinese hamster ovary (CHO) cells [cite Castan 2018]. Although different cells can be used as hosts, such as bacterial, plant-based or yeast cells, mammalian cells remain the most popular choice [cite Beckman]. The reason behind this popularity resides in the fact that they can produce a diverse range of correctly folded proteins and most importantly they have high protein production rates. The productivity rate is measured in titre of produced protein, and CHO cells can reach 0.1 - 1 g/L in batch and 1 - 10 g/L in fed-batch [TODO add reference] cultures [cite Tihanyi 2020]. Mostly all of the mAbs are produced using CHO cells [cite Lalonde 2017]. Companies mostly use the same host cell line for all their productions because already checked and qualified cells simplify the road to the clinic [cite Tihanyi 2020]. This is why current research has a wide applicability.

However, there is a downside to using CHO as host cells - they are infamously unstable. As rapidly growing immortal cells CHO are also genetically unstable and extremely heterogeneous which usually leads to the main issue: production instability. The problem of choosing stable and high-production clones that simultaneously will be able to express protein qualitatively and quantifiably over time is essentially the main goal of current research. The challenge in manufacturing here is the time and the cost of production. Currently, a lot of research attention is dedicated to the reduction of both factors, as well as the development of techniques of high-throughput clone screening and characterization [cite Tihanyi 2020]. The latter is of interest for this thesis. With great amounts of data collected over time and the development of computational modelling and statistical analysis it is now possible to carry out the analysis *in silico*, meaning computationally without interfering with the cells instead of the usual *in vitro* analysis.

2.1.1.1 CLD steps



Figure 1: CLD process steps

The first step of CLD is called transfection - the introduction of the gene of interest (GOI or a DNA vector or alternatively an expression vector) into CHO cells. There are two main problems with this step: firstly, transfection mostly results in a vector being inserted into a random site within the host cell genome and secondly, it generally has low efficiency of integration [cite Tihanyi]. It is important to transfet a GOI into the optimal site of the genome to secure high protein expression over time during protein production. Practically however, GOI is transfected into a random location of genome. In cases where the gene was transfected into an inactive site of genome (essentially the majority of genome is transcriptionally inactive), the cell will likely be unable to express the gene [cite Castan, Hong 2018].

The second step of the process is the selection of cell minipools that have successful and stable gene integrations for further expansion and cloning. The reason for not all of them being suitable is that during the transfection step, only 80% of the cells will receive a GOI vector [cite Castan]. Only a small percentage of these cells actually integrate a vector into the genome and, as mentioned above, only a fraction of those are able to express the protein stably [a better reference needed Shin 2020]. After the best minipools are selected, they will be expanded.

The third step in CLD is to clone the cells. The selected stable pools of cells are phenotypically and genetically diverse. This means that they have different growth rates, metabolic profiles, and so forth. This is not ideal for industrial production - all the cells used for protein production should be derived from the same clone [cite [25] from Castan].

Once the cells are cloned, phenotypical and genetical heterogeneity is reduced, the next step is to characterize the cells for their expression of the GOI. One has to estimate the clones' productivity and stability. Such observations may take up to 90 days (usually the checks are made on the 30th, 60th and 90th days). If the clones remain stable after this time and are able to express enough of the protein, then they are suitable for further production. However, this last step costs a lot of time and maintenance costs for feeding and analysing the cells. Predicting productivity and stability of the cells in earlier stages would reduce this time significantly or even allow to avoid this process entirely.

2.1.2 Project specifications of cell line development for Merck KGaA

There are many different proteins that can be produced using such technologies, for example, vaccines, hormones, sugars etc., This research however is dedicated to the production of monoclonal antibodies (mAbs).

CHOZN® platform is a currently widely used product of Merk KGaA. CHOZN® is a CHO mammalian cell expression system for fast and easy selection and growth of clones producing high levels of recombinant proteins [cite tech-bulletin]. The processes of developing expression systems on this platform correspond to the general CLD process described in the previous subsection [put subsection number]. The scope of the project is to simplify the labour-intensive and time-consuming process of stability measurement of the expression system by inducing predictions of productivity and stability rates during early steps in the CLD process.

After the transfection step there are several quantities that are measured in minipools in order to select the best ones. For example, cell size, its complexity, cell surface protein expression, endoplasmic reticulum (ER) mass, mitochondria mass, etc. For qualitative and quantitative characterization of cells, fluorescent labeling is used. It is a process of covalently binding fluorescent dyes to biomolecules such as nucleic acids or proteins, so that they can be visualized via fluorescence imaging [cite <https://www.nature.com/subjects/fluorescent-labelling>]. A fluorophore is a chemical compound that can reemit light at a certain wavelength. These compounds are a critical tool in biology because they allow experimentators to capture particular components of a given cell in detail [cite O'reilly life sciences p113].

Unfortunately, fluorescence labeling is expensive, time-consuming and may kill the cell due to its phototoxicity [cite Fried et al., 1982; Patil et al., 2018; Progatzky et al., 2013]. Additionally, Yeo et al. [cite Tihanyi] found out that different selection markers affect the production stability of CHO cells. Other negative aspects the manual staining approach are: there is a limited number of available fluorescent channels in microscopes; some fluorophores have a spectral overlap, hence there is a limited number of detectable markers [cite Perfetto et al., 2004]; such labels can be inconsistent [cite Burry, 2011; Weigert et al., 1970], and depend a lot on reagent quality and require many hours of lab work. Toxicity, for instance, is a very dangerous factor, especially for medicine production as it may even affect the final product. Because of this there is a need for an approach of *in silico* fluorescent labeling - computationally and without affecting the cell.

For *in silico* labeling, the input data is a differential interference contrast (DIC) microscopy. This is an optical microscopy technique used to enhance the contrast in unstained, transparent samples [cite wikipedia?]. This is a much cheaper image acquisition technique than a staining process, and it has much less variability as well (for example, no dependency on the dye or antibody quality). The research carried out in the scope of this thesis is dedicated to predicting fluorescence signal from the DIC imaging directly without the need of actual cell staining. The measurements needed for selecting the minipools can be calculated as usual, with the exception of using the predicted images instead.

2.2 Deep learning and machine learning basics

2.2.1 Neural networks

Definition 2.1 (Image dataset). An image dataset in the scope of this thesis consists of input DIC images X and target fluorescence images Y . Combined, couples from each form (X and Y) construct a dataset:

$$D = (X, Y) = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\} \quad (1)$$

where both $x^{(i)}$ and $y^{(i)} \in \mathbb{R}^{W \times H}$ are single images, N is the size of the dataset. Generally input data has a shape of (N, C, H, W) , in this work $C = 1$.

Definition 2.2 (Model). A model is a function with learnable parameters $\theta = (\theta_1, \dots, \theta_K)$ where $\theta_i \in \mathbb{R}$ for $i \in 0, \dots, K$ which approximates the mapping of initial data X to target data Y .

$$M(X, \theta) = Y' \approx Y \quad (2)$$

Definition 2.3 (Loss function). A loss function is a function $L(y, M(x, \theta))$ of model's parameters θ , that for $(x^{(i)}, y^{(i)}) \in D$ outputs a scalar value measuring the difference between ground truth y and prediction $M(x, \theta)$. A loss function can normally be defined as an average over the training set:

$$J(\theta) = \mathbb{E}_{(x,y) \sim p_{data}} L(y, M(x, \theta)) \quad (3)$$

where p_{data} denotes an empirical distribution of the training data.

Definition 2.4 (Binary-cross entropy loss). Let $y \in \mathbb{R}^{W \times H}$ be a ground truth image and $y' \in \mathbb{R}^{W \times H}$ be a prediction. Binary-cross entropy loss is defined as:

$$L(y, y') = -\frac{1}{N^2} \sum_{i=1}^H \sum_{j=1}^W y_{i,j} \cdot \log(y'_{i,j}) + (1 - y_{i,j}) \cdot \log(1 - y'_{i,j}) \quad (4)$$

Definition 2.5 (MSE (mean squared error) loss). Let $y \in \mathbb{R}^{W \times H}$ be the ground truth and $y' \in \mathbb{R}^{W \times H}$ be the predicted images. The MSE loss is defined as:

$$L(y, y') = \sum_{i=1}^H \sum_{j=1}^W (y_{i,j} - y'_{i,j})^2 \quad (5)$$

Definition 2.6 (PCC (Pearson correlation coefficient) loss). Let $y \in \mathbb{R}^{WH}$ be a flattened ground truth and $y' \in \mathbb{R}^{WH}$ be a flattened predicted image. The PCC loss is defined as:

$$PCC(y, y') = \frac{\sum_{i=1}^{WH} (y_i - \bar{y})(y'_i - \bar{y}')}{\sqrt{\sum_{i=1}^{WH} (y_i - \bar{y})^2 (y'_i - \bar{y}')^2}} \quad (6)$$

$$L(y, y') = \frac{1 - PCC(y, y')}{2} \quad (7)$$

where \bar{y}, \bar{y}' are means of the ground truth and predicted images respectively.

There are two important values here, first Pearson correlation coefficient (PCC further) and PCC loss. PCC in a measure of similarity between two matrices in this case, with values between -1 and 1 , with 1 being a positive correlation. On the contrary PCC loss is a measure of dissimilarity between two matrices, with values between 0 and 1 , with 0 meaning that matrices are the same.

This loss is widely used in cell biology for comparison of co-localization between the proteins. PCC is also popular in computer vision where it is utilized for the determination of image similarity in terms of spatial-intensity [cite Cohen].

Definition 2.7 (Optimization). Optimization is a process of updating the parameters θ of the model $M(X, \theta)$ to minimize the loss function $L(y, M(x, \theta))$.

With a maximum likelihood estimation, we get:

$$\theta_{MLE} = \arg \max_{\theta} \sum_{i=1}^N \log p_{\text{model}}(x^{(i)}, y^{(i)}, \theta) \quad (8)$$

After maximizing the sum and taking a gradient one gets:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{x,y \sim p_{\text{data}}} \nabla_{\theta} \log p_{\text{model}}(x, y, \theta) \quad (9)$$

The exact gradient on a discretized data-generating distribution is then:

$$g = \nabla_{\theta} J^*(\theta) = \sum_x \sum_y p_{\text{data}}(x, y) \nabla_{\theta} L(y, M(x, \theta)) \quad (10)$$

Here one can obtain an unbiased estimator of a true gradient on a mini-batch of i.i.d. samples $\{x^{(i)}, \dots, x^{(m)}\}$

$$\hat{g} = \frac{1}{m} \nabla_{\theta} \sum_i L(y^{(i)}, M(x^{(i)}, \theta)) \quad (11)$$

Definition 2.8 (Stochastic gradient descent). Stochastic gradient descent is an optimization algorithm where the parameters θ are iteratively updated every mini-batch of data by the following rule:

$$\theta_{k+1} = \theta_k - \alpha \frac{1}{m} \nabla_{\theta} \sum_i L(y^{(i)}, M(x^{(i)}, \theta)) \quad (12)$$

where α is a tuneable parameter called *learning rate*.

Definition 2.9 (Adadelta optimizer). An Adadelta optimizer is a more sophisticated optimization technique, that follows the subsequent algorithm for the parameter update:

Definition 2.10 (Stochastic gradient descent (SGD) optimizer). Stochastic gradient descent is an optimization algorithm where the parameters θ are iteratively updated every mini-batch of data by the following rule:

$$\theta_{k+1} = \theta_k - \alpha \frac{1}{m} \nabla_{\theta} \sum_i L(y^{(i)}, M(x^{(i)}, \theta)) \quad (13)$$

Algorithm 1 Adadelta optimization

1. $E[g]^2_0 = 0$ and $E[\Delta\theta^2]_0 = 0$ In order to update the parameters compute:
 2. Compute gradient: \hat{g}_t
 3. Accumulate gradient: $E[g]^2_t = \rho E[g]^2_{t-1} + (1 - \rho)g_t^2$
 4. Compute update: $\Delta\theta_t = \frac{\text{RMS}[\Delta\theta]_{t-1}}{\text{RMS}[g]_t} \hat{g}_t$
 5. Accumulate updates: $E[\Delta\theta^2]_t = \rho E[\Delta\theta^2]_{t-1} + (1 - \rho)\Delta\theta_t^2$
 6. Apply update: $\theta_{t+1} = \theta_t + \Delta\theta_t$
- RMS here is the root mean square. [cite Zeiler 2012]
-

where α is a tuneable parameter called *learning rate*.

Definition 2.11 (Feedforward fully connected layer). A feedforward fully connected layer is a trainable function with parameters $W \in \mathbb{R}^{N \times M}$ (weights) and $b \in \mathbb{R}^M$ (biases) that, in this case, maps a vector $x \in \mathbb{R}^N$ to an output $a \in \mathbb{R}^M$ via the following transformation:

$$a = W^T x + b \quad (14)$$

This is one of the simplest layers in a feedforward neural networks and input and output in it as mentioned above are vectors. However, in this study inputs and outputs are images, that are represented in memory as square matrices $x^{(i)}, y^{(i)} \in \mathbb{R}^{W \times H}$. One could simply flatten the image into a vector and use it as an input to a fully connected feedforward neural network. Nevertheless this would be a suboptimal approach.

Since essentially one of the main tasks of this research is to create a deep learning model that is able to predict a fluorescence image from a DIC image, the problem statement could be narrowed down to the following: predict an intensity high-resolution image from another intensity high-resolution image based on the features of the object morphology in it. Such problem is very common in the field of image analysis and one of the popular deep learning tools for solving such problems is convolutional neural network (CNN) or more specifically a UNet.

CNNs are able to capture nonlinear relationships over large areas of images, they greatly improve performance for image recognition tasks in comparison to classical machine learning methods [TODO cite Oukomol]. The word "convolutional" suggests that the convolution operation should be used in at least one of the layers there.

Definition 2.12 (Convolutional layer). A convolutional layer is a trainable function with parametrized kernel $K \in \mathbb{R}^{F \times F \times C}$ and bias $b \in \mathbb{R}$ that is usually denoted via the operator $(\cdot * \cdot)$. By transforming an input $x \in \mathbb{R}^{W, H, C}$ it produces an output S

$$S = K * x + b \quad (15)$$

that is called a *feature map* where an element on position (i, j) is defined as follows:

$$S_{i,j} = \sum_w \sum_h x_{m,n} K_{i-m, j-n} \quad (16)$$

Convolutional layer like a fully connected layer can be viewed a linear transformation as well. However, there are three main advantages that leverage convolutional layers for image processing in comparison to fully connected layers: sparse interactions, parameter sharing and equivariant representations. Image is a very redundant way of representing the semantic meaning hidden within it. Having a value of one pixel, the probability that the neighboring one will be of the same color is very high. Sparsity of interactions can be described by an example: usually a high-resolution image might have millions of pixels, however it is possible to detect smaller and very important features like contrast changes, edges, and shapes using a kernel consisting of only a few hundred pixels. By applying kernels (or filters) on the image locally, one will infer many of these features across the whole image. Such an approach reduces the memory needed for parameter storing and improves its statistical efficiency [cite DL-book]. Parameter sharing refers to the fact that instead of learning a separate set of parameters for every location within the image, will be learned only one set of parameters will be learned and applied across all image locations. Lastly, equivariance here means that convolution operation is equivariant to the shifts in the image.

Definition 2.13 (Stride). During the computation of convolution, the kernel starts sliding at the upper left corner of the input tensor, covering all locations while heading to the right and down. The step with which the window slides is called *stride*.

Definition 2.14 (Padding). When convolution is applied several points on the perimeter of the input tensor will be lost and the output tensor will have smaller spatial dimension than the input one. One can fix this by adding a few more pixels outside the perimeter, to preserve the dimension of the output to be same as input. The amount of pixels added is called *padding*.

Definition 2.15 (Max-pooling layer). Maximum pooling operation reports the maximum output within a rectangular neighborhood [cite DL-book].

Since adding two linear functions together would produce a linear function, it is important to use activation functions (or non-linearities) after each convolutional or linear layer like RELU, ELU, Tahn, Sigmoids and so forth. In CNNs they are also often combined with max-pooling layers and dropouts to escape overfitting.

Definition 2.16 (Batch normalization layer). Let's denote $B = \{x^{(i)}, \dots, x^{(m)}\}$ to be a mini-batch of data. Then batch normalizing transform applied to this input data would be:

$$\begin{aligned} a^{(i)} &= \gamma \frac{x^{(i)} - \mu_B}{\sigma_B^2 + \epsilon} + \beta \\ \sigma_B^2 &= \frac{1}{m} \sum_i^m (x^{(i)} - \mu_B)^2 \\ \mu_B &= \frac{1}{m} \sum_i^m x^{(i)} \end{aligned} \tag{17}$$

where γ and β are learnable parameters, μ_B and σ_B^2 are the mean and standard deviation of the batch. [cite Ioffe and Szegedy, 2015]

Definition 2.17 (Dropout layer). Dropout is a technique that randomly sets some weights (units) to zero [cite Srivastava, Hinton 2014]. It leads to the training of several smaller networks that share the parameters. If a mask vector μ specifies which units are included in training, then dropout's objective to be minimized becomes: $\mathbb{E}_\mu J(\theta, \mu)$. Visually dropout is presented in the Figure 2.

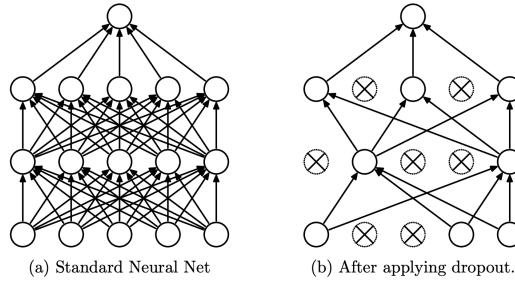


Figure 2: Dropout

Definition 2.18 (Activation function). An activation function is an element-wise non-linear function $f(\cdot)$. Some examples are:

$$f(x) = \frac{1}{1 + e^{-x}} \quad \text{Sigmoid} \quad (18)$$

$$f(x) = \max(0, x) \quad \text{Rectified linear unit (ReLU)} \quad (19)$$

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha * (e^x - 1), & \text{if } x \leq 0 \end{cases} \quad \text{ELU} \quad (20)$$

The models in this project mostly use ELU activations as ELU provides a better signal flow between the layers by not cutting off the negative values completely.

Definition 2.19 (UNet). UNet is fully convolutional neural network with U-shaped encoder-decoder network architecture. [cite Ronneberger].

The encoder is a common CNN, consisting of the repeated block of two 3×3 convolutions, followed by an activation function, and a 2×2 max-pooling operation with stride 2. At each encoder step the number of feature channels doubles. The decoder is also a CNN, consisting of repeated blocks of transposed convolution, that halves the number of feature channels, followed by a concatenation with a corresponding output from an encoder, and two 3×3 convolutions, followed by a ReLU. The last decoder layer is a 1×1 convolution to map the tensor to the number of output image channels needed. Skip-connections is a very important part of UNet as they allow to the flow of high-resolution features from the encoder to the decoder that in turn allows to restore a corresponding high-resolution image.

Definition 2.20 (Autoencoder). Autoencoder is an unsupervised learning technique in neural networks for the representation learning purposes. Autoencoder consists of an encoder that compresses data into a lower dimensional representation and a decoder that restores the original input from the encoded representation.

Definition 2.21 (Overfitting). "[A] Hypothesis overfits the training samples if some other hypothesis that fits the training samples less well actually performs better over the entire distribution of instances" (cite p67 Mitchell Machine Learning 1997). The way to avoid overfitting that happened to the models in Sections [TODO cite sections] are discussed in Section [cite regularization section].

2.2.2 Dimensionality reduction methods

This research additionally provides a study of the embeddings of a trained UNet and an autoencoder in Chapter [TODO cite chapter]. In order to better understand the visualizations, all dimensionality reduction methods that were used are listed and explained in this subsection.

Definition 2.22 (Embedding). An embedding in this context is an output tensor from the encoder part of the UNet or from an encoder part of an autoencoder.

The encoder output of the UNet is a tensor of size $16 \times 16 \times 256$ and after its flattening it turns into a vector of size 65536. The smallest autoencoder embedding was of size 200 [TODO check] which is also high-dimensional. One of the tasks of this research is to determine whether there are any interesting patterns or grouping based on various criteria hidden within the bottleneck embeddings, and whether they could be useful for further research. Yet in order for humans to comprehend the embeddings we need to map them either to 2D or 3D vectors. In this context dimensionality reduction algorithms are essential.

2.2.2.1 UMAP

Dimension reduction algorithms mostly form two main categories: ones are stronger preserving the pairwise distance globally — meaning they are trying to preserve the structure among all the data samples; others prefer to save local distances. For example, PCA [cite Hotelling] are assigned to the first category, while t-SNE [cite Ulyanov] and Isomap are assigned to the latter one.

Uniform Manifold Approximation and Projection (UMAP) was built in a way to preserve both and it is a competitor of the t-SNE approach. However is much faster and provides a transformation that can be used on the new data. UMAP is a graph-based algorithm and uses a k-nearest graph as its foundation. As with any graph-based algorithm, its structure also includes two main steps:

- Graph construction procedure. During this stage a weighted k-neighbour graph will be constructed from the data. Specific transformations are applied on its edges to surround local distance. And the strong asymmetry common to k-neighbour graphs will be reduced.
- Graph layout building. In this stage one first needs to define an objective function that can preserve the desired graph characteristics and then find a low dimensional representation of the graph that will minimize the objective.

In short, UMAP optimizes a low-dimensional graph from the high-dimensional one to be structurally very similar to each other. The algorithm has two important hyperparameters, which should be chosen carefully: $n_neighbors$ and min_dist . The first one balances the local versus the global structure of the graphs; the higher the values the more fine details will be lost. The latter one controls how densely points will be located to one another. Higher values of this parameter result in a looser structure that preserves a broader topology of the data.

2.2.2.2 PaCMAP

Pairwise Controlled Manifold Approximation (PaCMAP) is another dimensionality reduction method that is able to preserve both local and global data structure in a lower dimension space. Unlike other methods that regulate the stronger preservance of global structure by using more neighbors, PaCMAP uses mid-near pairs to first capture global structure and then refine local structure, which both preserve global and local structure. It introduces the following parameters: neighbor pairs (pair_neighbors), mid-near pair (pair_MN), and further pairs (pair_FP). [cite Yingfan] The neighbor pairs parameter is used during the building of the k-nearest neighbor graph. It is recommended to use a value of around 10 for datasets with a size smaller than 10000 [cite their repo]. The mid-near pair ratio parameter is the ratio of the number of mid-near pairs to the number of neighbors, whereas further pairs ratio is the ratio of the number of further pairs to the number of neighbors. Configuring these parameters allows the user to achieve the desired ratio between preserving local and global structure. This method also works faster than UMAP, which allows to try out more hyperparameter options.

2.2.2.3 PCA

Principal component analysis (PCA) is an algorithm for linear dimensionality reduction [cite Pearson 1901]. PCA maximizes the variance in data's low-dimensional representation in order to keep as much information as possible. Essentially, PCA gives projections $\tilde{x}^{(i)}$ for input samples $x^{(i)}$ that would be very similar to them, however have a much smaller dimensionality. Eigenvectors of the data covariance matrix are the directions of the most variance within the data, and the eigenvalues corresponding to them are the amount of variance hidden in each dimension. That is why by projecting the data using the eigenvectors with the largest eigenvalues, one will preserve the most variance of the data possible [cite MML-book].

The steps of PCA algorithm are the following:

- Subtract mean μ_d . Centering the input data is not a necessary step, but it is recommended to do so to avoid numerical problems.
- Standardize the data. Calculate the standard deviation σ_d and standardize the data to have unit variance for every dimension.
- Do an eigendecomposition of the data covariance matrix. To do so one must first compute the covariance matrix itself, since the covariance matrix is symmetric from

the spectral theorem one can always find an orthonormal basis of eigenvectors.

- Project the data. First, standardize the point $x^* \in \mathbb{R}$ using μ_d and σ_d :

$$x_d^* \leftarrow \frac{x_d^* - \mu_d}{\sigma_d} \quad (21)$$

where $d = 1, \dots, D$ and x_d^* is a d -th component of vector $x^* \in \mathbb{R}^D$. Get the projection as

$$\tilde{x}^* = BB^T x^* \quad (22)$$

with coordinates $z^* = B^T x^*$. Here B is a matrix of eigenvectors associated with the biggest eigenvalues of a covariance matrix.

[cite MML-book]

2.2.3 Clustering

After visualizing the embeddings, there is an interest in checking whether they form any kind of clusters. This question is discussed in Section [TODO cite the section] using DBSCAN algorithm. This part will provide the theory needed to understand how this algorithm works and how it can be set up.

2.2.3.1 DBSCAN

A density-based algorithm for discovering clusters (DBSCAN) does not require the provision of the number of clusters in advance. Although this is a nice quality of this algorithm it is not that important for current research. The goal of Section [cite Section] is to check whether different phenotypes form different clusters or, for example, whether corrupted images would fall into a separate cluster. In all cases the number of ground truth clusters is known in advance. However, the fact that corrupted images may form a denser region in their embeddings in comparison to non-corrupted ones [cite Section] may be an indicator that exactly DBSCAN would give a good clustering result.

This algorithm uses two hyperparameters = {eps, min_samples} to define the clusters. The first is the distance threshold, which is used to determine whether a point is located in the neighborhood of the other point. The latter one is the minimum number of points that are needed to form one cluster. DBSCAN clusters points not only into several clusters but also determines the points that could not be assigned to any cluster, which is also very useful in this research.

2.3 Imaging

2.3.1 Digital imaging

Digitally an image is represented as an array of size (H, W, C) where H is the height, W is the width and C is the number of channels of the image. In this work, $C = 1$ and $W = H$. A digital image A can be represented with the matrix:

$$A = \begin{bmatrix} a_{0,0} & \cdots & a_{0,W-1} \\ \vdots & \ddots & \vdots \\ a_{H,0} & \cdots & a_{H-1,W-1} \end{bmatrix} \quad (23)$$

where $a_{i,j} \in \mathbb{R}$. Both DIC and fluorescence images were provided in tag image file format (TIFF). For the processing convenience purpose all images were normalized to be in the range of $[0, 1]$:

$$a_{i,j}^{\text{norm}} = \frac{a_{i,j} - \min(A)}{\max(A) - \min(A)} \quad (24)$$

for $\forall i \in \{0, \dots, W-1\}$ and $\forall j \in \{0, \dots, H-1\}$

2.3.2 Microscopy imaging

2.3.2.1 Image acquisition peculiarities

The cells used in this research are growing in 96-well plates. A plate or a microplate in biology is a flat plate with multiple tubes ("wells"). The microscope used in the experiments takes photos of the well plate in random locations. The reason for that hides in the focusing settings of a microscope. To get a reasonably good, and not blurry, photo, a microscope has to focus on a specific location of the plate. The choice of this location however happens automatically, therefore the location of the focus is random (see Figure 3).

It might be problematic in the following sense: photos taken in such a manner do not guarantee that the focus will land in distinct spots all the time. Meaning that some cells present in one of the photos might appear in the other ones as well. Since the photos are high-resolution they will first be split into crops of size 256×256 each during the preprocessing. It might happen that the same cells appear in several crops. That is why after the split of the image data between train, test and validation sets it might also happen that the same set of cells will once land in the train set and another time in the validation set, which will lead to a not completely fair and representative validation metrics during training.

In order to overcome this problem much more expensive equipment is needed. Since it does not cause fundamental problems in this case, except for the fact that the validation metrics might be lower than what they should have been, there was no need to purchase more expensive tools.

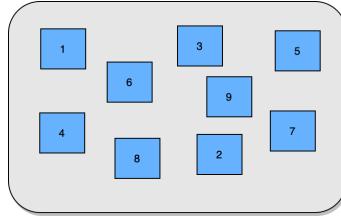


Figure 3: Way in which photos of the well-plate were taken

2.3.2.2 Crops combination technique

Due to the restricted amount of memory on the GPU deep learning models cannot have a high-resolution image as their input in the scope of this research. Yet this is also not obligatory: as the image contains dozens of cells within it, its processing can be limited to a crop of a smaller size. After the model has predicted fluorescence signal for each of the crops, output fluorescence images can be combined together to form a high-resolution image again. In this thesis the architecture of the model assumes an input size of (256, 256) or more specifically (*None*, 1, 256, 256), where the first dimension is responsible for the batch size and the second one states that the input is a 1-channel image.

There are several ways of how one can split the image, the easiest approach would be to use a sliding window of size w . This algorithm is depicted in Figure 4. A small window starts sliding the image from the upper left to the lower right corner with step size s feeding the selected crops into a deep learning model. From the output of the model only a center part of such a crop is accepted to form a full fluorescence image. Border size b in this case is the size of the edges of the crop that are not accepted from the predictions of the deep learning model.

Accepted areas from each of the crops have to follow each other without any space in between. In order to achieve that if the border size has been defined in advance, one has to set the step size to in the following way:

$$s = w - 2 * b \quad (25)$$

When step size s is equal to window size w , there is no overlap between the windows.

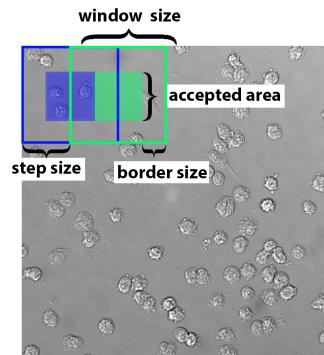


Figure 4: Sliding window approach for fluorescence prediction

The reason why the full prediction is not accepted to form the output lies in the following: trained models are less accurate on the borders of the crops rather than in the center. Most of the times there are cells on the borders of the crops that were sliced and therefore it might be impossible to make a good prediction for them just due to the lack of input information. Therefore, the step size has to be smaller than the window size, so that the windows are overlapping and for each prediction we use only the image center and are allowed to ignore predictions on the border (see the comparison between different border sizes in Figure 5). This is discussed in more detail in Section [TODO reference the section]. Such an approach helps to reduce the effect of grid visibility on the image composed of many small crops. This can be seen in the left part of Figure 5 as opposed to the non-visible borders in the same Figure on the right. This would of course take more time to create the predictions, however, the speed is less crucial in comparison to the accuracy of the predictions.

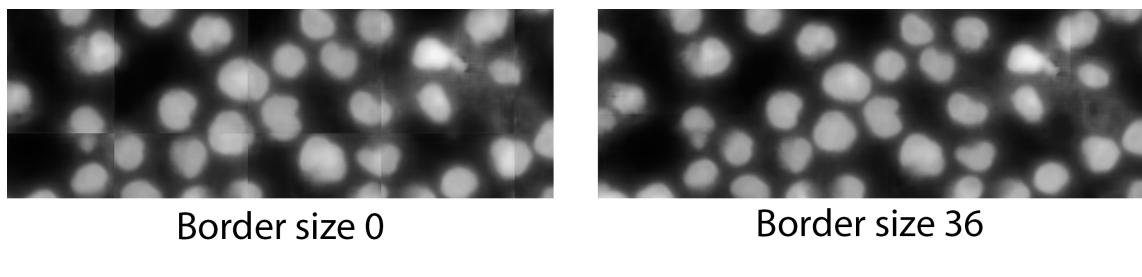


Figure 5: Difference of overlap between predictions on the resulting image

3 Implementation and experiments

In this chapter the results of all experiments performed for predicting fluorescence signal from 4 cell organelles: nuclei, endoplasmic reticulum, Golgi apparatus, and full cell fluorescence are provided and discussed. It starts first with a description of the models and data used in the experiments, followed by 4 subsections dedicated to each of the organelles. Each subsection describes its own different approaches in, for example, pre- or postprocessing needed, difficulties that occurred during preparation or training steps and well as results obtained for each organelle separately. At the end of this chapter an alternative way of the models performance estimation is provided, which is very important for the practical application of them.

3.1 Model training

3.1.1 Neural network architecture

As was described in section 2.2 the choice of the model architecture fell onto the UNet structure. Here will be provided a detailed description of the architecture and the different layers used. An architecture used in this research was chosen based on paper [cite LaChance]. The input to this network is a 256×256 -pixel DIC image that should be already preprocessed based on the corresponding to the desired organelle preprocessing pipeline. Specifications about different preprocessings are described separately in the next subsections dedicated to different organelles.

The encoder part of the UNet (Figure 6) step by step compresses spatial dimensions of the image (the spatial dimension size is denoted by a number on the left of each green block) into tensors or so-called feature maps with an increasing amount of filters (number of filters is denoted on the top of each green block). This allows to reduce the spatial information in the image and capture semantics. Decoder part on the contrary decompresses feature maps gradually increasing the amount of spatial information in tensors and reducing the number of filters. All convolutional layers use convolutions of size 3×3 with the corresponding number of filters. Downsampling in encoder reduces the spatial dimension twice during each step and implemented using max-pooling with a size of 2×2 . Upsampling in decoder increases the spatial dimension also twice during each step and is implemented using transposed convolutions with a size of 2×2 . After the first convolutional, that follows each max-pooling step, a batch normalization layer was used as it is well-known for speeding up the training process [cite Ioffe]. One should not forget though that using batch normalization might be sometimes dangerous due to the hidden information leaks it induces [cite Fetterman 2020]. Additionally dropouts were used for example for actin predictions as the model would encounter overfit quite easily there, however in the default architecture dropouts are not present. That is another thing that differs this architecture from the original one in [TODO Cite LaChancel] paper. The last layer of the UNet here is a sigmoid activation function.

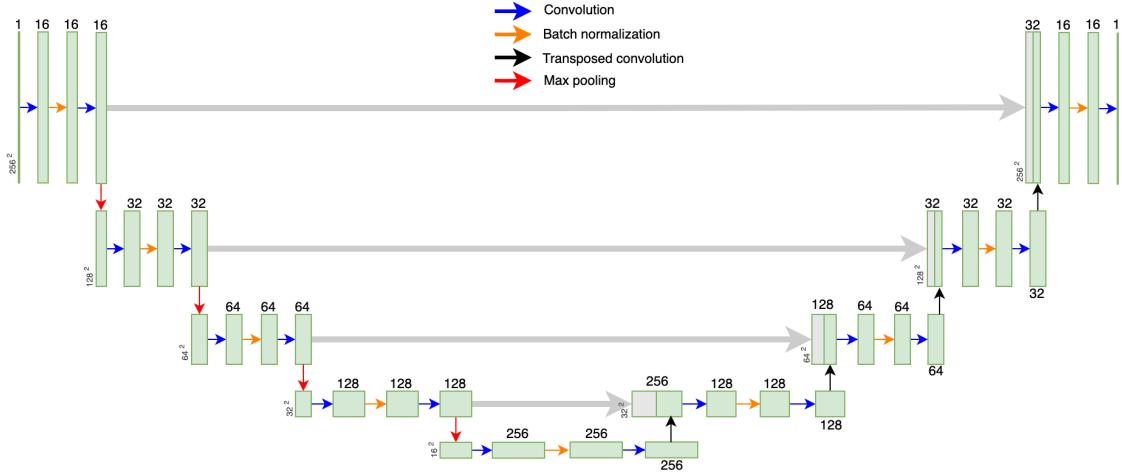


Figure 6: Unet

There is a space for potential improvements regarding the model architecture used in this research: for example, [Shiyui 2021] recommends to use special dense-block after each convolutional layer. They consist of another 3 convolutional layers with 24 filters, batch normalization layer and ReLU activations each. This could potentially facilitate efficient training of the model, still most probably the efficiency comes mostly from Batch normalization layers that are already used in our architecture. Nevertheless the idea of using a bigger model such as one in [Shiyui 2021], or more specifically a model with more filters, indeed improves the predictions (shown in Section [TODO ref section]). That leaves the place available for further research and improvements regarding the size of the model and additional use of dense-blocks.

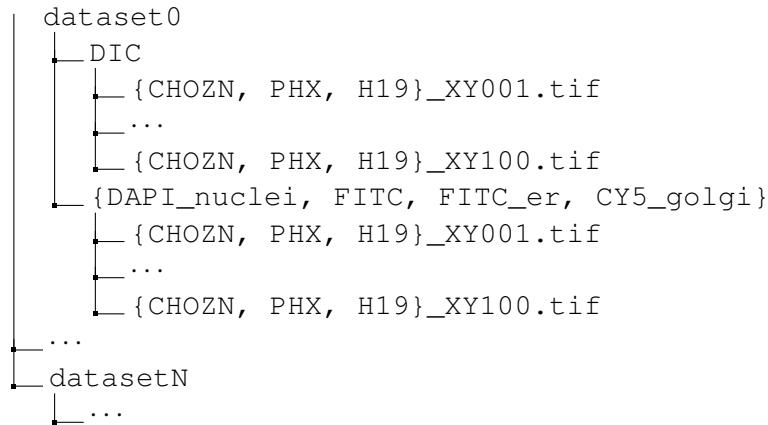
An interesting question that automatically rises here is what do UNet embeddings (output tensors from the encoder) represent. There is a big difference between any representation learning network such as an autoencoder and a UNet — a UNet model uses skip-connections that allow to propagate information between its encoder and a decoder. Meaning that embeddings do not contain purely semantic information, because essentially the network is not pushed towards compressing the information severely (as an autoencoder would do), but it should only extract relevant for segmentation features. One of the questions solved in this work was whether or not UNet embeddings are clustering based on the following classes: cell phenotypes, any kind of corruption within the data. For example, it would be very useful to be able to not only predict the data itself, but also to say whether the prediction is reliable or not. The initial hypothesis here would be that if the predictions are not of a good enough quality this would be also reflected in the embeddings.

3.1.2 Available data

There were 4 targets studied in this research: nuclei, endoplasmic reticulum, Golgi apparatus and a full cell target. First 3 targets provide many insights on the state of cell based

on their quantity, size and the intensity of fluorescence signal that they can produce by binding with the protein, the latter one is useful in general for locating cells and calculating their area. There are of course other possible targets that could be used during the selection step of CLD, but the goal of this particular study was to provide the proof of concept on whether the network could potentially substitute manual fluorescence staining in general. The hypothesis on which we rely here is that if the model can successfully predict fluorescence signal from some of the targets, then this research could be extended to other cell organelles as well.

The imaging data comes always in pairs (DIC + fluorescence imaging) and has the following structure:



Here a dataset $\{0, \dots, N\}$ is one 96-well plate. There are only several datasets for each of organelle and each of them contains 100 images on random locations within this plate. Each dataset includes 2 subfolders — DIC and a fluorescence subfolder. The name of the latter one corresponds to the name of the fluorescent binding molecule as they are different for each target within the cell. Almost all plates contain pairs of images from one fluorescent staining target only. Each filename within the subfolder includes its index number between $\{1, \dots, 100\}$ and the phenotype of the cell. "Cellular phenotype is the conglomerate of multiple cellular processes involving gene and protein expression that result in the elaboration of a cell's particular morphology and function" [TODO cite Jai-Yoon Sul 2009]. There are only 3 different phenotypes present across the datasets: CHOZN, PHX and H19. The latter however one is present only for full cell target.

Table 1: Available data for each of the organelles

	Total images	Training crops	Validation crops	Test crops
Nuclei	595	27,264	3,008	7,616
Actin	400	18,432	2,048	5,120
Golgi	761	23,036	2,336	6,347
H19	400	18,432	2,048	5,120
Nucleolei	?	?	?	?

For the training all images are cropped into smaller crops of size 256×256 and split between training, test and validation sets with crops from one image being present in only one set. However it is still possible that the cells themselves might be mixed up between the sets, because of the randomness of the microscopy focusing system (see

subsection 2.3.2.2). The summary of the total number of crops in each set is presented in Table 1.

3.1.3 Training costs estimation

For training purposes in this research cloud computing services were used, or more specifically - Amazon Web Services (AWS). These remotely located servers provide an possibility to train your models on a variety of graphic cards paying for a minute rate. In order to keep the costs under the control it was important to estimate the time needed for training in advance to choose the most efficient GPU possible.

Cost estimation is additionally important for the model inference as well during the production. Although for production purposes the lambda functions from AWS can be used. They can be triggered only when the inference request arrives and can be turned off automatically shortly after.

For both training and inference purposes two GPU models were tested g3.4xlarge (NVIDIA Tesla M60) and p3.2xlarge (NVIDIA Tesla V100). The datasets on which the experiments were performed are the full cell target training and validation dataset. The resulting costs are presented in the Tables 2, 3.

Table 2: Costs estimations of AWS use for training models

	Runtime (1 epoch)	Dataset size	Costs per minute	Cost per epoch
g3.4xlarge	6.5min	18,432	0.07125\$	0.3
p3.2xlarge	57sec	18,432	0.1911\$	0.18

Table 3: Costs estimations of AWS use for inference purposes

	Runtime	Dataset size	Costs per minute	Cost of inference
g3.4xlarge	?mins	2,048	0.07125	?\$
p3.2xlarge	?mins	2,048	0.1911	?\$

In conclusion even though a p3 instance seems to be much more expensive, it is also much more efficient and the costs estimated to training times are significantly lower. That is why all the training experiments in this research were conducted on an NVIDIA Tesla V100 GPU.

3.1.4 Augmentations

Augmentations is a powerful regularization technique that helps the network to generalize better [cite Wang 2017] and is an effective solution for a situation with the lack of labeled data [cite Yang 2022]. The main idea behind the augmentation is to increase the diversity of training data, when the acquisition of the real training data is expensive. Augmenting existing images creates the new synthetic samples which are hypothetically very close to the original true distribution of images. However one should be very cautious regarding the type of augmentations that can be used. Augmentation must enrich the dataset, however

should not change the semantics hidden in each image. The peculiarity of current research is the pair-wise correspondense between the DIC input and the output fluorescence. One should not change the input in such a way that the fluorescence signal could not be inferred from it. Therefore the following augmentations have been used on cropped images:

- **Flipping:** Flip the crop horizontally (30% chance), vertically (30% chance) or both.
- **Random rotation:** Rotates the crop by a random angle.
- **Random scaling:** Reduces crop size by 100 (20% chance), 50 (20% chance) or 20 (60% chance) pixels from each side (down, top, left, right).
- **Contrast:** Decreases an image contrast in twice (50% chance), or triples it (50% chance). Applied with 20% chance.
- **Defocus blur:** Imitates a defocus blur of severity level 4 (see section [TODO cite section]) on an image with 20 % chance.

3.1.4.1 Special augmentations for rotation and scaling

Augmentations are chosen to be applied during training and the reason for that is to preserve the same conditions regarding the size of the datasets in order to have a fair comparison between approaches. For example, one decides to augment 20% of images and add them to an original dataset in order to expand it. Then the comparison between training with and without augmentations would be unfair as the sizes of the datasets are different, and it is not clear whether the performance improvement or decrease comes from the longer training (enlarged dataset) or augmentations.

Since augmentations are applied on crops directly during training there is a possibility to improve the rotation and scale of crops. The problem with this augmentations are depicted in the Figure 7 - after applying them there will appear a gray background, that would be filled with zeros in PyTorch implementation. However since one has an original image where the crop has been cut out, one can easily restore background with original values and escape this problem altogether.

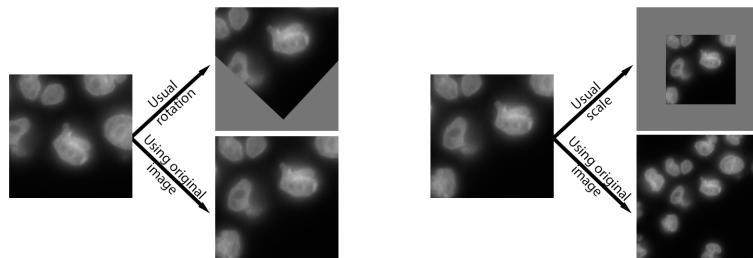


Figure 7: Using original image for rotation and scaling augmentations

Applying augmentations in combination with other regularization techniques has proven to be helpful against overfitting for the nuclei and ER training. Therefore it is recommended to use it in further research as well.

3.1.5 Model setup

3.1.5.1 Weight Initialization

In order to achieve best predictions results it is very important to pre-setup a model correctly. Since the architecture used here is very similar to the one used in LaChance paper, the setup configuration is similar as well. Weight initialization plays a very important role in model training. Even on the simplest model wrongly initialized weights (for example all constant or too large or too small) can lead to very slow convergence or prevent the model from converging at all [cite Kumar 2017].

Xavier initialization that is usually a default choice in many neural networks works well mostly for fully connected layers with tanh as activation function. There is also a study providing some insights into why Xavier initialization may not be the optimal choice for ReLU activations [cite Kumar 2017]. In the following samples *fan_in* denotes the maximum number of input signal units to a given layer and *fan_out* is the maximum number of output signal units from it. You can find a definition of Xavier initialization below:

Research of [cite Kaiming He 2015] also notices the problems with Xavier initialization for ReLU activations. The authors suggest a new robust method called He initialization that enables training of even extremely deep or wide network architectures with ReLU activations. This method was suggested by the [LaChance 2020] paper and has been used in this research as well. He initialization draws samples from a truncated normal distribution:

$$N(0, \sqrt{\frac{2}{\text{fan_in}}}) \quad (26)$$

Default weight initialization of Conv2D layers in Python claims to use the initialization method of [cite He 2015], by calling it Kaiming uniform initialization, however after careful study of the source PyTorch code one can quickly find out that used initialization approach has nothing to do with the He initialization at all. Even the samples are even drawn from the uniform distribution, although the paper clearly states that a normal distribution should be used. The definition of Kaiming uniform is provided below:

$$\text{std} = \sqrt{\frac{2}{\text{fan_in}}} \quad (27)$$

$$\text{bound} = \sqrt{3 * \text{std}} \quad (28)$$

$$\text{Uniform}(-\text{bound}, \text{bound}) \quad (29)$$

However the naming is quite confusing and therefore two experiments have been conducted: first the model predicting nuclei target was trained with the default weight initialization provided by PyTorch and then the initialization was switched to a true He initialization.

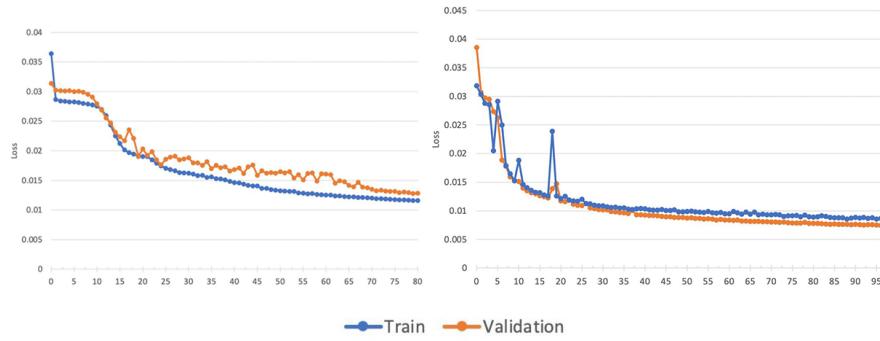


Figure 8: Nuclei training without (left) and with (right) custom weight initialization

The results of the experiments are presented in Figure 8. As one can see the loss in the left plots stagnates during the first few epochs and then starts to converge later. This is a symptom of a wrong initialization of the weights. Even after convergence starts the model still has a higher loss than the one on the right (around 0.012 in comparison to 0.007). However the model on the right is not completely perfect as the loss still doesn't converge at the same speed everywhere. Yet this might be not related to the weight initialization but more to instability of the training in general as there were only few images used for these experiments.

3.1.5.2 Regularization

Regularization is mostly used to prevent the deep learning model to overfit on the training data and to be able to generalize well. The overfitting has happened to the models used in this research and therefore it is important to understand the techniques that can be used against it. There are several approaches to regularize the model and they will be explained below.

- Early-stopping

Overfitting can be detected via visualizing train and validation losses. Training behaviour at first will be the usual one, meaning that both train and validation loss are gradually decreasing, however at some point the train loss continues to decrease, whereas the validation loss suddenly starts to increase (see Figure 23). Since the model hasn't seen any of the data from the validation set, it means that it loses its capability to generalize on the unseen data, while improving its performance on the seen data (train set). This doesn't happen during earlier epochs. Assume that during training the model learns a complex decision surface, with the correct weights initialization the weights of the model will be quite small and random and therefore the best decision surface during the first epochs would be a smooth one. But during the later ones the difference in values of the weights grows and they become not similar anymore which also means that the decision surface becomes more complex and the model is now able to fit not only the training data itself, but

also its noise [cite p111 Mitchell Machine Learning 1997]. And that is why stopping before the model became too complex, meaning to stop before the overfitting point, mitigates this problem.

- *L1- L2-regularization*

The complexity of the deep model grows with the number of features it uses, sometimes the model may pay attention to the features that are not important to the outcome, or even considers a noise to be a feature. To prevent this one should decrease the weights associated with useless features, however one cannot know ahead which of them should be ignored, therefore one may limit them all [cite Ying 2019]. In order to do that, a penalty term in loss function is added:

$$\tilde{L}(Y, M(X, \theta)) = L(Y, M(X, \theta)) + \lambda R(\theta) \quad (30)$$

for some $\lambda > 0$. This is called a *soft-constraint* optimization. When $R(\theta)$ is of the form $R(\theta) = \|\theta\|_2^2 = \sqrt{\sum_i \theta_i^2}$ this is called *L2-regularization*. When it is of form $R(\theta) = \|\theta\|_1 = \sum_i |\theta_i|$ this is called *L1-regularization*. *L2-regularization* used in combination with backpropagation is equivalent to weight decay. Weight decay is defined by [cite Hanson and Pratt 1988] as follows:

$$\theta_{t+1} = (1 - \lambda)\theta_t - \alpha \frac{\partial L}{\partial \theta_t} \quad (31)$$

where α is a learning rate. Weight decay successfully affects more those weights the gradient change along which is smaller [cite DL-book p229]. *L1-regularization* induces sparsity of the weights by assining some of them to zero, this could be also considered as a feature selection approach.

- Regularization layers

Batch normalization and dropout layers are also considered to be a form of regularization.

- Network reduction

Since learning a too complex and noise-fitting decision surface might be an often a cause of an overfit, another way to mitigate it would to be reduce the space of the possible decision surfaces and therefore make the surface simpler so that it cannot fit into the noise from the data. By changing the number of adaptive parameters in the network, the complexity can be varied [cite Page 332 Bishop book].

- Expansion of the training data

For a successfull training a model needs to have a sufficient amount of quality samples. An expanded dataset can improve the quality of the predictions [cite Ying 2019], however only when the model has already performed well on the initial dataset. If the model was performing badly initially, adding more data will not solve the problem. Here having 27,264 crops of data the model was trained on 5,376 crops only (2 96-well plates) to find the best structure and regularization first, afterwards the model was retrained using more data and the PCC loss improves from 0.77, to 0.93.

3.1.5.3 Optimizers

It is also important to choose a correct optimizer in order for model to converge as fast as possible. Here 3 different optimizers have been tried out — SGD, Adam and Adadelta optimizers. As the result SGD optimizer performed the worst, while Adam and Adadelta optimizer performed quite similarly with Adadelta converging to slightly better values at the end. Adam optimizer has required some fine-tuning of the learning rate from 0.001 to 0.0001 to achieve the best result. Both Adadelta and Adam can be used for model optimization in for this dataset. The experiments were conducted on the truncated dataset of nuclei images using PCC loss.

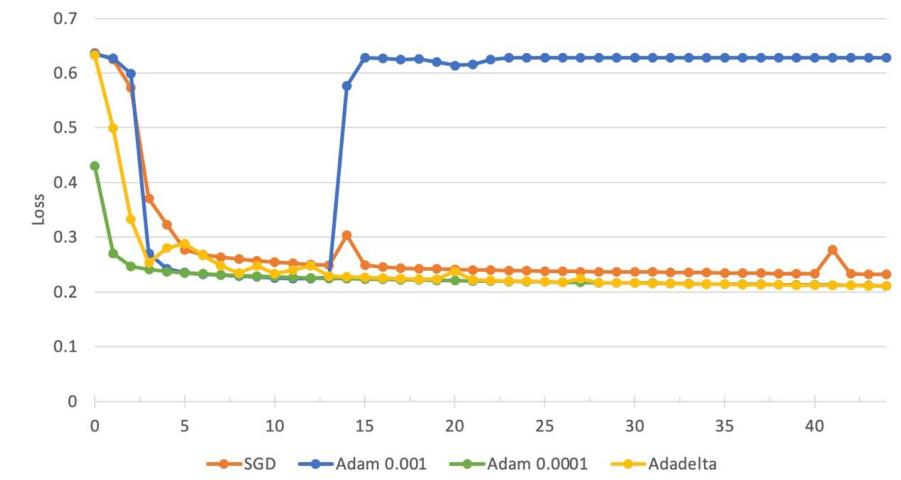


Figure 9: Comparison of convergence for different optimizers

3.2 Nuclei

A nucleus (plural nuclei), as related to genomics, is the membrane-enclosed organelle within a cell that contains the chromosomes. The nucleus is one of the easiest organelle to detect within the cell as it is usually located in the middle and occupies quite a big area of the cell. Nucleus contains all of the cell's chromosomes, which in their turn encode the genetic material, therefore nucleus is a very important organelle (*Nucleus n.d.*). In order to stain it, DAPI was added to the cells. This is a fluorescent stain that binds strongly with some regions in DNA. Analysis of cell's nucleus can provide many valuable insights, for example the radius of living cells is on average bigger than in dead ones (Christiansen et al., 2018). With fluorescence labeling one can derive useful features for determining whether the well plate should or should not be selected during the selection step in CLD process.

3.2.1 Preprocessing

As not all of the dataset samples were of the same quality it was important to first filter the dataset to remove the bad imaging samples. Even with the normal conditions many of the images contain a lot of background, that creates a big background vs. foreground class imbalance (see Figure 10a). Overexposure is also a typical problem, that gives samples of a too high intensity and without details inside the nucleus (see Figure 10b, c). Lastly, underexposure is as problematic as an overexposure (see Figure 10d).

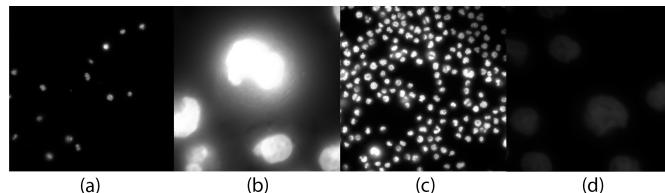


Figure 10: Samples to be filtered out

Once the images have been filtered out, they were normalized to have the values between 0 and 1.

3.2.2 Training and predictions

3.2.2.1 Convergence

As the nuclei dataset is one of the biggest ones (see Table 1), the experiments would first be performed on the subset of the data, and only once the training pipeline and good results were established, the training would be done using the full dataset. Figure 8 (right) represents the training of nuclei using only two 96-well plates with MSE loss. The training is very unstable in the beginning, but one can clearly see that the model

successfully converges. Our hypothesis behind the instability was the clear lack of the amount of training data, which has been proved by further training using the full data and as a result getting a more stable PCC loss there (see Figure 11).

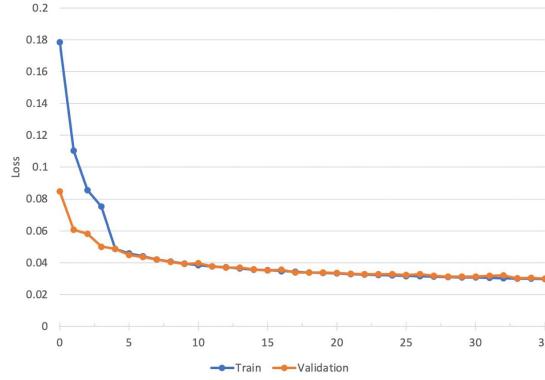


Figure 11: Having more data makes training more stable

As mentioned in definition 2.6 for correct understanding of further plots, one should be careful with differentiating PCC loss from PCC itself. PCC loss converts PCC to be between 0 and 1 with 0 being an optimal value.

Seeing that the model significantly stabilizes with the use of more data and that the prediction results become much more similar to the ground truth (see Figure 14 *small dataset* vs. *full dataset*), it was decided to try the use of augmentations in order to enlarge the dataset even more. In this case the augmentations were used not in order to regularize or stabilize the training (by providing more difficult, for instance, blurred samples), but to simply have more data. Training and validation PCC losses from the train with the use of the augmented data are presented in Figure 12. The augmentations used here are horizontal and vertical flips, rotations and crops (described in detail in section 3.1.4).

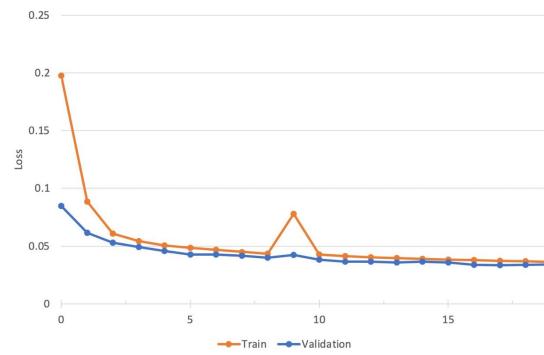


Figure 12: Adding simple augmentations in the dataset

Validation PCC loss has slightly increased to 0.038 in comparison to a previous 0.0365. However the validation set on which this loss was estimated also includes augmentations

mentioned above and, therefore, presents a bit more difficult task, then the original validation set. This is confirmed by measuring the loss of the models on the same dataset, where PCC for has improved from 0.92 to 0.93 (or PCC loss from 0.0365 to 0.032).

In next experiment the model has additionally been regularized by adding dropout layers and using a weight decay of 0.0001 (see Figure 13). This didn't bring a better result, but have only made it more difficult for model to capture needed feature to reproduce the fine details within the nuclei. However, this brought up a new hypothesis, that the model simply might have not enough of capacity to capture enough of the details. In order to confirm this a bigger model had to be trained.

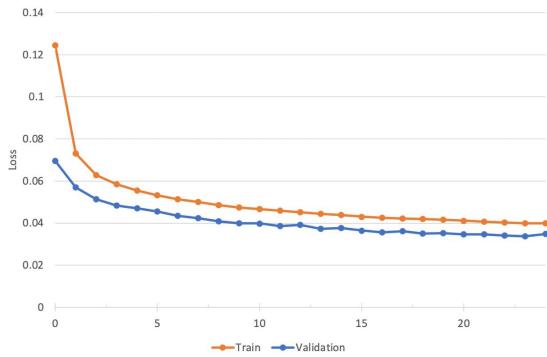


Figure 13: With regularization and augmentations

Indeed, increasing the number of filters in the first convolutional layer from 16 to 64 made the model to be able to capture more details. Predictions became better both visually (see Figure 14) and based on both metrics (PCC and MSE losses).

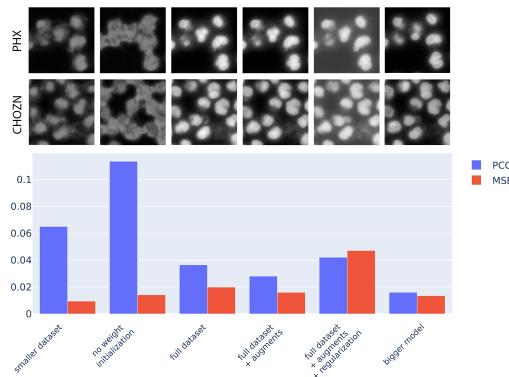


Figure 14: Different models predictions and scores comparison

Interestingly observations made in this section made it clear that indeed metrics used for training (PCC and MSE losses) are not representative enough to derive any conclusions regarding the model quality from them. Just by looking at the Figure 14 one can see that the for example training on the full dataset of data gives much better results than training

on the small dataset. Yet MSE seems to be bigger for this experiment. As well as it is not representative in comparison of the model without the correct weights initialization vs. the regularized model with augmentations. MSE loss is much higher there, because in general the image became somewhat brighter, even though the quality of the nuclei is significantly better. PCC loss seems to represent the desired quality of the model better. This trend has been noticed by Lachance et al., 2020 as well. They notice that values of PCC lack the practical context — which value would be good enough and good enough in which context? This issue has been addressed here as well in section 3.6, where more practical metrics are suggested and the model evaluation on them is provided.

3.2.2.2 Predictions quality

Blurry, boundaries, not enough of details and possible improvements

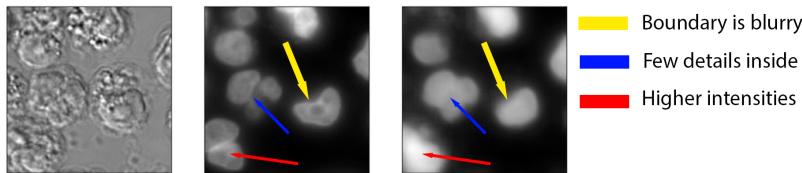


Figure 15: Problems in predictions

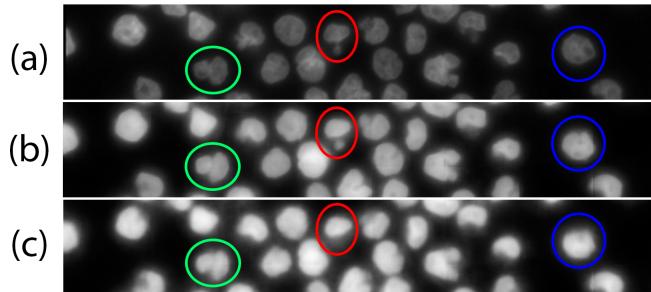


Figure 16: Predictions improvement

3.2.3 Postprocessing for nuclei segmentation

The most extreme corruptions mentioned in section 3.2.1 were removed, yet some of the images that are corrupted not that severely (meaning they still have all the visible features needed for learning) were left in the dataset in order to not critically reduce the amount of data. The examples of these images are shown in Figure 17.

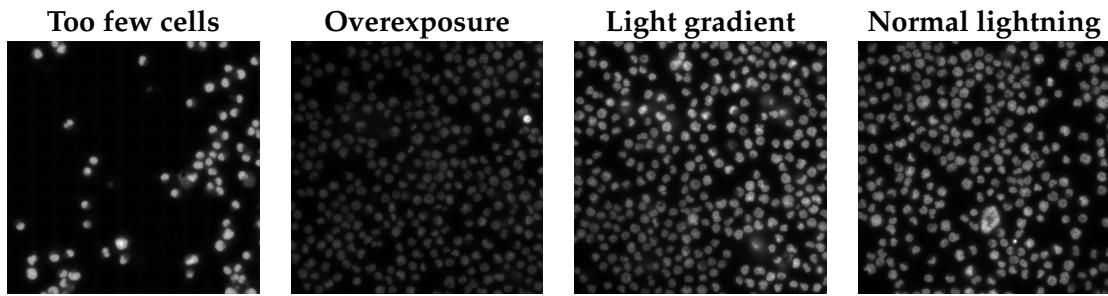


Figure 17: Different lightning conditions

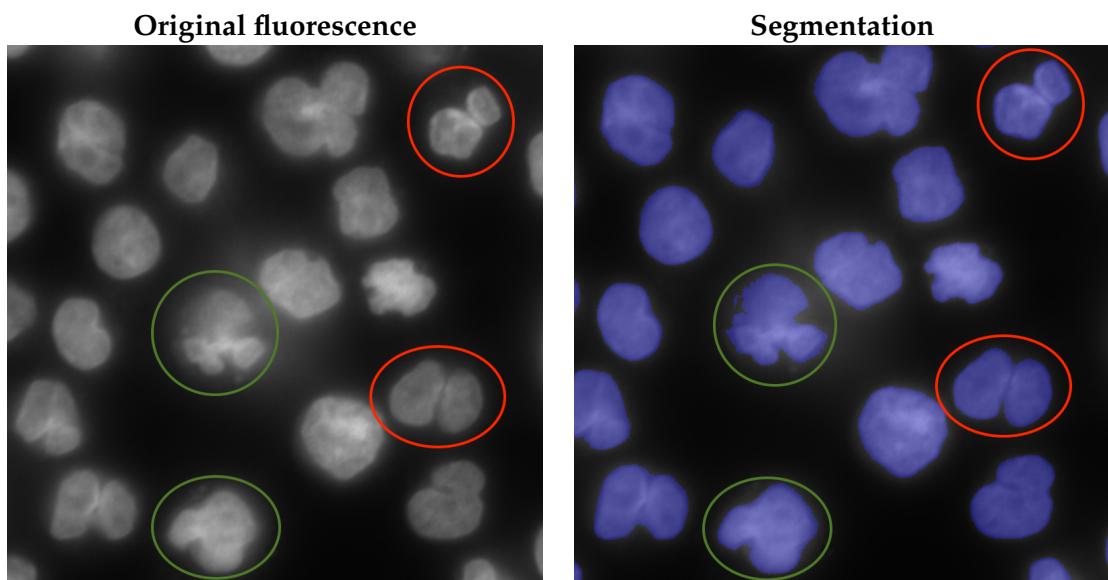


Figure 18: Closely located cells

Overall algorithm

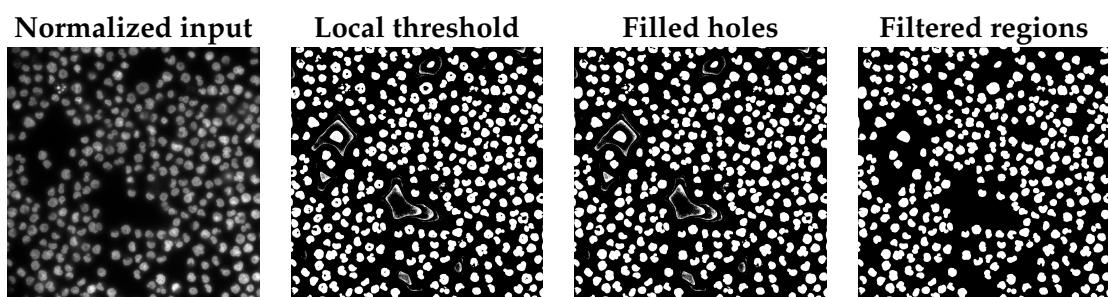
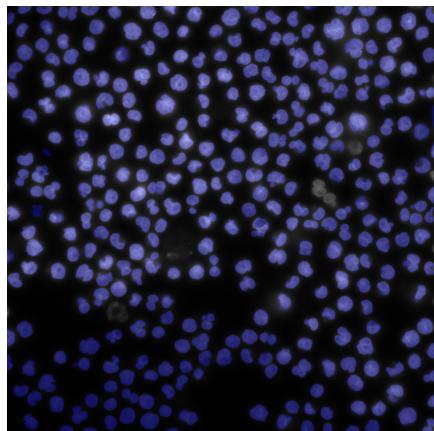


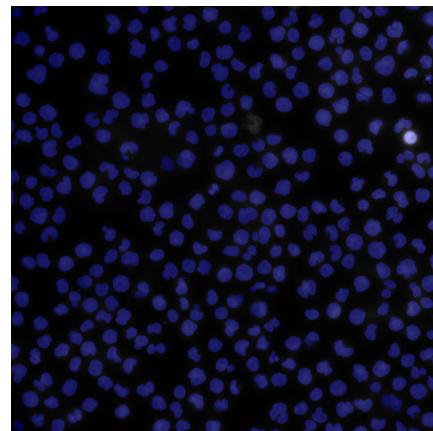
Figure 19: Fluorescence segmentation

3.2.3.1 Thresholding algorithms

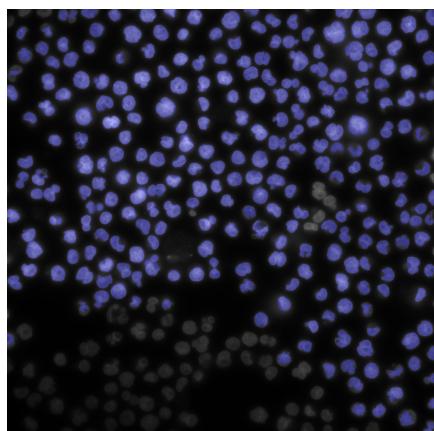
Global and local thresholding



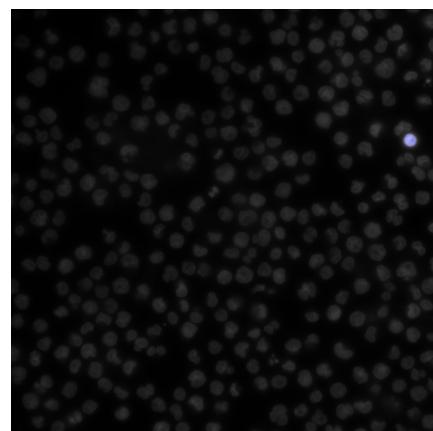
(a) Local



(b) Local



(c) Global, minimum



(d) Global, maximum

Figure 20: Local vs. Global thresholding

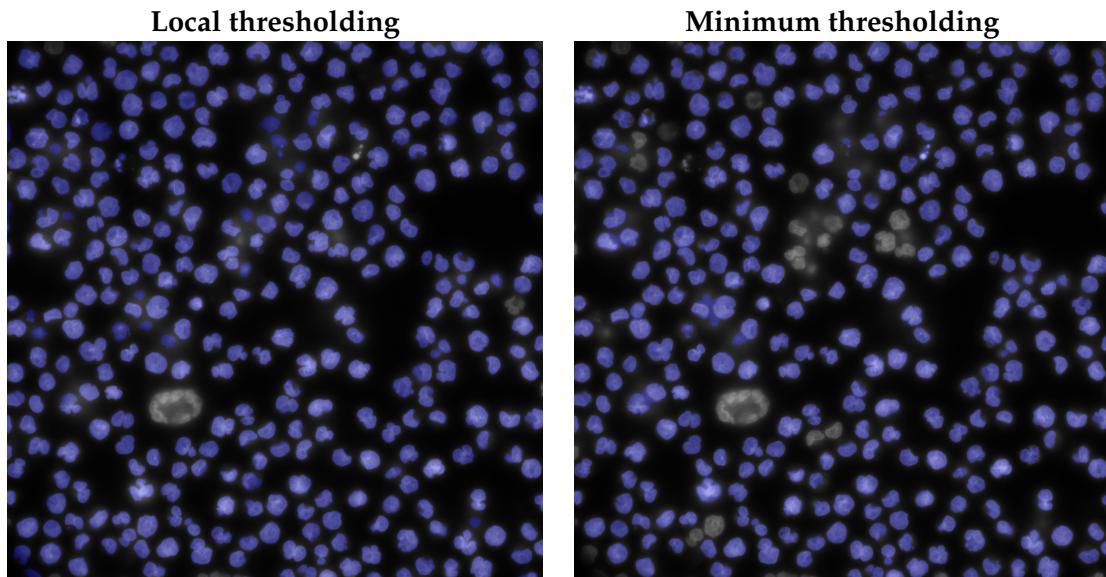


Figure 21: Local vs. Global thresholding (normal conditions)

3.2.4 Influence of scaling on predictions quality

Examples of predictions quality with different scales.

Table 4: Pearson correlation coefficients for downstream tasks for different scaling factors

	1.3 scale	0.7 scale	Train (1.0 scale + augments) Predict (1.3 scale)	Train (1.3 scale) Predict (1.0 scale)	Train (1.3 scale) Predict (0.7 scale)
Number of nuclei	0.987	0.995	0.975	0.971	?
Total intensity	0.902	.88	0.861	0.856	?
Mean intensity	0.922	0.906	0.88	0.872	?
Area	0.991	0.992	0.961	0.952	?

3.3 Endoplasmic Reticulum

3.3.1 Preprocessing

Algorithm 2 Fluorescence segmentation

1. Normalize image
 2. Apply global *threshold_mean* to receive initial mask.
 3. Zero out pixels outside the mask
 4. Apply local thresholding.
 5. Apply *fill_holes* transformation.
 6. Morphological opening from OpenCV and Gaussian blur.
 7. Run *findContours* from OpenCV in order to obtain separate regions and filter out too small regions.
-

Segmentation steps are also illustrated in Figure

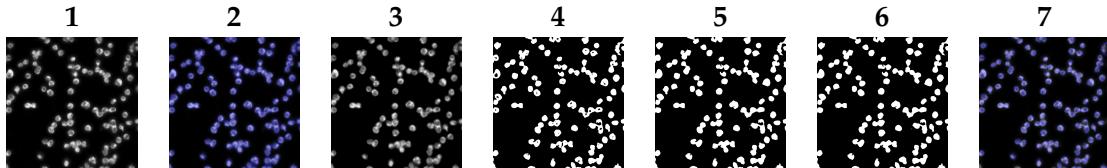


Figure 22: ER prediction

3.3.2 Training and predictions

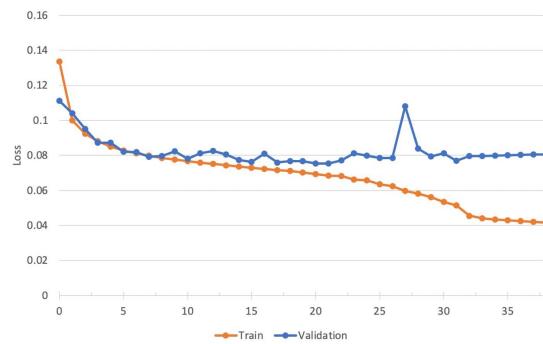


Figure 23: Overfit

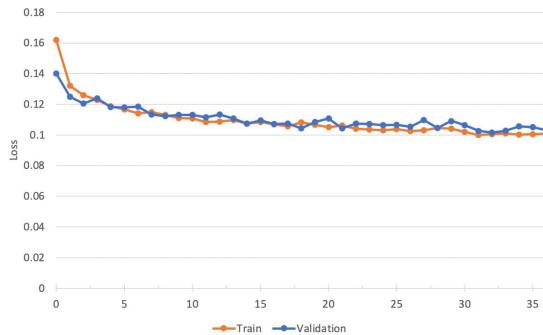


Figure 24: No overfit with augmentations

3.3.3 Combination of nuclei and actin predictions

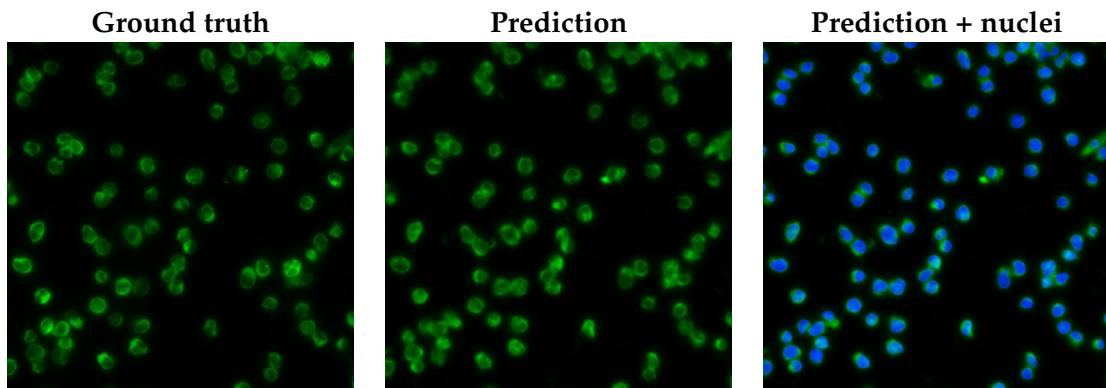


Figure 25: ER prediction

3.3.4 Generalizability across phenotypes

TODO train the model on one phenotype and predict on the other, compare predictions (visually?) postprocessing with metrics then? Add metrics

3.4 Golgi

3.4.1 Preprocessing

Enhancement

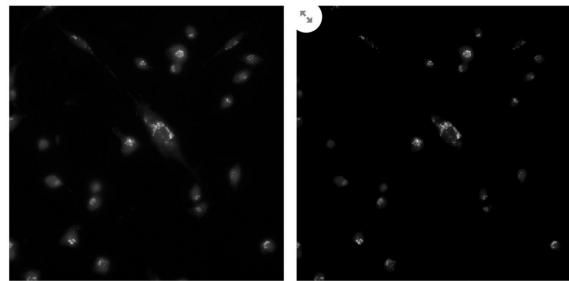


Figure 26: Golgi enhancement

3.4.1.1 Background removal algorithms

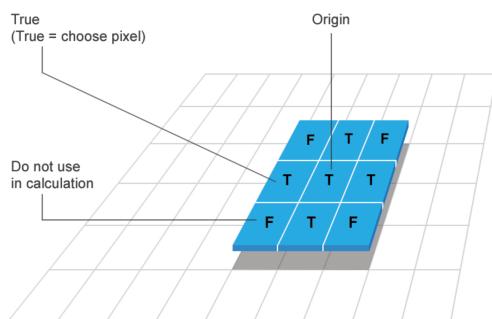


Figure 27: Structuring Element

Rolling ball algorithms

Rolling ball still leaves some noise

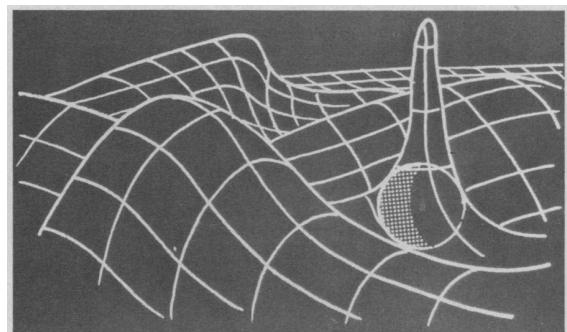


Figure 28: Rolling Ball

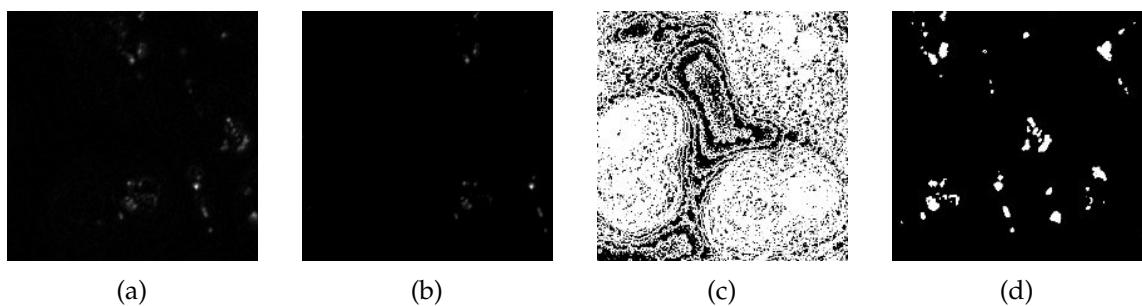


Figure 29: (a) Vanilla pre-processing with automatic background removal algorithm only; (b) Additional clipping of lower intensities after vanilla pre-processing; (c) masked or subfigure (a); (d) mask of subfigure (b)

3.4.2 Training and predictions

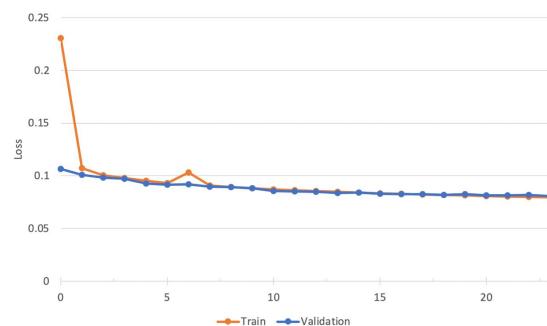


Figure 30: Straightforward training doesn't work

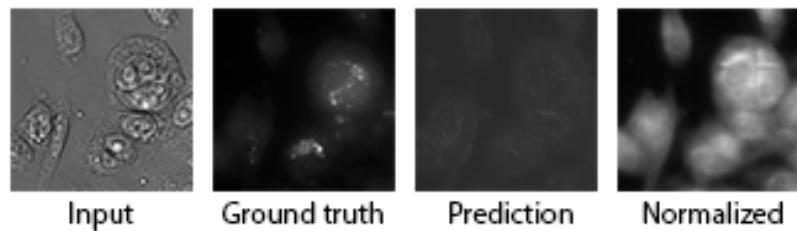


Figure 31: Training on original data

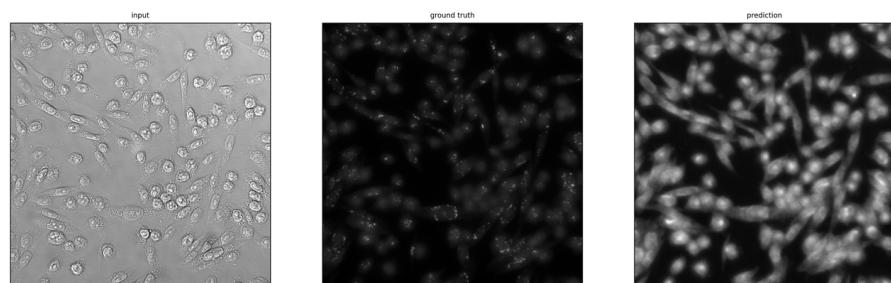


Figure 32: Full size predictions

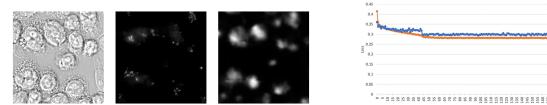


Figure 33: Training on the enhanced data

3.4.3 Alternative ways to improve predictions

3.4.3.1 Asymmetrical losses

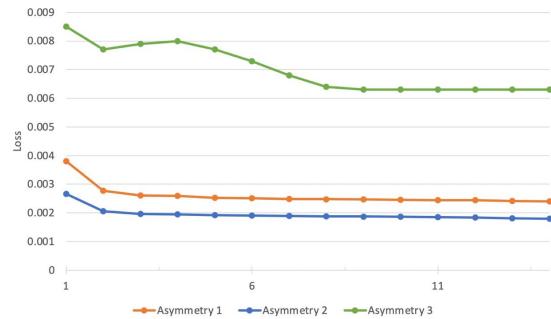


Figure 34: Asymmetrical training

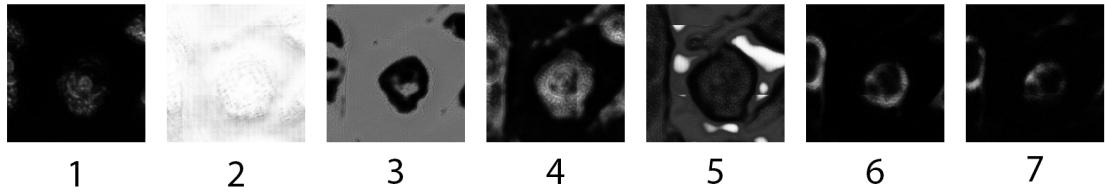


Figure 35: Asymmetrical training predictions

3.4.3.2 Use of gradient in loss

3.4.3.3 Noise reduction methods

3.5 GFP

3.5.1 Preprocessing

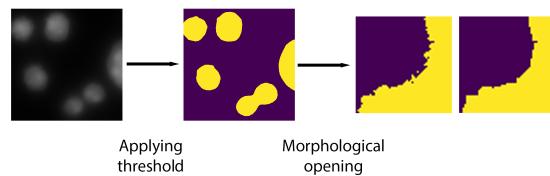


Figure 36: Converting GFP to a binary mask

3.5.2 Predictions

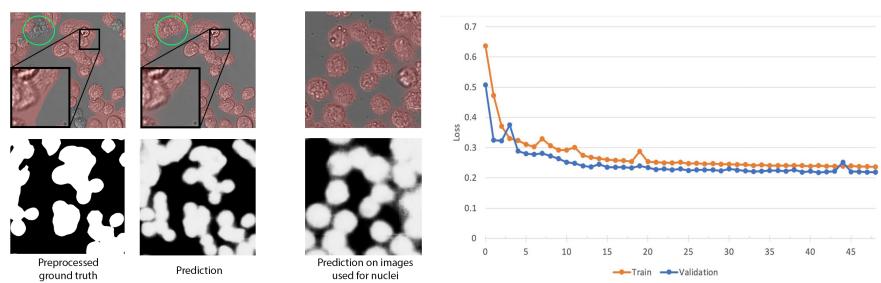


Figure 37: Training with BCE loss

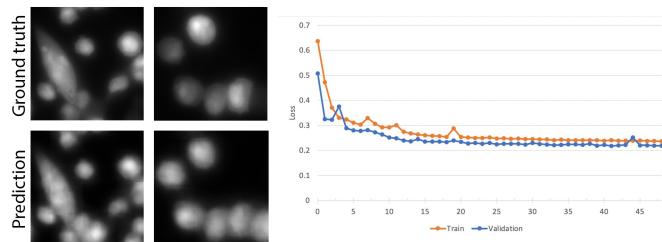


Figure 38: Training with Pearson correlation loss

Table 5: Correlation coefficients for downstream tasks

Binary training	Pearson	Spearman
Number of ER	0.67	0.64
Area	0.82	0.75
Continuos training	Pearson	Spearman
Number of ER	0.57	0.55
Area	0.26	0.64

3.5.3 Downstream metrics

TODO move to separate chapter?

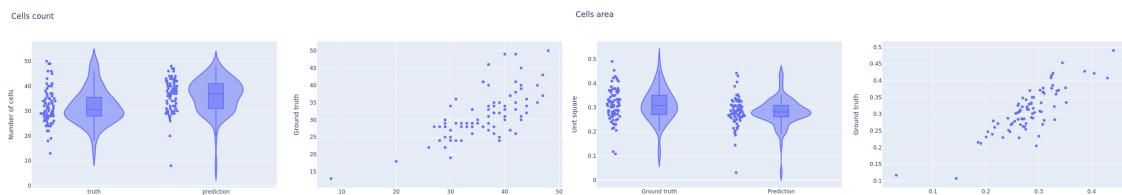


Figure 39: Downstream metrics

3.5.4 Combination of GFP, nuclei and ER

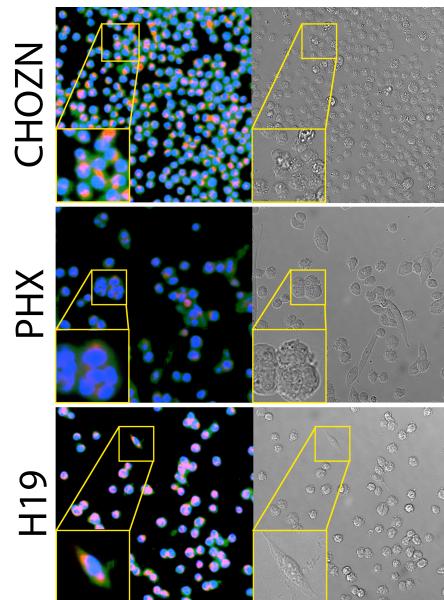


Figure 40: GFP, Nuclei and ER combined

3.6 Model evaluation

3.6.1 Metrics for downstream tasks

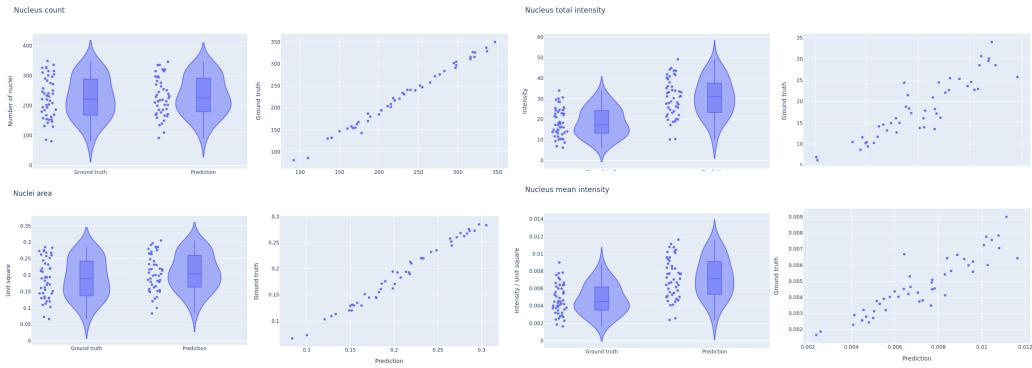


Figure 41: Metrics for downstream tasks on nuclei

Table 6: Correlation coefficients for downstream tasks on nuclei

	Pearson	Spearman
Number of nuclei	0.995	0.994
Total intensity	0.902	.911
Mean intensity	0.907	0.904
Area	0.992	0.990

3.6.2 Influence of different loss functions on metrics for downstream tasks

4 Stability study

4.1 Stability study

4.1.1 Artificial corruptions

Description of artificial corruptions.

Table 7: Hyperparameterization for different artificial corruption severities

Corruption \ Severity	-5	-4	-3	-2	-1	0	1	2	3	4	5
Defocus blur (radius)	-	-	-	-	-	0	0.5	1.0	1.5	2	3
Contrast (gain)	3.5	3.0	2.5	2.0	1.5	1	0.9	0.8	0.7	0.5	0.3
Brightness (bias)	-150	-135	-120	-90	-50	0	50	90	120	135	150

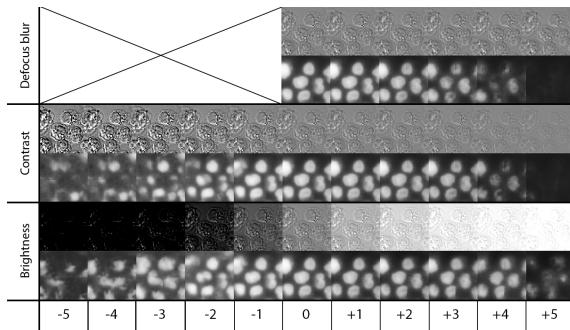


Figure 42: Influence of artificial corruptions on the predictions

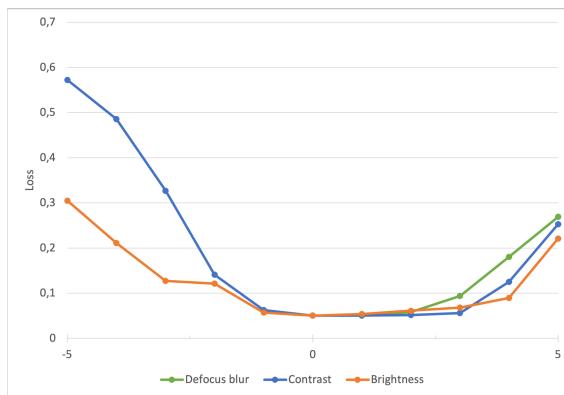


Figure 43: Change of Pearson correlation loss for artificial corruptions

4.1.2 Real corruptions

4.1.2.1 Not fixed cells imaging as corrupted input

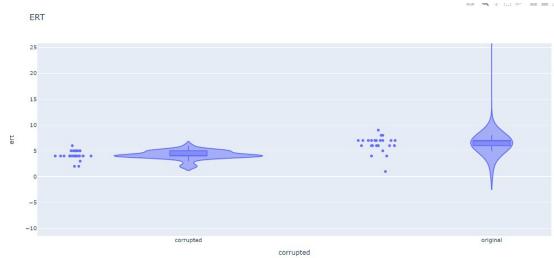


Figure 44: Online drift detection of not fixated cells

Scores of 0.91 however the threshold is 6, not corrupted data (fixed cells) mostly ert of 7 whereas corrupted data (not fixed cells) have an ert of 4. The threshold is therefore 6.

4.1.2.2 Real-world examples of corruptions

4.1.3 Influence of corruptions on metrics for downstream tasks

Calculate how metrics worsen when the evaluation stays the same, but the input is corrupted.

4.1.4 Improving predictions with additional corruption augmentations

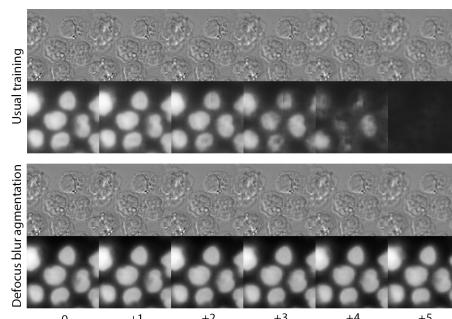


Figure 45: Using corruptions as augmentations improves predictions

4.2 UNET embeddings study

4.2.1 Application of various dimentionality reduction methods

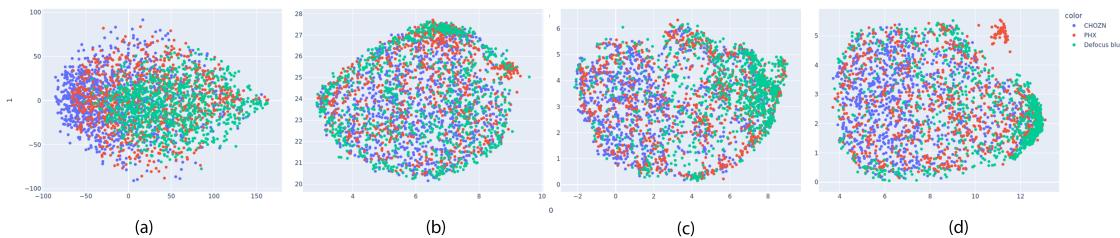


Figure 46: (a) PCA, (b) UMAP, (c) combination of PCA and UMAP with 10 and (d) 50 components

4.2.2 Autoencoder embeddings as an alternative

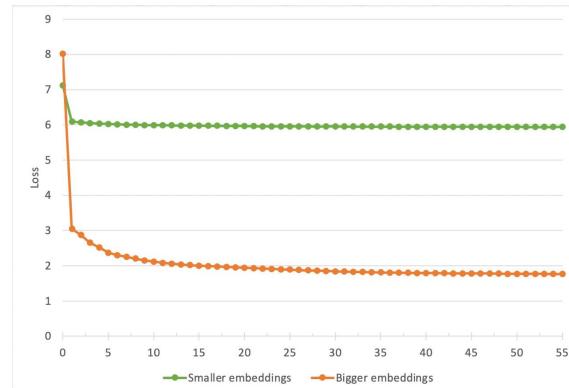


Figure 47: Autoencoders training convergence

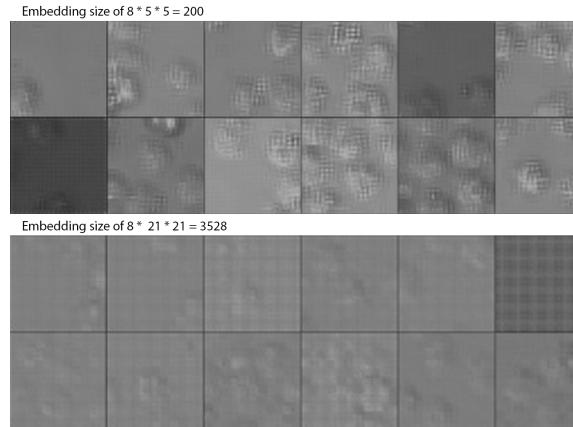


Figure 48: Samples drawn from the trained autoencoder

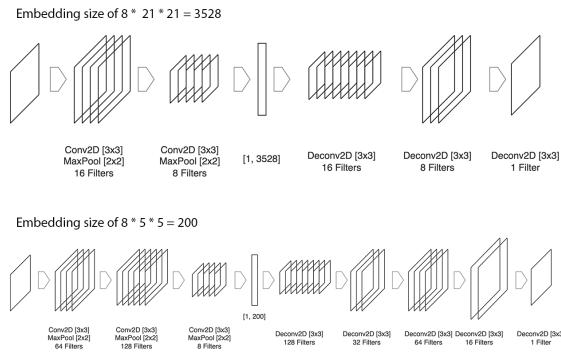


Figure 49: Architectures of two autoencoders

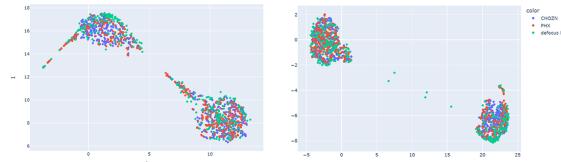


Figure 50: Autoencoder embeddings after applying PCA with 10 components and UMAP afterwards. Earlier epoch VS later epoch

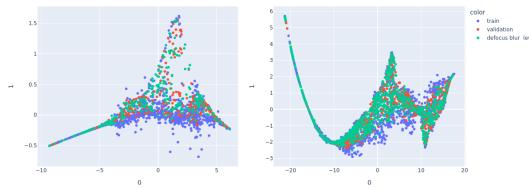


Figure 51: PacMAP does not provide information on the corruption

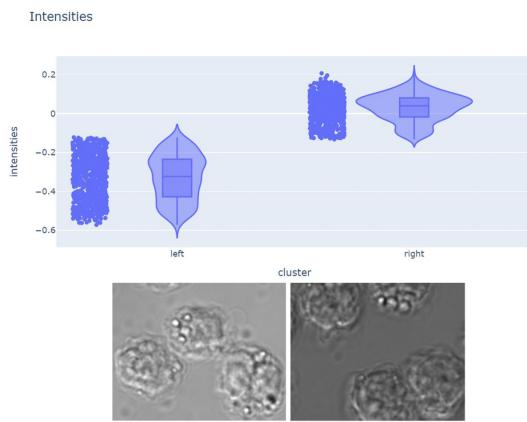


Figure 52: What do two UMAP clusters represent

4.2.3 Clustering of PacMAP embeddings

4.2.3.1 Clustering on UNet embeddings

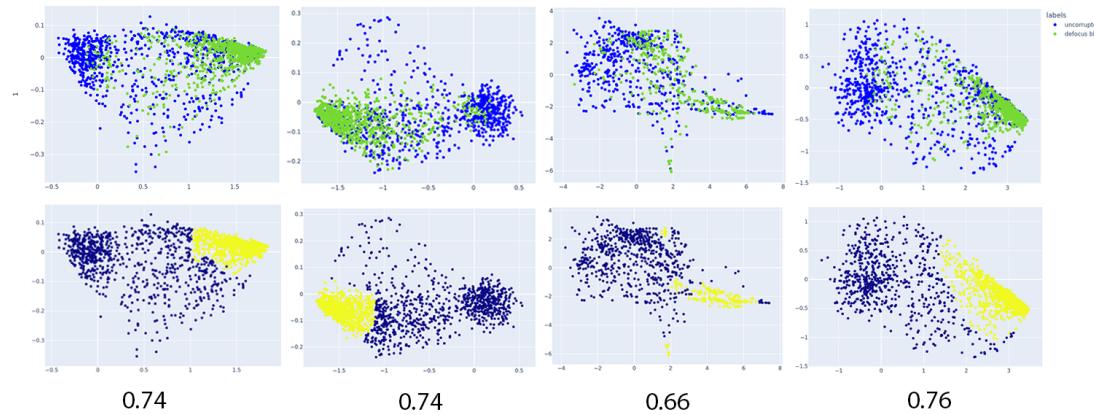


Figure 53: Clustering of UNet embeddings after PacMAP

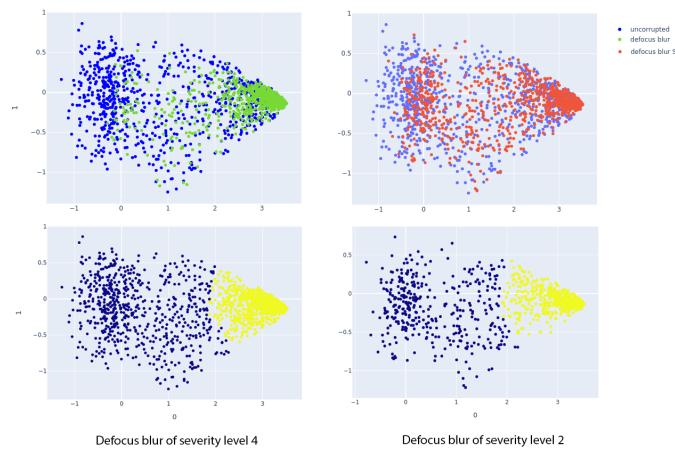


Figure 54: Clustering of UNet embeddings after PacMAP for different severities levels

TABLE with F1-score: 0.76 VS 0.64

4.3 Drift detection

4.3.1 A need to detect drift

4.3.2 Maximum mean discrepancy for drift detection

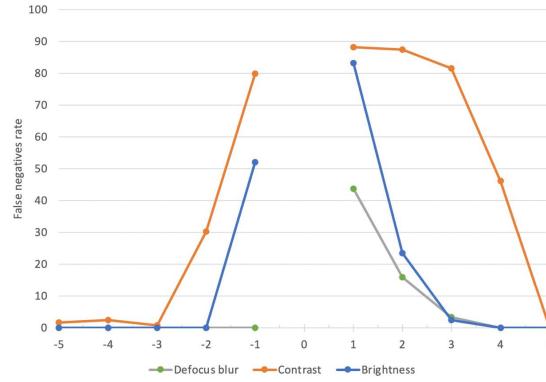


Figure 55: False negatives rate for drift detection on artificial corruptions

4.3.3 Online version of MMD algorithm

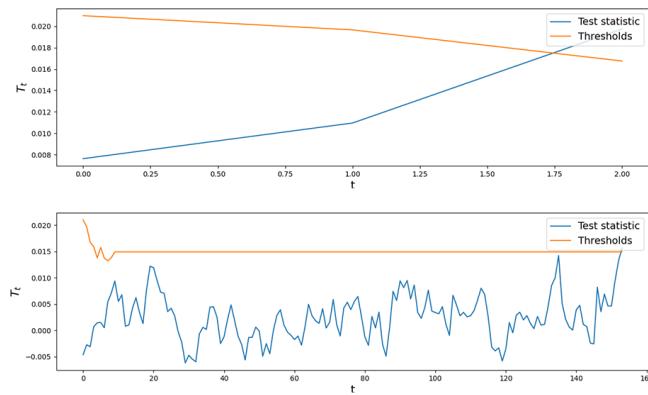


Figure 56: Expected runtime (ERT) for corrupted and in-distribution data

Table 8: Test window size influence on separability

W	2	5	10	15	20
Auc-Roc	0.85	0.92	0.98	0.90	0.88

Table 9: ERT influence on separability

W	32	64	128	256
Auc-Roc	0.90	0.95	0.98	0.98

Table 10: Severity of corruptions on separability

W	Level 2	Level 3	Level 4
Auc-Roc	0.84	0.92	0.98

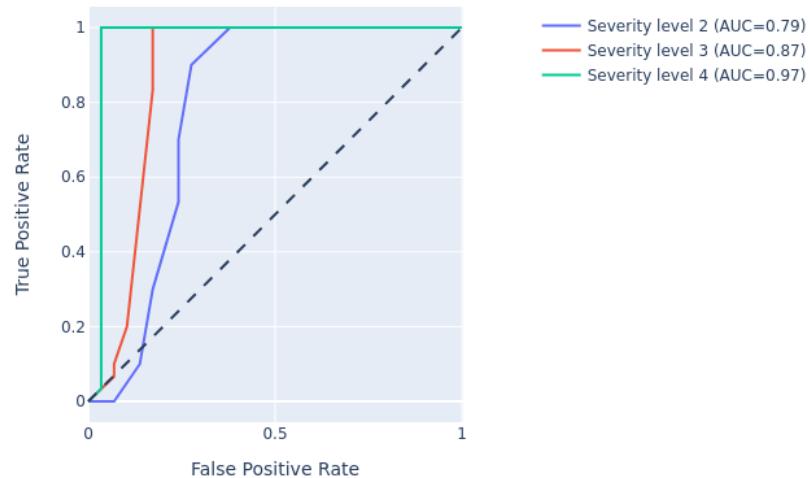


Figure 57: AUC ROC scores for various defocus corruptions severities

5 Software Tools

5.1 Foundry. Palantir

5.2 AWS

5.3 Streamlit

6 Future research

"One limitation of our current work is that it is based on fixed cells that does not allow longitudinal imaging. This can be overcome by using fluorescent reporter cell lines or live cell dyes to provide the fluorescence ground truth (10) and enable dynamic observation. Another limitation of the DL framework we used here is that it cannot be generalized to different types of cells. Techniques based on transfer learning (<https://doi.org/10.1038/s41551-019-0362-y>, <https://downloads.spj.sciencemag.org/bmef/2020/9647163.pdf>) and domain adaptation (38) will be investigated in our future work to overcome this limitation."

TODO rephrase <https://www.science.org/doi/10.1126/sciadv.abe0431>

7 Summary

References

- Eric M. Christiansen, Samuel J. Yang, D. Michael Ando, Ashkan Javaherian, Gaia Skibinski, Scott Lipnick, Elliot Mount, Alison O'Neil, Kevan Shah, Alicia K. Lee, Piyush Goyal, William Fedus, Ryan Poplin, Andre Esteva, Marc Berndl, Lee L. Rubin, Philip Nelson, and Steven Finkbeiner (Apr. 2018). "In Silico Labeling: Predicting Fluorescent Labels in Unlabeled Images". In: *Cell* 173.3, 792–803.e19.
- Julienne Lachance and Daniel J. Cohen (Dec. 2020). "Practical fluorescence reconstruction microscopy for large samples and low-magnification imaging". In: *PLOS Computational Biology* 16.12, pp. 1–24.
- Nucleus* (n.d.).

List of Figures

1	CLD process steps	5
2	Dropout	11
3	Way in which photos of the well-plate were taken	16
4	Sliding window approach for fluorescence prediction	17
5	Difference of overlap between predictions on the resulting image	17
6	Unet	19
7	Using original image for rotation and scaling augmentations	22
8	Nuclei training without (left) and with (right) custom weight initialization	24
9	Comparison of convergence for different optimizers	26
10	Samples to be filtered out	27
11	Having more data makes training more stable	28
12	Adding simple augmentations in the dataset	28
13	With regularization and augmentations	29
14	Difefrent models predictions and scores comparison	29
15	Problems in predictions	30
16	Predictions improvement	30
17	Different lightning conditions	31
18	Closely located cells	31
19	Fluorescence segmentation	31
20	Local vs. Global thresholding	32
21	Local vs. Global thresholding (normal conditions)	33
22	ER prediction	34
23	Overfit	34
24	No overfit with augmentations	35
25	ER prediction	35
26	Golgi enhancement	36
27	Structuring Element	36
28	Rolling Ball	37
29	(a) Vanilla pre-processing with automatic background removal algorithm only; (b) Additional clipping of lower intensities after vanilla pre-processing; (c) masked or subfigure (a); (d) mask of subfigure (b)	37
30	Straightforward training doesn't work	37

31	Training on original data	38
32	Full size predictions	38
33	Training on the enhanced data	38
34	Asymmetrical training	39
35	Asymmetrical training predictions	39
36	Converting GFP to a binary mask	40
37	Training with BCE loss	40
38	Training with Pearson correlation loss	40
39	Downstream metrics	41
40	GFP, Nuclei and ER combined	41
41	Metrics for downstream tasks on nuclei	42
42	Influence of artificial corruptions on the predictions	43
43	Change of Pearson correlation loss for artificial corruptions	43
44	Online drift detection of not fixated cells	44
45	Using corruptions as augmentations improves predictions	44
46	(a) PCA, (b) UMAP, (c) combination of PCA and UMAP with 10 and (d) 50 components	45
47	Autoencoders training convergence	45
48	Samples drawn from the trained autoencoder	46
49	Architectures of two autoencoders	46
50	Autoencoder embeddings after applying PCA with 10 components and UMAP afterwards. Earlier epoch VS later epoch	46
51	PacMAP does not provide information on the corruption	47
52	What do two UMAP clusters represent	47
53	Clustering of UNet embeddings after PacMAP	48
54	Clustering of UNet embeddings after PacMAP for different severities levels	48
55	False negatives rate for drift detection on artificial corruptions	49
56	Expected runtime (ERT) for corrupted and in-distribution data	49
57	AUC ROC scores for various defocus corruptions severities	50

List of Tables

1	Available data for each fo the organelles	20
---	---	----

2	Costs estimations of AWS use for training models	21
3	Costs estimations of AWS use for inference purposes	21
4	Pearson correlation coefficients for downstream tasks for different scaling factors	33
5	Correlation coefficients for downstream tasks	41
6	Correlation coefficients for downstream tasks on nuclei	42
7	Hyperparameterization for different artificial corruption severities	43
8	Test window size influence on separability	49
9	ERT influence on separability	50
10	Severity of corruptions on separability	50