

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

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>  | <b>1</b>  |
| 1.1      | Motivation . . . . .   | 1         |
| 1.2      | Notation . . . . .   | 2         |
| <b>2</b> | <b>Domain knowledge</b>  | <b>3</b>  |
| 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 . . . . .  | 14        |
| 2.3.1    | Digital imaging . . . . .  | 14        |
| 2.3.2    | Microscopy imaging . . . . .   | 15        |
| 2.3.2.1  | Image acquisition peculiarities . . . . .                            | 15        |
| 2.3.2.2  | Crops combination technique . . . . .                                | 15        |
| <b>3</b> | <b>Implementation and experiments</b>                                | <b>18</b> |
| 3.1      | Model training . . . . .   | 18        |
| 3.1.1    | Neural network architecture . . . . .                                | 18        |
| 3.1.2    | Loss functions . . . . .   | 18        |
| 3.1.3    | Available data . . . . .   | 18        |
| 3.1.4    | Training costs estimation . . . . .                                  | 19        |
| 3.1.5    | Augmentations . . . . .  | 19        |
| 3.1.5.1  | Special augmentations for rotation and scaling . . . . .             | 19        |

|         |   |    |
|---------|---|----|
| 3.1.6   | Model setup . . . . .   | 19 |
| 3.1.6.1 | Weight Initialization . . . . .                                       | 19 |
| 3.1.6.2 | Regularization . . . . .  | 19 |
| 3.1.6.3 | Optimizers . . . . .  | 20 |
| 3.2     | Nuclei . . . . .  | 21 |
| 3.2.1   | Preprocessing . . . . .   | 21 |
| 3.2.1.1 | Thresholding algorithms . . . . .                                     | 21 |
| 3.2.2   | Training and predictions . . . . .                                    | 21 |
| 3.2.2.1 | Convergence . . . . .   | 21 |
| 3.2.2.2 | Predictions quality . . . . .   | 24 |
| 3.2.3   | Postprocessing for nuclei segmentation . . . . .                      | 24 |
| 3.2.4   | Influence of scaling on predictions quality . . . . .                 | 25 |
| 3.3     | Endoplasmic Reticulum . . . . .                                       | 26 |
| 3.3.1   | Preprocessing . . . . .   | 26 |
| 3.3.2   | Training and predictions . . . . .                                    | 26 |
| 3.3.3   | Combination of nuclei and actin predictions . . . . .                 | 27 |
| 3.3.4   | Generalizability across phenotypes . . . . .                          | 27 |
| 3.4     | Golgi . . . . .   | 28 |
| 3.4.1   | Preprocessing . . . . .   | 28 |
| 3.4.1.1 | Background removal algorithms . . . . .                               | 28 |
| 3.4.2   | Training and predictions . . . . .                                    | 29 |
| 3.4.3   | Alternative ways to improve predictions . . . . .                     | 31 |
| 3.4.3.1 | Asymmetrical losses . . . . .   | 31 |
| 3.4.3.2 | Use of gradient in loss . . . . .                                     | 31 |
| 3.4.3.3 | Noise reduction methods . . . . .                                     | 31 |
| 3.5     | GFP . . . . .   | 32 |
| 3.5.1   | Preprocessing . . . . .   | 32 |
| 3.5.2   | Predictions . . . . .   | 32 |
| 3.5.3   | Downstream metrics . . . . .  | 33 |
| 3.5.4   | Combination of GFP, nuclei and ER . . . . .                           | 33 |
| 3.6     | Model evaluation . . . . .  | 34 |
| 3.6.1   | Metrics for downstream tasks . . . . .                                | 34 |
| 3.6.2   | Influence of different loss functions on metrics for downstream tasks | 34 |

|  |           |
|--|-----------|
| <b>4 Stability study</b>   | <b>35</b> |
| 4.1 Stability study . . . . .  | 35        |
| 4.1.1 Artificial corruptions . . . . .                                   | 35        |
| 4.1.2 Real corruptions . . . . .   | 36        |
| 4.1.2.1 Not fixed cells imaging as corrupted input . . . . .             | 36        |
| 4.1.2.2 Real-world examples of corruptions . . . . .                     | 36        |
| 4.1.3 Influence of corruptions on metrics for downstream tasks . . . . . | 36        |
| 4.1.4 Improving predictions with additional corruption augmentations .   | 36        |
| 4.2 UNET embeddings study . . . . .                                      | 37        |
| 4.2.1 Application of various dimentionality reduction methods . . . . .  | 37        |
| 4.2.2 Autoencoder embeddings as an alternative . . . . .                 | 37        |
| 4.2.3 Clustering of PacMAP embeddings . . . . .                          | 40        |
| 4.2.3.1 Clustering on UNet embeddings . . . . .                          | 40        |
| 4.3 Drift detection . . . . .  | 41        |
| 4.3.1 A need to detect drift . . . . .                                   | 41        |
| 4.3.2 Maximum mean discrepancy for drift detection . . . . .             | 41        |
| 4.3.3 Online version of MMD algorithm . . . . .                          | 41        |
| <b>5 Software Tools</b>  | <b>43</b> |
| 5.1 Foundry. Palantir . . . . .  | 43        |
| 5.2 AWS . . . . .  | 43        |
| 5.3 Streamlit . . . . .  | 43        |
| <b>6 Future research</b>   | <b>44</b> |
| <b>7 Summary</b>   | <b>45</b> |
| <b>List of Figures</b>   | <b>46</b> |
| <b>List of Tables</b>  | <b>47</b> |



# 1 Introduction

## 1.1 Motivation

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

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

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

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

## 1.2 Notation

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

$X_{train}$  A set of training examples

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

$\mathbb{R}$  A set of real numbers

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

$p_{data}$  Data generating distribution

## 2 Domain knowledge

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

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

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

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

## 2.1 Biology

### 2.1.1 Cell line development process

Cell line development (CLD) is a process of generating single cell-derived clones that produce high and consistent levels of target therapeutic protein [TODO cite [pharma.lonza.com/offers/mammalian/cell-line-development](http://pharma.lonza.com/offers/mammalian/cell-line-development)]. Therapeutic proteins in this case are so-called recombinant proteins and they are widely used in the biomedical research, drug production and for various therapeutic needs like, for example, vaccines and monoclonal antibodies (mAbs) IWNLP<sup>iqb12007underwater</sup> [TODO cite Ohtake 2013, Jafferis 2021, Funaro 1996]. A recombinant protein is defined by [cite Barbeau, J] as a modified or manipulated protein encoded by a recombinant DNA. Recombinant DNA, in its 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 will be 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, in both industry and research there is a great need for production of high volumes of good-quality recombinant proteins [TODO cite Tihanyi 2020]. That is why the goal of many research efforts 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 work specifically are chinese hamster ovary (CHO) cells [cite Castan 2018]. Altough different cells can be used as hosts, such as bacterial, plant-based or yeast cells, mammalian cells remain the most popular [cite Beckman]. The reason behind this popularity hides in the fact that they can produce a diverse range of correctly folded proteins and most importantly, they have high protein production rates. 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]. That is why current research has a wide applicability.

However there is a downside to using CHO as host cells - they are infamously unstable. As rapidly growing immortal cells CHO are also genetically unstable and extremely heterogenous which ususally leads to the main issue: production instability. The problem of choosing stable and high-production clones that simutaneously will be able to express protein qualitatively and quantifiably over time is essentially the main goal of current research. The big challenge in manufacturing here is the time and the costs of production. Currently, a lot of research attention is dedicated to the reduction of both, as well as to developing techniques of high-throughput clone screening and characterization [cite Tihanyi 2020]. The latter is of interest for this thesis. With the great amounts of data acquired over time and the development of the computational modelling and statistical analysis it is possible now to do the analysis *in silico*, meaning - computationally without interferring 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. It has two main problems: first is that transfection mostly results in a vector being inserted into a random site within the host cell genome and second, that it generally has low efficiency of integration [cite Tihanyi]. It is important to transfet a GOI into the optimal site of the genome to secure high protein expression over time during protein production, however practically, transfection happens 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 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 percent of these cells actually integrate a vector into the genome and, as mentioned above, only a fraction of those are able to stably express the protein [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. Chosen stable pools of cells are phenotypically and genetically diverse - meaning they have different growth rates, metabolic profile and etc. This is not ideal for industrial production - all the cells used for protein production should be derived from the same clone [cite [25] from Castan].

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

### 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., however this research 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 of 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. Therefore there exists 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 is dedicated to predicting fluorescence signal from the DIC imaging directly without the need of actual cell staining. The measurements needed for selection of minipools can be calculated as usual, but 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 current thesis consists of input DIC images  $X$  and target fluorescence images  $Y$ . Together pairs from each form a dataset:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

With maximum likelihood estimation one has that:

$$\theta_{MLE} = \arg \max_{\theta} \sum_{i=1}^N \log p_{\text{model}}(x^{(i)}, y^{(i)}, \theta) \quad (7)$$

After maximizing the sum and taking a gradient one gets:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{x,y \sim p_{\text{data}}} \nabla_{\theta} \log p_{\text{model}}(x, y, \theta) \quad (8)$$

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

$$g = \nabla_{\theta} J^*(\theta) = \sum_x \sum_y p_{\text{data}}(x, y) \nabla_{\theta} L(y, M(x, \theta)) \quad (9)$$

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

$$\hat{g} = \frac{1}{m} \nabla_{\theta} \sum_i L(y^{(i)}, M(x^{(i)}, \theta)) \quad (10)$$

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

$$\theta_{k+1} = \theta_k - \alpha \frac{1}{m} \nabla_{\theta} \sum_i L(y^{(i)}, M(x^{(i)}, \theta)) \quad (11)$$

where  $\alpha$  is a tuneable parameter called *learning rate*.

**Definition 2.9** (Adadelta optimizer). Adadelta optimizer is a more sophisticated optimization technique, that follows the following algorithm for the parameters update: Stochastic

---

#### Algorithm 1 Adadelta optimization

---

1.  $E[g]^2_0 = 0$  and  $E[\Delta\theta^2]_0 = 0$  In order to update the parameters compute:

2. Compute gradient:  $\hat{g}_t$

3. Accumulate gradient:  $E[g]^2_t = \rho E[g]^2_{t-1} + (1 - \rho) g_t^2$

4. Compute update:  $\Delta\theta_t = \frac{\text{RMS}[\Delta\theta]_{t-1}}{\text{RMS}[g]_t} \hat{g}_t$

5. Accumulate updates:  $E[\Delta\theta^2]_t = \rho E[\Delta\theta^2]_{t-1} + (1 - \rho) \Delta\theta_t^2$

6. Apply update:  $\theta_{t+1} = \theta_t + \Delta\theta_t$

RMS here is the root mean square. [cite Zeiler 2012]

---

gradient descent is an optimization algorithm where the parameters  $\theta$  are iteratively updated every mini-batch of data by the following rule:

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

where  $\alpha$  is a tuneable parameter called *learning rate*.

**Definition 2.10** (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 maps in this case a vector  $x \in \mathbb{R}^N$  to an output  $a \in \mathbb{R}^M$  via the following transformation:

$$a = W^T x + b \quad (13)$$

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 boiled 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" in its name suggests that the convolution operation should be used in at least one of the layers there.

**Definition 2.11** (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 operator  $(\cdot * \cdot)$ . By transforming an input  $x \in \mathbb{R}^{W, H, C}$  it produces an output  $S$

$$S = K * x + b \quad (14)$$

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

$$S_{i,j} = \sum_w \sum_h x_{m,n} K_{i-m, j-n} \quad (15)$$

Convolutional layer like a fully-connected layer can be viewed a linear transformation as well. However there are 3 main advantages that leverage convolutional layers for image processing in comparison to fully-connected layers: sparse interactions, parameter sharing, equivariant representations. Image is a very redundant way of representing the semantic meaning hidden in 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 pixels, however it is possible to detect smaller and very important features like contrast changes, edges, and shapes using a kernel consisting only of a hundred of pixels. By applying kernels (or filters) on the image locally one will infer many of these features across the whole image. Such 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 the parameters and applied across all image locations. Lastly, equivariance here means that convolution operation is equivariant to the shifts in the image.

**Definition 2.12** (Stride). During the computation of convolution, kernel starts sliding at the upper-left corner of the input tensor, all over all locations to the right and down. The step with which the window slides is called *stride*.

**Definition 2.13** (Padding). When convolution is applied several points on the perimeter of the input tensor will be lost. One can fix this by adding few more pixels of the perimeter, to preserve the dimension of the output same as input. The amount of pixels added is called *padding*.

**Definition 2.14** (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 etc. In CNNs they are also often combined with max pooling layers and dropouts to escape overfitting.

**Definition 2.15** (Batch normalization layer). Let's denote  $B = \{x^{(i)}, \dots, x^{(m)}\}$  to be a mini-batch of data. Then Batch Normalizing transform applied to this input data would be:

$$\begin{aligned} a^{(i)} &= \gamma \frac{x^{(i)} - \mu_B}{\sigma_B^2 + \epsilon} + \beta \\ \sigma_B^2 &= \frac{1}{m} \sum_i^m (x^{(i)} - \mu_B)^2 \\ \mu_B &= \frac{1}{m} \sum_i^m x^{(i)} \end{aligned} \tag{16}$$

where  $\gamma$  and  $\beta$  are learnable parameters,  $\mu_B$  and  $\sigma_B^2$  are the mean and standard deviation of the batch. [cite Ioffe and Szegedy, 2015]

**Definition 2.16** (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  $\mu$  specifies which units are included in training, then dropout's objective to be minimized becomes:  $\mathbb{E}_\mu J(\theta, \mu)$ . Visually dropout is presented in the Figure 2.

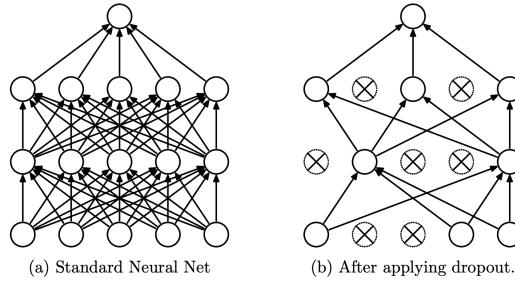


Figure 2: Dropout

**Definition 2.17** (Activation function). An activation function is an element-wise non-linear function  $f(\cdot)$ . Some examples with are:

$$f(x) = \frac{1}{1 + e^{-x}} \quad \text{Sigmoid} \quad (17)$$

$$f(x) = \max(0, x) \quad \text{Rectified linear unit (ReLU)} \quad (18)$$

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha * (e^x - 1), & \text{if } x \leq 0 \end{cases} \quad \text{ELU} \quad (19)$$

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

**Definition 2.18** (UNet). UNet is fully convolutional neural network with U-shaped encoder-decoder network architecture. [cite Ronneberger].

The encoder is a usual CNN, consisting of the repeated block of two  $3 \times 3$  convolutions, followed by an activation function, and a  $2 \times 2$  max-pooling operation with stride 2. At each encoder step the number of feature channels doubles. The decoder is also a usual 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  $3 \times 3$  convolutions, followed by a ReLU. The last decoder layer is a  $1 \times 1$  convolution to map the tensor to the number of output image channels need. 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 2.19** (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 2.20** (Overfitting). "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). They way to avoid overfitting that happened to the models in Sections [TODO cite sections] are discussed in Section [cite regularization section].

### 2.2.2 Dimensionality reduction methods

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

**Definition 2.21** (Embedding). An embedding in this context is an output tensor from the encoder part of the UNet or from an encoder part of an Autoencoder.

The encoder output of the UNet is a tensor of size  $16 \times 16 \times 256$  and after its flattening it turns into a vector of size 65536. The smallest autoencoder embedding was of size 200 [TODO check] which is also high-dimensional. One of the tasks of this research is to determine whether there are any interesting patterns or grouping based of 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 and that is where 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 try to preserve the structure amongst all the data samples; others prefer to save local distances. For example PCA [cite Hotelling] are assigned to the first category, while t-SNE [cite Ulyanov] and Isomap are assigned to a latter one.

Uniform Manifold Approximation and Projection (UMAP) was built in a way to preserve both and it is a competitor of 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 a foundation. As 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 needs first to define an objective function that can preserve 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 results 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 the value around 10 for datasets with 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 allow the user to achieve the desired ration between preserving local and global structure. Such method also works faster than UMAP, which allows to try out more hyperparameter options.

### 2.2.2.3 PCA

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

The steps of PCA algorithm are the following:

- Subtract mean  $\mu_d$ . To center the input data is not a necessary step, but it is recommended to do so to avoid the numerical problems.
- Standardize the data. Calculate the standard deviation  $\sigma_d$  and standardize the data to have unit variance for every dimension.
- Do an eigendecomposition of the data covariance matrix. For that one first must 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. For that first standardize the point  $x^* \in \mathbb{R}$  using  $\mu_d$  and  $\sigma_d$ :

$$x_d^* \leftarrow \frac{x_d^* - \mu_d}{\sigma_d} \quad (20)$$

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

$$\tilde{x}^* = BB^T x^* \quad (21)$$

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 the Section [TODO cite the section] using DBSCAN algorithm. Here will be provided 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 to provide the number of clusters in advance. Although this is a nice quality of this algorithm it is not that important for current research. The goal of Section [cite Section] is to check whether different phenotypes form different clusters or for example whether corrupted images would fall into a separate cluster. In all cases the number of ground truth clusters is known in advance. However the fact that corrupted images form a more dense region in rather than non corrupted ones [cite Section] is 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, that is used to find 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  $C$  is the number of channels of the image. In this work  $C = 1$  and  $W = H$ . A digitally represented image  $A$  then is the matrix:

$$A = \begin{bmatrix} a_{0,0} & \cdots & a_{0,W-1} \\ \vdots & \ddots & \vdots \\ a_{H,0} & \cdots & a_{H-1,W-1} \end{bmatrix} \quad (22)$$

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

$$a_{i,j}^{\text{norm}} = \frac{a_{i,j} - \min(A)}{\max(A) - \min(A)} \quad (23)$$

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

### 2.3.2 Microscopy imaging

#### 2.3.2.1 Image acquisition peculiarities

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 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 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 be first splitted into crops of size  $256 \times 256$  each during the preprocessing. And it might happen that same cells might appear in several crops. That is why after the split of the image data between train, test and validation sets it might also happen that the same set of cells will once land in the train set and another time in the validation set, which will lead to a not completely fair and representative validation metrics during training.

In order to overcome this problem a much more expensive equipment is needed. Since in this case it doesn't bring too huge problems except for the fact that validation metrics might be lower than they should have been, there was no need to purchase a more expensive equipment.

#### 2.3.2.2 Crops combination technique

Due to the restricted amount of memory on GPU deep learning models cannot have a high-resolution image as their input within current 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 of size  $(256, 256)$  or

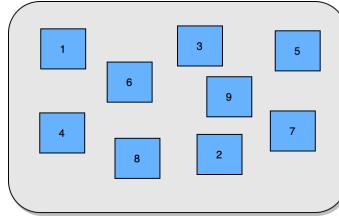


Figure 3: Way in which photos of the well-plate were taken

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 the Figure 4. A small window starts sliding the image from the left upper to the right down 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.

Having a desired border size, in order to accepted areas to be overlaped with each other without blank spaces, one has to adjust the step size to be:

$$s = w - 2 * b \quad (24)$$

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

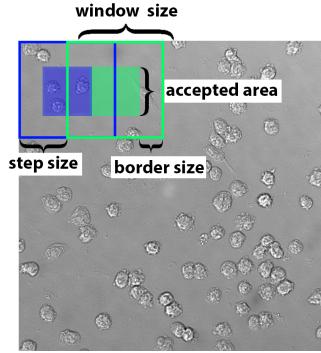


Figure 4: Sliding window approach for fluorescence prediction

The reason why not the full prediction is accepted to form the output hides 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 the Figure 5). This is discussed in more detail in Section [TODO reference the section]. Such approach helps to reduce the effect of the grid visible on the image composed of many small crops, which one can see in the Figure 5 on the left to almost non-visible borders as in the same Figure on right. This would of course take longer time in predictions, however as the speed is less crucial in comparison to the accuracy of the predictions.

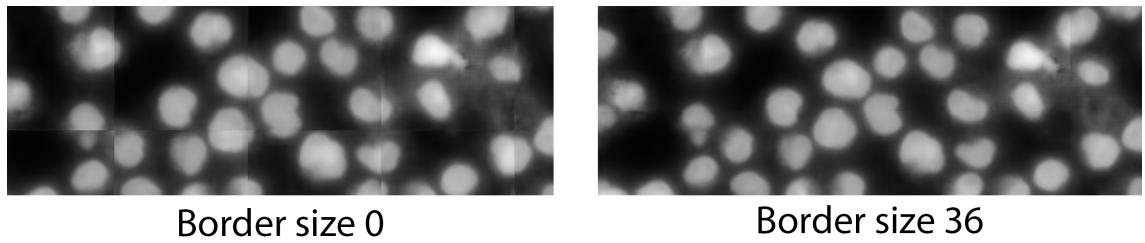


Figure 5: Difference of overlap between predictions on the resulting image

### 3 Implementation and experiments

#### 3.1 Model training

##### 3.1.1 Neural network architecture

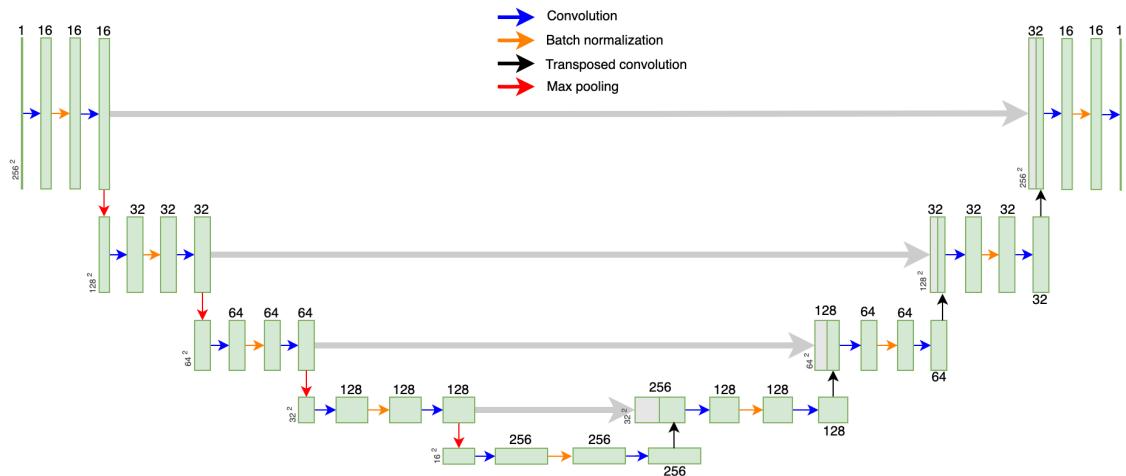


Figure 6: Unet

And information on the embeddings, output sizes, amount of parameters, etc.  
take description from here file:// /Users/anniepank/Downloads/media-1.pdf

##### 3.1.2 Loss functions

Which loss functions were used, Pearson correlation coefficient explained.

##### 3.1.3 Available data

Description of the datasets and the amount of images in each category.

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            | 5120       |
| Golgi     | 761          | 23,036         | 2,336            | 6,347      |
| H19       | 400          | 27,264         | 3,008            | 7,61       |
| Nucleolei | ?            | ?              | ?                | ?          |

### 3.1.4 Training costs estimation

Table with the estimation of costs and times for AWS

### 3.1.5 Augmentations

Description of all augmentations used

#### 3.1.5.1 Special augmentations for rotation and scaling

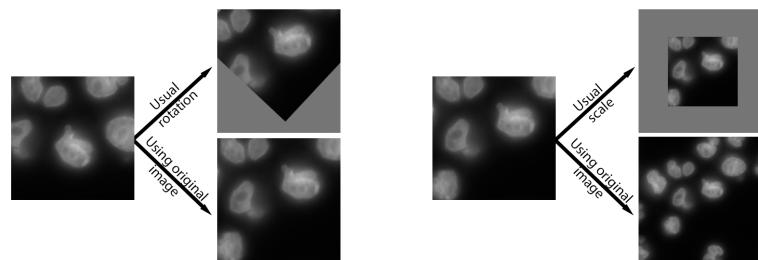


Figure 7: Using original image for rotation and scaling augmentations

### 3.1.6 Model setup

#### 3.1.6.1 Weight Initialization

These plots represent MSE

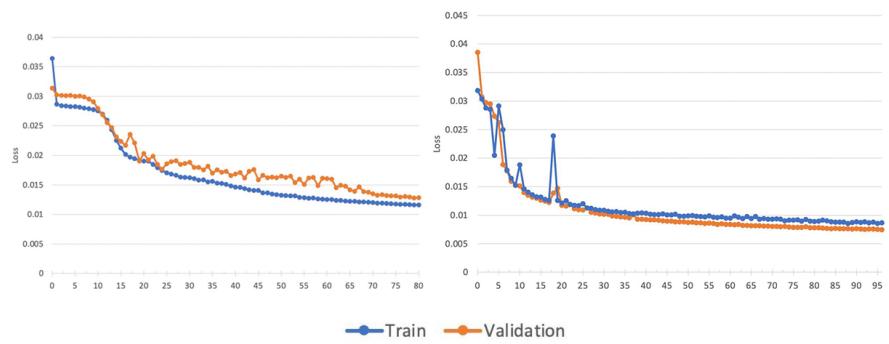


Figure 8: Nuclei training without and with custom weight initialization

#### 3.1.6.2 Regularization

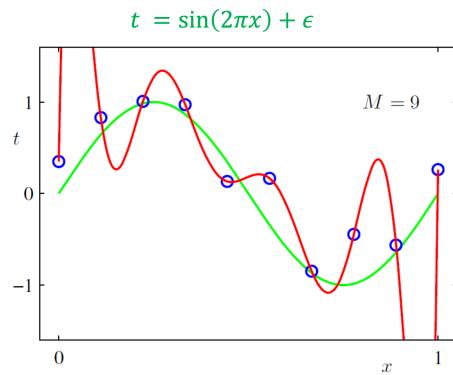


Figure 9: Overfitting

### 3.1.6.3 Optimizers

Comparison of different optimizers

## 3.2 Nuclei

### 3.2.1 Preprocessing

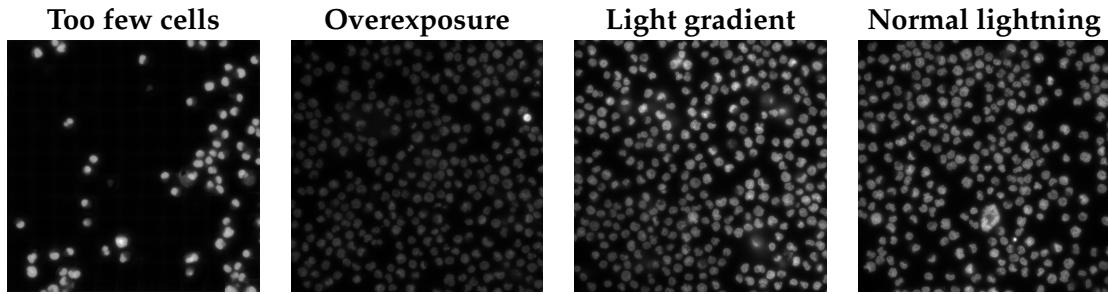


Figure 10: Different lightning conditions

#### 3.2.1.1 Thresholding algorithms

Global and local thresholding

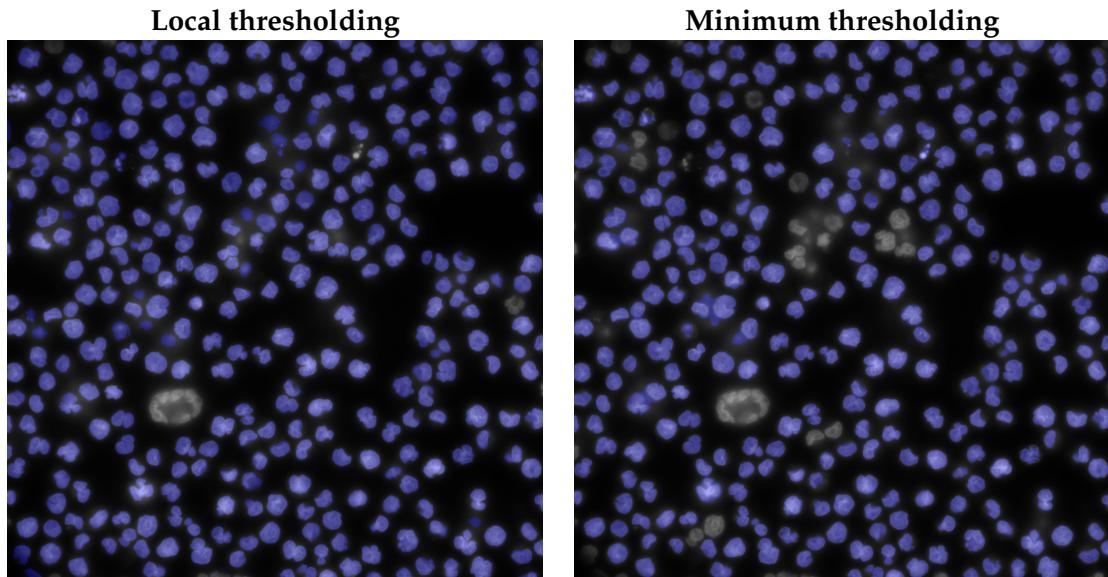


Figure 12: Local vs. Global thresholding (normal conditions)

### 3.2.2 Training and predictions

#### 3.2.2.1 Convergence

Has the model converged or not. Will more data help?

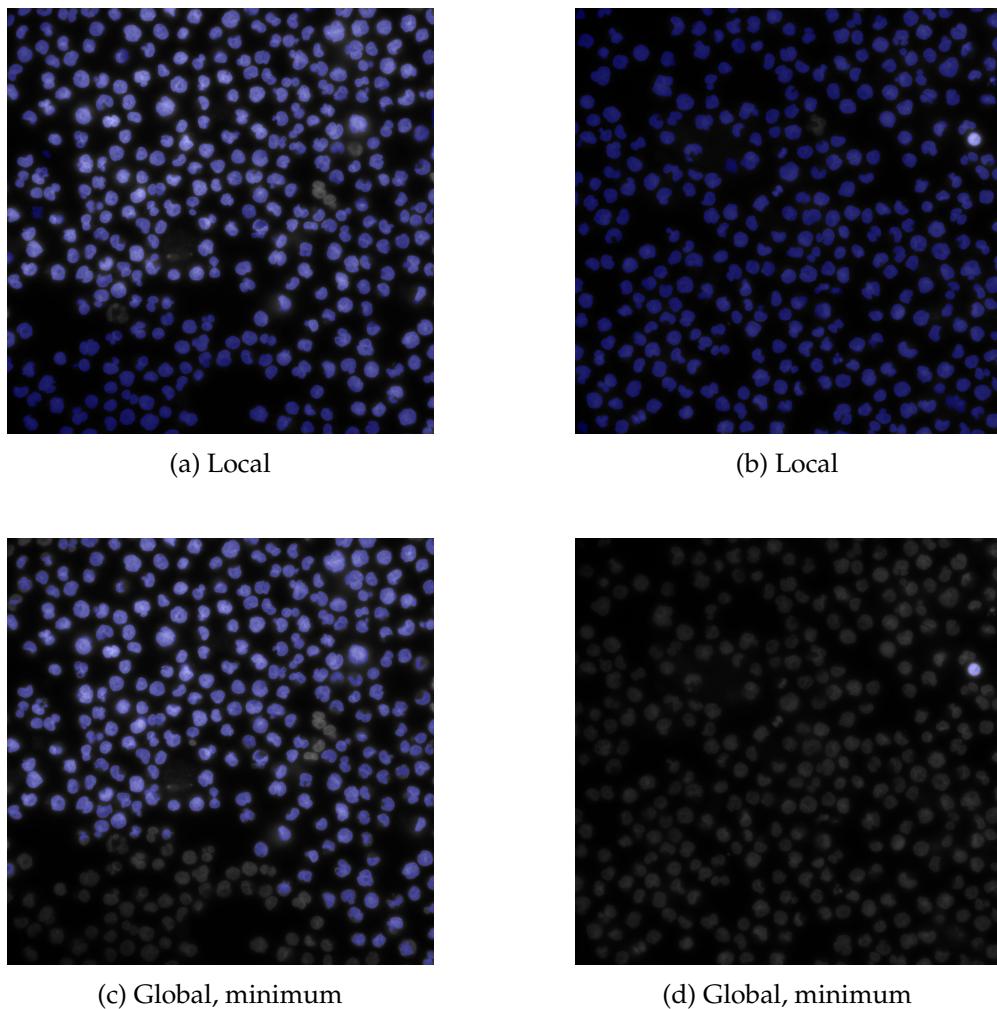


Figure 11: Local vs. Global thresholding

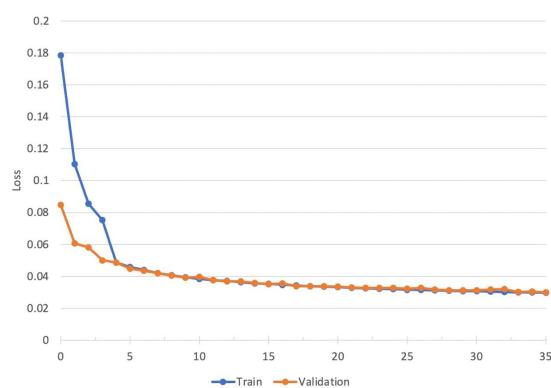


Figure 13: Having more data makes training more stable

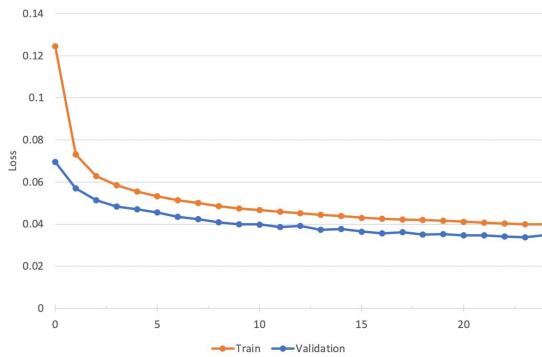


Figure 14: With regularization and augmentations PCC

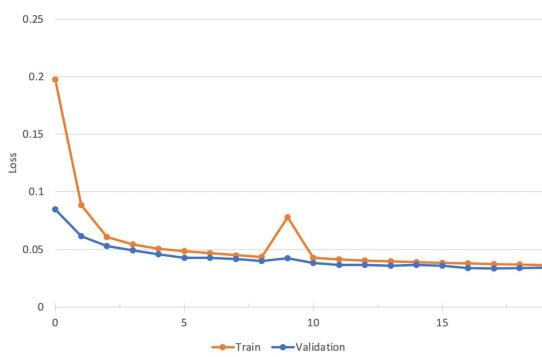


Figure 15: No regularization but augmentations

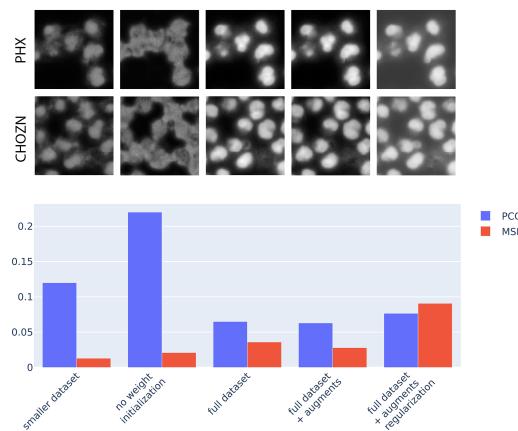


Figure 16: Difefrent models predictions and scores comparison

### 3.2.2.2 Predictions quality

Blurry, boundaries, not enough of details and possible improvements

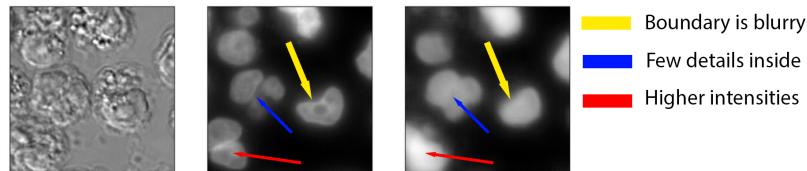


Figure 17: Problems in predictions

### 3.2.3 Postprocessing for nuclei segmentation

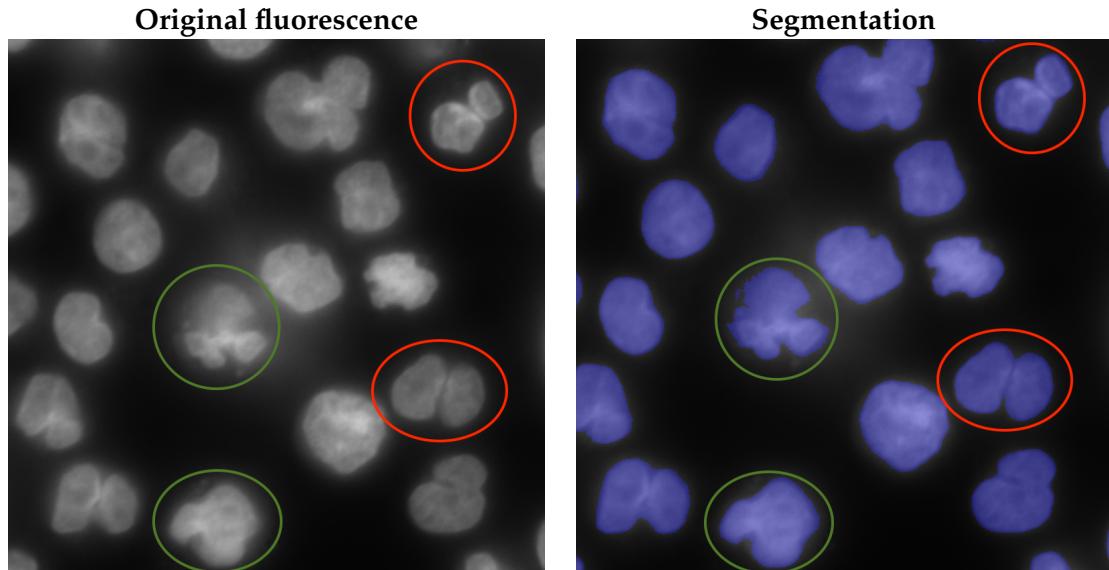


Figure 18: Closely located cells

Overall algorithm

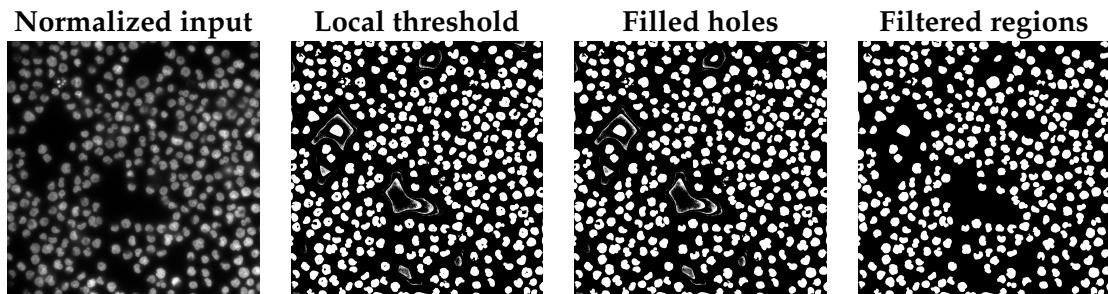


Figure 19: Fluorescence segmentation

### 3.2.4 Influence of scaling on predictions quality

Examples of predictions quality with different scales.

Table 2: Pearson correlation coefficients for downstream tasks for different scaling factors

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

### 3.3 Endoplasmic Reticulum

#### 3.3.1 Preprocessing

---

**Algorithm 2** Fluorescence segmentation
 

---

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

Segmentation steps are also illustrated in Figure

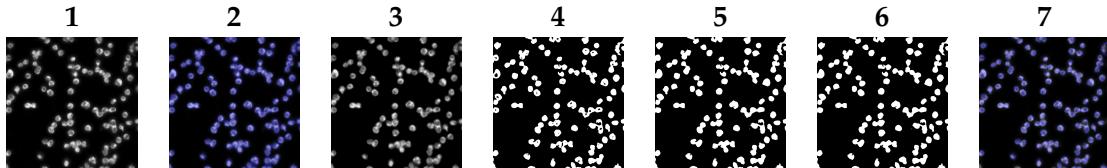


Figure 20: ER prediction

#### 3.3.2 Training and predictions

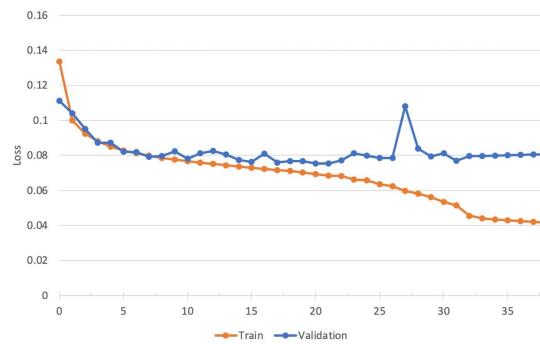


Figure 21: Overfit

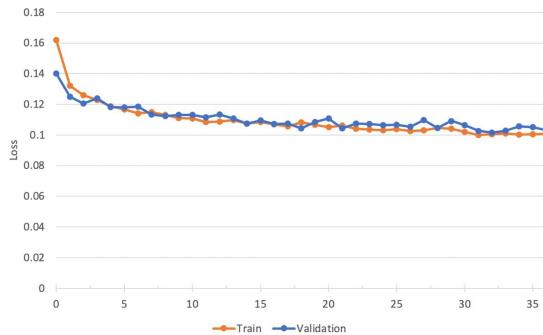


Figure 22: No overfit with augmentations

### 3.3.3 Combination of nuclei and actin predictions

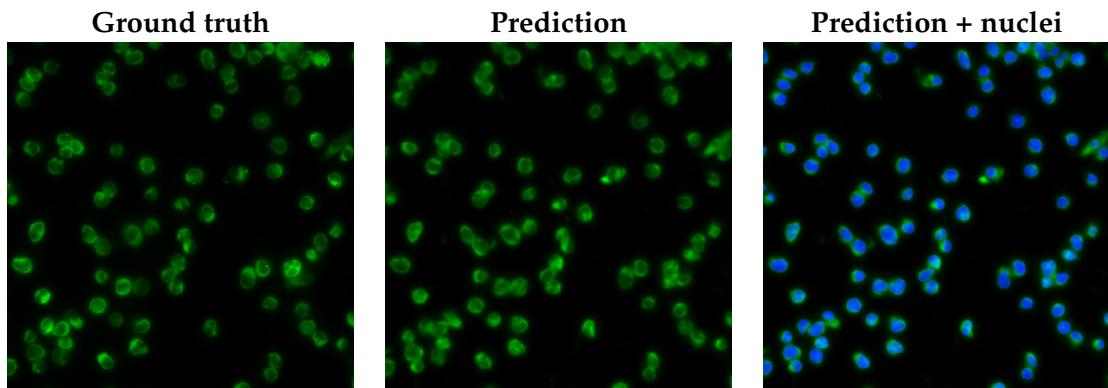


Figure 23: ER prediction

### 3.3.4 Generalizability across phenotypes

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

## 3.4 Golgi

### 3.4.1 Preprocessing

Enhancement

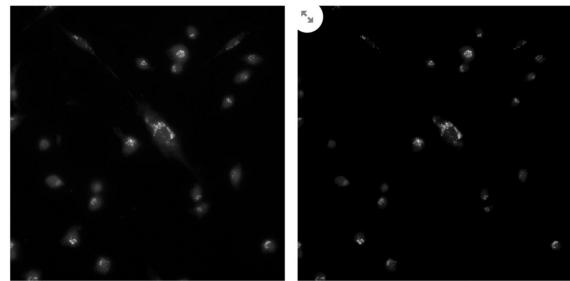


Figure 24: Golgi enhancement

#### 3.4.1.1 Background removal algorithms

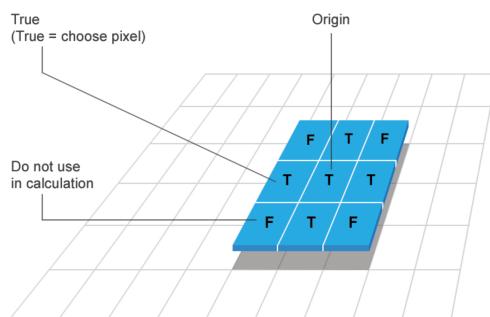


Figure 25: Structuring Element

Rolling ball algorithms

Rolling ball still leaves some noise

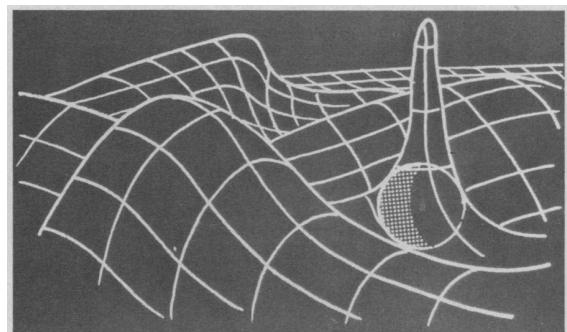


Figure 26: Rolling Ball

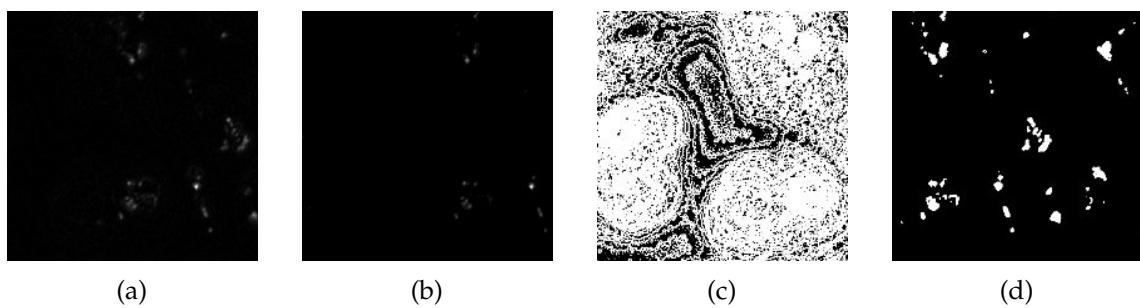


Figure 27: (a) Vanilla pre-processing with automatic background removal algorithm only; (b) Additional clipping of lower intensities after vanilla pre-processing; (c) masked or subfigure (a); (d) mask of subfigure (b)

### 3.4.2 Training and predictions

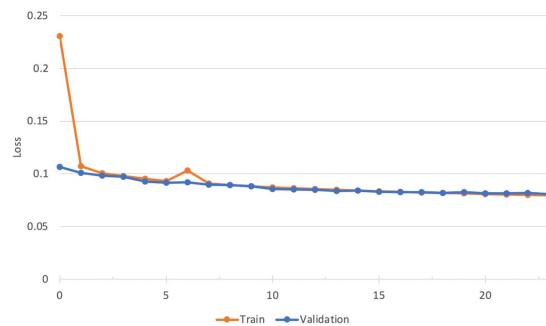


Figure 28: Straightforward training doesn't work

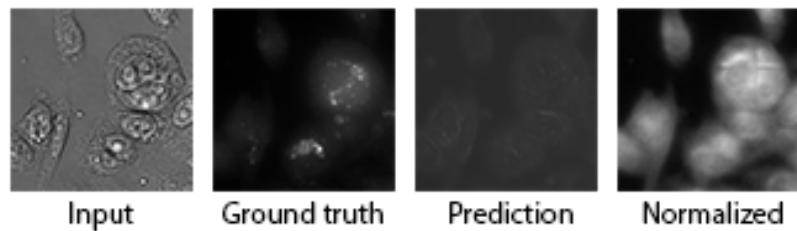


Figure 29: Training on original data

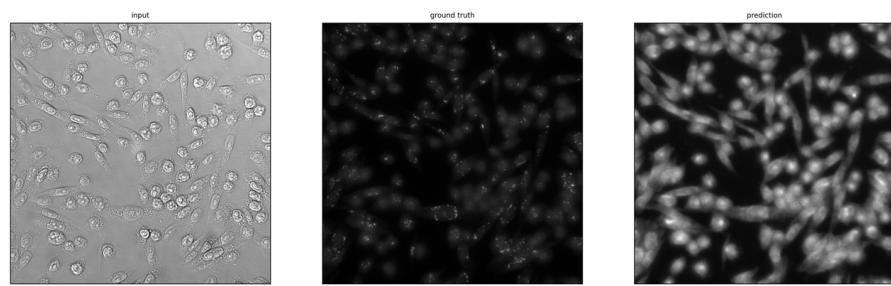


Figure 30: Full size predictions

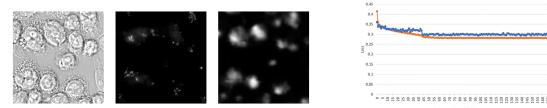


Figure 31: Training on the enhanced data

### 3.4.3 Alternative ways to improve predictions

#### 3.4.3.1 Asymmetrical losses

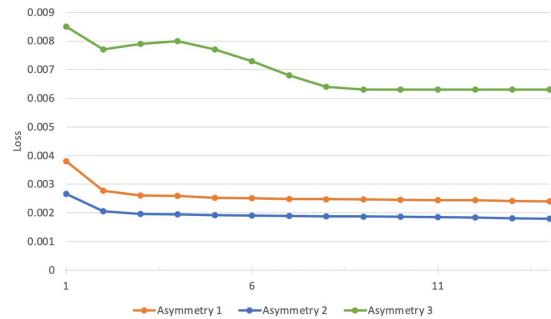


Figure 32: Asymmetrical training

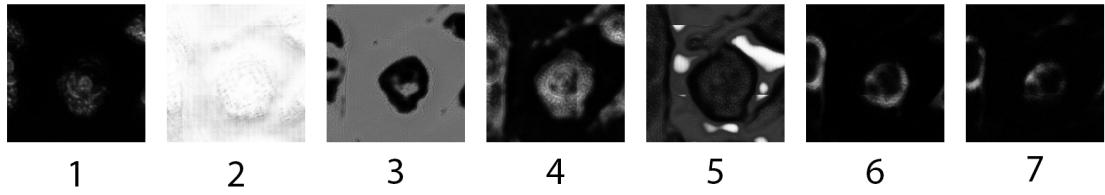


Figure 33: Asymmetrical training predictions

#### 3.4.3.2 Use of gradient in loss

#### 3.4.3.3 Noise reduction methods

### 3.5 GFP

#### 3.5.1 Preprocessing

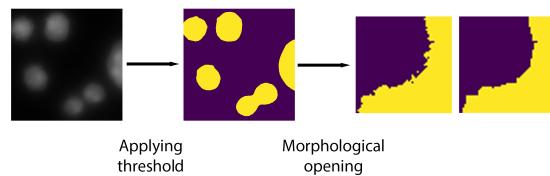


Figure 34: Converting GFP to a binary mask

#### 3.5.2 Predictions

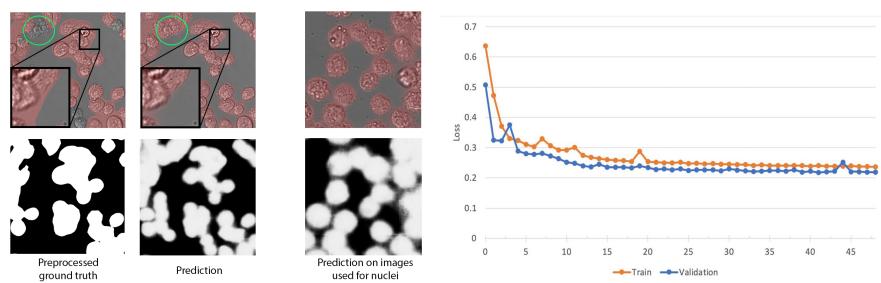


Figure 35: Training with BCE loss

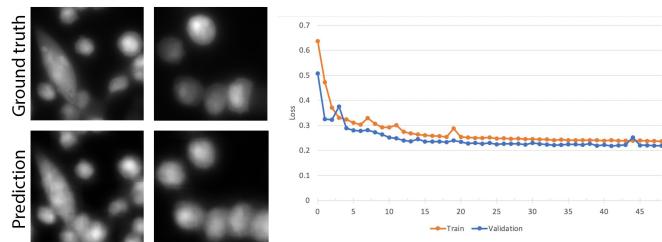


Figure 36: Training with Pearson correlation loss

Table 3: Correlation coefficients for downstream tasks

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

### 3.5.3 Downstream metrics

TODO move to separate chapter?

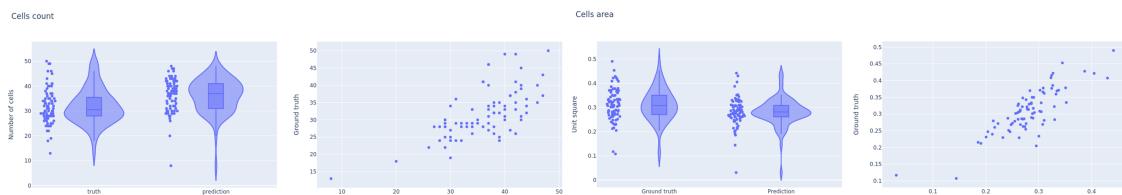


Figure 37: Downstream metrics

### 3.5.4 Combination of GFP, nuclei and ER

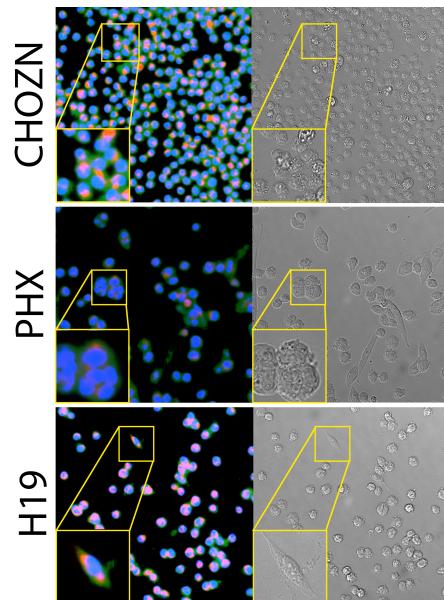


Figure 38: GFP, Nuclei and ER combined

### 3.6 Model evaluation

#### 3.6.1 Metrics for downstream tasks

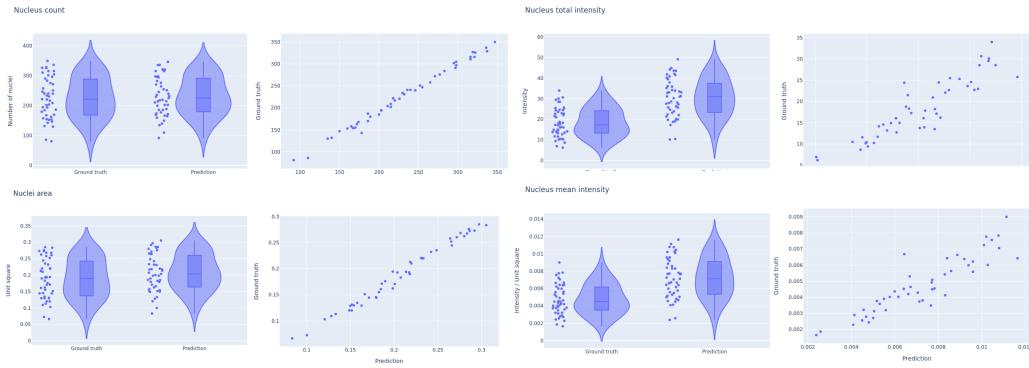


Figure 39: Metrics for downstream tasks on nuclei

Table 4: Correlation coefficients for downstream tasks on nuclei

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

#### 3.6.2 Influence of different loss functions on metrics for downstream tasks

## 4 Stability study

### 4.1 Stability study

#### 4.1.1 Artificial corruptions

Description of artificial corruptions.

Table 5: Hyperparameterization for different artificial corruption severities

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

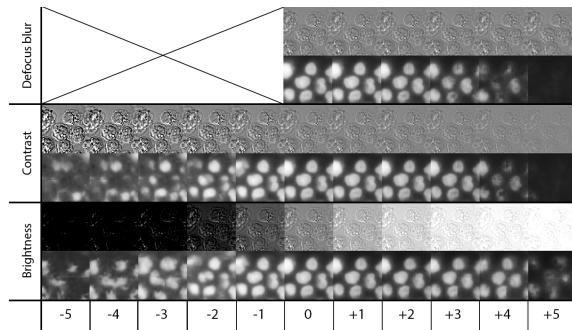


Figure 40: Influence of artificial corruptions on the predictions

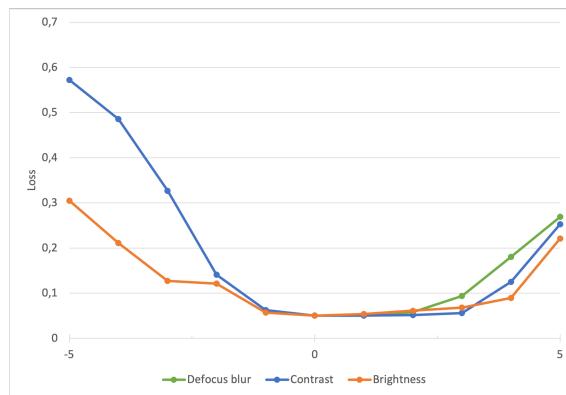


Figure 41: Change of Pearson correlation loss for artificial corruptions

### 4.1.2 Real corruptions

#### 4.1.2.1 Not fixed cells imaging as corrupted input

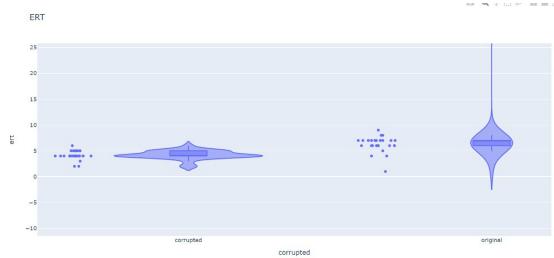


Figure 42: Online drift detection of not fixated cells

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

#### 4.1.2.2 Real-world examples of corruptions

#### 4.1.3 Influence of corruptions on metrics for downstream tasks

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

#### 4.1.4 Improving predictions with additional corruption augmentations

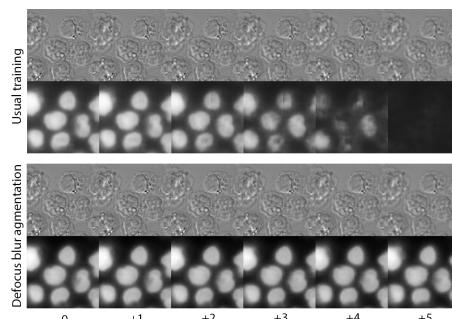


Figure 43: Using corruptions as augmentations improves predictions

## 4.2 UNET embeddings study

### 4.2.1 Application of various dimentionality reduction methods

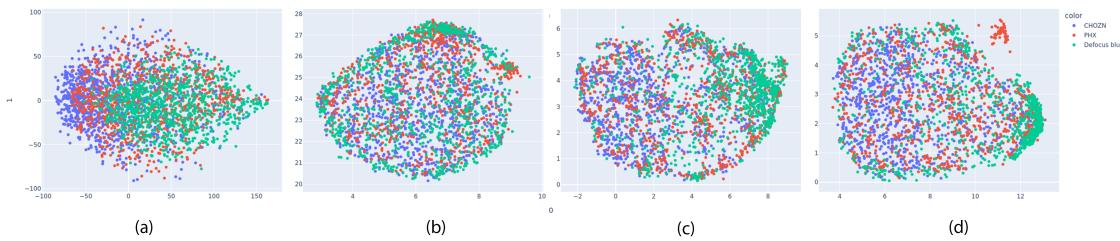


Figure 44: (a) PCA, (b) UMAP, (c) combination of PCA and UMAP with 10 and (d) 50 components

### 4.2.2 Autoencoder embeddings as an alternative

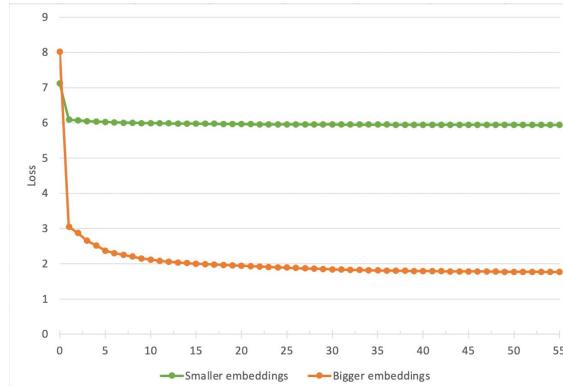


Figure 45: Autoencoders training convergence

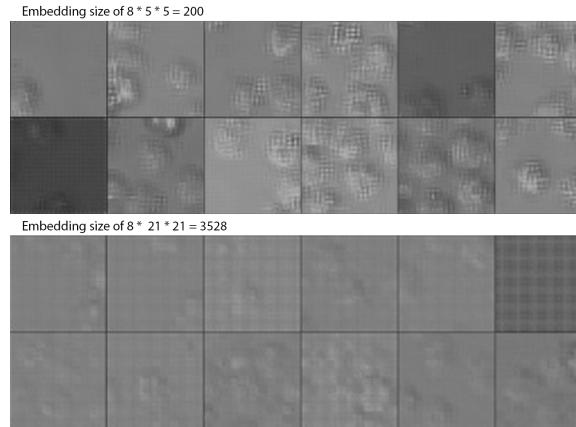


Figure 46: Samples drawn from the trained autoencoder

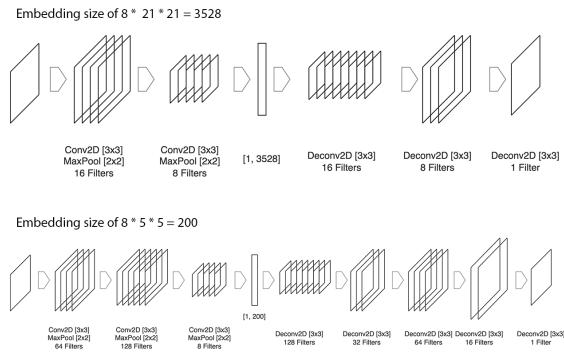


Figure 47: Architectures of two autoencoders

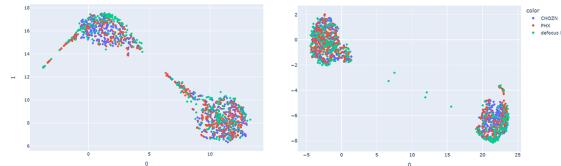


Figure 48: Autoencoder embeddings after applying PCA with 10 components and UMAP afterwards. Earlier epoch VS later epoch

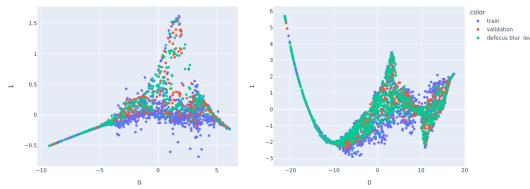


Figure 49: PacMAP does not provide information on the corruption

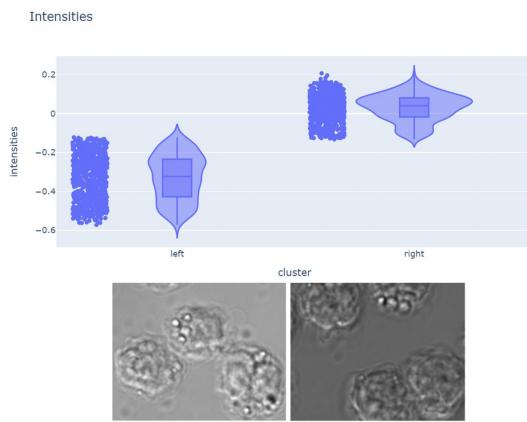


Figure 50: What do two UMAP clusters represent

### 4.2.3 Clustering of PacMAP embeddings

#### 4.2.3.1 Clustering on UNet embeddings

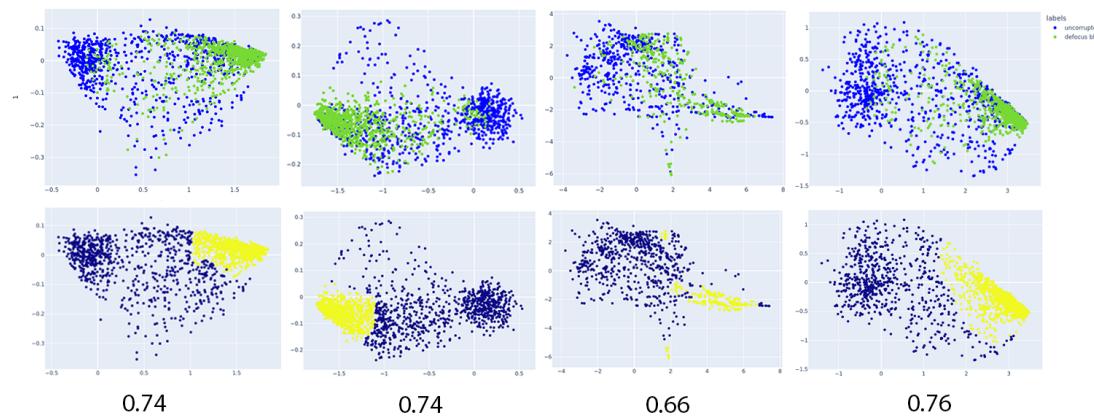


Figure 51: Clustering of UNet embeddings after PacMAP

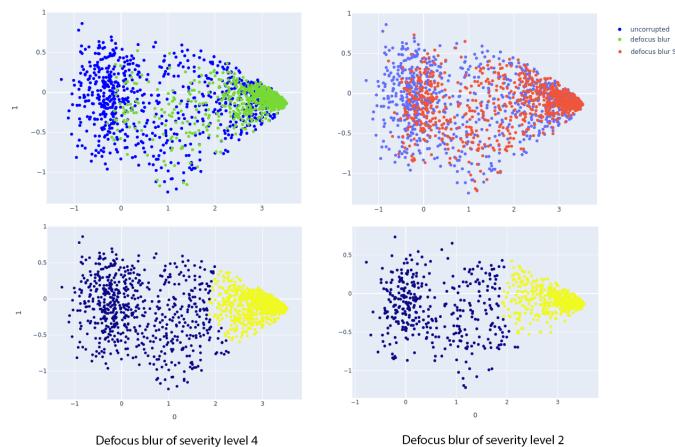


Figure 52: Clustering of UNet embeddings after PacMAP for different severities levels

TABLE with F1-score: 0.76 VS 0.64

### 4.3 Drift detection

#### 4.3.1 A need to detect drift

#### 4.3.2 Maximum mean discrepancy for drift detection

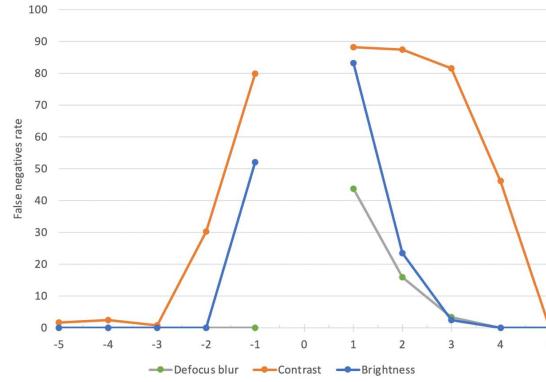


Figure 53: False negatives rate for drift detection on artificial corruptions

#### 4.3.3 Online version of MMD algorithm

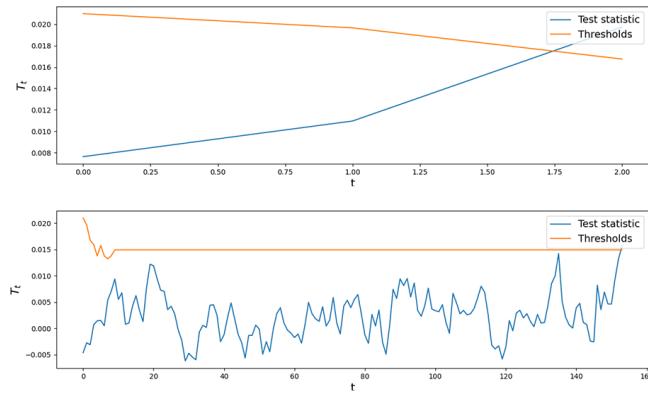


Figure 54: Expected runtime (ERT) for corrupted and in-distribution data

Table 6: 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 7: ERT influence on separability

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

Table 8: Severity of corruptions on separability

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

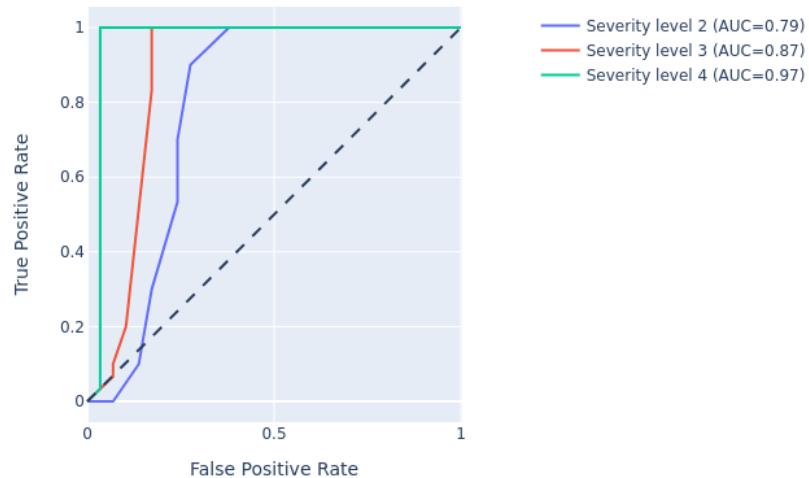


Figure 55: AUC ROC scores for various defocus corruptions severities

## 5 Software Tools

5.1 Foundry. Palantir

5.2 AWS

5.3 Streamlit

## 6 Future research

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

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

## 7 Summary

## 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 . . . . .  | 16 |
| 5  | Difference of overlap between predictions on the resulting image . . . . .   | 17 |
| 6  | Unet . . . . .   | 18 |
| 7  | Using original image for rotation and scaling augmentations . . . . .  | 19 |
| 8  | Nuclei training without and with custom weight initialization . . . . .  | 19 |
| 9  | Overfitting . . . . .  | 20 |
| 10 | Different lightning conditions . . . . .   | 21 |
| 12 | Local vs. Global thresholding (normal conditions) . . . . .  | 21 |
| 11 | Local vs. Global thresholding . . . . .  | 22 |
| 13 | Having more data makes training more stable . . . . .  | 22 |
| 14 | With regularization and augmentations PCC . . . . .  | 23 |
| 15 | No regularization but augmentations . . . . .  | 23 |
| 16 | Difefrent models predictions and scores comparison . . . . .   | 23 |
| 17 | Problems in predictions . . . . .  | 24 |
| 18 | Closely located cells . . . . .  | 24 |
| 19 | Fluorescence segmentation . . . . .  | 25 |
| 20 | ER prediction . . . . .  | 26 |
| 21 | Overfit . . . . .  | 26 |
| 22 | No overfit with augmentations . . . . .  | 27 |
| 23 | ER prediction . . . . .  | 27 |
| 24 | Golgi enhancement . . . . .  | 28 |
| 25 | Structuring Element . . . . .  | 28 |
| 26 | Rolling Ball . . . . .   | 29 |
| 27 | (a) Vanilla pre-processing with automatic background removal algorithm only; (b) Additional clipping of lower intensities after vanilla pre-processing; (c) masked or subfigure (a); (d) mask of subfigure (b) . . . . . | 29 |
| 28 | Straightforward training doesn't work . . . . .  | 29 |
| 29 | Training on original data . . . . .  | 30 |
| 30 | Full size predictions . . . . .  | 30 |

|                       |    |
|-----------------------|----|
| <i>LIST OF TABLES</i> | 47 |
|-----------------------|----|

|    |  |    |
|----|--|----|
| 31 | Training on the enhanced data . . . . .  | 30 |
| 32 | Asymmetrical training . . . . .  | 31 |
| 33 | Asymmetrical training predictions . . . . .  | 31 |
| 34 | Converting GFP to a binary mask . . . . .  | 32 |
| 35 | Training with BCE loss . . . . .   | 32 |
| 36 | Training with Pearson correlation loss . . . . .   | 32 |
| 37 | Downstream metrics . . . . .   | 33 |
| 38 | GFP, Nuclei and ER combined . . . . .  | 33 |
| 39 | Metrics for downstream tasks on nuclei . . . . .   | 34 |
| 40 | Influence of artificial corruptions on the predictions . . . . .   | 35 |
| 41 | Change of Pearson correlation loss for artificial corruptions . . . . .  | 35 |
| 42 | Online drift detection of not fixated cells . . . . .  | 36 |
| 43 | Using corruptions as augmentations improves predictions . . . . .  | 36 |
| 44 | (a) PCA, (b) UMAP, (c) combination of PCA and UMAP with 10 and (d) 50 components . . . . .                               | 37 |
| 45 | Autoencoders training convergence . . . . .  | 37 |
| 46 | Samples drawn from the trained autoencoder . . . . .   | 38 |
| 47 | Architectures of two autoencoders . . . . .  | 38 |
| 48 | Autoencoder embeddings after applying PCA with 10 components and UMAP afterwards. Earlier epoch VS later epoch . . . . . | 38 |
| 49 | PacMAP does not provide information on the corruption . . . . .  | 39 |
| 50 | What do two UMAP clusters represent . . . . .  | 39 |
| 51 | Clustering of UNet embeddings after PacMAP . . . . .   | 40 |
| 52 | Clustering of UNet embeddings after PacMAP for different severities levels . . . . .                                     | 40 |
| 53 | False negatives rate for drift detection on artificial corruptions . . . . .   | 41 |
| 54 | Expected runtime (ERT) for corrupted and in-distribution data . . . . .  | 41 |
| 55 | AUC ROC scores for various defocus corruptions severities . . . . .  | 42 |

## List of Tables

|   |   |    |
|---|---|----|
| 1 | Available data for each fo the organelles . . . . .   | 18 |
| 2 | Pearson correlation coefficients for downstream tasks for different scaling factors . . . . . | 25 |

|   |  |    |
|---|--|----|
| 3 | Correlation coefficients for downstream tasks . . . . .                        | 33 |
| 4 | Correlation coefficients for downstream tasks on nuclei . . . . .              | 34 |
| 5 | Hyperparameterization for different artificial corruption severities . . . . . | 35 |
| 6 | Test window size influence on separability . . . . .                           | 41 |
| 7 | ERT influence on separability . . . . .  | 42 |
| 8 | Severity of corruptions on separability . . . . .                              | 42 |