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 PaCMAP . . . 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
3.1.6 Model evaluation: metrics for downstream tasks . . . 26
3.2 Nuclei . . . 28
3.2.1 Preprocessing . . . 28
3.2.2 Training and predictions . . . 29
3.2.2.1 Convergence . . . 29
3.2.2.2 Predictions quality . . . 31
3.2.3 Postprocessing for nuclei segmentation . . . 32
3.2.3.1 Thresholding algorithms . . . 34
3.2.4 Downstream metrics . . . 38
3.2.5 Influence of scaling on predictions quality . . . 39
3.2.6 Generalizability across phenotypes . . . 39
3.2.7 Conclusions . . . 39
3.3 Endoplasmic Reticulum . . . 41
3.3.1 Training and predictions . . . 41
3.3.2 Combination of nuclei and actin predictions . . . 42
3.3.3 Postprocessing for ER segmentation . . . 42
3.3.4 Downstream metrics . . . 43
3.3.5 Conclusions . . . 44
3.4 Golgi . . . 46
3.4.1 Preprocessing . . . 46
3.4.1.1 Background removal algorithms . . . 47
3.4.2 Training and predictions . . . 48
3.4.3 Alternative ways to improve predictions . . . 51
3.4.3.1 Asymmetrical losses . . . 51
3.4.3.2 Use of gradient in loss . . . 53
3.4.3.3 Noise reduction methods . . . 53
3.4.4 Conclusions . . . 54
3.5 GFP . . . 55
3.5.1 Preprocessing . . . 56
3.5.2 Predictions . . . 56

3.5.3 Downstream metrics . . . 57
3.5.4 Combination of GFP, nuclei and ER . . . 58
3.5.5 Conclusions . . . 59

4 Stability study **60**
4.1 Stability study . . . 60
4.1.1 Artificial corruptions . . . 60
4.1.2 Defocus Blur . . . 60
4.1.3 Brightness . . . 61
4.1.4 Contrast . . . 61
4.1.5 Real corruptions . . . 63
4.1.5.1 Real-world examples of corruptions . . . 63
4.1.6 Improving predictions with additional corruption augmentations . 64
4.1.7 Influence of corruptions on metrics for downstream tasks . . . 64
4.2 UNET embeddings study . . . 65
4.2.1 Application of various dimentionality reduction methods . . . 65
4.2.2 Autoencoder embeddings as an alternative . . . 66
4.2.3 Clustering of PacMAP embeddings . . . 69
4.2.3.1 Clustering on UNet embeddings . . . 69
4.3 Drift detection . . . 72
4.3.1 Drift detection vs. outliers detection . . . 72
4.3.2 Kernel methods and two-sample testing . . . 73
4.3.3 Maximum mean discrepancy for drift detection . . . 75
4.3.4 Drift detection experiments . . . 76
4.3.5 Online drift detection experiments . . . 77
4.3.5.1 Not fixed cells imaging as corrupted input . . . 78

5 Future research **79**

6 Summary **80**

References **81**

List of Figures **82**

List of Tables **84**

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 neeeds 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

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 succeful 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/offerings/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 genomically 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

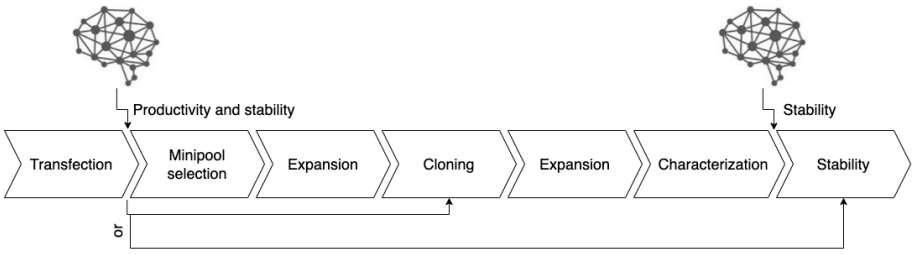


Figure 1: CLD process steps

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 transfect 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 30^{th}, 60^{th} and 90^{th} 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 constists of input DIC images X and target fluorescence images Y. Combined, couples from each form (X and Y) construct a dataset:

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

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 $\theta = (\theta_1, ..., \theta_K)$ where $\theta_i \in \mathbb{R}$ for $i \in 0, ..., K$ which approximates the mapping of initial data X to target data Y.

$$M(X, \theta) = Y' \approx Y \tag{2}$$

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:

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

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:

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

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:

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

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:

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

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

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 similiraity 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 esimation, we get:

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

After maximizing the sum and taking a gradient one gets:

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

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

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

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

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

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

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

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

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

Algorithm 1 Adadelta optimization

1. $E[g]_0^2 = 0$ and $E[\Delta\theta^2]_0 = 0$ In order to update the parameters compute:
2. Compute gradient: $\hat{g}_t$
3. Accumulate gradient: $E[g]_t^2 = \rho E[g]_{t-1}^2 + (1-\rho)g_t^2$
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:

$$a = W^T x + b \tag{14}$$

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

$$S = K * x + b \tag{15}$$

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

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

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 equivarient 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 ouput 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].

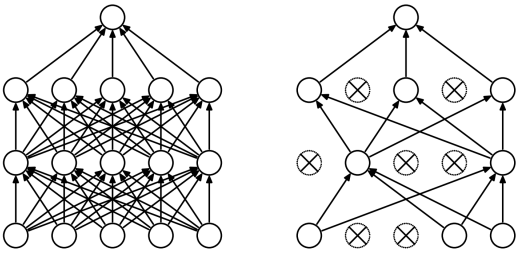
Since adding two linear functions together would produce a linear function, it is important to use activation functions (or non-linearities) after each convolutional or linear layer like RELU, ELU, Tahn, Sigmoids and so forth. In CNNs they are also often combined with max-pooling layers and dropouts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

Definition 2.16 (Batch normalization layer)**.** Let's denote $B = \{x^{(i)}, ..., x^{(m)}\}$ 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.



(a) Standard Neural Net (b) After applying dropout.

Figure 2: Dropout

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

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

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

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

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 ecnoder 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 reseach 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 655536. 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 caterogy, while t-SNE [cite Ulyanov] and Isomap are assinged 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 correspoding 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 varience 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:

$$x_d^* \leftarrow \frac{x_d^* - \mu_d}{\sigma_d} \tag{21}$$

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

$$\tilde{x^*} = BB^T x^* \tag{22}$$

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 clusters. 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 represented with the matrix:

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

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 $[0, 1]$:

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

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

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 validaiton 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.

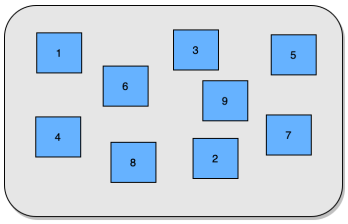


Figure 3: Way in which photos of the well-plate were taken

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 oder to achieve that if the border size has been defined in advance, one has to set the step size to in the following way:

$$s = w - 2 * b \tag{25}$$

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

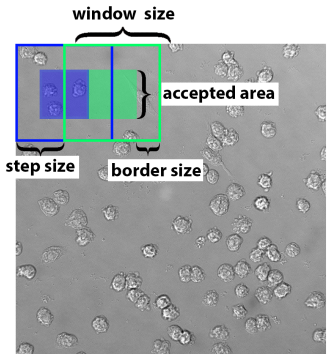


Figure 4: Sliding window approach for fluorescence prediction

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). 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 created the predictions, however, the speed is less crucial in comparison to the accuracy of the predictions.

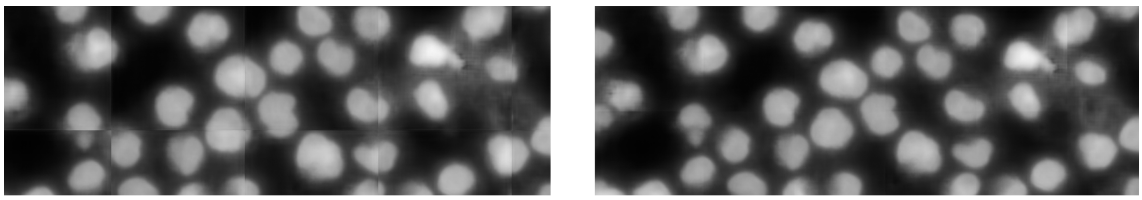


Figure 5: Difference of overlap between predictions on the resulting image

3 Implementation and experiments

In this chapter the results of all experiments performed for predicting fluorescence signal from 4 cell oganelles: nuclei, endoplaspic 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 occured 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 decription 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 detoned 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 pressent. That is another thing that differs this architecture from the original one in [TODO Cite LaChance] paper. The last layer of the UNet here is a sigmoid activation function.

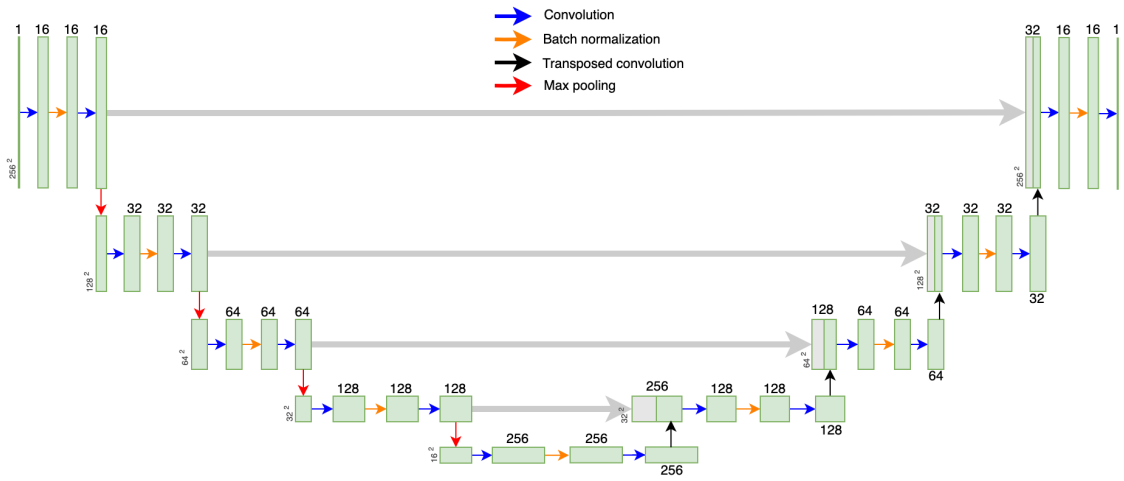


Figure 6: Unet

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 efficience comes mostly from Batch normalization layers that are already used in our architecture. Nethertheless 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 wether 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 bery usefull to be able to not only predict the data itselt, but also to say wether the prediction is reliable of 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, endoplaspic reticulum, Golgi apparatus and a full cell target. First 3 targets provide many insights on the state of cell based

on their quaintity, size and the intensity of fluorescence signal that they can produce by binding with the protein, the latter one is usefull 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 wether the network could potentially substitute manual fluorescence staining in general. The hypothesis on which we rely here is that if the model can successully predict fluorescence signal from some of the targets, then this research could be exended to other cell organelles as well.

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

```
dataset0
├── DIC
│   ├── {CHOZN, PHX, H19}_XY001.tif
│   ├── ...
│   └── {CHOZN, PHX, H19}_XY100.tif
└── {DAPI_nuclei, FITC, FITC_er, CY5_golgi}
    ├── {CHOZN, PHX, H19}_XY001.tif
    ├── ...
    └── {CHOZN, PHX, H19}_XY100.tif
...
datasetN
└── ...
```

Here a dataset$\{0, \ldots, N\}$ is one 96-well plane. 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 form one fluorescent staining target only. Each filename within the subfolder includes its index number between $\{1, \ldots, 100\}$ 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 fo 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 the 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 presened 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 unter 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 used. They can be triggered only when the inference request arrives and ca be turned off automatically shortly after.

For both training an 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 netwrk 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 hypotetically very close to the original true distribution of images. However one should be very cautios 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 infered 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**: Descreases an image constrast 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 chosed 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 wether 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 wuold be filled with zeros in PyTorch implementation. However in 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 alltogether.

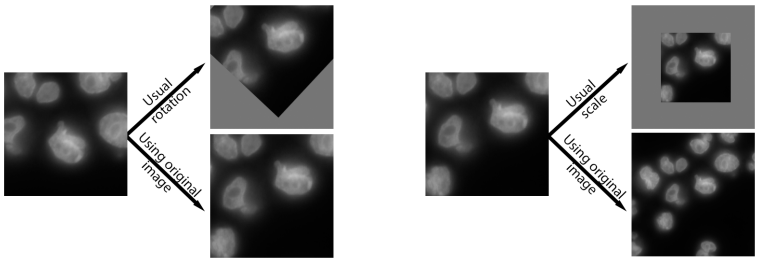


Figure 7: Using original image for rotation and scaling augmentations

Applying augmentations in combination with other regularization techniques has proven to be helpful against overfitting for the nuclei and ER training. Therefore it is recommened 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 reseach as well. He initialization draws samples from a truncated normal distribution:

$$N(0, \sqrt{\frac{2}{\text{fan\_in}}}) \tag{26}$$

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:

$$std = \sqrt{\frac{2}{fan\_in}} \tag{27}$$

$$bound = \sqrt{3 * std} \tag{28}$$

$$Uniform(-bound, bound) \tag{29}$$

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.

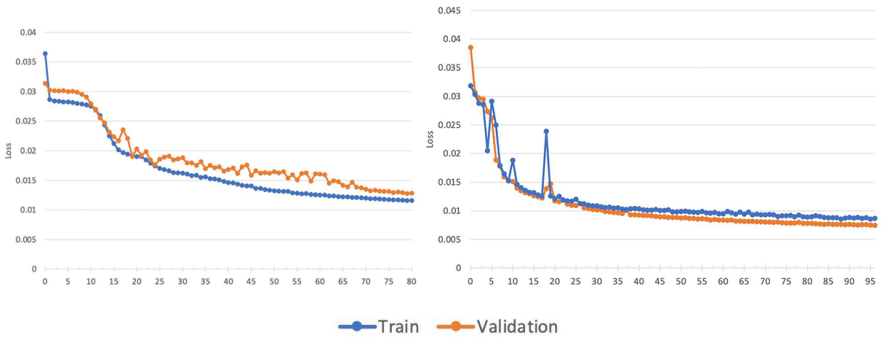


Figure 8: Nuclei training without (left) and with (right) custom weight initialization

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 the 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 improtant to understant the tenchniques that can be used against it. There are 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 **??**). Since the model haven't seen any of the data from the validation set, it means that it losses its capability to generalize on the unsees data, while improving its perfomance 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:

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

  for some $\lambda > 0$. This is called a *soft-constrait* 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:

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

  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 optmizers 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 expreiments were conducted on the truncated dataset of nuclei images using PCC loss.

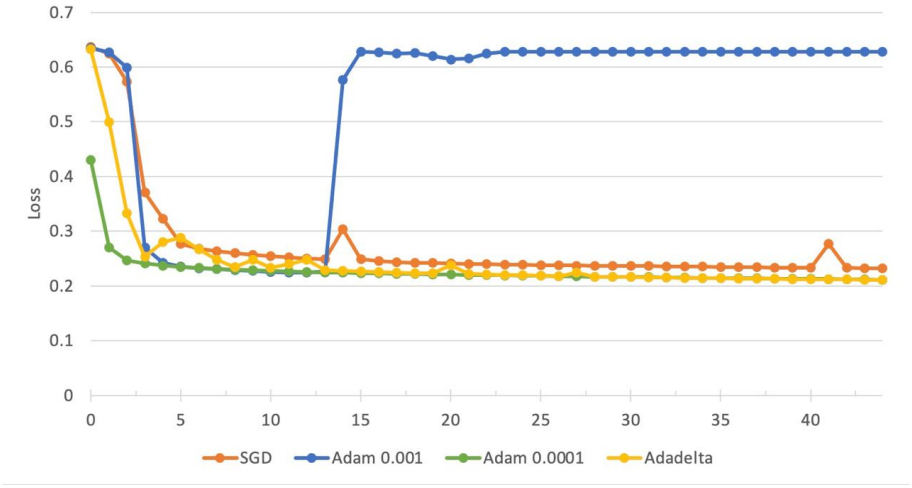


Figure 9: Comparison of convergence for different optimizers

3.1.6 Model evaluation: metrics for downstream tasks

As Lachance et al., 2020 notices evaluation of models predicting flureoscence signal in terms of the PCC or MSE loss is not quite an objective approach. Even when one model might have a smaller loss, it doesn't nessecarily will perform better than another model. Practically there are other every-day metrics that have more value to the end-user rather than some abstract performance measurements typically used in computer vision. Therefore here the evaluation of the model is additionally checked in terms of the following metrics:

- Number of nuclei / ER / Golgi / cells
- Area of nucleus / ER / Golgi / cell
- Total intensity of nuclei / ER / Golgi
- Mean intensity of nuclei / ER / Golgi

These quantities can be compareed with each other in a much more understandable way. On the contrary, when one recieves a precision value P (let it be PCC loss in this example)

from the model performance evaluation there is no way to appreciate how good this model is. Usually one can easily raise the value of P by simply training on more data or using a better resolution microscopy, but all of this increases the amount of lab work and expenses.

Highly practical metrics mentioned above would be calculated for each image in the test set and for corresponding ground truth images. This would give us two distributions: one is a distribution of predicted values and another — of ground truth values. Ideally these two should be the same distribution. Here they will be compared visually with violin plots (simply checking the form and range of the two), scatter plots and quantitevely by correlation coefficients, or more specifically, with the help of Spearman rank coefficient and and Pearson correlation coefficient.

In each subsection dedicated to each organelle these metrics are presented under the according "Downstream metrics" section.

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 (see Figure 10). 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 then in dead ones (Christiansen et al., 2018). With fluorescence labeling one can derive some useful features used for determining whether the well plate should or should not be selected during the selection step in CLD process.

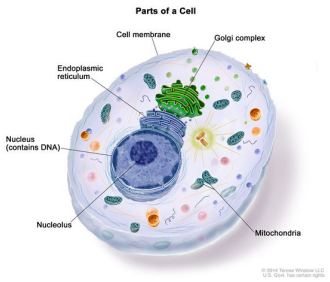


Figure 10: Cell structure

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 11a). Overexposure is also a typical problem, that creates samples of a too high intensity and without details inside the nucleus (see Figure 11b, c). Lastly, underexposure is as problematics as an overexposure (see Figure 11d).

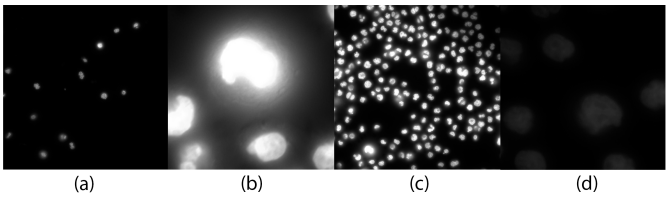


Figure 11: Samples to be filtered out

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 begining, but one can clearly see that the model successfully converges afterwards. 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 12).

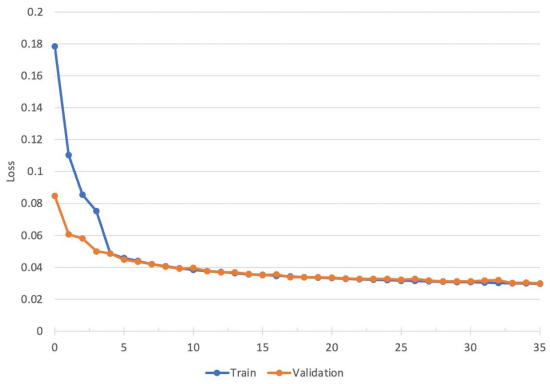


Figure 12: Having more data makes training more stable

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 15 *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 of 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 the augmented data are presented in Figure 13. The augmentations used here are horizontal and vertical flips, rotations and crops (described in detail in section 3.1.4).

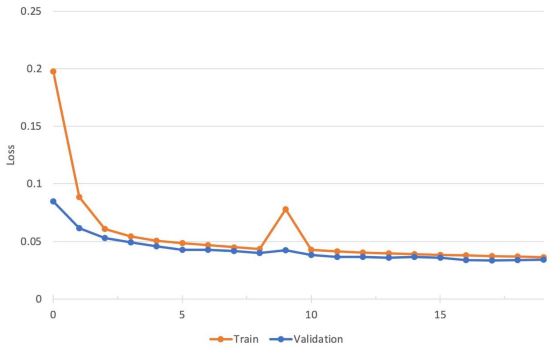


Figure 13: Adding simple augmentations in the dataset

Validation PCC loss has slightly increased to 0.0381 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. True improvement is confirmed by measuring the loss of the models on the same dataset, where PCC loss for has improved from 0.0365 to 0.0322(or PCC from 0.92 to 0.93).

In next experiment the model has additionally been regularized by adding dropout layers and using a weight decay of 0.0001 (see Figure 14). 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 might simply have not enough of capacity to capture enough of the details. In order to confirm this a bigger model has to be trained.

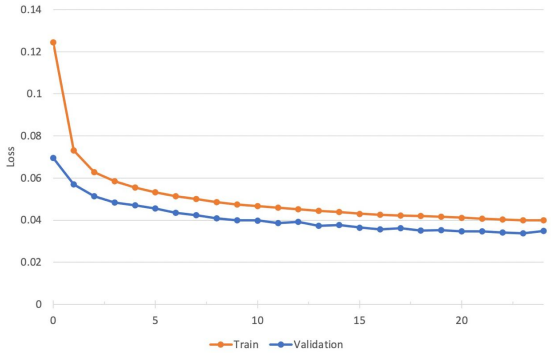


Figure 14: With regularization and augmentations

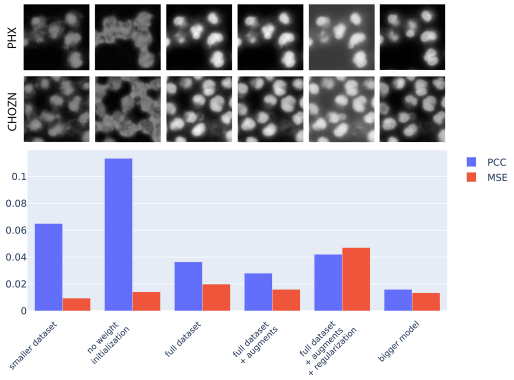


Figure 15: Difefrent models predictions and scores comparison

Interestigly observations made in this section made it clear that indeed metrics used for trainnig (PCC and MSE losses) are not representative enough to derive any conclusions regarding the model quality from them. Just by looking at the Figure 15 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 intialization vs. the regularized model with augmentations. MSE loss if 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 state 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 3.1.6, where more practical metrics are suggested and the model evaluation on them is provided.

3.2.2.2 Predictions quality

A more thorough visual evaluation of nuclei fluroescence predictions is provided in Figure 16. There are tree main visible problems that should be adressed in further research:

- The form of the nucleus is well-captured, but the texture inside is not (the change of intensities) is not predicted very well.
- The border around the nucleus is quite blurry
- The overall intensities of predictions seem to be higher than the ground truth ones.

All of these problems are quite similar and can be summarized under the statement that the model does not have enough of resolution. Which can be solved by making a model larger and providing more data to it. In order to confirm this a bigger model has been trained.

Indeed, increasing the number of filters in each convolutional layer by a factor of 4 (switching from from 16 to 64 in the first one) the model was able to capture more details. Pre-

dictions became better both visually (see Figure 17) and based on both metrics (PCC and MSE losses, see Figure 15).

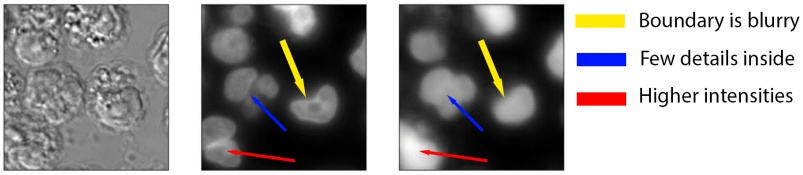


Figure 16: Problems in predictions

Another rather small issue is that predictions on the border of the crops seem to be worse than in the middle. As has already been mentioned in section 2.3.2.2, when the cell is not fully present in the crop (split in halves for ex.), the network doesn't have enough data to give out a good enough prediction. However, when combining crops into a high-resolution images, this problem can be mitigated by the method described in section 2.3.2.2.

The problem of the higher intensities can also be addressed by a more careful dataset filtration, as few several images with overexposed cells are still used for training and that might be a reason of intensity overprediction. Blurry borders is also a signal of an overexposed nucleus. Even if they are present in the dataset, they can be removed using background removal techniques described in section 3.4.1.1. This was not done in the scope of this research as the predictions have had good enough qulity to be used in CLD process.

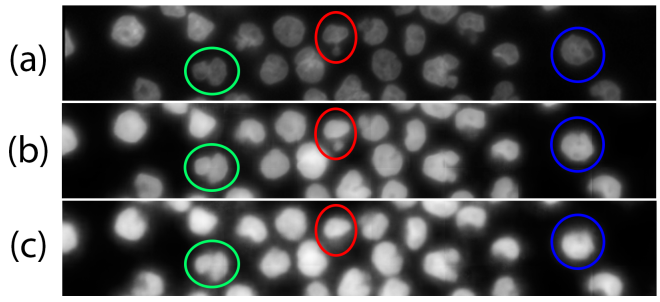


Figure 17: Predictions improvement: (a) ground truth fluorescence; (b) bigger model with four times more filters in convolutional layers; (c) standard architecture

3.2.3 Postprocessing for nuclei segmentation

To properly evaluate practical downstream metrics described in section 3.1.6 on model predictions, one need to be able to segment nuclei from fluorescence (as well as from predictions) first. By segmentation here the creation of mask is understood. It should constist of 0s and 1s, with 1 assigned to pixels that are part of nucleus and 0 otherwise.

Altough this might be a straight-forward task for our eyes, it is not that easy to select separate nuclei via post-processing. There are several edge cases where the nuclei are difficult to segment.

Even though the most extreme corruptions mentioned in section 3.2.1 were filtered out, some of the images that are corrupted not that severely (meaning they still have all the visible features needed for learning) are still present in the dataset. This allows to not reduce the amount of data significantly.

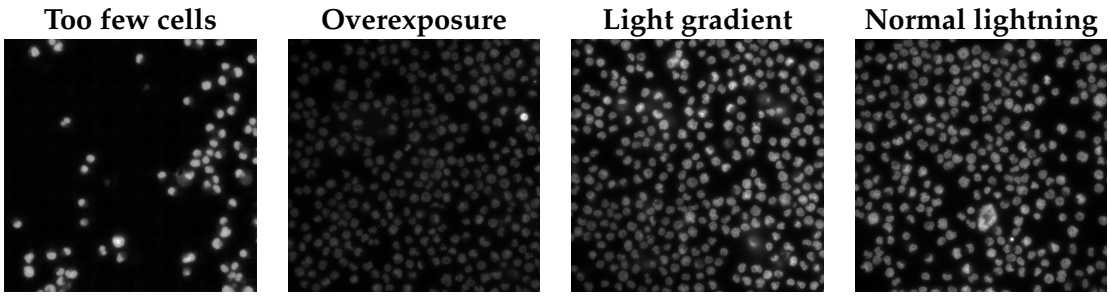
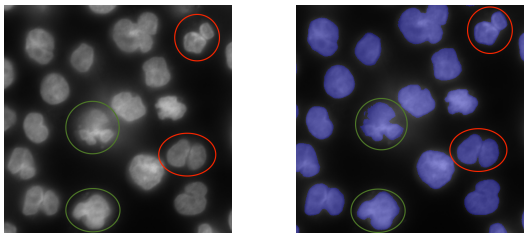


Figure 18: Different lightning conditions

The examples of difficult for segmentation cases are presented in Figure 18 from left to right: image contains too few cells, which leads to background being much darker than usually; overexposure of one cell, which leads to difficulties of segmenting the rest of the cells as they are hard distinguishable from the background; lighting gradient from darker (left bottom corner) to brighter (upper right corner) region; normal lighting conditions.

Another challenge for segmentation bring nuclei that are very close to each other. This might happen sometimes because some of the cells are currently in the process of the division. Also when some have already fully divided, they might still be located close to one another. The example of such situations is presented in Figure 19.



(a) Original fluorescence

(b) Segmentation (violet mask)

Figure 19: Closely located and dividing cells

Here cells, that are not yet fully divided are highlighted with the green cirles, and ones, that are fully divided, but just located too close to one another are highlighted with red circles. You can see that the segmentation algorithm (see Algorithm 2) recognises both such cases as one nuclei. This algorithm is described below and its step are vidualized in Figure 20.

Algorithm 2 Fluorescence segmentation

1. Normalize image.
2. Apply local thresholding and get a threshold T or a set of local thresholds $\{T_i\}$ and create an initial mask: 1 if $x_i > T$ or 0 otherwise.
3. Apply *fill_holes* transformation to the initial mask in order to get rid of unneeded details insides the nuclei.
4. Run *findContours* from opencv in order to obtain separate regions and filter them based on the following criteria: filter out too big regions (measure the biggest possible nuclei manually), too small regions (measured manually as well), regions that have a shape that it not very similar to convex circular type of nuclei. The last filter is done by checking the ratio of the area of the region to the area of the convex hull of the region.

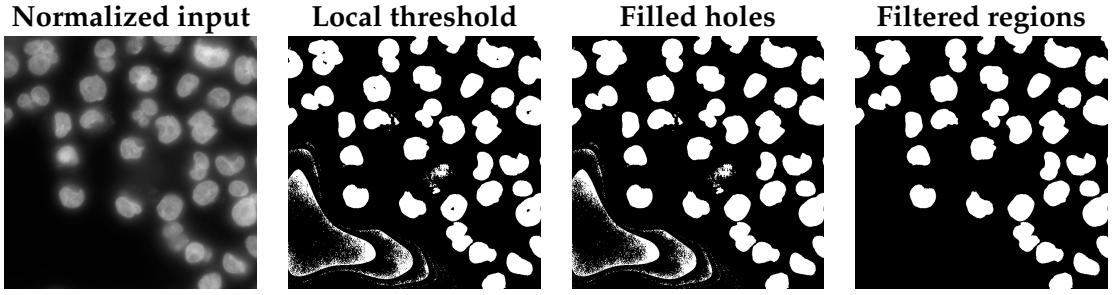


Figure 20: Fluorescence segmentation

A more detailed description of the step 2 of the algorithm (thresholding) as well as the general theory behind it is provided in the next subsection.

3.2.3.1 Thresholding algorithms

There are in general two types of thresholding that exist to binarize an image (create its mask): global and local.

Global thresholding is a an algorithm that simply choses one threshold T for the whole histogram of the image. All pixels that are smaller than this threshold $x_{i,j} < T$ are assigned to be of class 0 (background) and all pixels that are larger than this threshold $x_{i,j} > T$ are assigned to be of class 1 (foreground). It is very difficult to find a good threshold manually that would work out for all the images, especially in this case, when the brightness does differ not only between the images, but also within the image itself (light gradient in Figure 18). To find a good threshold automatically (at least to some extect) Gonzalez et al., 2006 proposed the following algorithm:

Nevertheless, it is not obvious how to preselect an initial threshold in step 1. There are several options here, however keep in mind that there is also no single best solution among them. For example, when an assumption that the foreground occupies approximately the same area as the background holds, than initial threshold T should be chosen to be an average gray level and etc.

After trying out many different global threshold approaches, it has been derived that a

Algorithm 3 Global thresholding

1. Select an initial estimate for T.
2. Segment the image using T. This will produce 2 groups of pixels G_1 (all pixels $x_i > T$) and G_2 (all pixels $x_i < T$).
3. Computer the average gray values μ_1 and μ_2 for the pixels in regions G_1 and G_2.
4. Compute a new threshold value $T' = \frac{\mu_1 + \mu_2}{2}$
5. Repeat steps 2-4 until difference in the change of value T is smaller that a predefined parameter.

global minimum thresholding is the best one. It is implemented in *skimage.filters* and according to the documentation (Prewitt et al., 1965) works in the following way: it assumes that the histogram $p = (p_0, \ldots, p_M)$ of the image is bimodal, meaning that it has two clearly defined peaks (background and foreground). Afetrwards the histogram is iteratively smoothed using a running average of size $k = 3$. The points on the histogram are updated with the value a_k from Equation 32 until only 2 local maximas (a_l and a_r) are left.

$$a_k = \frac{1}{k} \sum_{i=n-k+1}^{n} p_i \tag{32}$$

Then the threshold is taken as the minimum between the two local maximas:

$$a_l \leq T \leq a_r \tag{33}$$

One clear downside of this approach though is that images which histograms have very unequal peaks or a broad and flat valley will be unsuitable for this method (*Thresholding¶* n.d.).

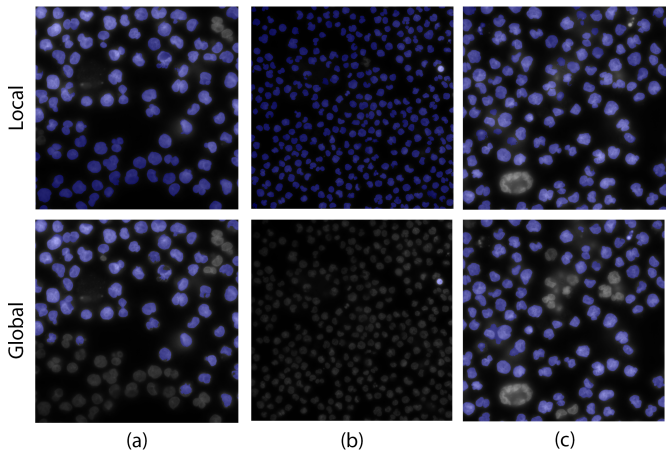


Figure 21: Local vs. global thresholding

Unfortunately this method did not perfom well even for the images that have an equal amount of foreground and background. And the reason for that is a mentioned above non-uniform illumination within the images. For visual comparison of global thresholding

applied to difficult images see Figure 21a, b, c Global. 21c is an image with equal brightness level, however even there some mistakes do appear.

However there is another better approach that performs well in such conditions: **local thresholding**. For nuclei segmentation exactly this algorithm was chosen. It is implemented in *skimage.filters* and can be used in the following way:

```
skimage.filters.local_threshold(img, block_size=7,
    method='gaussian', offset=0)
```

The main idea behind it is the following: instead of selecting one threshold for the whole image (global one), one can select several thresholds for each local region with a predefined size. The comparison between global and local thresholding is presented in Figure 21, where (a), (b) denote exteme corruption cases and (c) represents a normal illumination.

"The threshold value is the weighted mean for the local neighborhood of a pixel subtracted by a constant (Gonzalez et al., 2006)". With the image size of 2136×2136, the local neighborhood (or a *block_ size*) by experimenting with different values was chosen equal to 111. The default *method* used on for local thresholding is *gaussian*, *offset* value is a constant that will be subtracted from weighted mean of neighborhood during the calculation of the local threshold, by default this value is 0 *Adaptive thresholding¶* n.d.

Let z be a random variable that quantifies a gray-level value of the pixel, then the histogram of the image is a probability density function (PDF) $p(z)$. Since we assume that the image contains a background and a foreground, then this PDF is a mixture of two densities $p_1(z)$ and $p_@(z)$ weighted by the relative areas of these two classes (their number of pixels) P_1 and P_2. Then

$$p(z) = P_1 p_1(z) + P_2 p_2(z) \tag{34}$$

By assuming Gauassian model for both $p_1(z)$ and $p_2(z)$, one gets a Gaussian Mixture Model (GMM). Since we have assumed that each pixel can be asssigned to either a background of foreground only, $P_1 + P_2 = 1$ must hold.

Probability to falsely classify an background pixel as a foreground then is:

$$E_1(T) = \int_{-\infty}^{T} p_2(z)\, dz \tag{35}$$

And probability to falsely classify a foreground pixel as a background then is:

$$E_2(T) = \int_{T}^{+\infty} p_1(z)\, dz \tag{36}$$

The overall error is:

$$E(T) = P_1 E_1(T) + P_2 E_2(T) \tag{37}$$

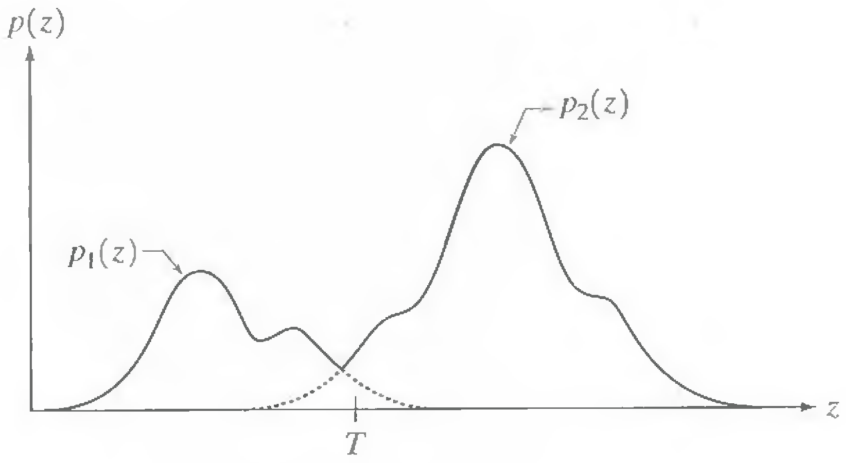


Figure 22: Histogram as a probability density function (Gonzalez et al., 2006)

By differentiating $E(T)$ wrt. to T and equating the result to zero the optimal solution will be:

$$P_1 p_1(T) = P_2 p_2(T) \tag{38}$$

Since Gaussian distributions have been assumed, it will hold that:

$$p(z) = \frac{P_1}{\sqrt{2\pi}\sigma_1} e^{-\frac{(z-\mu_1)^2}{2\sigma_1^2}} + \frac{P_2}{\sqrt{2\pi}\sigma_2} e^{-\frac{(z-\mu_2)^2}{2\sigma_2^2}} \tag{39}$$

With μ_i and σ_i^2 for $i \in \{1, 2\}$ being the mean and variance of the Gaussian distribution $p_i(z)$. This results in the following solution for T:

$$\begin{aligned} &AT^2 + BT + C = 0 \\ &A = \sigma_1^2 + \sigma_2^2 \\ &B = 2(\mu_1\sigma_1^2 - \mu_2\sigma_2^2) \\ &C = \sigma_1^2\mu_2^2 - \sigma_2^2\mu_1^2 + 2\sigma_1^2 2\sigma_2^2 ln\left(\frac{\sigma_2 P_1}{\sigma_1 P_2}\right) \end{aligned} \tag{40}$$

To escape two optimal solutions of the quadratic equation, it may be assumed that $\sigma_1 = \sigma_2 = \sigma$ and then:

$$T = \frac{\mu_1 + \mu_2}{2} + \frac{\sigma^2}{\mu_1 - \mu_2} ln\left(\frac{P_2}{P_1}\right) \tag{41}$$

Such threshold search is then applied to all of the subregions of the image with overlaps. Thresholds are calculated only for the regions that contain two clear peaks in their histograms and interpolated to the other pixels from the regions that do not contain them. If

Local Threshold	Global Threshold
0.3 sec	17 sec

Table 4: Threshold timing

the subregions does not contain two peaks, it simply means that there is no foreground or background object on it.

Of course local thresholding approach has a longer runtime time (see Table 4). Therefore when the inference speed is crutial one can still use *global minimum thresholding*. It does performs visually a bit worse that a local threshold (especially for the extreme corrupted cases), however for the normal conditions the performance is quite similar ot the local thresholding (Figure 21c). One should also keep in mind, that after the model is trained due to its generalization ability the gradient in intensities or overexposure will not be present anymore, as it is related to the image acquisition technique and does not depend on the DIC image itself.

3.2.4 Downstream metrics

In this subsection the evaluation of the model trained on full nuclei dataset with full set of augmentations is presented. Distributions of number of nuclei and their area are very similar in shape, the corresponding scatter plots form almost linear dependance. Distributions of intensities are less similar, the higher intensity of predicted images is proved here. However, high scores in Pearson and Spearman correlation coefficients (see Table 5) suggest that distributions are indeed close to each other. The difference between predictions and ground truth might be caused by an absolute value shift, that can be easily fixed.

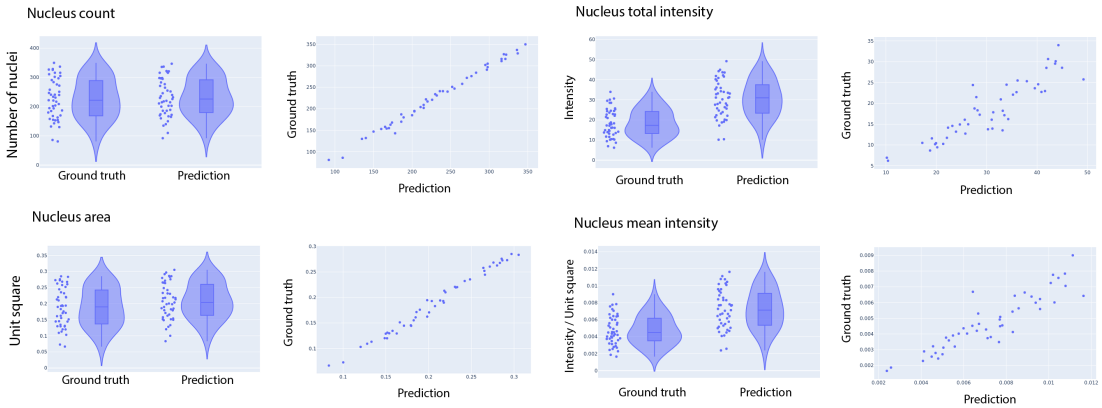


Figure 23: Metrics for downstream tasks on nuclei

Table 5: Correlation coefficients for downstream tasks on nuclei

	Pearson	Spearman
Number of nuclei	0.995	0.994
Total intensity	0.902	0.911
Mean intensity	0.907	0.904
Area	0.992	0.990

3.2.5 Influence of scaling on predictions quality

Additionally the quality of the model have been checked for different scales of the input. It was interesting to know whether or not the quaility of predictions differ based on the size of the cell within the crop. For that two additional datasets were prepared: one, where the images were first scaled up by a factor of 1.3 and another one, where they were scaled down by a factor of 0.7. Afterwards the image swere cropped as usual into 256×256 crops. Predictions quality were measured based on the downstream metrics described in Section 3.1.6 and results are presented in Table 6.

Table 6: 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	0.88	0.861	0.856	?
Mean intensity	0.922	0.906	0.88	0.872	?
Area	0.991	0.992	0.961	0.952	?

As a result, the bigger the cell is within the image, the better its intensity can be predicted (both total and mean). However the number of nuclei and their area, are mostly not affected by the size of the cell. Also training on the bigger cells and predicting on usual ones as well as vice versa, doesn't change the accuracy as long as the difference is not significant. However when the difference becomes bigger (training on the enlarged cells and predicting on the smaller ones), the accuracy of the model decreases. By the accuracy for all these value PCC is assumed.

3.2.6 Generalizability across phenotypes

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

3.2.7 Conclusions

Fluorescence staining of nucleus in CHO cells used for recombinant protein production at Merk KGaA can be successfully replaced with *in silico* labeling using deep neural network. Provided dataset is big enough to achieve accurate predictions, nevertheless a further research could be dedicated to improving fluoreschence image preprocessing in order to

remove the blur around the nucleus, as well as to cleaning the data from the under- or overexposed images.

The model successfully converges, and the use of augmentations improve its performance. It is recommended to use Person correlation coefficient or downstream metrics to evaluate the model's predictions as MSE is not representative of a model's quality in this case. The performance of a model is evaluated in terms of practical downstream metrics. A further research can be also conducted to advance the architecture of the model, as the use of more parameters improved the predictions significantly both visually and in metrics.

In order to perform segmentation of the nucleus it is recommended to use a local thresholding approach when the inference time is not crucial and global minimum thresholding algorithm when the inference time is critical. Training the model on cells of a larger size slightly improves the intensity accuracy. Yet a slight changes in the cell size do not influence model's performance significantly.

3.3 Endoplasmic Reticulum

Endoplasmic reticulum (ER) is a network of membranes inside a cell, through which proteins and other molecules move. Ribosomes are small and round organelles with the main function to produce the protein needed for the cell. They are located in a continuous membrane system that forms series of flattened sacs, this membrane is called ER (see Figure 10) (*Endoplasmic Reticulum (smooth)* n.d.). ER itself is moslty located around the nuclei. In order to stain ER, CHO cells were treated with Donkey Anti-Rabbit IgG antibody. This is a fluorescent stain that binds strongly with ER. The analysis of this cell organelle is also important for the CLD as ER is directly related to the process of protein production within the cell: it is responsible for the synthesis, folding, modification, and transport of proteins (Kara, n.d.). The proximity of the ER to the nucleus essentially allows to control the protein production. For example, when the protein is incorrectly folded it will be accumulated in the ER lumen and it will serve as a signal to activate misfolded protein response from the cell. In contrast to other imaging datasets ER dataset does not require special preprocessing steps as the images are of a good quality and without any visible problems (see the ground truth image in Figure 25).

3.3.1 Training and predictions

During the training with PCC loss the model has successfully converged already after 35 epochs, however a clear overfit was encountered (see Figure 24 (left)) with the best PCC loss before the overfit being 0.0713. Overfitting happens due to the lack of data, as for ER case there were much fewer samples than for nuclei for example (see Table 1). Even though one could use early-stopping approach and simply choose an earlier epoch before overfit, the better approach would be to use regularization methods described in section 3.1.5.2. Additional regularization in terms of data augmentations was introduced. The new learning curve is shown in Figure 24 (middle). Overfit happens now much later (after 120 epochs) and PCC loss improves to 0.0701. Introducing a stronger regularization with the use of weight decay in adadelta optimizer and dropout layers with a dropout rate of 0.1 (see Figure 24 (right)) reduces the overfit completely, however at the same time increases the loss to 0.099.

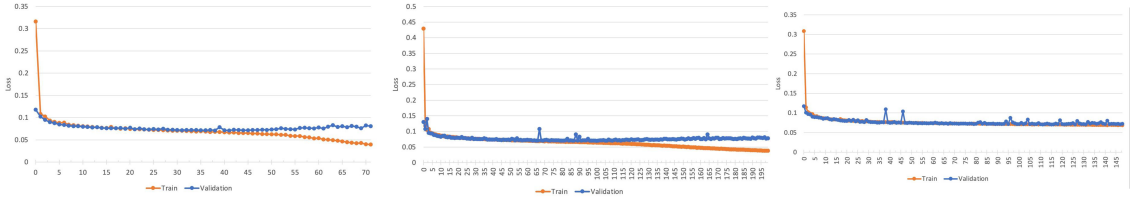


Figure 24: Default model (left), slightly regularized model(middle), strongly regularized model (right)

Although the validation loss seem to be higher in comparison to 0.0701, in this case the growths were not explained by the difference between the validation sets, where the loss

was measured on. PCC losses on the same validation set for both were 0.0701 and 0.0742. It might have been the case that the regularization was too strong. In this case it would be better to use early-stopping approach with the epoch that has the lowest loss.

3.3.2 Combination of nuclei and actin predictions

Now having two models for the predictions of two organelles: nuclei and ER, it is interesting to visualize the predictions together (see Figure 25). It is clear from the image that ER indeed is located around the nucleus. Unfortunately, the cells were stained only for separate oragnelles and different fluorescent target do not intersect, meaning there were no images with both nucleus and ER stained at the same time. A further research can be applied here to in order to evaluate the models even better measuring the intersection error between the two for instance.

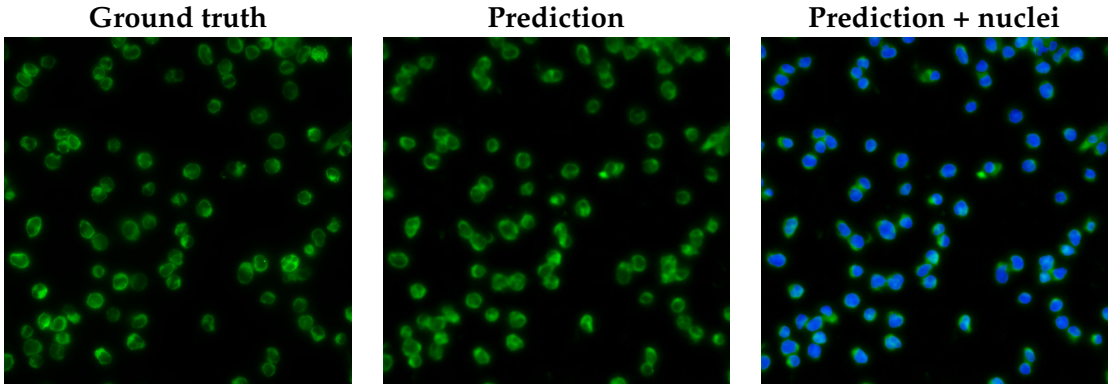
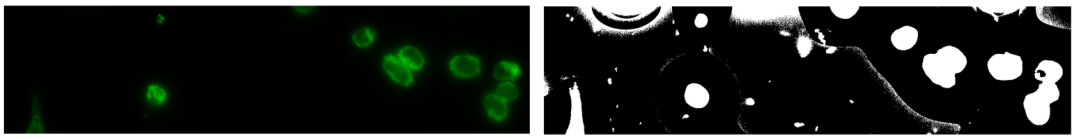


Figure 25: Combination pof ER with nuclei

3.3.3 Postprocessing for ER segmentation

The process of ER fluorescence segmentation is somewhat different in comparison to nuclei segmentation. These fluorescence staining has a stronger "shining" around the ER itself and therefore any method for background removal would be helpful to reduce it. One of such methods based on the rolling ball algorithm is described in more details in section 3.4.1.1.

Initially it was established that a local thresholding approach for segmentation would be nessecary here as well, as the images also have a non-uniform illumitation. However a downside of a local thresholding algorithm is the appearance of the artifacts briefly mentioned in the previous section and presented in Figure 26. Even though the background in fluorescence imaging appears to be completely black, it still contains some slight signal (non-zero values), that are boosted by a local threshold and becomes an unwanted artifact.



Fluorescence input Local threshold artifacts

Figure 26: Artifacts from local thresholding algorithm

Such falsely recognized regions appeared in the nuclei as well, yet there they were easily filtered out based on the shape criteria. All nuclei are almost round and convex objects, whereas background artifacts are prolonged completely non-convex objects. Unfortunately, such filtration cannot be applied to ER imaging as very often ER from one cell is located very close one to the ER from another cell, and together they may form a long non-convex object, that now could not be filtered out this easily. That is why it is very important to remove the background noise here first before applying a local threshold.

In order to do that one can first apply a rough "over-predictive" global thresholding, that will cover a true signal fully, including the "shine" around the ER, but will ignore the background noise. In the role of "over-predictove" global mean thresholding algorithm can be used (see Figure 27.2). The mask created with the mean thresholding approach is used to zero out all the pixels that are not covered by it. And after that the local thresholding can be successfully applied with the *block_size* of 181 (Figure 27.4). Then algorithm fills in all the holes in the middle of ER that might have appeared during the thresholding. Morphological opening (see Section TODO cite section) and Gaussian Blur with the squared kernel 3×3 are applied. Connected components are detected afterwards and filtered based on the limit of the area they occupy, this filters out mostly very small components from the mask which might be produced by the left out background noise. The whole algorithm overview is described below:

Segmentation steps are described in Algorithm 4 and also illustrated in Figure 27.

Algorithm 4 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.

3.3.4 Downstream metrics

In this subsection the evaluation of the model trained on ER dataset with stronger regularization is presented. The accuracy of all downstream metrics are even better than

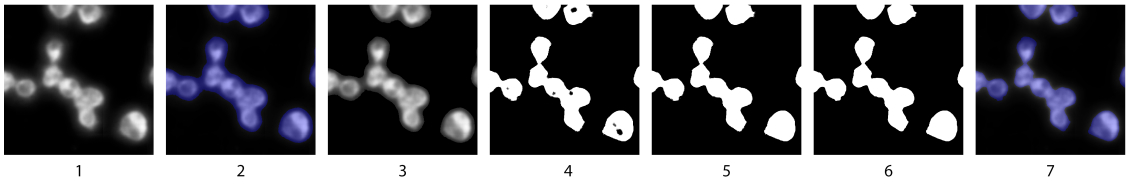


Figure 27: Segmentation steps in ER postprocessing

for nuclei, espesially mean and total intensities correlations are higher. Visually predictions do not look significantly brigher as well. Pearson and Spearman rank correlation coefficients are presented in Table 7.

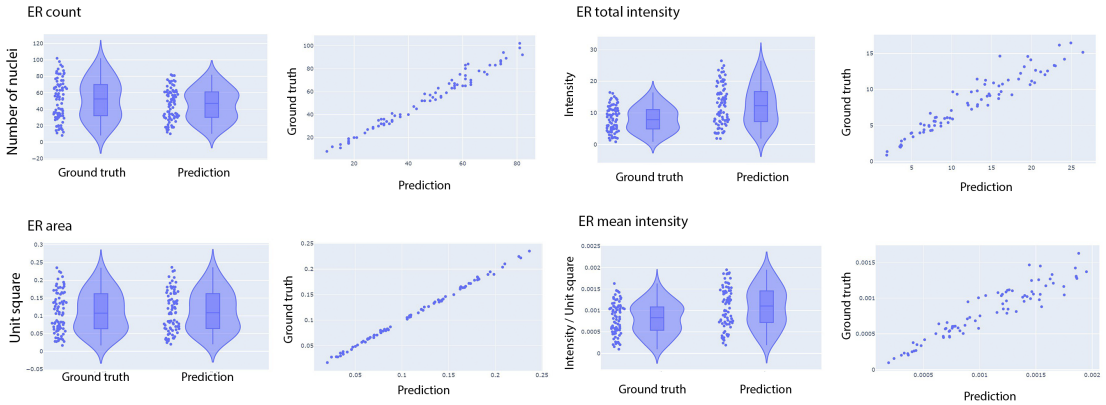


Figure 28: Metrics for downstream tasks on nuclei

Table 7: Correlation coefficients for downstream tasks on ER

	Pearson	Spearman
Number of ER	0.988	0.984
Total intensity	0.952	0.955
Mean intensity	0.941	0.933
Area	0.991	0.990

3.3.5 Conclusions

Training the model on the ER stain images produced very successfull results that can be used as as replacement for manual staining with *in silico* labeling. Overfit of the model has been encountered and several experiments were performed to overcome it, the best checkpoint was selected and evaluated.

The segmentation of ER for further downstream metrics evaluation was proposed. The problem of presence of the overlap between the cells causing the segmented regions to merge together is discovered and a better preprocessing algorithm was proposed. This algorithm helps to avoid the artifacts appearing from a local thresholding.

The combination of ER with nuclei predictions was visualized and analysed, it was confirmed that the models produce realistic results confirming biological definition of their mutual arrangement within the cell. However, it is recommended to test the models using the simultaneous staning of both organelles to get a quantative metrics of the accuracy.

3.4 Golgi

Golgi apparatus (or Golgi, or Golgi complex) is another organelle inside the cell that packages proteins into membrane-bound vescicles, that will be exported from the cell. Golgi is located also near the nucleus and ER as well, it is even called a perinuclear body. This location is explained by the bilogical processes: after a protein comes out of the ER, it goes into the Golgi for further processing (*Golgi body* n.d.). The staining of Golgi apparatus has turned out to be the most difficult of all. Two antibodies were tried out for stating here, however both of them resulted in a low signal-to-noise ratio. Many images were underexposed, and the density of the cells after their fixating was pretty low. The hypothesis why this was the case was that the choice of target protein in Golgi to which antibody can bind to was not the best. Golgi appartus is a difficult target even having a fluorescence imaging with high signal-to-noise ratio (lowest scores among all cell organelles in state-of-the-art Cheng et al., 2021 paper were achieved exactly on Golgi).

3.4.1 Preprocessing

One can see an example of how Golgi fluorescence staining looks like in Figure 29. It is evident that there is a lighter foreground fluorescence and a bit darker one in the background. A true Golgi signal here is considered to be only the lighter part of fluorescence lightning. The light gray background present here is called a non-specific fluorescence lightning. It comes from the cell itself, and might occur when the antigen is impure and contains antigenic contaminants (Borek, 1984). And brightness of it may vary due to longer or shorter exposure times.

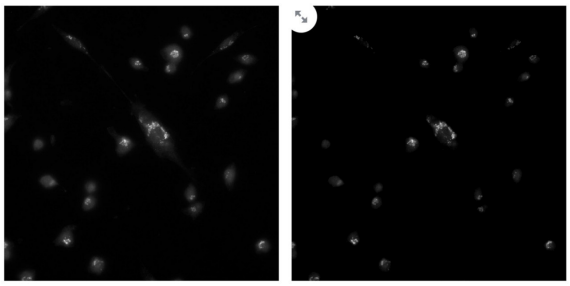


Figure 29: Golgi enhancement: left — original signal, right — enhanced image

Having such a non-specific fluorescence background has two challenges for training:

- The relative area of the background fluorescence is bigger than the area of Golgi themselves. Therefore quite a big part in the loss during training will be dedicated to teaching model to restore this background fluorescence instead of the Golgi itself.
- It introduces difficulties during the post-processing of the predictions. As well as for nuclei, the mask of the predicted Golgi apparatus is needed for further evaluation of the downstream metrics. Using the same algorithm for post-processing segmen-

tation that was used for nuclei, the mask of the Golgi Apparatus will consider the background noise to be relevant, although this is an unwanted behavior.

On the left of Figure 29 one can see the original ground truth image and on the right — the images after the background was substracted. One of the great background subtraction algorithms is the rolling ball algorithm which is described below.

Additionally, all of the crops used for training were filtered base on the amount of background they contain. Without filtering the crops that contain a lot of background, a big portion of a dataset would be black crops only, which creates very strong class imbalance.

3.4.1.1 Background removal algorithms

Rolling ball algorithm was introduced by Sternberg, 1983 and is still widely used for processing medical and biological data. The idea of this algorithm is based on morphological opening of the image.

Definition 3.1 (Morphological opening)**.** Morphological opening is an operation in image processing, when an image is first eroded and then dilated using the same structuring element.

Morphological opening is helpful for removing noisy elements, thin lines, while preserving bigger objects in the image (*Types of morphological operations* n.d.).

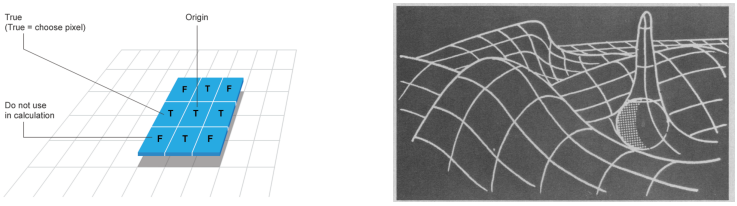
A structuring elements is an analogue of a kernel in image processing. It is a matrix of zeros and ones (true and false), where ones represent the elements that will be used to perform the morphological operation and others will be ignored (see an example in Figure 30 (left)). Such a structuring element is applied across the whole input image producing a new image based on the rules of a morphological operation it performs.

For example, morphological dilation takes a new value of the pixel as the maximum value of its neighbours within the structing element. Therefore after this operation lines will be thicker and in general objects will appear bigger.

Whereas morphological erosion makes the pixel value as the minimum value of its neighbours within the structuring element. After this operation the floating pixels will be removed and all object become smaller and thinner.

Sternberg, 1983 has extrapolated the operation of morphological opening from 2D into 3D space. He defines a new interpretaion of a 2D image in 3D world called umbra. Umbra can be described as a 3D plane, where the height of ech point is determined by its intesity value.

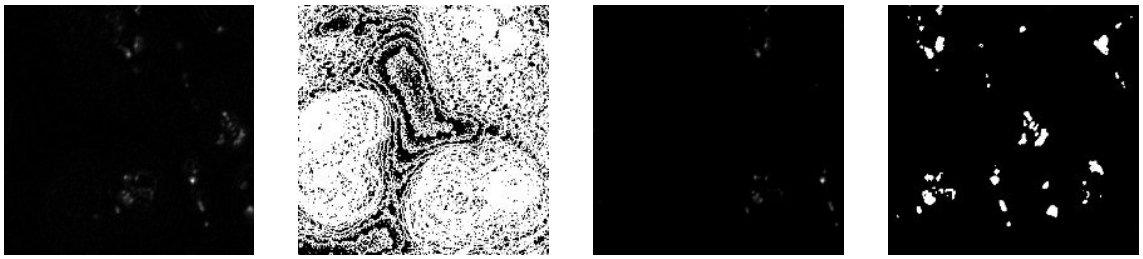
The structuring element for morphological opening of an umbra has to be then also a 3D object — in this case a ball. Morphological opening of an umbra is a union of translations of the 3D structuring element that can be entirely contained inside it (see Figure 30). One can image the ball freely moving inside the volume constrained by the upper surface of an umbra. The opening then consists of all the pixels then can be reaches by the ball. The radius of the ball is a hyper-parameter which has to be tuned.



(a) Structuring element (b) Rolling ball

Figure 30: Background removal

Subtracting the background with rolling ball algorithm unfortunately is still not enough to get a reasonably clean signal of Golgi Apparatus from fluorescence imaging, as a lot of background noise will still be present there. In Figure 31ba one can see an fluorescence image preprocessed with a rolling ball algorithm. Let's turn it into a binary image that you can in Figure 31bb. It clearly still contains a lot of background noise of low intensity and



(a) (b) (c) (d)

Figure 31: (a) Vanilla pre-processing with automatic background removal algorithm only; (b) masked or subfigure (a); (c) Additional clipping of lower intensities after vanilla pre-processing; (d) mask of subfigure (c)

which was not visible for an eye. In order to get rid of it one could clip lower intensities of the image via the following approach:

```
import numpy as np

def clip(image):
  minval = np.percentile(image, 90)
  image = np.clip(image, minval, image.max())
  image = (image - image.min()) / (image.max() - image.min())
  return image
```

The result of the clipped image and its mask are illustated in the Figure 31c, 31d correspondingly. It contains almost no background noise now. Such additional clipping might improve the results slightly.

3.4.2 Training and predictions

Applying the usual model architecture with PCC loss too train the model to predict fluorescence signal from DIC imaging for Golgi did not bring a good results. Model seems to

converge (see Figure 32), however there is no learning happening.

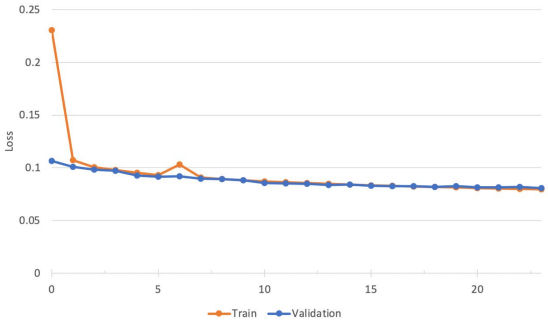


Figure 32: Straightforward training doesn't work

This was also depicted in visual predictions where the predicted fluroescence mostly contains dark pixels only (see Figure 33). Altough it seems that some pattern is hidden behind the dark pixels, visualizing it simply shows that the model picks up on the cell outline itself and does not give any useful information on the location of Golgi. One can see this by normalizing the predicted dark image to the range $[0, 1]$ (Figure 33).

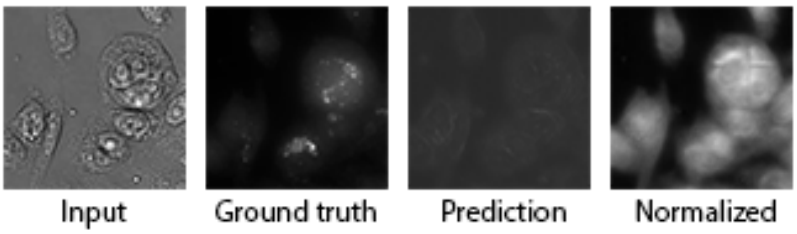


Figure 33: Training on original data

While in Figure 33 only the results of predictions for one crop are presented, it would be interesting to see how the image with combines crops would look like. This is presented in Figure 34. One can notice an interesting pattern there: essentially the model is predicting some signal across the whole cell with a brighter regions in some of them. Yet these regions do not have a correct location wrt to Golgi. Also it is important to keep in mind, that the image of the right in this Figure is a normalized one and a true image is almost fully black.

This experiment brought a hypothesis that the background removal algorithm might have reduces the signal-to-noise ratio and remove some important fluroescence signal along the way. Therefore an alternative approach of a background removal had been tried, where only enhancing via clipping described above has been applied directly on fluprescence imaging (without using a rolling ball algorithm). This produces images that still have much more non-specific fluorescence background (see Figure 35 (second image)), yet this

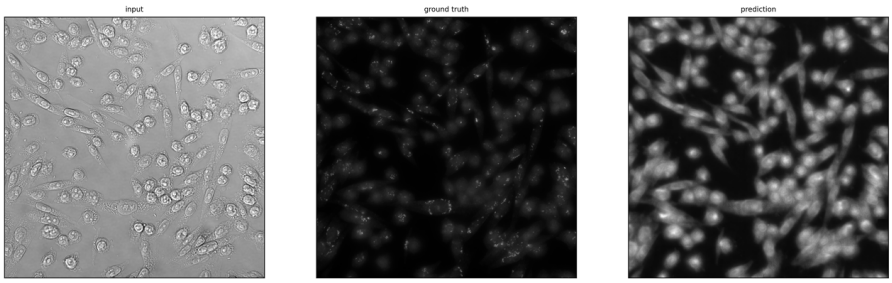


Figure 34: Full size predictions

preprocessing alternates the initial image to a much lower extent.

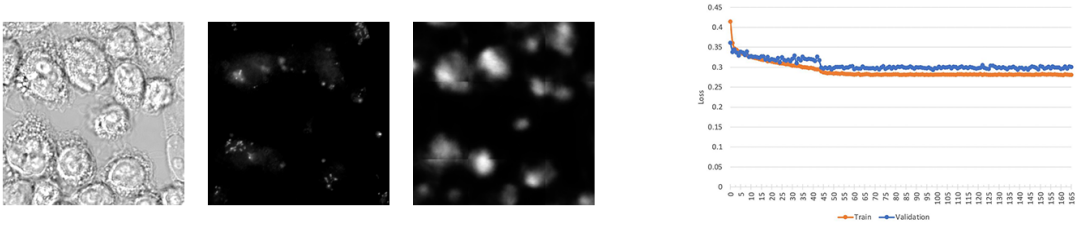


Figure 35: Training on the enhanced data

In this case predictions are not completely black anymore, however their quality is still very low. Images used in these training experiments were effectively coming from different staining procedures, using different antibodies, fixation process and microscopy settings. This brought up an idea to train the model on few selective datasets only that were created with the first staining approach only. The preprocessing was chosen as in the previous example — with the use of enhancement and without rolling ball algorithm as it has shown the best results.

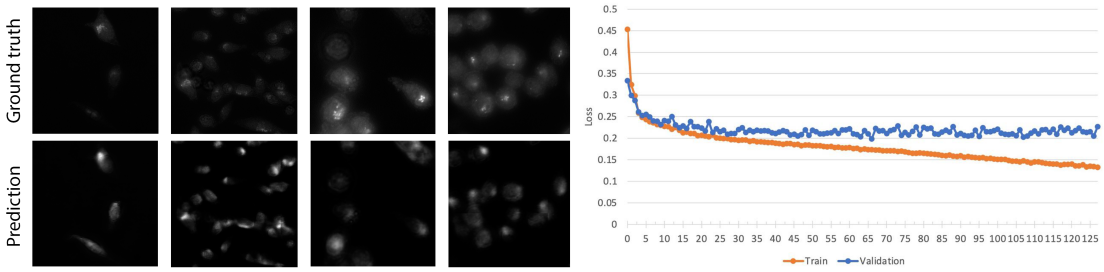


Figure 36: Small subset of the best staining

The predictions from this experiment you can find in Figure 36. Due to the stricter filtering and the use of much less data there were much smaller crops which were used for training. In this case only 4251 crops were used for training and 406 for validation purposes,

which directly led to overfitting after 70 epochs. However, the predictions did improve visually as well as the loss dropped to 0.19. Nevertheless the losses from these two experiments should not be compared directly as they were evaluated on two different validation sets. The improvement of the results indicated the careful choice of data samples taken from the same staining approach with the same settings, that does not require a severe preprocessing and already has a high signal-to-noise ratio might help the predictions significantly. There is a potential here for a better regularization of the model trained on the small subset of data. Yet it is clear that the acquisition of a better fluorescence staining is much more crucial here.

3.4.3 Alternative ways to improve predictions

3.4.3.1 Asymmetrical losses

The earliest learnings from the experiments have shown that the severe class imbalance is present and the model tends to predict mostly pure background — black images. In order to overcome this one can use an asymmetrical loss during training. Similarily to a weighted loss for a classification problem with class imbalance issue present, a weigted loss for segmentation task can be introduced. In this case different pixels from the prediction will recieve a different weight based on some criteria. Yet the use of weights cannot be easily defined for a Pearson correlation coefficients, it is possible to use them with MSE loss, because weights coefficients there can be added directly in front of the squared difference between pixels.

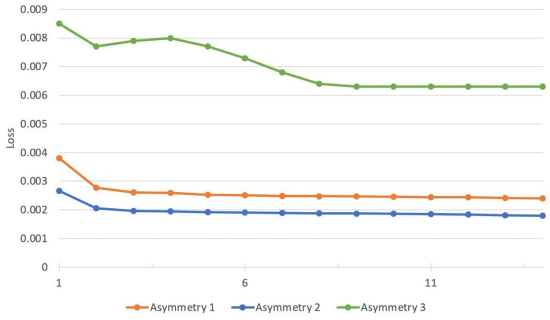


Figure 37: Punishing over and under predictions with asymmetrical MSE loss

In Figure 37 training learning curves from three asymmetrical approaches are presented.

Asymmetry 2 is aimed to punish the errors on bright pixels more: when a models pixel prediction is higher than a true one the loss will be higher. Yet this loss also encourages the model to underpredict and results in completely black images even though it has the lowest loss.

```
residual = prediction - ground_truth
loss = torch.where(residual < 0, residual ** 2, 2 * (residual) **
    2)
loss = torch.mean(loss)
```

Asymmetry 1 is aimed to do the opposite and punish an underprediction: when a model's pixel prediction is lower than a true one the loss will be higher. This resulted in a slightly brighter images

```
residual = prediction - ground_truth
loss = torch.where(residual > 0, residual ** 2, 2 * (residual) ** 2)
loss = torch.mean(loss)
```

Asymmetry 3 is a stronger version of Asymmetry 1 and it results in

```
    residual = prediction - ground_truth
    loss = torch.where(residual > 0, residual ** 3, 20 * (residual)
        ** 2)
    loss = torch.mean(loss)
```

All of the approaches above do not bring a big change in the performance and a mostly black image remained to be an output. Interestingly, punishing underprediction has essentially backfired, as loss then supports an overprediction. Because setting the weights of one class to be smaller is the same as setting the weights of the other class to be larger, and here as a result the model is more likely to overpredict. That is why the second asymmetry has a better loss, even though the logic behind it is not that obvious at first.

There were other interesting approaches in asymmetrical losses tested that are depicted in Figure 37

1. Adjusting overall brightness. Leads the an absence of bright spots and brightness gradients in the prediction. The illumination of the cell becomes same across all the cells.

```
loss = loss + prediction.sum() / ground_truth.sum()
```

2. Adjusting overall brightness with a reversed division. Leads to fully white images as this would minimize the fraction in loss.

```
loss = loss + ground_truth.sum() / prediction.sum()
```

3. Multiplying loss with prediction wiill result in black images again, however multiplying with the ground truth gives a bit more interesting result. However the model is pushed to predictin average gray color scross the whole image mostly.

```
loss = MSE(ground_truth, prediction)
loss = torch.mul(loss, ground_truth)
```

4. Multiplying loss with 1 - ground_truth also results in completely black images as such loss puts more weights on the correct prediction of the background.

```
loss = MSE(ground_truth, prediction)
loss = torch.mul(loss, 1 - ground_truth)
```

5. This improves the asymmetry approach 3, however now the highlighted regions simply include mostly the whole cell.

```
loss = MSE(ground_truth, prediction)
loss = torch.mul(loss, ground_truth) + ground_truth.sum() /
    prediction.sum()
```

6. Usual MSE.

7. Usual PCC.

From the experiments it was clear that the best approach remain to be PCC, pure MSE or MSE with the adjustment of overall brightness.

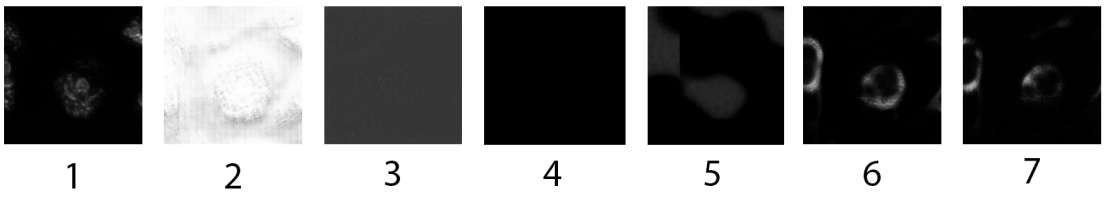


Figure 38: Results of advanced versions of MSE training

3.4.3.2 Use of gradient in loss

One of the insights from the predictions is that there is not enough details inside the predicted fluorescence. The texture and the gradient seem to be missing and predictions are too smooth. Although this is not very crucial for other organelles, Golgi apparatus stain on the other hand has a very granular structure and consists from the small dots (see ground truth image in Figure 33). In order to introduce the granularity we suggest to try to introduce gradient of an image into the loss function. For example, by calculcating image gradient with Sobel filters for ground truth and prediction and incorporating the difference into the loss function. Sobel filter is a convolutional operation on the image with two filters, one for detecting horizontal derivative approximations:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A, G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{42}$$

where A is an image. Then the resulting gradient is:

$$G = \sqrt{G_x^2 + G_y^2} \tag{43}$$

3.4.3.3 Noise reduction methods

Cheng et al., 2021 puts a special attention on the significant importance of a good signal-to-noise ratio for quality Golgi fluorescence predictions. This paper uses an advanced

denoising algorithm called *noise2void* (Krull et al., 2019). This paper suggest a strong method that does not require pairs target clean signal and noisy images available, but simply assumes that that there are only noisy images available. They use the idea that if one could acquire multiple images with the same signal but with different realizations of noise, then after averaging across these images the resulting image would approach the true signal.

This might be a useful approach for denoising Golgi imaging as well instead of the use of enhancement and rolling ball approached. It is highly advisable to try this method in future research.

3.4.4 Conclusions

Golgi apparatus the most difficult target in this research and the models trained on Golgi data require futher research. The difficulty in predictions of Golgi is explained due to the low signal-to-noise ratio of the acquired ground truth fluorescence imaging, overall low density of the cell in images and the low quality of the antibodies chosed for cell staining.

Golgi imaging requires a strong preprocessing such as enhancement and background removal with rolling ball algorithm due to the presence of non-specific fluorescence lightning. However even these approaches do not significantly improve the quality of training data and the predictions are still not accurate. It is recommened to use a *noise2void* denoising algorithm called (Krull et al., 2019) as well as explore other losses that would include the granularity of the Golgi stains, for example a loss that uses Sobel filters.

3.5 GFP

Green fluorescent protein (GFP) is protein that produces bright green fluroescence from the whole cell surface. Some cells express this protein naturally, therefore there is no need to perform cell staining or fixation in this case and the imaging can be done on living cells. Fixation of the cells is a processes of destructing their membrane and fixing them on the plate in order for the stain antibody to be able to enter the cell. The difference in DIC imaging between fixed and not fixed cells is presented in Figure 39. Clearly, cell membrane is intact and clearly defined in not fixed cells, whereas in not fixed ones there is almost no definite membrane present.

Nevertheless, in order to get training data for all other targets (nuclei, ER, Golgi apparatus) the staining procedure is unavoidable. And staining requires the cells first to be fixed. That brings up the limitation of current *in silico* labeling research — cells have to be fixated in any case. DIC imaging of living and fixed cells look very different and models trained on fixed cells do not generalize well to not fixed ones. Luckily, fixing cells is not a cumbersome lab procedure and is far easier than staining the cells, which is escaped with the help of *in silico* fluorescence labeling. After successfull training of the model on living GFP expressing cells, we found out that other models cannot perform that well on living cells. Therefore, the cells were fixed in order to look alike with previously acquired data and the experiments were repeated. Nevertheless, we recommend to look into the possibilities of transfer learning from fixed to live cells. The results of training on the fixated cells are presented below.

For this experiment another cell phenotype was chosen — H19. Training the model to predict GFP fluorescence essentially allows to predict the area of the whole cell, that is used in further downstream analysis. There is no need to capture the intensities as they do not bring any useful features for selection step in CLD. However they might help for algorithms like watershed to find separate cells in order to count them. Yet the task can be simplified to predicting binary mask of GFP signal too. Binarized images are well-suited for cell area predictions. Altough one has to find a corresponding image preprocessing pipeline in order to convert training intesity fluorescence imaging into masks first. Both tranining variations are provided in this chapter — with and without prediction of the intensities.

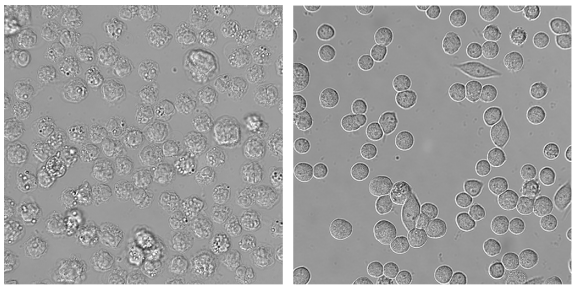


Figure 39: Examples of fixed and not fixed cells DIC imaging

3.5.1 Preprocessing

In order to convert intensity fluorescence image into a binary image any of the previously described threshold algorithms can be applied. Here intensity of GFP in different cells may differ due to some of the cells being in focus and others being outside of it, that is why local thresholding algorithm has better results than any global thresholding approach.

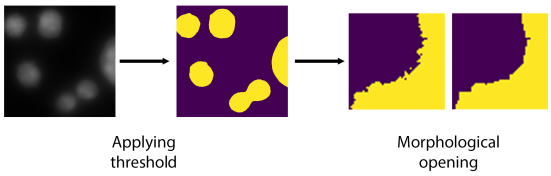


Figure 40: Converting GFP into a binary mask

Figure 40 displays the pipeline of converting an intensity image into a binary mask. After applying local threshold algorithm, additional morphological opening operation has been performed in order to smooth the borders of the cell and remove noise.

3.5.2 Predictions

Figure 41 presents the result of training with PCC loss on intensity images. The model convergence and visual comparison between ground truth and predictions is displayed. One can see that some of the cells are present in predictions, but are missing in ground truth images. These are dead cells that do not express GFP anymore, however the cell body is still present in DIC. Additional observation is that the boundary is generally blurry than in ground truth fluorescence, which is typical for all previous organelles as well. However all cells are clearly visible and can be separated using additional image postprocessing pipelines.

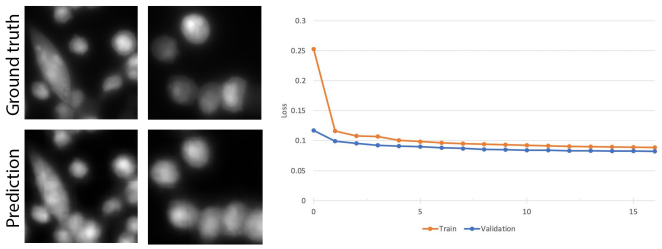


Figure 41: Training with Pearson correlation loss

In Figure 42 one can find results of training on the binarized image dataset. Model does converge and predictions are successfull as well. Without the intensities it might be more difficult to visually separate the cells from the mask, rather then in the image with intensity values. The predicted masks are not binarized yet and contain continuous values from

$[0, 1]$ interval. One can see how the model generalizes in the following observation: preprocessed ground truth image still has not very strict boundaries in its masking even after local thresholding binarization. After zooming in one can see that some cell's boundaries were oversegmented. This happens due to the different intensity values for different cells. However, this is not a an issue as the model is still able to generalize well and predicts the middle part of the cell confidently, then smoothly reduces the confidence on the cell boundary. After thresholding such predictions one can choose such a threshold that will get just enough of the cell boundary needed. In our case the value was chosen to be 0.8.

An interesting detail here are dead cells mentioned above. They are present in DIC, but do not express GFP. One can see a clear example of such cells selected with green circles in Figure 42. The model generalizes in a way that it continues to segment all of the cells regardless of their state. This is an expected behavior as the amount of dead cells in the dataset is pretty low and ocasional mistakes do not push the model strongly enough to be able to learn the features that separate dead from alive cells. We proceed with the existing model to futher evaluation, however it might be useful for further research to address this issue. For example, Christiansen et al., 2018 mentions the ability of their model to detect dead cells from alive by staning the dead ones and then traning the model to differentiate between them.

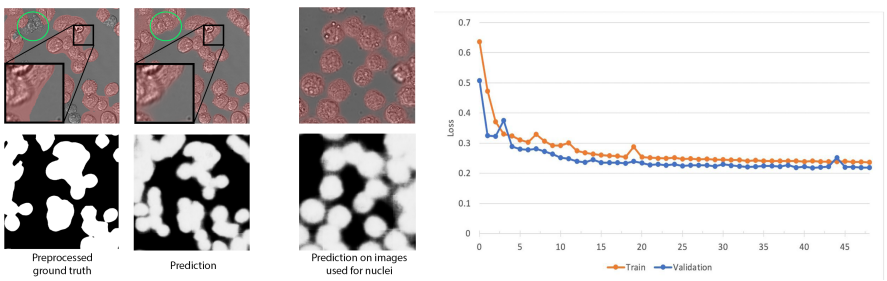


Figure 42: Training with BCE loss

3.5.3 Downstream metrics

TODO move to separate chapter?

For the train models we provide the resulting metrics on the downstream tasks. In this case only two metrics are important for evaluation — cell area and cell count (see Table 8). Both Pearson correlation and Spearman rank coefficients are lower than in previous experiments. Influence of additional segmentation of dead cells by the model lowers correlation scores, however they still signalize the presence of strong correlation between prediction and ground truth.

Table 8: Correlation coefficients for downstream tasks

BCE loss	Pearson	Spearman
Number of ER	0.67	0.64
Area	0.82	0.75

Violin and scattter plots depicting two metrics mentioned above are presented in Figure 43.

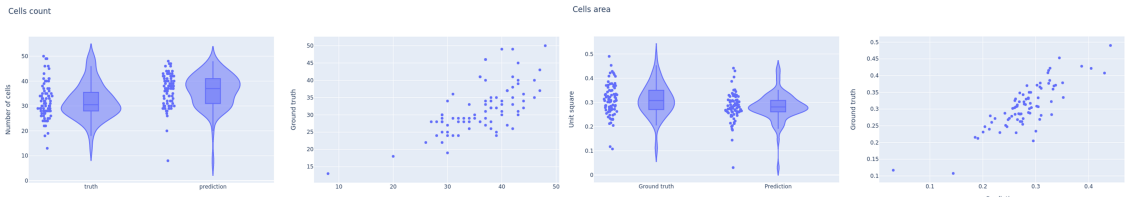


Figure 43: Downstream metrics

3.5.4 Combination of GFP, nuclei and ER

Now having three successfull models that are able to prediction GFP fluorescence from the whole cell, nuclei and ER, one can combine their predictions together for the same image. For that an output RGB image was contructed, where each prediction takes one channel. In this case channel correspondence is the following: red — ER, green — GFP, blue — nuclei. The resulting image is shown in Figure 44. Additionally one can see, that GFP model has successfully generalized on other cell phenotypes (CHOZN, PHX), although it was train on H19 only.

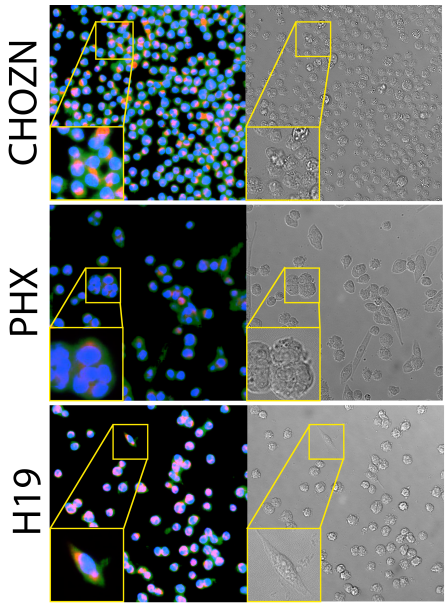


Figure 44: GFP, Nuclei and ER combined

3.5.5 Conclusions

GFP *in silico* fluorescence labeling can successfully be used for counting number of cells and estimating their size. The created models can both predict the intensity images as well as binarized masks. Intensity predictions might be more useful for number of cell estimation as the cells separate there visually better. The models was evaluated on two donstream metrics: cell size and cell count. Correlation coefficient suggest a strong correlation between the predictions and ground truth. The model has limitation in its diability to differentiate between dead and alive cells. This issue should be addressed in further reasearch after acquiring data for labeling dead cells. Despite the difficulty of clear determination of cell boundary during image preprocessing for fluorescence image binarization (its overprediction in some cases) the model can successfully generalize and predict a correct boundary for all cells. The images of not fixed cells were provided for the first time during GFP experiments and it was clear that previously trained model do not generalize on them well, therefore the limitation of research in terms of the cell fixation need was determined. However, its recommened to look into possible transfer learning approaches to get rid of fixation step completely.

4 Stability study

4.1 Stability study

For practical reasons it is important to not evaluate the models on high-quality data only, but also to know how the predictions will degrade when input's data quality decreases. Having a model for fluorescence *in silico* labeling that can additionally alarm when the predictions should not be relied upon is very useful in practice. Although the DIC microscopy is a relatively easy technique, there are still setting up procedures taking place that can be prone to errors. Additionally, as the models are not easily generalizable across phenotypes as well as between fixed and not fixed cells, an alarming system that is able to catch these situations would be useful to save time and costs of lab work. In order to measure the stability or robustness of the models towards data degeneration they were evaluated on the corrupted or "bad" input DIC images. There are two sources of "bad" images that can be used for such estimations. One are real corrupted images made in the laboratory. Such corruptions may arise from different sources: for example, bubble of oil that landed on the microscope lenses, low density of the cell on the image, over- or underexposure during image acquisition. Another source of image corruptions would be images with artificial or pseudocorruptions created manually via image processing.

4.1.1 Artificial corruptions

In this subsection results from evaluating models on 3 types of artificial image corruption are presented, namely: defocus blur that imitates the defocus of the microscope lenses, changes in brightness and changes in contrast. Every corruption has different effects on the prediction of the model based on its severity level. Therefore it is important to evaluate the error-rate (in this case a loss function) for the predictions for different severity levels of each of corruption types presented. It is also important to perform a visual evaluation of model's predictions on corrupted data. Each corruption c has severity levels s, where $-5 \leq s \leq 5$ ($0 \leq s \leq 5$ for defocus blur corruption). 0 here corresponds to an original image without corruption. It is important to keep in mind that although severity levels were chosen to be as much comparable between each other as possible, they still might have differences in their strength. For example, contrast has much stronger effect on predictions than brightness changes. Three types of artificial image corruption are presented below.

4.1.2 Defocus Blur

Defocus blur corruption imitates the effect of defocus on the microscope. The blur is applied to the image by convolving it with a special defocus kernel. There are two tunable parameters for this corruption type: first one is the readius of the circle in the kernel r, and the second one is the blur strength parameter s. Examples of the kernel with radius r is shown in the Figure 45. Such kernel is then simply applied to an image via $cv2.filter2D$

function.

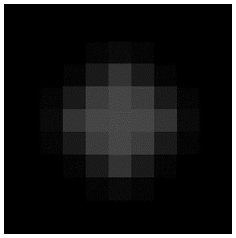


Figure 45: Defocus blur kernel

4.1.3 Brightness

Different brightness levels is also an important image corruption to test on. Brightness variations appear often in the dataset during image accquisition. In oder to change the brightness, an image from the RGB format was translated into HSV format, which stands for hue, saturation and value. This is also one of the popular formats to represent an image. To make an image brighter or darker, one can simply add or subtract a parameter s in a value channel for each pixel $x_{i,j}$ correspondingly. This parameter is often called bias. The bigger absolute value of this change the stronger a corruption will be.

$$\hat{x}_{i,j} = x_{i,j} + s \tag{44}$$

4.1.4 Contrast

In contrast to adding a constant value pixelwise to an image in order to change a contrast level one can perform a multiplication of an image with another constant s. This parameter is often called gain.

$$\hat{x}_{i,j} = s * x_{i,j} \tag{45}$$

For both contrast and brightness changes one can use $cv2.convertScaleAbs()$ from OpenCV library. This method directly accepts gain and bias parameters and clips the image to stay within the allowed values range.

The values of hyperparameters used in corruptions (kernel radius, gain and bias) are streched across the range of severity levels and presented in Table 9.

Table 9: 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

Severity level of -5 for contrast represents an highly contrastive image, while 5 is a very low contrast image. For brightness corruption levels of -5 and 5 correspond to images with very low and high brightness respectively. And defocus blur corruption has only 5 levels of severity from original image (level 0) to the image with a stronger blur (level 5).

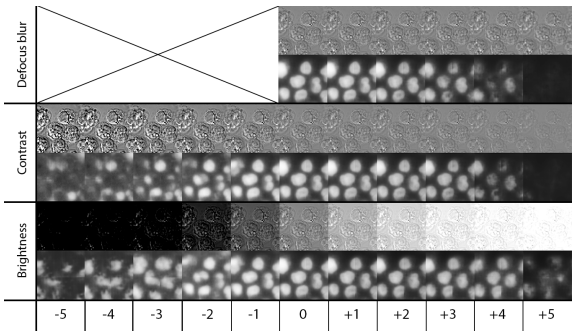


Figure 46: Influence of artificial corruptions on the predictions

One can observe input image change for each of the corruptions along with the change of prediction of nuclei model in Figure 46. One can clearly notice that model's predictions are quite stable towards different brightness levels and contrast. However, the predictions on the crops are very sensible towards defocus blur corruption: DIC images with defocus blur levels $1-4$ are almost indistinctable from the original image, yet the model predictions degrade quite soon. This can be explained by fact that training dataset contains quite diverse data in terms of contrast and brightness levels and, as the result, the model is more stable towards these changes. Using defocus blur as an augmentation will help to solve this problem. This will be described in more detail in Section 4.1.6.

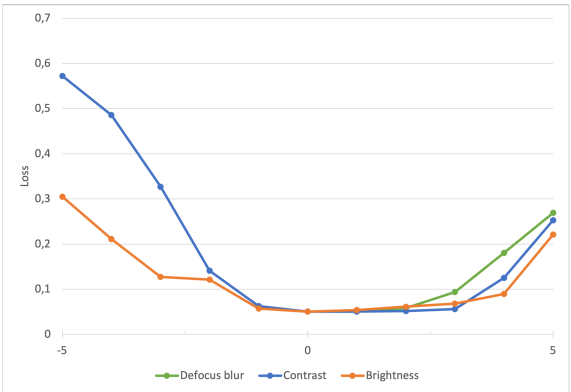


Figure 47: Change of PCC loss for artificial corruptions

Additionally, a change in PCC loss is presented in Figure 47. This is a plot of PCC loss for different artificial corruptions. Loss increases for stronger severity levels. We can see that in positive direction with defocus blur corruption the model degerates more quickly and that contrast corruptions change predictions more severely in the negative one.

4.1.5 Real corruptions

4.1.5.1 Real-world examples of corruptions

A more interesting set of experiments was conducted on the image data that exhibits real corruptions that can happen in the lab due to the wrong settings of the microscopy image acquition approach. The following data was provided:

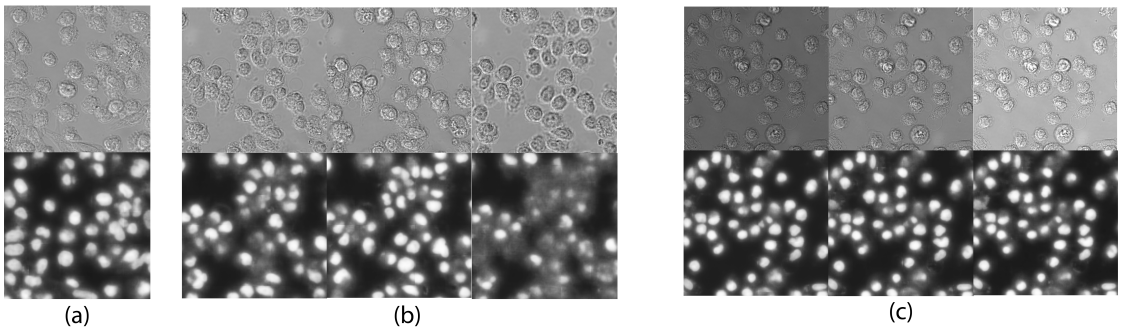


Figure 48: Real corrupted data from the lab

- Figure 48 (a) presents an example of the fixation of the cells in formalin for 15 minutes instead of the usual 10. These cells are a bit "lumpier" and stronger clumped together. This is a good exmaple of the image that cannot be simulated artificially with image processing.
- Figure 48 (b) presents the same image with different microscope focus adjusments. From left to right: microscope stage lowered by 5 microns bringing the subject out of focus, normal height, higher microscope stage by 5 microns bringing the subject out of focus again but in the other direction.
- Figure 48 (c) presents the same image with different exposure times. From left to right: 20ms — underexposure, 30ms — normal setting, 40ms — overexposure.

The model tested here is the model trained on nucleus with simple augmentations (scale, rotation, flips). It seems that time of formalin fixing does not really matter for nucleus outline, however amount of details inside the nuclei seems to be slightly higher than in usual fixation procedure. Focus of the microscope is clearly very important for good predictions. Altough the negative direction of the focus loss seems to have better predictions than the positive one, many of the nuclei are simply gone in the fluorescence image in both cases. Over and underexposure do not influence the predictions that drastically. Visually different exposure seems to change image brightness and the model is stable towards these changes.Nucleus outline stays mostly the same here as well, but the shining around it is dependent on the exposure times.

4.1.6 Improving predictions with additional corruption augmentations

From observations of models stability towards different image brightness which are present in the datasets a new hypothesis was drawn. Introducing corruptions that we test on into the training should improve the predictions on corrupted data. Unfortunately it is not possible to use real lab corruptions here as the data was provided only for testing on these difficult cases and was not stained. Without staining one cannot give a quiantitative measure of the quality of the predictions. However, artificial corruptions can be applied here easily. Random changes in contrast, brightness and defocus blur os severity levels -4 and 4 were added to training augmentations. After the improved model was trained the predictions on the corrupted dataset became much better indeed (see Figure 49).

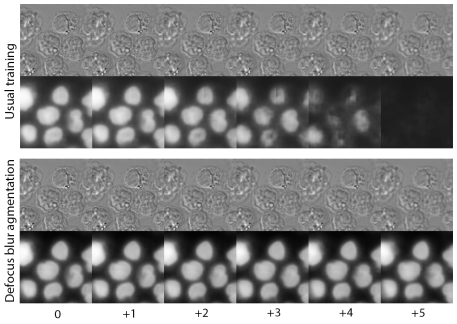


Figure 49: Using corruptions as augmentations improve predictions

Additionally, since these corruptions are artifical and the ground truth data from staining is present, the difference in downstream metrics for models with and without augmentations has been measured (see Table TODO). Calculation of downstream metrics remained the same, the only change happened in the input data (artifical corruptions were applied).

TO BE ADDED.

4.1.7 Influence of corruptions on metrics for downstream tasks

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

4.2 UNET embeddings study

Similarly to studying autoencoder embeddings that represent a high-dimentional input in lower dimentional space, one can study UNet embeddings. However, it is important to keep in mind that the dimensionality of embeddings in UNet case is not lower than dimensionality of the input and is even often higher (see the UNet architecture in Figure 6). The goal of a UNet in contrast to an autoencoder is not to compress the input, but to extract useful features that are helpful for high-resolution segmentation. UNet embeddings do not contain rich image semantics in them as embeddings of an autoencoder do. UNet compresses the spatial dimention of the input, but at the sme time it gradually increases the number of filters that capture of information need for segmentation. As it has been proven in section 3.2.2.2 having more filters only helps to get better predictions, therefore there is no need for a UNet to have low-dimentional embeddings. Nevertheless, it is still interesting to see if the embeddings do contain any information about the input that one could use. There were two hypothesis put in question: the first one is whether embeddings of a trained UNet form any kind of clusters based on cells phenotype. And a second one is whether embeddings of corrupted images can be clustered together further away from not corrupted ones. If the latter hypothesis would hold, one could alarm the end-user about the outliers in the dataset based on their distance from both of the clusters.

4.2.1 Application of various dimentionality reduction methods

It is important that any image is fed into the network crop by crop, meaning that for each crop there is separate embedding. In this section image embeddings are not combined in any way together and were analysed separately. However, it is strongly recommended for a further research to evaluate all the experiments by finding projections in a lower-dimensional space not only of separate crops, but of the image as the whole. For example, simply by averaging embeddings for all crops from the same image. In all experiments here a model trained on nuclei dataset was used.

UNet embedding is of size $16 \times 16 \times 256$ and can be flattened into a 655536-dimensional vector. In order to comprehend the embeddings for us as humans, any dimensionality reduction algorithm has to be applied. One option would be to compress a vector to 2D or 3D-dimentional representation, which is easily comprehendable by humans.

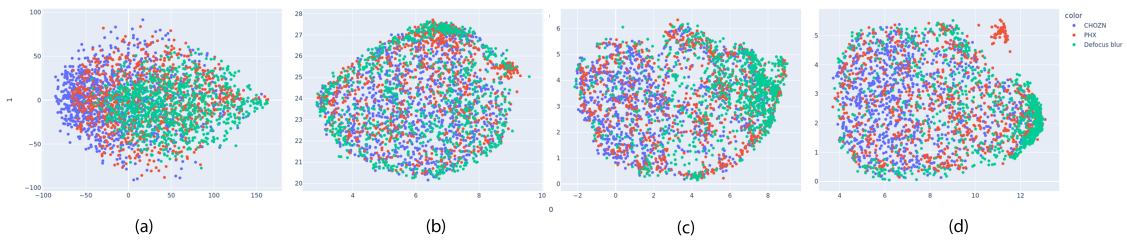


Figure 50: (a) PCA, (b) UMAP, (c) combination of PCA and UMAP with 10 and (d) 50 components

In this case a two-dimensional representation was chosen. With the help of PCA, UMAP and their combination embeddings were projected into a 2D space. In Figure 50 one can see the clouds of dots, where each dot represents a projected UNet embedding of a crop. Both research questions are addressed here: clustering based on phenotypes (CHOZN or PHX) and clustering based on input corruptions (defocus blur of severity level 4). It is very important here that corrupted data was not used in training of any of dimensionality reduction methods. The goal was to use only the data available in trianing dataset, find the transformation of high-dimentional data into a lower-dimentional space and apply it to new samples. That is also why methods like t-sne (t-sne) cannot be used here, because the transformation t-sne learns cannot be applied to new samples.

From Figure 50 it is clear that there is no clustering based on the phenotype is happening. However, green corrupted dots seem to clump more in groups, occupying either one side (a, d) or area around the center (b). It is also clear that the combination of UMAP with previously applied PCA works better with the increasing amount of components in PCA: dots in (d) seem to form a better cluster than dots in (c). Yet it is still not intuitive from this Figure how many not corrupted dots are hidden behind the cluster of the green ones — meaning whether non corrupted crops cluster intersects severely with a corrupted one. In order to visualize that better one can use a kernel density estimate (KDE) plot presented in Figure 51. Additionally, it is clear that vanilla UMAP is not the best approach for the extreme number of dimentions as one has in this case.

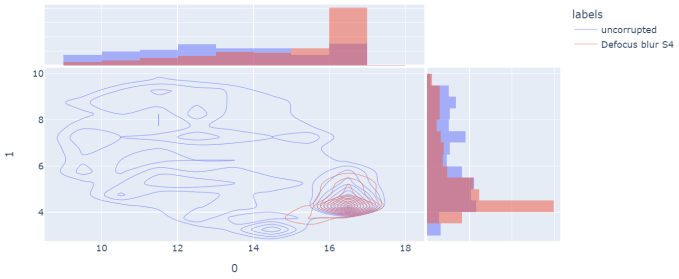


Figure 51: KDE plot of UMAP applied after PCA with 200 components

a cluster of corrupted images is clearly present here, however it also intersects with many non corrupted crops. The quantitive evaluation of how this cluster if separable from the rest of the points is provided in section 4.2.3.1. Altough one can already state that there is a clear opportunity to differentiate between corrupted and not corrupted images, due to clusters being not well separable the accuracy cannot be high. For further research we suggest to additionally check whether clusters may form into a high-dimensional space before projecting them into a 2D space.

4.2.2 Autoencoder embeddings as an alternative

Since UNet embeddings seem to not exhibit any exceptional results in termns of clustering, it was decided to train a vanilla autoencoder directly on image crops. Since autoencoder's embeddings contain dense semantic information of the input they might provide more

insights for clustering hypotheses mentioned before. Figure 52 presents the architecture of two convolutional autoencoders used for these experiments. One compresses 256×256 input crops into embeddings vector of size 3528 and another one compresses them into a vector of a smaller size 200. Both autoencoders were trained using MSE loss. The results of their convergence are presented in Figure 52 on the right.

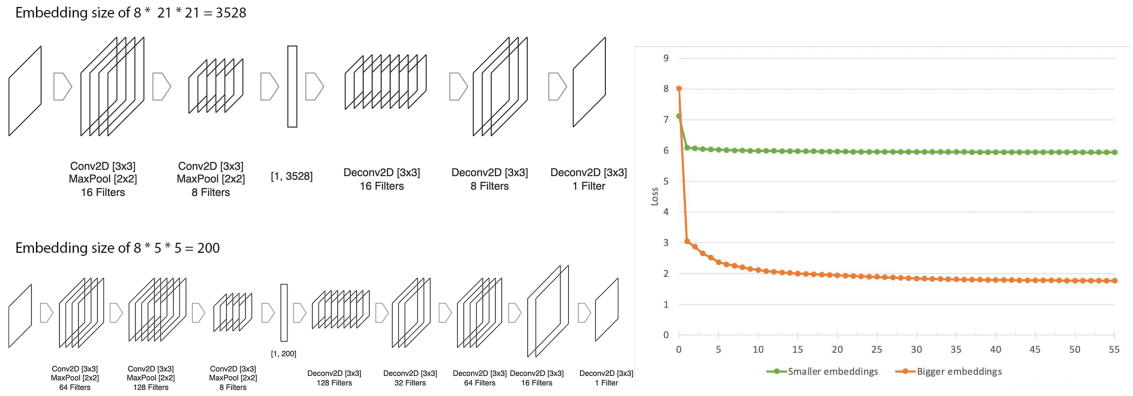


Figure 52: Architectures of two autoencoders and their training convergence

An autoencoder with embeddings of bigger size was able to achieve a lower loss as well as samples reconstructed from it were of a better quality (see Figure 53). Clearly samples reconstruction will not have a high resolution as there are no skip-connections in this architecture. However, this is also not needed, the main goal here would be to find out whether autoencoder embeddings provide any insights on the data.

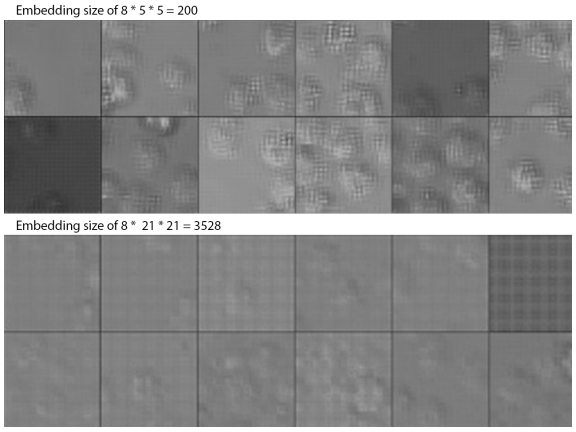


Figure 53: Samples drawn from trained autoencoders

Since an autoencoder with bigger embeddings size seems to be able to reconstruct crops much better we have proceeded with its achtitecture. Embeddings were projected into a 2-dimensional space using first PCA with 10 components and then applying UMAP on PCA's projections. The results of such projection are presented in Figure 54. Two clearly

defined clusters appear: left plot presents projections from an earlier epoch, the right one — from a later one. Embeddings separate gradually into two clusters throughout the training.

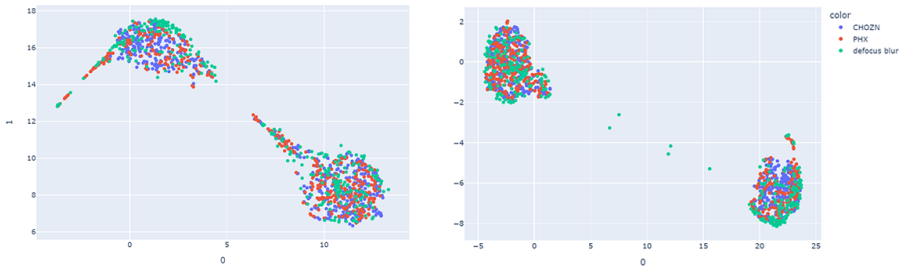


Figure 54: Autoencoder embeddings after applying PCA with 10 components and UMAP afterwards. Earlier epoch VS later epoch.

However, these two clusters are based neither on cell phenotype nor on input corruption. All points of both phenotype as well as corruptions seem to be equally mixed between two custers. By looking at the images correponding to each of the clusters it soon became clear that the main difference between them is their brightness level. To prove this theory distirbutions of average image intensity of images in both clusters are presented in Figure 55. From violin plots it becomes clear that distribution of the crops on the left has a much lower brightness level than distribution of the crops on the right.

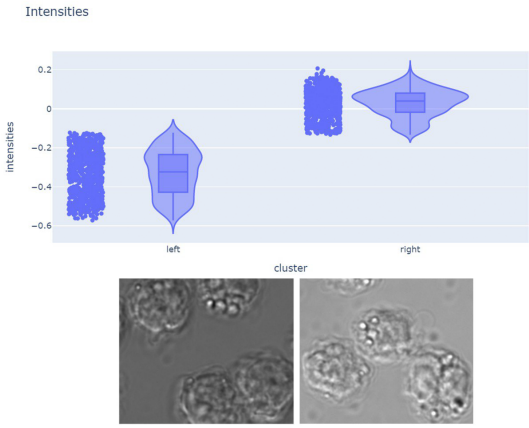


Figure 55: What do two UMAP clusters represent

Since an autoencoder picks up on brightness difference within the crops, it is worth trying to normalize crops brightness across all dataset first. Nevertheless, it is not a trivial task as images have different cell density in them. That is why some images that contain primarily background pixel will always be darker than the ones that contain enough of foreground. We suggest to filter the crops based on amount of cells critea (which can be done using GFP model that can detect cells present in DIC) and normalize them after-

wards. Retraining autoencoder on new training data might provide more insights when difference in brightness will be gone.

It is also clear why autoencoder embeddings do not provide any clustering for corrupted crops. Corruption severities neither really change the image semantics nor they are significantly different visually (see defocus blur in 46). Therefore they do not alter the ability of an autoencoder to restore input correctly. In contrast, UNet's fluorescence predictions do suffer significantly for severy corruption levels, its predictions strongly changes — outline of the organelle becomes more blurry, additional shine appears in fluprescence prediction. These changes happen not only during the decoding part, but they also might bring unsual values in the embedding representation. Therefore when UNet embeddings have more information on the "trustworthiness" of predictions. That is why when defocus corruptions are used as training augmentations, drift detection for the model trained with these corruptions stops alarming about the drift, although it did for the model, which did not have these augmentations present TODO add reference. This happens simply because models predictions degrade and start looking different, which triggers a "drift alarm", with the imroved predictions, drift alarm would not be triggered even when using the same data.

4.2.3 Clustering of PacMAP embeddings

4.2.3.1 Clustering on UNet embeddings

Since the UNet embeddgins are the most promising for clustering based on input corruptions we will proceed with this approach. Apart from dimensionality reduction methods used in section 4.2.1, PaCMAP clustering was tried. You can find its detailed description in section 2.2.2.2.

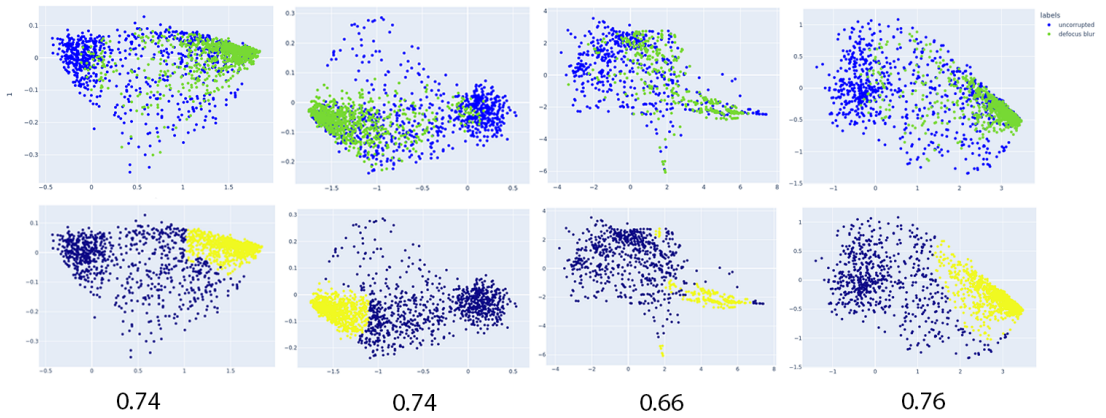


Figure 56: Clustering of UNet embeddings after PacMAP

PaCMAP allows a more flexible hyperparameter tuning in order to preserve local and global relations from high-dimentional data and as result a better clustering can be found. As usual PaCMAP was trained using "good" training data only, meaning no corruptions

were introduced. And only when the transformation from high-dimensional into a lower dimensional space was found, corrupted crops were projected using this transformation. Figure 56 presents results of training PaCMAP with 4 different hyperparametrization settings. There are 3 hyperparameters that were changed here: MN_ratio, FP_ratio and n_neighbors. Detailed explanation of their influence was also described in section 2.2.2.2. From left to right in Figure 56:

- MN_ratio=0.5, FP_ratio=0.1, n_neighbors=10
- MN_ratio=0.1, FP_ratio=0.1, n_neighbors=10
- MN_ratio=0.5, FP_ratio=0.5, n_neighbors=2
- MN_ratio=0.1, FP_ratio=0.5, n_neighbors=10

In this case cluster of green dots represents projected UNet embeddings of images corrupted with defocus blur with severity level 4, which is already a strong corruption and leads to unacceptable predictions of the model, that did not have defocus blur augmentations. However, these point are still strongly mixed with not corrupted ones. In order to check how separable they are unsupervised clustering DBSCAN alogorithm was used. Results of this clustering are shown in Figure 57. Density based clustering approach used here was described in details in section 2.2.3.1.

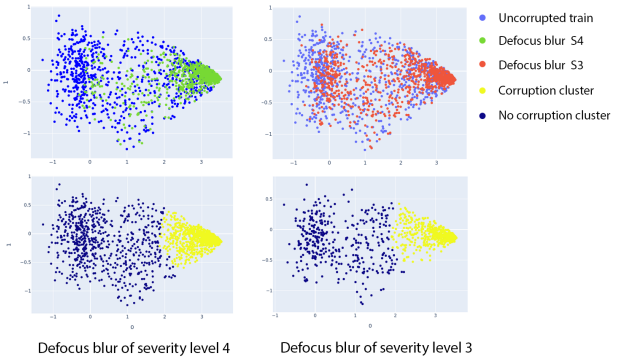


Figure 57: Clustering of UNet embeddings after PacMAP for different severities levels

After training DBSCAN on not corrupted crops embeddings it has recognized a class on the right (yellow dots) as a cluster and the rest of the points (blue ones) as noise, because they have quite low density in comparison to the yellow cluster. This is not a problem if we consider noisy points simply as a separate cluster. For such clustering defocus blur of level 4 splits the crops between two clusters with an F1-score of 0.74. In Figure 57 on the right red points represent projection of image embeddings after corrupting them with a defocus blur of severity level 3. In this case they are mixed with not corrupted projections even stronger. Here prediction of already trained DBSCAN drops to F1-score of 0.64.

Overall UNet embeddings do express clustering of corrupted embeddings to a small extent, however not strongly enough to use it in practice. Autoencoder here would be a

better approach, however brightness normalization has to take place first. It is recommended to proceed the research here in the direction of contrastive learning algorithms, specifically Tack et al., 2020 approach.

4.3 Drift detection

Assume that during training labeled data comes from a distribution p, meaning $\{(x^{(1)}, y^{(1)}), ..., (x^{(n)}, y^{(n)})\} \sim p$ and during deployment unlabeled data comes from a distribution q, meaning $\{x^{(1)\prime}, ..., x^{(1)\prime}\} \sim q$. The goal of the drift detection is to determine if $q(x\prime)$ is the same data distribution as $p(x)$. Or, putting it more formally, determine which hypothesis holds: null-hypothesis H_0 or and alternative hypothesis H_A, where $H_0 : p(x) = q(x)$ and $H_A : p(x) \neq q(x)$.

Having samples from both distributions or representation of these samples in lower dimension, one can then choose a statistical hypothesis test to compare these distributions (Muandet et al., 2017).

4.3.1 Drift detection vs. outliers detection

After developement phase when model training is finished, the model will be moved into a deployment or a production, where it is supposed to maintain an expected quality of predictions. However input data is not always a stable source of input. One should constantly maintain quality of predictions and do a regular check-ups for outliers as well as to alert an end-user about a drift in input data. Drift detection happens on raw data in absence of the ground truth labels and serves as a signal that the input data differs a lot from the data used for training, meaning that predictions became not reliable.

There is a significant difference between distinguishing drift of the whole source of data in comparison to detecting single outliers. In drift detection, one looks at the whole new input data as a distribution and checks if there is a significant shift in comparison to the data used during training.

There are two possible reactions after the drift is detected: alert the user that predictions became unreliable, and therefore she should consider expanding the dataset by adding more labeled data from a new drifted distribution in training or apply some different logic on the model outputs. When an outlier is detected, a model might request human assistance for some particular input, because this input is too unfamiliar to the model and possibly it won't give a good prediction on this one.

The goal of outlier detection is to decide on the single instances whether or not it is different from training data or unusual in some way or another. Outliers might appear both in training and prediction datasets.

Data drift and outlier detection can co-exist. It might be that the input is drifted, but there are no outliers, it might be that there are a lot of outliers, but the data was not drifted. (Samuylova, 2021).

Important observation here is that the drift detector should be robust to outliers. The system should not send an alert as soon as it sees a suspicious sample due to the fact that outlier might be present in the original data distribution as well. But the alert should happen when there are many such samples. To compare original training data distribution and the new one from inputs different statistical tests like Kolmogorov-Smirnov, Chi-squared and etc. can used.

The need of maintaining drift detection or outlier detection depends on the costs on the errors. If the cost of a single error is too high, one should use an outlier detection, but when one needs a test to decide when to label new data - drift detection would be a better approach.

In summary, the drift detection is needed only when the meaningful shifts of the input data distribution from the training distribution need to be detected, whereas the outlier detector aims at finding unusual single instances in inputs. Here this is exactly the case, we train models assuming correct set up of microscopy image acquisition, however changes in exposure, illumination, cell fixation procedure might alternate DIC imaging. In this case user has to be informed about it and choose afterwards whether he wants to add more data to the training set or not to use model's predictions.

4.3.2 Kernel methods and two-sample testing

The test used in this work for determinig wether two distributions are the same or not is one of the multivariate kernel two-sample tests and called Maximum Mean Discrepancy or shotly MMD. The idea behid any two-sample testing is to choose two random samples, where each was taken from one of the two different distributions and afterwards to decide wether the difference in them is statistically significant.

MMD is a kernel-based method that can distinguish between two distributions based on their kernel mean embeddings in a reproducing kernel Hilbert space (RKHS) (Rabanser et al., 2018).

The idea behind a Hilbert space embedding distribution (or a kernel mean embedding) is to map a distribution into a point in a reproducing Hilbert space. After this step, one is allowed to use all powerful kernel methods for probability measures, resulting in methods like kernel two-sample testing. One of the widely known kernel methods if a support vector machines (SVM).

To understand why kernel mean embeddings are so successful one has to first understand what a kernel function is. With the help of kernel functions an inner product of elements $x, y \in \mathcal{X}$ in some high-dimensional feature space can be calculated. If kernel function is positive definite, then there always exists a dot product space $\mathscr{H}$ along with a function that maps a space $\mathcal{X}$ into space $\mathscr{H}$: $\phi : \mathcal{X} \to \mathscr{H}$ such that $k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathscr{H}}$ and most importantly there is no need for explicit computation of ϕ (Schölkopf et al., 2002). Therefore if there exists an algorithm that can be expressed through dot product of $\langle x, y \rangle$ then kernel function can be applied to this dot product and this is called a *kernel trick* (Schölkopf et al., 2002).

Now, kernel mean embedding actually extends the above mentioned feature map ϕ to the space of probability distributions. In this space each probability distribution will be mapped to a mean function defined as follows:

$$\phi(\mathbb{P}) = \mu_{\mathbb{P}} := \int_{\mathcal{X}} k(x, \cdot) d\mathbb{P}(x) \tag{46}$$

Here $k(x, \cdot)$ is a positive definite symmetric kernel function. Main goal here is to map

a distribution $\mathbb{P}$ to an point in the feature space $\mathscr{H}$ and this feature space is exactly an RKHS that corresponds to a kernel k. Such a mapping might be useful because it captures all information about the initial distribution $\mathbb{P}$. This mapping $\mathbb{P} \to \mu_{\mathbb{P}}$ is injective. This means that $||\mu_{\mathbb{P}} - \mu_{\mathbb{Q}}||_{\mathscr{H}} = 0$ if and only if $\mathbb{P} = \mathbb{Q}$. Here it means that $\mathbb{P}$ and $\mathbb{Q}$ is the same distribution. Additionally since the mapping is injective, it is possible to use such characterization of a distribution to be used in two-sample homogenneity tests, which is exactly what is needed here.

To estimate a kernel mean embedding is much easier than to estimate a distribution itself. This approach is successfully used in data-generating processes, it also improves some statistical inference methods like two-sample testing. Such approach is also useful when instead of data points in testing and training datasets there are probability distributions.

Inner product $\langle x, y\rangle$ can be viewed as a similarity measure between x and y. This inner product includes a class of linear functions and this class is too restrictive for many applications, however there is a simple possible extension to add non-linearities to it with the mapping:

$$\phi : \mathcal{X} \to \mathcal{F} \tag{47}$$

where

$$\phi : x \to \phi(x) \tag{48}$$

Here $\mathcal{F}$ is high-dimensional feature space and it is possible to evaluate then:

$$k(x, y) := \langle \phi(x), \phi(y)\rangle_{\mathcal{F}} \tag{49}$$

with $\langle \cdot, \cdot \rangle_{\mathcal{F}}$ denoting an inner product in of $\mathcal{F}$.

Now $k(x, y)$ is already a non-linear similarity measure between x and y. In order to get a non-linear version of the algorithms that use dot product simply sustitute $\langle x, y\rangle$ with $\langle \phi(x), \phi(y)\rangle_{\mathcal{F}}$.

Let's define the following mapping that represents in $\mathcal{X}$ any probability measure $\mathbb{P}$ and denote it as $\mu_{\mathbb{P}}$. This mapping is called a kernel mean embedding.

Definition 4.1 (Kernel mean embedding)**.** cite Berlinet and Thomas Agnan 2004 The kernel mean embedding of probability measure in $M_+^1(\mathcal{X})$ into RKHS $\mathscr{H}$ endowed with a reproducing kernel $k : \mathscr{H} \times \mathscr{H} \to \mathbb{R}$ is defined by a mapping

$$\mu : M_+^1(\mathcal{X}) \to \mathscr{H}, \mathbb{P} \to \int k(x, \cdot) d\mathbb{P}(x) \tag{50}$$

However, usually there is no access to the distribution $\mathbb{P}$ and that is why one cannot directly compute $\mu_{\mathbb{P}}$. Fortuntel, there are samples that can be drawn from this distribution and with their use one can make a good approximation of $\hat{\mu}_{\mathbb{P}}$ of a true kernel mean embedding $\mu_{\mathbb{P}}$. One of such approximations could be the following unbiased estimate:

$$\hat{\mu}_{\mathbb{P}} := \frac{1}{n} \sum_{i=1}^{n} k(x_i, \cdot) \tag{51}$$

Moreover, this estimator $\hat{\mu}_{\mathbb{P}}$ will converge to $\mu_{\mathbb{P}}$ by the law of large numbers as $n \to \infty$.

Definition 4.2 (Characteristic kernel)**.** A kernel k is a characteristic kernel if the map $\mu : \mathbb{P} \to \mu_{\mathbb{P}}$ is injective. If the reproducing kernel of the RKHS $\mathscr{H}$ is characteristic, then RKHS is called characteristic as well.

In machine learning applications and statistics kernel mean embedding is as a metric for the probability distributions. And mean embeddings metric is actually just a specific case of a more general so-called integral probability metric (IPM) (Müller, 1997).

Definition 4.3 (IPM)**.** Let $\mathbb{P}$ and $\mathbb{Q}$ be two probability measures on some measurable space $\mathcal{X}$. Then IPM is defined as follows:

$$\gamma[\mathcal{F}, \mathbb{P}, \mathbb{Q}] = \sup_{f \in \mathcal{F}} \left\{ \int f(x) d\mathbb{P}(x) - \int f(y) d\mathbb{Q}(y) \right\} \tag{52}$$

with $\mathcal{F}$ being a space of real-value bounded functions.

4.3.3 Maximum mean discrepancy for drift detection

Let's assume that $\mathcal{F} := \{f \mid \|f\|_{\mathscr{H}} \leq 1\}$, it means that if supremum in Definition 52 is taken over functions in the unit ball in RKHS, then mean embedding metric is called *maximum mean discrepancy* (MMD).

Definition 4.4 (IPM)**.** Maximum mean discrepancy is defined as a distance between two mean embeddings of distributions:

$$\begin{aligned} \text{MMD}[\mathscr{H}, \mathbb{P}, \mathbb{Q}] &= \sup_{\|f\| \leq 1} \left\{ \int f(x) d\mathbb{P}(x) - \int f(y) d\mathbb{Q}(y) \right\} \\ &= \sup_{\|f\| \leq 1} \{ \langle f, \mu_{\mathbb{P}} - \mu_{\mathbb{Q}} \rangle_{\mathscr{H}} \} \\ &= \|\mu_{\mathbb{P}} - \mu_{\mathbb{Q}}\|_{\mathscr{H}} \end{aligned} \tag{53}$$

From 53: if $\mathscr{H}$ is characteristic, then $\text{MMD}[\mathscr{H}, \mathbb{P}, \mathbb{Q}] = 0$ if and only if $\mathbb{P} = \mathbb{Q}$.

Similarily to kernel mean embedding estimation, through samples drawn from a distribution a biased empricial estimator of MMD can be obtained. Assume that there are two sets of samples $X = \{x^{(1)}, ..., x^{(n)}\}$ and $Y = \{y^{(1)}, ..., y^{(m)}\}$ drawn from two distributions $\mathbb{P}$ and $\mathbb{Q}$ corrspondingly. Then the biased estimate of MMD can be calculated as follows:

$$\widehat{\text{MMD}}_b^2[\mathscr{H}, X, Y] := \sup_{\|f\|_{\mathscr{H}} \leq 1} \left\{ \frac{1}{n} \sum_{i=1}^{n} f(x^{(i)}) - \frac{1}{m} \sum_{j=1}^{m} f(y^{(j)}) \right\} \tag{54}$$

By replacing $\frac{1}{n} \sum_{i=1}^{n} f(x^{(i)})$ with the empirical estimators of mean embeddings one would get:

$$\widehat{\text{MMD}}_b^2[\mathscr{H}, X, Y] = \|\widehat{\mu}_{\mathbb{P}} - \widehat{\mu}_{\mathbb{Q}}\|_{\mathscr{H}}^2 \tag{55}$$

If an unbiased estimator of MMD through the kernel function k is needed then one could use the following estimator from Corollary 2.3 in **Borgwardt**.

The most common application of MMD in statistics is a two-sample testing. Particularily, in testing the null hypothesis $H_0 : \|\widehat{\mu}_{\mathbb{P}} - \widehat{\mu}_{\mathbb{Q}}\|_{\mathscr{H}} = 0$ that two samples come from the same distribution against an alternative hypothesis $H_1 : \|\widehat{\mu}_{\mathbb{P}} - \widehat{\mu}_{\mathbb{Q}}\|_{\mathscr{H}} \neq 0$. But one has to be cautios here, even when both samples came from the same distribution it still might be that the MMD will be not zero due to the fact that this is an etimate and not a precise value and we have a finite number of smaples for this estimate.

$$\begin{aligned}\widehat{\mathrm{MMD}}_b^2[\mathscr{H}, X, Y] =& \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k(x_i, x_j) \\ &+ \frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{j \neq i}^{n} k(y_i, y_j) \\ &- \frac{2}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} k(x_i, y_j)\end{aligned} \tag{56}$$

4.3.4 Drift detection experiments

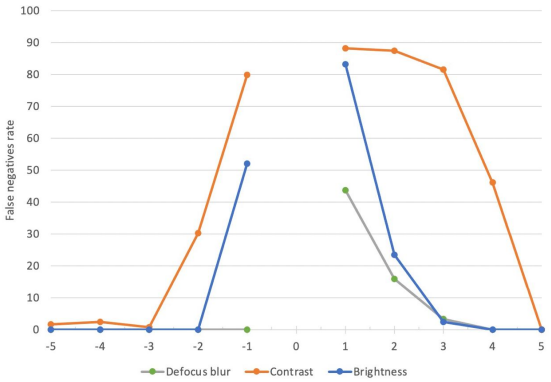


Figure 58: False negatives rate for drift detection on artificial corruptions

4.3.5 Online drift detection experiments

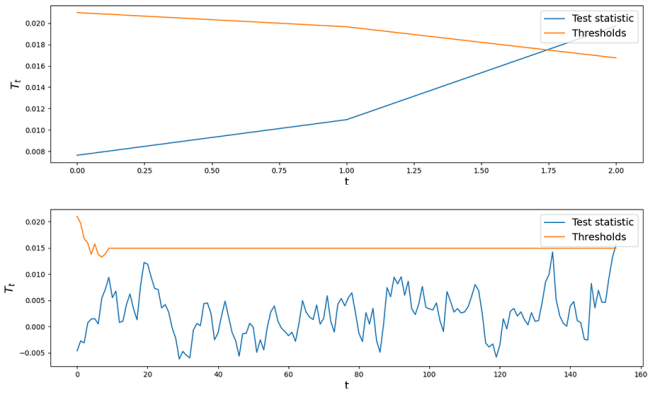


Figure 59: Expected runtime (ERT) for corrupted and in-distribution data

Table 10: 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 11: ERT influence on separability

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

Table 12: Severity of corruptions on separability

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

Figure 60: AUC ROC scores for various defocus corruptions severities

4.3.5.1 Not fixed cells imaging as corrupted input

In section 3.5 the difference between fixed and not fixed cells was mentioned. Visual analysis of model's predictions for not fixed cells after training it on fixed ones has shown that the model was not able to generalize well on them. That is why it would be important to alarm the end-user to not rely on predictions when such situation occurs. In this case an online drift detector trained using not corrupted data used for ER training first and tested on not fixed ER cells. The results of this test are shown in Figure 61.

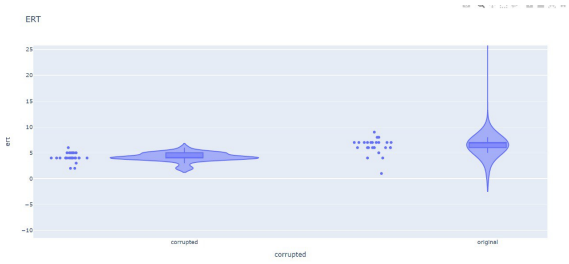


Figure 61: Online drift detection of not fixated cells

The ERTs for corrupted data (left) are lower from ERT for true input. ROC-AUC score for the separability is 0.91 and the best threshold is 6. However not corrupted data (fixed cells) mostly have ert of 7, whereas corrupted data (not fixed cells) have an ert of 4. Both classes have ERTs that are very close to the threshold.

5 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

6 Summary

References

Adaptive thresholding¶ (n.d.).

F Borek (Oct. 1984). "Immunofluorescence in medical science". en. In: *J. Immunol. Methods* 73.1, p. 232.

Shiyi Cheng, Sipei Fu, Yumi Mun Kim, Weiye Song, Yunzhe Li, Yujia Xue, Ji Yi, and Lei Tian (Jan. 2021). "Single-cell cytometry via multiplexed fluorescence prediction by label-free reflectance microscopy". In: *Science Advances* 7.3.

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.

Endoplasmic Reticulum (smooth) (n.d.).

Golgi body (n.d.).

Rafael C. Gonzalez and Richard E. Woods (2006). *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc.

Rogers Kara (n.d.). *Endoplasmic Reticulum*.

Alexander Krull, Tim-Oliver Buchholz, and Florian Jug (June 2019). "Noise2Void - learning denoising from single noisy images". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

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.

Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, and Bernhard Schölkopf (2017). "Kernel Mean Embedding of Distributions: A Review and Beyond". In: *Foundations and Trends® in Machine Learning* 10.1-2, pp. 1–141.

Alfred Müller (1997). "Integral Probability Metrics and Their Generating Classes of Functions". In: *Advances in Applied Probability* 29.2, pp. 429–443. (Visited on 07/30/2022).

Nucleus (n.d.).

Judith M. S. Prewitt and Mortimer L. Mendelsohn (Jan. 1965). "The Analysis of Cell Images". In: *Annals of the New York Academy of Sciences* 128.3, pp. 1035–1053.

Stephan Rabanser, Stephan Günnemann, and Zachary C. Lipton (2018). *Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift*.

Elena Samuylova (Nov. 2021). *What is the difference between outlier detection and data drift detection?*

Bernhard Schölkopf, Alex Smola, Alexander Smola, and A Smola (Apr. 2002). "Support Vector Machines and Kernel Algorithms". In: *Encyclopedia of Biostatistics, 5328-5335 (2005)*.

Sternberg (Jan. 1983). "Biomedical Image Processing". In: *Computer (Long Beach Calif.)* 16.1, pp. 22–34.

Jihoon Tack, Sangwoo Mo, Jongheon Jeong, and Jinwoo Shin (2020). *CSI: Novelty Detection via Contrastive Learning on Distributionally Shifted Instances*.

Thresholding¶ (n.d.).

Types of morphological operations (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 Cell structure 28
11 Samples to be filtered out 28
12 Having more data makes training more stable 29
13 Adding simple augmentations in the dataset 30
14 With regularization and augmentations 30
15 Difefrent models predictions and scores comparison 31
16 Problems in predictions 32
17 Predictions improvement: (a) ground truth fluorescence; (b) bigger model with four times more filters in convolutional layers; (c) standard architecture 32
18 Different lightning conditions 33
19 Closely located and dividing cells 33
20 Fluorescence segmentation 34
21 Local vs. global thresholding 35
22 Histogram as a probability density function (Gonzalez et al., 2006) 37
23 Metrics for downstream tasks on nuclei 38
24 Default model (left), slightly regularized model(middle), strongly regularized model (right) 41
25 Combination pof ER with nuclei 42
26 Artifacts from local thresholding algorithm 43
27 Segmentation steps in ER postprocessing 44
28 Metrics for downstream tasks on nuclei 44
29 Golgi enhancement: left — original signal, right — enhanced image 46
30 Background removal 48

31 (a) Vanilla pre-processing with automatic background removal algorithm only; (b) masked or subfigure (a); (c) Additional clipping of lower intensities after vanilla pre-processing; (d) mask of subfigure (c) 48
32 Straightforward training doesn't work 49
33 Training on original data 49
34 Full size predictions 50
35 Training on the enhanced data 50
36 Small subset of the best staining 50
37 Punishing over and under predictions with asymmetrical MSE loss 51
38 Results of advanced versions of MSE training 53
39 Examples of fixed and not fixed cells DIC imaging 55
40 Converting GFP into a binary mask 56
41 Training with Pearson correlation loss 56
42 Training with BCE loss 57
43 Downstream metrics 58
44 GFP, Nuclei and ER combined 58
45 Defocus blur kernel 61
46 Influence of artificial corruptions on the predictions 62
47 Change of PCC loss for artificial corruptions 62
48 Real corrupted data from the lab 63
49 Using corruptions as augmentations improve predictions 64
50 (a) PCA, (b) UMAP, (c) combination of PCA and UMAP with 10 and (d) 50 components 65
51 KDE plot of UMAP applied after PCA with 200 components 66
52 Architectures of two autoencoders and their training convergence 67
53 Samples drawn from trained autoencoders 67
54 Autoencoder embeddings after applying PCA with 10 components and UMAP afterwards. Earlier epoch VS later epoch. 68
55 What do two UMAP clusters represent 68
56 Clustering of UNet embeddings after PacMAP 69
57 Clustering of UNet embeddings after PacMAP for different severities levels 70
58 False negatives rate for drift detection on artificial corruptions 76
59 Expected runtime (ERT) for corrupted and in-distribution data 77
60 AUC ROC scores for various defocus corruptions severities 78

61 Online drift detection of not fixated cells 78

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 Threshold timing 38
5 Correlation coefficients for downstream tasks on nuclei 39
6 Pearson correlation coefficients for downstream tasks for different scaling factors 39
7 Correlation coefficients for downstream tasks on ER 44
8 Correlation coefficients for downstream tasks 57
9 Hyperparameterization for different artificial corruption severities 61
10 Test window size influence on separability 77
11 ERT influence on separability 77
12 Severity of corruptions on separability 77