Deeper Understanding of Autophagy and pseudo-Autophagy through Latent-Space Analysis

Sigurjón Ísaksson
Clare Hall

UNIVERSITY OF CAMBRIDGE

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Machine Learning, Speech, and Language Technology

University of Cambridge
Engineering Department
Trumpington Steet
Cambridge CB2 1PZ
United Kingdom

Email: si318@cam.ac.uk

August 16, 2018

Declaration

I Sigurjón Ísaksson of Clare Hall, being a candidate for the M.Phil in Machine Learning, Speech, and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 13094

Signed:

Date:

Acknowledgements

First of all, I would like to thank my industrial supervisor, Dr. Matthew Johnson, both for proposing this project, and for his indispensable support and guidance. I would also like to thank Dr. Rupert Bale, who supervised the biological aspect of the project. Without his guidance, this project would have never been possible. Working with both of them has been a pleasurable experience. Secondly, I would like to thank my girlfriend, Tinna Björk, for her unconditional support and patience throughout this MPhil. Most importantly, I would like to thank my loving parents. Without you, none of this would have been possible. Your love, believe in me, support and guidance are the reasons I am where I am, and for that, I am grateful.

Abstract

Autophagy is a critical biological process due to its importance for the survival of cells. Despite it being widely studied in the biological research community, our understanding of it remains limited. This is partly due to the fact that processes happening inside cells follow a complex, stochastic behaviour, which is hard for human researchers to identify, or distinguish between. In this thesis, we attempt to address this by taking advantage of recent advances in deep learning and computer vision. A large amount of imaging data is available where these processes are known to be happening. However, to our best knowledge, there has been no research directed towards using machine learning to understand autophagy and its causes better. We show that with the use of discriminative models, we can find distinguishable patterns and features in the cell images, enabling us to discriminate between cell types with different processes happening inside of them. We also show that by using generative latent variable models, we can model a latent space of the cell images that contains interpretable generative features with regards to autophagy. Findings in this thesis indicate that machine learning can be used to improve understanding of autophagy, and with further research, it has the potential to help make important discoveries in this field.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Autophagy is a biological process in which dysfunctional cellular machinery is degraded into its core components and then used to build new parts. It can take many different forms, and other biological processes behave similarly. These processes happen simultaneously inside cells, making distinguishing between them a tedious and difficult task, while quantifying the processes is even more difficult. This consumes valuable research time, resulting in slower and less effective research. In this project, we aim to exploit recent advances in deep neural networks and computer vision to create methods that allow researchers to better understand autophagy, as well as automating time consuming tasks.

We propose two different ways of approaching the problem. The first method is to use supervised learning to train a classifier that will discriminate between cells that are known to have different processes happening inside them. This gives us an indication of how different the cell types we are working with are, and whether a machine learning algorithm could find distinguishable patterns in these cells. A more interesting approach is to use unsupervised learning, training a generative latent variable model that models the latent space of the cell images. If the latent space is properly disentangled, our hypothesis is that it will contain valuable information about the cells, which will not only help distinguish between cell types, but also allow us to learn which of the cells' features are responsible for different processes. Mapping the cell images to their latent space representation also gives us a robust method for quantifying these features inside the cells. This has the potential to help us gain a better understanding of autophagy and its causes, which is a very active and important research area in biology. The goal of this thesis is not to solve this problem comprehensively, as that would require a much more thorough study, using more time and resources then are available for this project. Instead the aim is to find indications which show that machine learning can be helpful for this problem, and that it is possible to model a disentangled latent space for the cell images that contains informative features with regards to autophagy. According to our best knowledge, this project is the first of its kind in this research area, making it an especially interesting study, and we hope that it can offer a stepping stone to further research.

The dataset for this thesis was gathered using an imaging flow cytometer, which is able to capture a large amount of individual cell images in a short amount of time, which is convenient in machine learning. However, the data that it generates is very raw, containing images that need to be preprocessed before we can learn anything valuable from them. Moreover, some of the images are simply not suitable for research, as they contain clumps of cells or even objects that are not cells. Therefore, an important task is to preprocess the cell images, and develop a method that allows us to distinguish between images that contain a single cell and images that do not. We propose using machine learning to distinguish between these images, filtering them with a binary classifier that outperforms current methods used to filter these images.

1.2 Contributions

The objective of this project is to develop methods that will enable us to carry out the processes described above. Our contributions include:

1. A method to preprocess the initial raw dataset provided by the imaging flow cytometer.
2. A binary classifier which separates single cell images from images that contain multiple cells or objects that are not cells.
3. A discriminative model that is able to distinguish between different cell types of interest.
4. Generative models that are able to model the latent space of the cell images.

1.3 Thesis outline

The rest of this thesis will be laid out as follows: In chapter 2, we describe the relevant background knowledge necessary to understand the machine learning methods used in this thesis and related work. Chapter 3 explains how the dataset was preprocessed, and chapter 4 describes the binary classification of useful cell images from unusable ones. Finally, chapter 5 explains the discriminative model that was built as well as discussing results, while chapter 6 includes a description of the generative models and a discussion of the results. The thesis is then concluded in chapter 7 with a summarisation of the work that was done, and a discussion of future work. All code written for this thesis can be found in the MLSALT machines, located in the authors home directory. All code was written in Python, using the deep learning library TensorFlow. All computing took place using Azure virtual machines, provided by the MPhil.

Chapter 2

Background

2.1 Multi-Layer Perceptron

A multi-layer perceptron (MLP), often referred to as a feed-forward neural network, is a powerful machine learning algorithm, whose behaviour is loosely based on how a biological brain works. The main building block of the algorithm is the perceptron, also called a neuron, as it is a simplified version of a biological neuron. The neuron is a function that takes in a weighted linear combination of signals as an input, adds a bias, and then maps it to an output signal using an activation function, which is usually non-linear. This can be expressed mathematically by the following equation:

$$a = h\left(\sum_{k=1}^{K} w_k * x_k + b\right) \tag{2.1}$$

Using just one neuron, it is possible to create a classifier or a regression model, depending on which activation function is used. For example, using a unit step function, we can create a simple binary classifier with a linear decision boundary, classifying it as positive (1) if the weighted linear combination of inputs is larger than the threshold, or negative (0) if not. This algorithm is called the Perceptron, and can be seen in Figure 2.1.

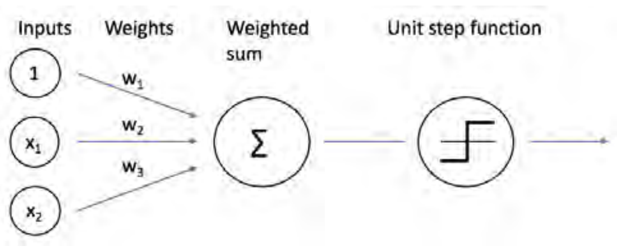


Figure 2.1: A demonstration of the Perceptron algorithm.

An MLP consists of a stack of layers, l = 1,2,..L, where each layer contains k^l neurons. The first layer is the input layer, where each neuron represents a value of the input vector. The layers in-between the first and last layer are referred to as the hidden layers of the network, as they are not observed, and the final layer is called the output layer, as it represents the output of the network. Each layer in the network is fully connected to the previous layer, meaning that each neuron in layer l+1 receives a signal from all neurons in layer l. As we add more hidden layers, the network becomes deeper, resulting in deep learning. Figure 2.2 shows an example of a 3-layer MLP.

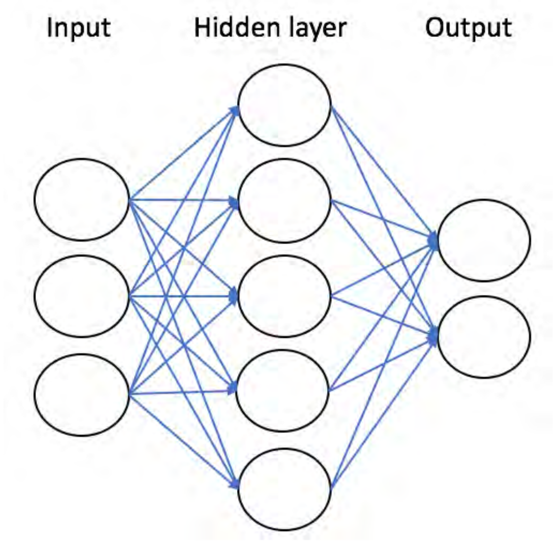


Figure 2.2: A demonstration of a 3-layer MLP

By stacking the neurons together like this, the network is able to generate a complex, non-linear mapping of the input data. In fact, the universal approximation theorem states that an MLP with a single layer can represent any continuous function. However, this layer may be infeasibly large and may fail to learn and generalise correctly [10]. Therefore, adding hidden layers for complexity has been shown to significantly improve performance. The output of a 3-layer MLP, using a sigmoid activation function, σ, in the output layer and any activation function in the hidden layer, can be described mathematically with equation 2.2

$$y(x, w) = \sigma \left(\sum_{j=1}^{M} w_{kj}^{(2)} h \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \tag{2.2}$$

To train the algorithm, we define an objective function, depending on our application, that takes in the output of the network and labels that represent the ground truth, and optimise that function w.r.t the networks parameters. A common choice of objective function is cross entropy for classification problems, and mean squared error for regression problems. To optimise the function, we use stochastic gradient optimisation methods, *e.g.* stochastic gradient descent, where the gradients w.r.t all parameters in the network are computed using backpropagation [25].

2.2 Convolutional Neural Networks

A convolutional neural network (CNN) is a variant of the multi-layer perceptron algorithm (MLP), and has gained increasing popularity for use in solving computer vision problems in recent years, achieving state of the art results in various tasks. In MLP, each neuron is fully connected to all the neurons in the previous layer, which quickly becomes a problem for high dimensional data, such as images. For example, the input features of a 256x256x3 image have a dimension of 196,608. By adding a hidden layer consisting of just 1000 neurons, the network would already have 196,608,000 parameters in the first layer. This increases the capacity of the system, and for larger networks, the computational cost quickly becomes infeasible, as well as requiring a larger training set to generalise. Furthermore, MLPs are not invariant with respect to translations in the input; *e.g.* different positioning of the object in the image or rotation. Training a network to recognise all these variations would be inefficient and would require a large amount of data. Moreover, in images, pixels that are close to each other are often highly correlated, while pixels that are far from each other are not. Since the MLP architecture is fully connected, it does not capture this correlation [18].

CNNs offer a solution to the problems described above by introducing three factors that distinguish them from MLP: local receptive fields, shared weights and spatial or temporal sub-sampling [18]. Firstly, each neuron is constrained to only depend on a subset of variables that are close to each other in the previous layer. For example, for a 256x256x3 image, each neuron might only depend on a 5x5x3 window of that image, often referred to as the *receptive field* of the neuron. Secondly, a set of neurons with different receptive fields are constrained to share the same parameters. Note that a neuron computes a weighted linear combination of the input inside its receptive field, which is then mapped using a nonlinear activation function, similarly to MLP. By constraining all neurons with different receptive fields to share parameters, evaluating the neurons can be thought of as sliding the same filter, whose weights are learned by the network, across different windows of the input. Sliding the filter across the image is equivalent to convolving the image with the filter, which is why these layers are called convolutional layers, also explaining the name of the algorithm. The output of this convolution is then used to construct a feature map, where each value of the feature map is the output of a neuron. The filters learn to look for certain features in the image, and the feature map tells us where these features are positioned. This provides equivariance to translation in the input. Figure 2.3 demonstrates how a filter, K, is convolved with an image, I, to form a feature map, $I \circledast K$.

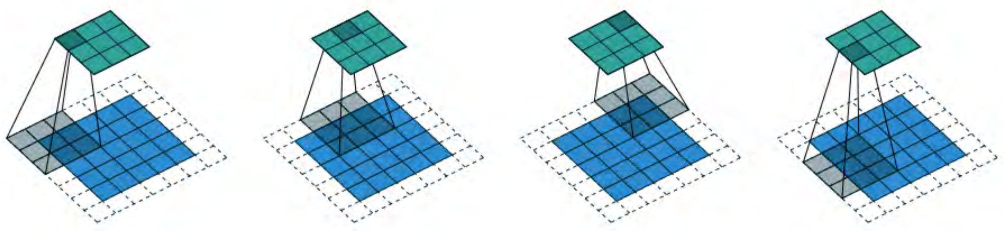


Figure 2.3: An example of a 3x3 convolution over a 5x5 input, using stride = 2 and zero padding the boarders with 1 frame, resulting in a 3x3 feature map[7].

A complete convolutional layer generates several feature maps, computed with different filters, allowing the network to extract many different features for each input. The feature maps are stacked together to form the output of the layer, and the input for the next layer. When computing convolution, there are two hyperparameters that control the size of the resulting feature map. The first one is called *stride*, which refers to the number of pixels we move over when sliding the filter. For example, if the stride is set to 1, the filter is moved one pixel at a time, while if the stride is 2, the filter will jump two pixels at a time when it slides over the input. The second hyperparameter is *zero padding*, where zeros are added to the borders of the input. By doing this, we can for example make sure that the output is the same size as the input after convolution. It is possible to calculate the output of the convolution using:

$$O = \frac{(W - F + 2P)}{S} + 1 \tag{2.3}$$

Where W is the input size, F is the receptive field size (filter size), S is the stride and P is the amount of zero padding.

The final factor that distinguishes a CNN from an MLP is the sub-sampling layers. The purpose of these layers is twofold: to reduce the dimensionality of the feature maps and to introduce translational invariance to the network. These layers are often referred to as pooling layers, and they can be explained very similarly to convolutional layers. The layer slides a fixed size window across each feature map, usually with stride = 2, and performs a mathematical operation, most commonly finding the maximum or average value. For example, a max pooling layer would slide across the feature map and return the maximum value of each receptive field. The new feature map then consists of these maximum values. Figure 2.4 describes the max pool operation.

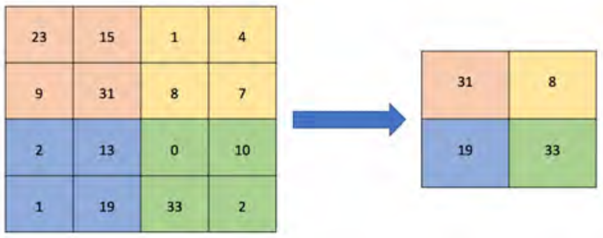


Figure 2.4: An example of a 2x2 max pooling, using stride = 2 and no zero padding, resulting in a 2x2 feature map

A typical CNN consists of a number of different layers, both convolutional and pooling, allowing us to detect complex high-level features as we increase the number of layers. The output of the last layer is then typically flattened into a 1D vector, which is then fed through one or more fully connected layers, ending with the output layer. As with MLPs, the network is trained by optimising an objective function with stochastic gradient based methods, where the gradients w.r.t all parameters are computed using back propagation [25]. Figure 2.5 demonstrates an example of a multi-layer CNN.

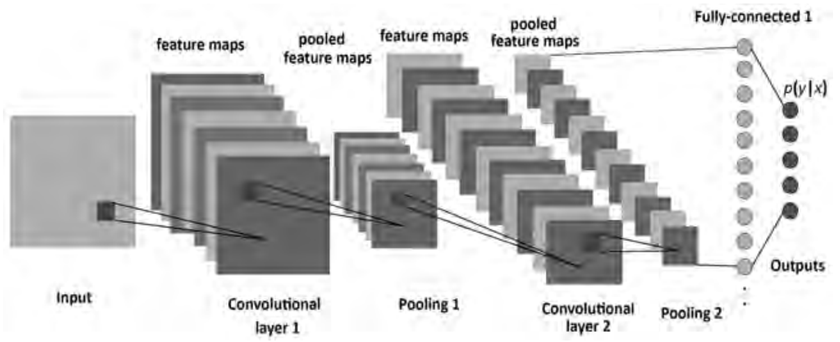


Figure 2.5: An example of a multi-layer CNN, consisting of two convolutional layers followed by max pooling, and one fully connected layer.

2.2.1 Upsampling using transposed convolution

For some applications, especially image synthesis, the goal is to increase the size of the input, instead of decreasing it. This can be accomplished by using transposed convolution [27]. A transposed convolutional layer is very similar to a convolutional layer. First, the input is mapped to a sparse, enlarged version of itself. How this transformation is computed depends on both the zero padding and the stride used to separate each pixel. Next, a filter is convolved with the sparse matrix, generating the new, enlarged feature map. The weights of the filters are learned in the same way as they are for regular convolutional layers. Figure 2.6 demonstrates the transposed deconvolution.

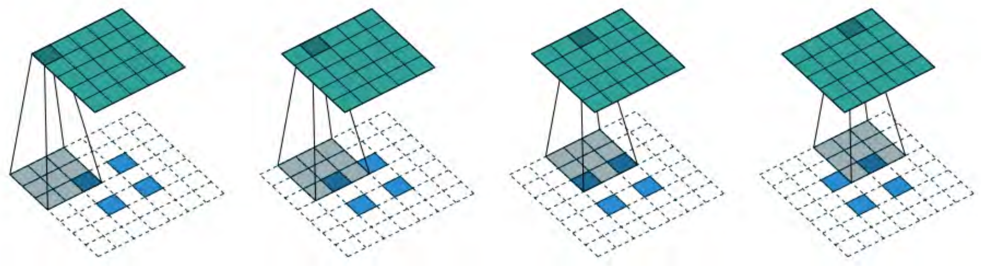


Figure 2.6: An example of a 3x3 transposed convolution over a 2x2 input, using stride = 2 and zero padding the boarders with two frames, resulting in a 5x5 feature map[7].

2.3 Generative latent variable models

Latent variable models (LVM) are probabilistic models which incorporate unobserved variables that affect the observed variables, referred to as latent variables. By defining a joint distribution over the sets of variables, complex distributions of interest can be modelled; *e.g.* the marginal distribution of the observed variables or conditional distributions [13].

For this thesis, we assume that our dataset consists of i.i.d datapoints, our cell images, where each datapoint $x^{(i)}$ is generated by a generative model that involves a latent variable, $z^{(i)}$. The process can be explained in two steps [15]:

- $z^{(i)}$ is drawn from a prior distribution, $p_{\theta*}(z)$
- The datapoint, $x^{(i)}$, is generated according to the conditional distribution $p_{\theta*}(x|z)$

where we assume that $p_{\theta*}(z)$ and $p_{\theta*}(x|z)$ are valid parametrised probability distributions. The space that the latent variable lies in is commonly referred to as the latent space of the observed data, and this definition will be used throughout this thesis. By constraining the latent space to have lower dimensions than the high dimensional data space, the latent variables form a condensed representation of the data, which we hope represents important distinguishable generative features. This can also be thought of as a directed graphical model, where our datapoint depends on the latent variable, and is conditionally independent of everything else when the latent variable is observed, as Figure 2.7 shows.

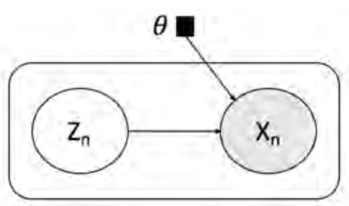


Figure 2.7: Directed graphical model that demonstrates the dependencies we assume our data follows.

Since the true parameter value θ_* and the latent variables z given the observed variable x are unknown, we are interested in estimating the model parameters θ, as well as doing posterior inference on the latent variable z given the observed variable x. In practice, this can prove difficult, since $p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$ is often intractable, especially when we have a complicated likelihood function $p_\theta(x|z)$. However, a number of approximation techniques exist for this problem, and the ones used in this thesis will be discussed below.

2.3.1 Disentangled latent space

An important field of research for generative latent variable models is to obtain disentangled latent space, meaning that each dimension contains interpretable semantic information about the data, independent of other dimensions [5]. For example, a dimension in the latent space might represent the size of the cell, and mapping the cell image to its latent space therefore gives us quantitative information about the cell's size. Another dimension could tell us how certain components inside the cell are distributed, which gives us interesting information about processes happening inside the cells, such as autophagy.

2.4 Auto-encoding variational bayes

Auto-Encoding Variational Bayes (AEVB) [15] is a method that provides an efficient way of inferring the latent variables z, and doing maximum likelihood (ML) or maximum a posteriori (MAP) inference on the parameters, in graphical models as described above in Figure 2.7, where the posterior distribution $p_\theta(z|x)$ is intractable and we have a large dataset. It does so by introducing a recognition model, $q_\phi(z|x)$, which is an approximation of the true intractable posterior, $p_\theta(z|x)$. This recognition model is often referred to as a probabilistic encoder, as it outputs a distribution over the possible latent space values that the data will be mapped to. Likewise, the distribution

$p_\theta(x|z)$ is referred to as a probabilistic decoder, as it generates a distribution of the possible values of x, given a latent variable z.

The probabilistic encoder, $q_\phi(z|x)$, is approximated using variational inference, which is a technique commonly used to approximate intractable probability distributions. We approximate the intractable posterior:

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)} = \frac{p_\theta(x,z)}{\int_z p_\theta(x,z)dz} \tag{2.4}$$

using another tractable distribution, $q_\phi(z|x)$, and then try to make these distributions as close to each other as possible. A known technique for measuring the distance between two distributions is the KL divergence. A useful feature of the KL divergence is that it is always larger than or equal to zero, and as it gets smaller, the two distributions become increasingly similar, achieving a KL divergence of zero when the two distributions are the same. Therefore, we can minimise the KL divergence between $p_\theta(z|x)$ and $q_\phi(z|x)$ and thus make them as close to each other as possible.

$$
\begin{aligned}
KL\big(q_\phi(z|x)||p_\theta(z|x)\big) &= \int_z q_\phi(z|x)log\left(\frac{q_\theta(z|x)}{p_\theta(z|x)}\right)dz \\
&= \int_z q_\phi(z|x)log\left(q_\phi(z|x)\frac{1}{p_\theta(z,x)/p_\theta(x)}\right)dz \\
&= \int_z q_\phi(z|x)log\left(q_\phi(z|x)\frac{p_\theta(x)}{p_\theta(z,x)}\right)dz \\
&= \int_z q_\phi(z|x)\left(log\left(\frac{q_\phi(z|x)}{p_\theta(z,x)}\right) + log\big(p_\theta(x)\big)\right)dz \\
&= \int_z q_\phi(z|x)log\left(\frac{q_\phi(z|x)}{p_\theta(z,x)}\right)dz + log\big(p_\theta(x)\big)\int_z q_\phi(z|x)dz \\
&= \int_z q_\phi(z|x)log\left(\frac{q_\phi(z|x)}{p_\theta(z,x)}\right)dz + log\big(p_\theta(x)\big) \\
\Leftrightarrow log\big(p_\theta(x)\big) &= KL\big(q_\phi(z|x)||p_\theta(z|x)\big) - \int_z q_\phi(z|x)log\left(\frac{q_\phi(z|x)}{p_\theta(z,x)}\right) \\
&= KL\big(q_\phi(z|x)||p_\theta(z|x)\big) + \int_z q_\phi(z|x)log\left(\frac{p_\theta(z,x)}{q_\phi(z|x)}\right)dz
\end{aligned}
\tag{2.5}
$$

$log\big(p_\theta(x)\big)$ does not depend on z, and can therefore be thought of as a constant. Thus, since the KL divergence is always non-negative, maximising the second term on the right hand side in Equation 2.5 is equivalent to minimising the KL divergence. This term is referred to as the evidence lower bound (ELBO), since it is always less than or equal to $log\big(p_\theta(x)\big)$, being equal when $p_\theta(z|x)$ and $q_\phi(z|x)$ are the same distributions. The ELBO can be written as:

$$
\begin{aligned}
\mathcal{L}(\theta, \phi; x) &= \int_z q_\phi(z|x) log\left(\frac{p_\theta(z,x)}{q_\phi(z|x)}\right) dz \\
&= \int_z q_\phi(z|x) log\left(\frac{q_\phi(x|z)p_\theta(z)}{q_\phi(z|x)}\right) dz \\
&= \int_z q_\phi(z|x)\left(log\big(p_\theta(x|z)\big) + log\left(\frac{p_\theta(z)}{q_\phi(z|x))}\right)\right) dz \\
&= \int_z q_\phi(z|x) log\big(p_\theta(x|z)\big) dz + \int_z q_\phi(z|x) log\left(\frac{p_\theta(z)}{q_\phi(z|x)}\right) \\
&= \mathbb{E}_{q_\phi(z|x)}\big[log\big(p_\theta(x|z)\big)\big] - KL\big(q_\phi(z|x)||p_\theta(z)\big)
\end{aligned}
\tag{2.6}
$$

We now have a joint optimisation problem. We want to optimise the ELBO w.r.t both the model parameters θ and the variational parameters ϕ, which is a difficult task. Solving the expectation analytically is not possible, and the usual approach is therefore to estimate the ELBO via use of Monte Carlo methods. However, the estimator for the derivatives w.r.t the variational parameters ϕ:

$$
\begin{aligned}
\nabla_\phi \mathbb{E}_{q_\phi(z)}[f(z)] &= \mathbb{E}_{q_\phi(z)}\big[f(z)\nabla_{q_\phi(z)} log\big(q_\phi(z)\big)\big] \\
&\approx \frac{1}{L}\sum_{l=1}^{L} f(z)\nabla_{q_\phi(z^{(l)})} log\big(q_\phi(z^{(l)})\big) \\
z^{(l)} &\sim q_\phi(z|x^{(i)})
\end{aligned}
\tag{2.7}
$$

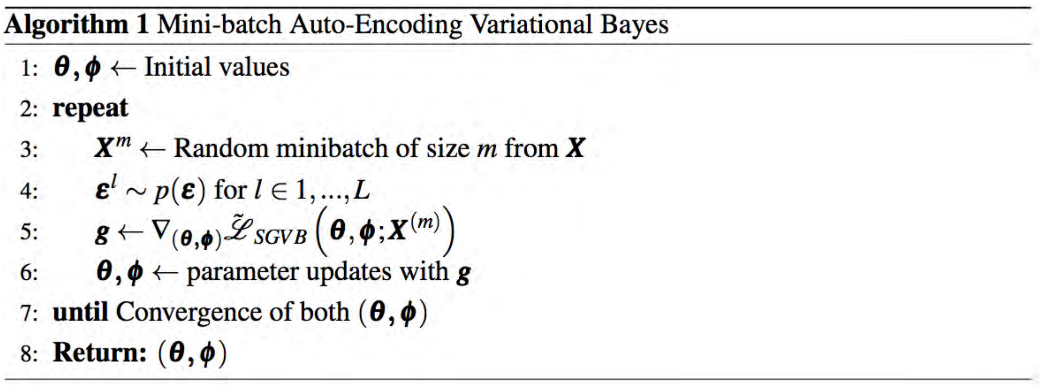
has been shown to exhibit high variance, making it impractical to optimise the ELBO [20]. The AEVB method introduces a practical estimator on the ELBO and its derivatives. It does so by reparameterising the random variable $z^{(l)} \sim q_\phi(z|x^{(i)})$ using a differentiable transformation $g_\phi(\epsilon, x)$, where ϵ comes from a distribution that we are able to sample from easily. By using this simple reparameterisation trick, we can define a Monte Carlo estimate for the expectation of a arbitrary function $f(z)$ w.r.t $q_\phi(z|x^{(i)})$

$$
\begin{aligned}
&\nabla_\phi \mathbb{E}_{q_\phi(z|x^{(i)})}[f(z)] = \mathbb{E}_{p(\epsilon)}[f(g_\phi(\epsilon, x^{(i)}))] \approx \frac{1}{L}\sum_{l=1}^{L} f(g_\phi(\epsilon^{(l)}, x^{(i)})) \\
&\epsilon^{(l)} \sim p(\epsilon)
\end{aligned}
\tag{2.8}
$$

Using this technique, we obtain an approximation of the ELBO, called the Stochastic Gradient Variational Bayes (SGVB) estimator , $\tilde{\mathcal{L}}(\theta, \phi; x^{(i)})$.

$$\begin{aligned} \tilde{\mathcal{L}}(\theta, \phi; x^{(i)}) &= \frac{1}{L}\sum_{l=1}^{L} log\big(p_\theta(x^{(i)}, z^{(i,l)})\big) - D_{KL}\big(q_\phi(z|x^{(i)})||p_\theta(z)\big) \\ z^{(i,l)} &= g_\phi(\epsilon^{(i,l)}, x^{(i)}), \qquad \epsilon^{(l)} \sim p(\epsilon) \end{aligned} \tag{2.9}$$

By using the SGVB estimator of the lower bound, we obtain low-variance gradient estimates, allowing us to use gradient based optimisation methods to optimise the ELBO w.r.t both the variational parameters ϕ and model parameters θ. Below, a pseudo code of the mini batch version of the AEVB algorithm is shown.



Algorithm 1 Mini-batch Auto-Encoding Variational Bayes

```
1: $\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Initial values
2: repeat
3:     $\boldsymbol{X}^m \leftarrow$ Random minibatch of size $m$ from $\boldsymbol{X}$
4:     $\boldsymbol{\varepsilon}^l \sim p(\boldsymbol{\varepsilon})$ for $l \in 1, ..., L$
5:     $\boldsymbol{g} \leftarrow \nabla_{(\boldsymbol{\theta},\boldsymbol{\phi})} \tilde{\mathscr{L}}_{SGVB}\left(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{X}^{(m)}\right)$
6:     $\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ parameter updates with $\boldsymbol{g}$
7: until Convergence of both $(\boldsymbol{\theta}, \boldsymbol{\phi})$
8: Return: $(\boldsymbol{\theta}, \boldsymbol{\phi})$
```

Figure 2.8: Pseudo code explaining the AEVB algorithm [15].

2.4.1 Variational Auto Encoder

The variational auto-encoder is an algorithm that uses neural networks to parameterise both the probabilistic encoder, $p_\theta(z|x)$, and the probabilistic decoder, $q_\phi(z|x)$, using the AEVB algorithm to optimise the parameters jointly.

We set the prior over the latent variables, $p_\theta(z)$, as a multivariate Gaussian with zero mean and unit variance, $\mathcal{N}(z; 0, I)$. The true posterior $p_\theta(z|x)$ is assumed to be approximately multivariate Gaussian with a diagonal covariance matrix. Assuming this is true, we can let the probabilistic encoder,

$q_\phi(z|x)$, be a multivariate Gaussian with a diagonal covariance matrix. We also assume the probabilistic decoder, $p_\theta(x|z)$, to be either a multivariate Gaussian or Bernoulli (depending on the data). Under these assumptions, we can define our model as:

$$\begin{aligned} p_\theta(x|z) &= \mathcal{N}(x; \mu_\theta(z), \sigma_\theta(z) \odot I) \\ p_\theta(z) &= \mathcal{N}(z; 0, I) \\ q_\phi(z|x) &= \mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x) \odot I) \end{aligned} \tag{2.10}$$

where $\mu_\phi(x)$ and $\sigma_\phi(x)$ are the outputs of a neural network, parameterised by ϕ and taking the observed variable (our data) as its input. Similarly, $\mu_\theta(z)$ and $\sigma_\theta(z)$ are also outputs of a neural network that is parameterised by θ, taking the latent variable z as its input.

Using the reparameterisation trick explained above, we can sample from the posterior distribution $z^{(i,l)} \sim q_\phi(z|x^{(i)})$ using $z^{(i,l)} = g_\phi(x^{(i)}, \epsilon^{(l)}) = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$ where $\epsilon^{(l)} \sim \mathcal{N}(0, I)$. As both distributions are Gaussian, the KL divergence can be computed and differentiated analytically. Therefore, we can derive the SGVB ELBO, which can be minimised using stochastic gradient optimisation techniques, where the derivatives w.r.t all parameters are obtained using backpropagation.

$$\begin{aligned} \mathcal{L}(\theta, \phi; x^{(i)}) \approx \frac{1}{2} \sum_{j=1}^{J} \Big(1 + log\big((\sigma_j^{(i)})^2\big) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\Big) \\ + \frac{1}{L} \sum_{l=1}^{L} log\big(p_\theta(x^{(i)}|z^{(i,l)})\big) \end{aligned} \tag{2.11}$$

$$z^{(i,l)} = g_\phi(x^{(i)}, \epsilon^{(l)}) = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}, \quad \epsilon^{(l)} \sim \mathcal{N}(0, I)$$

We note that the second term, $\frac{1}{L} \sum_{l=1}^{L} log\big(p_\theta(x^{(i)}|z^{(i,l)})\big)$, is simply the reconstruction error, in other words, how well the decoder is able to reconstruct the original input from the latent variables. It is therefore common to train the VAE with a mean squared error or sigmoid cross entropy objective function.

To summarise, for our application the VAE takes an image as an input, maps it to the mean and variance of its latent representation, which is assumed to be Gaussian distributed. It then draws a sample from that Gaussian distribution, and uses it as an input to the decoder, which is trained to reconstruct the original image given a latent vector, while keeping the KL divergence low. This forces the encoder to map the image to values that represent interpretable and informative generative features of the image. By successfully training this algorithm, we have modelled an approximation of the underlying data distribution:

$$P_\theta(x) = \int_z P_\theta(x, y) = \int_z P_\theta(x|z) P_\theta(z) \tag{2.12}$$

allowing us to generate new samples by drawing from the prior, $p_\theta(z)$, and then mapping it to a sample by using the trained decoder. Figure 2.9 shows an example of a VAE, using MLPs. It should be noted that CNNs can also be used, and will be in this thesis.

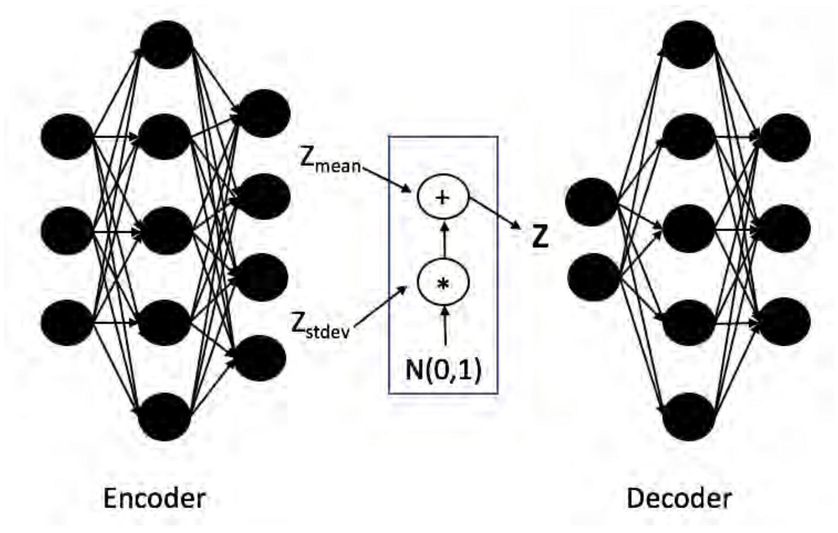


Figure 2.9: An example of a VAE, using one hidden layer for both the encoding and decoding networks.

2.4.2 β-VAE

β-VAE [3] is a variant of VAE, where a new parameter, β, is introduced to the SGVB, which is used to control how much to penalise the KL divergence term

$$\begin{aligned}\mathcal{L}(\theta, \phi; x^{(i)}) \approx \frac{\beta}{2} \sum_{j=1}^{J} \Big(1 + log\big((\sigma_j^{(i)})^2\big) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\Big) \\ + \frac{1}{L} \sum_{l=1}^{L} log\big(p_\theta(x^{(i)}|z^{(i,l)})\big)\end{aligned} \tag{2.13}$$

The authors of the method propose that if $q_\phi(z|x)$ is factorial, then the latent space that the VAE learns is disentangled, as the dimensions are independent of each other. Since $p_\theta(z)$ is factorial, increasing the penalty on the KL divergence should force the latent features to be more independent of each other, increasing the disentanglement in the latent space. Therefore, it is proposed that setting $\beta > 1$ encourages disentanglement. However, this extra pressure on the KL term can decrease the quality of the reconstruction, so a trade-off exists between disentanglement and reconstruction quality.

2.5 Generative Adverserial Networks

Generative adversarial networks (GANs) [9] are a powerful method that is capable of modelling the underlying data distribution without approximating the intractable encoding distribution $p_\theta(z|x)$. Instead, it proposes a method related to game theory that consists of two models:

1. The generative model, G, models the data distribution and can therefore generate data. It does so by defining a prior distribution over the latent variables, $p(z)$, and a neural network, $G(z; \theta_g)$, that maps the latent variable z to the data space.

2. The discriminative model, D, which is also represented by a neural network, $D(x;\theta_d)$, that computes the probability of whether a sample comes from the real data distribution or not.

These models are then simultaneously trained with the following objectives:

1. $D(x;\theta_d)$ is trained to discriminate between real and generated samples; *i.e.* assign high probability when given real samples as input, and assign low probability when given generated samples as input.
2. $G(z;\theta_g)$ is trained to fool the discriminator; *e.g.* generate images that are assigned high probability by the discriminator.

Intuitively, we can think of this as $D(x;\theta_d)$ and $G(z;\theta_g)$ playing a two-player minimax game with value function V(G,D):

$$\begin{aligned}\underset{G}{\text{Min}}\underset{D}{\text{Max}}V(D,G) = {} & \mathbb{E}_{x\sim p_{data}(x)}\big[log\big(D(x)\big)\big] \\ & + \mathbb{E}_{x\sim p_z(z)}\big[log\big(1-D(G(z))\big)\big]\end{aligned} \tag{2.14}$$

To optimise this objective, we alternate between steps of updating the discriminator, D, by maximising the objective, and one step of updating the generator, G, by minimising the objective. Since both D and G are neural networks, the entire model can be trained using gradient based optimisation methods. If D and G can represent any arbitrary function, we say that there exists a solution where Nash equilibrium is reached. For this solution, G has captured the underlying data distribution and therefore the discriminator cannot possibly discriminate between real and generated data, always outputting 0.5. Figure 2.10 shows the structure of a GAN.

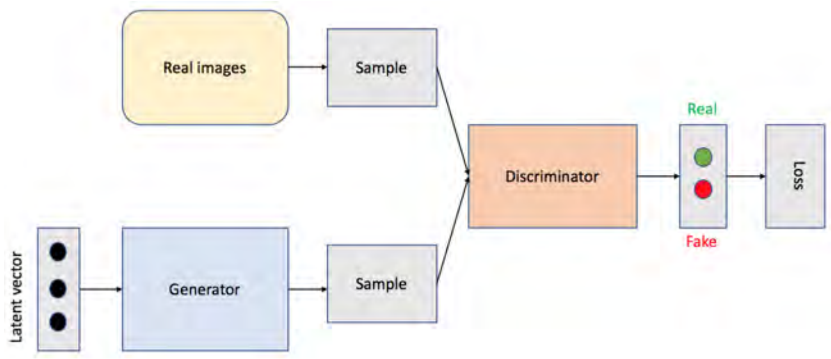


Figure 2.10: Explanation of how generative adverserial networks are structured

GANs have shown great promise in capturing underlying data distributions, especially for images, generating images of higher quality and detail than VAEs. However, GANs do not provide any efficient ways of mapping the data to the latent space, as they do not model $p_\theta(z|x)$. Methods have been developed to do this, and the techniques used in this thesis are explained below.

2.5.1 Stable GAN training

Training GANs is known to be a very unstable procedure, and whether a GAN will train successfully depends on various things; *e.g* choice of architecture, hyper-parameter setting and parameter initialisation. One of the reasons for this is that the support of the generative distribution G, and the real data distribution P, can be non-overlapping. A proposed solution to this problem is to add Gaussian noise to both generative and real samples, to ensure that the supports of the distributions overlap [2]. A problem with this method is that introducing high dimensional noise introduces variance when estimating the parameters. Kevin Roth et al. [24] propose an analytical approach, where both distributions G and P are convolved with Gaussian noise, which results in a weighted penalty function on the norm of the gradients, which is added to the loss function of the discriminator. This was used in all

GANs trained for this project.

$$\omega(P, G; D) = \mathbb{E}_P \left[(1 - D(x))^2 ||\nabla_\phi(x)||^2\right] + \mathbb{E}_Q \left[D(x)^2 ||\nabla_\phi(x)||^2\right] \tag{2.15}$$

2.5.2 infoGAN

InfoGAN [6] is a variant of GANs, where the idea is to provide disentangled latent space by decomposing the latent vector into:

1. The traditional input vector, z, simply considered as noise.
2. A new latent vector, c. This is the part that contains meaningful, disentangled representations, which is achieved by maximising the mutual information between c and the output of the generator.

The mutual information is maximised by introducing a new term to the objective function, which encourages maximum mutual information.

$$\underset{G}{\mathrm{Min}}\underset{D}{\mathrm{Max}} V_I(D, G) = V(D, G) - \lambda I(c; G(z, c)) \tag{2.16}$$

In practice, the mutual information term is hard to maximise, as it requires access to the intractable posterior distribution P(c|x). Using variational information maximisation, we can derive a lower boundary on the mutual information term by defining an auxiliary distribution Q(c|x) as an approximation of P(c|x).

$$\begin{aligned} L_I(c; G(z, c)) &= H(c) - H(c|G(z, c)) \\ &= \mathbb{E}_{x\sim G(z,c)} \left[\mathbb{E}_{c'\sim P(c|x)} \left[log\big(P(c'|x)\big)\right]\right] + H(c) \\ &= \mathbb{E}_{x\sim G(z,c)} \big[D_{KL}\big(P(\cdot|x)||Q(\cdot|x)\big) \\ &+ \mathbb{E}_{c'\sim P(c|x)} \left[log\big(Q(c'|x)\big)\right]\big] + H(c) \\ &\geq \mathbb{E}_{x\sim G(z,c)} \left[\mathbb{E}_{c'\sim P(c|x)} \left[log\big(Q(c'|x)\big)\right]\right] + H(c) \\ &= \lambda L_I(G, Q) \end{aligned} \tag{2.17}$$

This term can now be approximated using a Monte Carlo simulation. It can be optimised directly w.r.t Q and by using the reparametrisation trick w.r.t G. The auxiliary distribution Q is parametrised using a neural network. Typically, it shares all layers with the discriminator network, D, except for a fully connected layer at the end, which generates the parameters of the distribution. Using this setup, the computational cost added by this distribution is negligible. Furthermore, by successfully training $Q(c|x)$, we obtain a distribution that maps the data points to their latent vectors. The training objective now becomes:

$$\underset{G}{\mathrm{Min}}\,\underset{D}{\mathrm{Max}} V_I(D,G) = V(D,G) - \lambda L_I(G,Q) \tag{2.18}$$

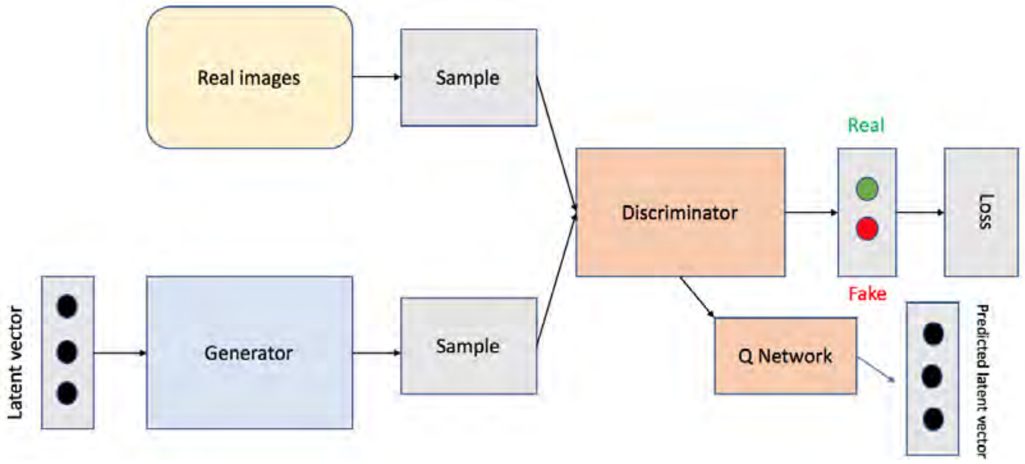


Figure 2.11: Demonstration of how the InfoGAN works

2.6 Related Work

After a thorough literature search, we found no published work where machine learning was used for autophagy research. However, a fair amount of work has been done on microscopy image segmentation. Although segmentation is not done in this thesis, it is worth mentioning the U-net architecture, introduced by Ronneberger et al. [23], that achieved great performance in microscopy imaging segmentation. It consists of an contracting path, which is paired with an expanding path, resulting in a U shaped architecture. It also passes feature maps from layers in the contracting path to their analogous layers in the expanding path, similarly to ResNet [11].

Interesting work has also been done using machine learning on the BBBC021v1 dataset [4], that consists of microscopy cell images and their corresponding labels that represent the effect a drug has on the cells. Pawlowski et al. [21] introduced the use of deep CNNs that are pre-trained on ImageNet images to extract features in the images that are informative with regards to the effect a drug has. Ando et al. [1] did the same things using different architectures as well as introducing a normalisation technique for these features. In terms of classification of the dataset, Kraus et al. [16] used convolutional multiple instance learning to distinguish between the cells.

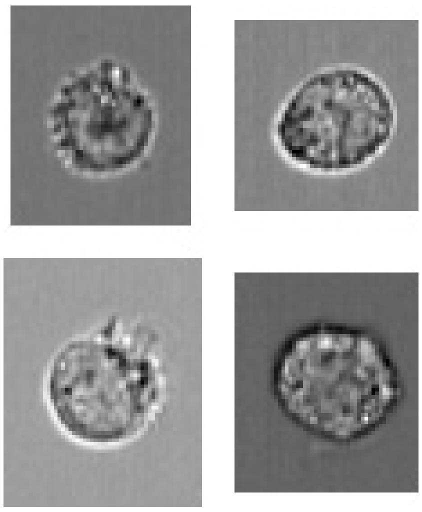
Zamparo et al. [26] investigated dimensionality reduction techniques on cell images before classifying them using support vector machine in the lower dimensional space. In terms of generative modelling, Osokin et al. [19] successfully applied GANs to generate cell images. Most relevant to our work, Goldsborough et al. [8] recently introduced the CytoGAN, where they compare the use of three different GANs to generate cell images, and then investigate how meaningful the features learned by the GAN are. However, their work is limited, as they do not incorporate any method to encourage disentanglement in the latent space. To subtract features from the network, they use the penultimate layer of the discriminator, without any mutual information maximisation. This leads to poor informative features, as their results show. Also, their work is focused around cancer cells, and these cell images differ considerably from ours, both in appearance and with regards to where important information lies in the cell.

Chapter 3

The dataset and preprocessing steps

The cell images contain two channels. The first channel is a bright-field microscopy image of the cell, and the second is the green HLA-FITC fluorescence channel. The important thing to know about the channels is that the first channel represents the shape and appearance of the cell. Currently, this channel is not used in autophagy research, as it is not known to carry any information about the process that is visible to the human eye. Therefore, it is interesting to see if a machine learning algorithm can find information in this channel. The second channel shows localisation of components (lysosomes) inside the cells. This channel contains valuable information, as the localisation should be different for cells where different processes are happening; *e.g.* autophagy. However, this difference is not always clear, and our aim is to find the complex patterns in these channels that suggest certain processes are happening.

Preprocessing of the dataset refers to every transformation of the data which takes place before using it in the machine learning algorithm. Preprocessing is a very important step, and is usually necessary before the machine learning algorithm can learn anything valuable. This chapter describes all steps taken to process the raw cell images generated by the imaging flow cytometer. Figure 3.1 shows an example of cell images for both channels before any preprocessing was done.


(a) First channel

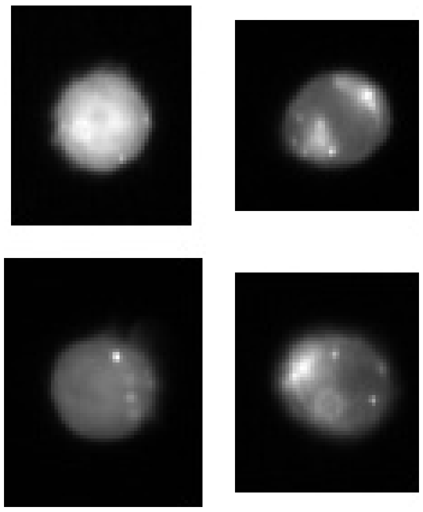

(b) Second channel

Figure 3.1: Sample images from the first and second channel, before doing any preprocessing.

3.1 Fixing the size of the images

The raw image dataset contains images of different sizes, which is not suitable for machine learning algorithms like neural networks, where the input vector needs to be a fixed size. Therefore, a fixed size had to be determined for the images, and then the sizes of the individual images had to be either reduced or enlarged to fit that size. In order to fix the size of the first channel, the following steps were computed:

1. Read all images into Python, and determine a suitable size for the images.
2. Compute the value of the pixels for all four edges of the image, and then calculate the standard deviation and mean of these pixels.

3. Compute an array of same size as the images are, that consists of normally distributed values, using the computed mean and standard deviation.
4. If either the columns or rows are too small, they are padded using the normally distributed pixels.
5. If either the columns or rows are too large, the images are cropped.

The same steps were performed for the second channel, except that instead of being padded with normally distributed pixels, the images were padded using a constant of 20. The reason for this is that when analysing the data, we noticed that when the sensors detected no fluorescence, the pixels had a value of 20.

3.2 Normalising pixel values

Normalising the pixel values to be distributed similarly is very important before feeding them into neural networks. If the inputs are on a different scale, the cost surface will be very steep in certain directions and shallow in others, which makes convergence for optimisation algorithms slow [17]. A simple fix for this is to subtract the mean of the input and divide by the standard deviation.

$$x_{normalised} = \frac{x - \mu_x}{\sigma_x} \tag{3.1}$$

For many applications, it is best to have every value between 0 and 1. This is true for us, since VAEs use cross entropy loss function, with the original input as target values. By definition, the pixel values therefore need to be between 0 and 1. An easy way to do this is by calculating the maximum and minimum pixel values, and normalise according to equation 3.2

$$x_{normalised} = \frac{x - min(x)}{max(x) - min(x)} \tag{3.2}$$

However, as our dataset contains problematic features that are not solved using the normalisation techniques described, more complicated methods had to be used, which we will discuss in the subsections below.

3.2.1 Normalising pixel values for the first channel

To normalise the first channel, a noticeable problem was that the background pixels of different images had different values. This was not ideal, as we wanted all images to have the same background colour, so the focus of the algorithm is on the cell and not the background. To make all backgrounds approximately the same colour, and normalise all pixels values between 0 and 1, the following steps were taken:

1. Compute the standard deviation for every pixel position over the entire dataset. Results in an array of same size as the images are, containing the standard deviation for every pixel position.

2. Find the indices of the standard deviation array where values are over some threshold. Positions with low standard deviation will correspond to positions where background values are usually located, high standard deviation to where cells are located.

3. Use the indices to create two mask arrays, one for the background images and another for the cells.

4. For every image, we used the mask array for background images to compute the median pixel value of the background, and subtract this value from every pixel. This should make each background value approximately 0.

5. Use the cell mask array to compute the standard deviation of the cell pixels, and divide each pixel by that value.

6. Clip the values so that they range from -3 to 3, removing outliers.

7. Normalise all pixel values to be between 0 and 1 by adding 6 to every pixel value and dividing by 3.

Figure 3.2 shows an example of 56 cell images after being normalised as described above.

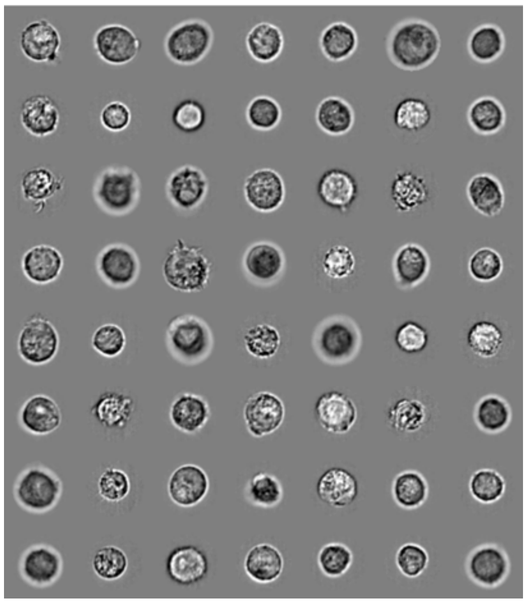


Figure 3.2: Sample images from the first channel after normalisation

3.2.2 Normalising pixel values for second channel

For the second channel, it is important to notice that when the flow cytometer detects something that expresses luminance, it is exponentially brighter than the pixels around it. By taking the log of the pixel values across the whole dataset, we can plot a histogram of the pixels and notice that the values appear to form a mixture of three normal distributions, one of which corresponds to background pixels, one for normal cell content and one for the dots distributed in the cell, see Figure 3.3.

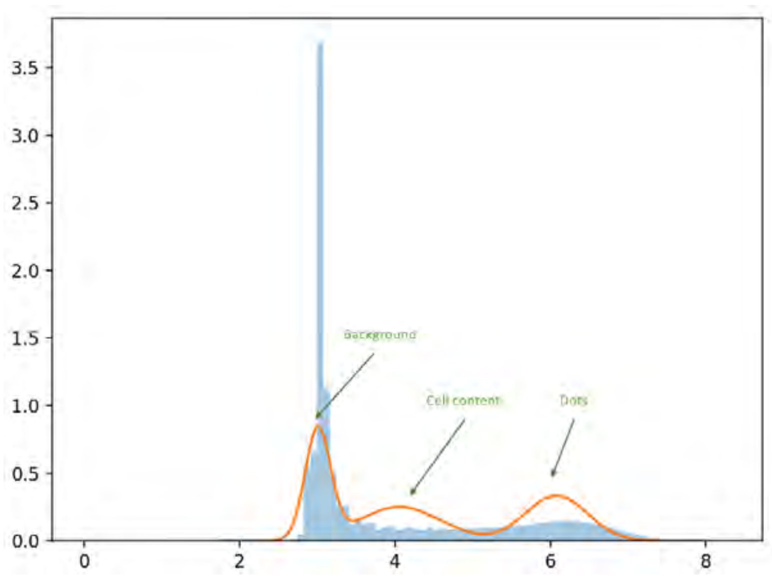


Figure 3.3: Histogram of the log pixel values and the three component Gaussian mixture model fitted to the pixels.

We would like to scale the images by the mean and standard deviation of the Gaussian corresponding to the dots pixels, and then omit all outliers, which will mainly be the background. Then the only non-zero pixel values will be the bioluminescent pixels. This was done the following way:

1. Take the logarithm of the pixel values for the whole dataset.
2. Fit a 3-component Gaussian mixture model to all pixels in the dataset.
3. Subtract the mean of the dots distribution from all log pixel values in the dataset.
4. Divide all log pixel values by the standard deviation of the same Gaussian.
5. Clip the values so that they range from -3 to 3
6. Normalise all pixel values to be between 0 and 1, using equation 3.2.

Figure 3.4 shows an example of 56 cell images after being normalised as described above.

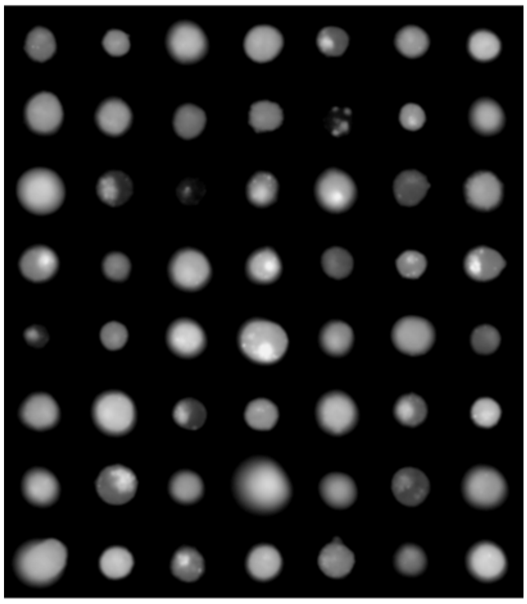


Figure 3.4: Sample images from the second channel after normalisation with log pixel values.

Chapter 4

Cell Filtering

To gather the dataset, an imaging flow cytometer was used. However, some of the images that it outputs are not suitable for research, as they contain clumps of cells or even objects that are not cells. Therefore, the dataset had to be filtered before it was used for our purpose. In order to do this, a discriminative binary classifier was trained, that filtered the images based on whether they were suitable for research or not. In order to train the classifier, a subset of 30,000 images was hand-labelled, since all the data was gathered by us and to our best knowledge, no similarly labelled dataset exists. After labelling the images, it was divided into training, validation and test sets.

The classifier was trained using only the first channel. The reason for this is that the first channel images do not differ significantly for cells with different processes, while the second channel does. By including the second channel, we risk of overfitting to our dataset. The goal was to create a robust method to filter these cells, independent from processes that are happening inside them or how they were prepared beforehand.

4.1 Model architecture

A simple convolutional neural network (CNN) was trained as the binary classifier to distinguish between the images. The network architecture is illustrated in Figure 4.1. It consists of two 3x3 convolutional layers, with 20 and 50 filters respectively, using 2x2 max pooling with stride 2 after each layer. The convolutional layers were followed by a fully connected layer with 500 neurons, followed by the output layer which generated the probability of the sample being positive or negative. The RELU non-linearity was used as an activation function for every layer, except for the output layer, where the sigmoid function was used. To train the network, the Adam optimisation algorithm [14] was used to minimise the sigmoid cross entropy loss function. The hyperparameters of the network were tuned using grid search, obtaining best performances using keep probability of 0.8 for dropout, batch size 50 and a learning rate of $1e^{-4}$. The number of epochs was set to 85.

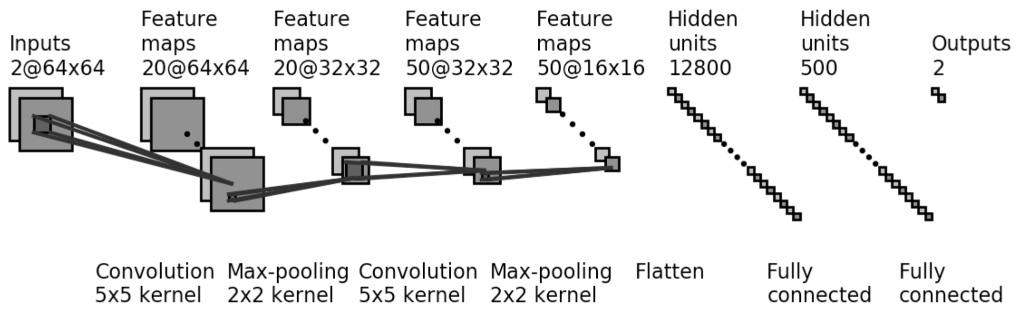
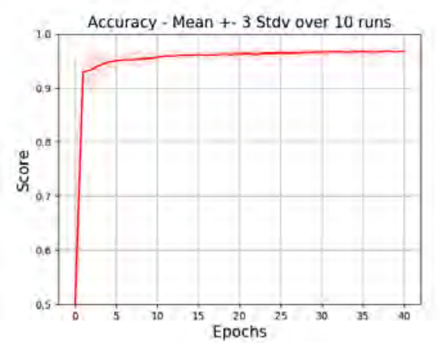
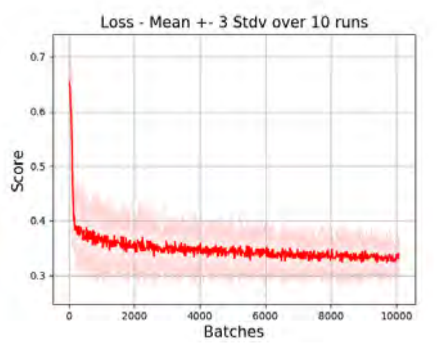


Figure 4.1: Explanation of the CNN architecture used for the filtering of cell images

4.2 Results and discussion



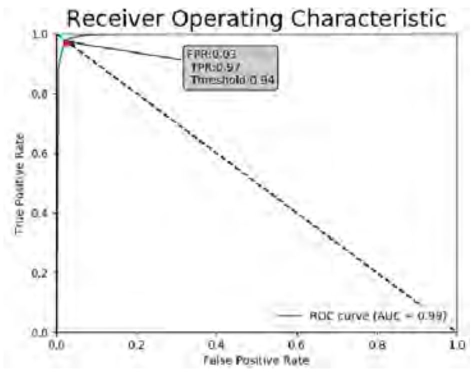
(a) Validation accuracy



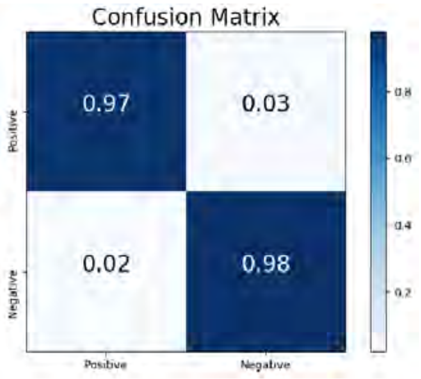
(b) Training loss

Figure 4.2: Average training loss plotted against number of batches, and average validation accuracy plotted against number of epochs for the binary classifier over ten different runs.

The average training loss plotted against the number of batches, and the average validation accuracy plotted against the number of epochs, for 10 different runs, are shown in Figure 4.2. The final test accuracy was 97%, showing us that this simple CNN can filter useful cell images from unusable ones almost perfectly. For comparison, a logistic regression algorithm was also trained, to see how a linear classifier performed with the dataset. The final accuracy was 91%, indicating that the dataset is quite close to being linearly separable. To ensure that the classifier was robust and not only useful for our dataset, a hold-out dataset was prepared, using cells that were processed differently. The dataset was not labelled, but by visually inspecting over 2000 cell images filtered by the classifier, we concluded that it was doing a good job on this dataset, almost only outputting single cell images. Figure 4.3 shows the confusion matrix (classification threshold of 0.5) and ROC curve for the CNN.



(a) ROC curve



(b) Confusion matrix

Figure 4.3: ROC curve and confusion matrix for the binary classifier using the test set.

The most important thing is that the false positive rate was low, as we would rather reject more good samples than accept bad samples and use them for research. The confusion matrix shows that the false positive rate was 3%, which is acceptable. This could be reduced by increasing the threshold of the classifier. However, since the neural network generates very confident predictions, and we see on the ROC break-even point that a threshold of 0.94 would still lead to a false positive rate of 3%, this was not done.

A common practice for biology researchers when working with this kind of data is to filter the cell images based on the size of the image. The method consists of setting three different constraints for the images: maximum size for row and columns, minimum size for row and columns, and a maximum for the ratio between the row and column. The argument for this is that single cell images should approximately have the same size, with images not suitable for research differing in both size and dimension ratio. These three constraints were tuned using a grid-search for every possible combination, yielding a best accuracy of 91.17%. This would not be acceptable for our research, and shows how machine learning can immediately improve quality of research in this area. According to our best knowledge, our model achieves better results than any method currently being used for this task.

Chapter 5

Discriminative Model

This chapter describes the discriminative model that was trained in order to see if it could find patterns in cell images that are known to have different processes happening inside them. Three different procedures were performed when preparing the cells used in this project. Firstly, we used cells that were given the drug monencin, which is known as an autophagy inhibitor. Secondly, we used cells that were given the drug doxorubicin, which is known as an autophagy inducer. Finally, we used cells that were not given any drugs, which we refer to as negative. Then, we also used cells from mutants, for which the same procedures were followed. We refer to the mutants as A1A2, and the normal cells as wild-type (WT). Therefore, in total we have 6 different type of cells: WTdox, WTmon, WTneg, A1A2dox, A1A2mon and A1A2neg.

As discussed above, before beginning the experiment, our understanding is that the second channel contains all information with regards to the processes happening, as it shows how components inside the cells are localised. Our prior knowledge about this localisation for the wild type cells was the following:

1. WTdox: The components should be grouped together at certain places inside the cells, appearing as dots in the cell images.

2. WTmon: Components should also be grouped together, but the distribution of dots should different from that in WTdox.

3. WTneg: The components should be spread all over, with very few dots showing in the cell images.

The behaviour of the mutants was less understood, and it was interesting to see if they would react differently to the drugs. The following things were expected before conducting the experiments:

1. A1A2dox: The components should be grouped together, forming dots, to a certain extent. Expect much fewer dots than for WTdox.

2. A1A2mon: Unknown behaviour, and therefore exciting with regards to our experiments. Components will most likely spread throughout the cell, similarly to WTmon.

3. A1A2neg: Components should be spread all over, showing very few dots in the cell images, similarly to WTneg.

5.1 Model architecture

A convolutional neural network (CNN) was trained as the discriminative model to distinguish between the different cell types. The network architecture is illustrated in figure 5.1. It consists of three 3x3 convolutional layers, with 16, 32 and 64 filters respectively, using 2x2 max pooling with stride 2 for each layer. The convolutional layers were followed by a fully connected layer with 512 neurons, followed by the output layer which generates the probability of each class, using the softmax activation function. The RELU non-linearity was used to activate all the other layers. Every layer was followed by batch normalisation, except for the output layer. To train the network, the Adam optimisation algorithm was used to minimise the softmax cross entropy loss function. Different architectures were not tested, and no real tuning of hyperparameters was conducted, both because of time and computing constraints, and also since the discriminator was only thought of as a way to ensure that the cells had distinguishable features, and to see which cell types behaved similarly. The networks used in the results reported used a keep probability of 0.8 for the dropout in the fully connected layer, a batch size of 100 and a learning rate of $1e^{-4}$. The number of epochs was set to 85.

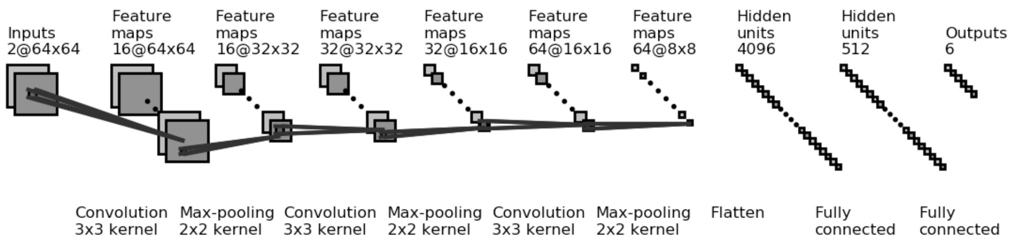
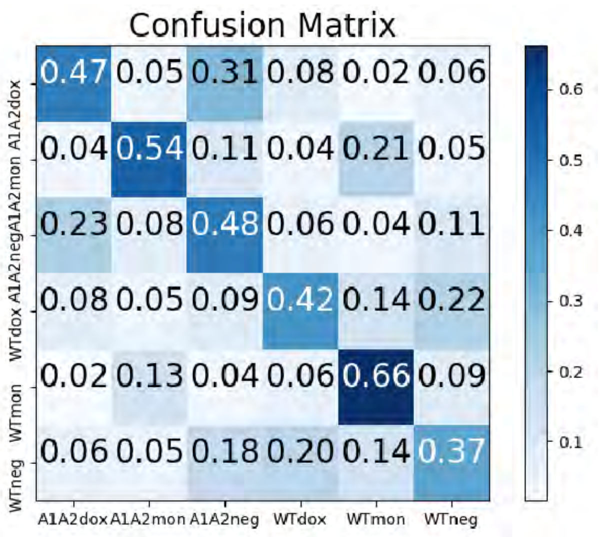


Figure 5.1: Explanation of the CNN architecture used for the discriminative model.

5.2 Results and discussion

5.2.1 Results training on the first channel

The final test accuracy was 49% for the 6 classes, showing that the CNN is able to find distinguishable features inside the cells using only the first channel. These results are interesting, as first channel images are currently not used for research purposes. Figure 5.2 shows the confusion matrix for the classifier.



	A1A2dox	A1A2mon	A1A2neg	WTdox	WTmon	WTneg
A1A2dox	0.47	0.05	0.31	0.08	0.02	0.06
A1A2mon	0.04	0.54	0.11	0.04	0.21	0.05
A1A2neg	0.23	0.08	0.48	0.06	0.04	0.11
WTdox	0.08	0.05	0.09	0.42	0.14	0.22
WTmon	0.02	0.13	0.04	0.06	0.66	0.09
WTneg	0.06	0.05	0.18	0.20	0.14	0.37

Figure 5.2: Confusion matrix for the discriminative model trained on the first channel.

5.2.2 Results training on the second channel

The final test accuracy was 62% for the 6 classes using only the second channel, confirming that the second channel contains more information than the first one with regards to the cell types. Figure 5.3 shows the confusion matrix for the classifier.

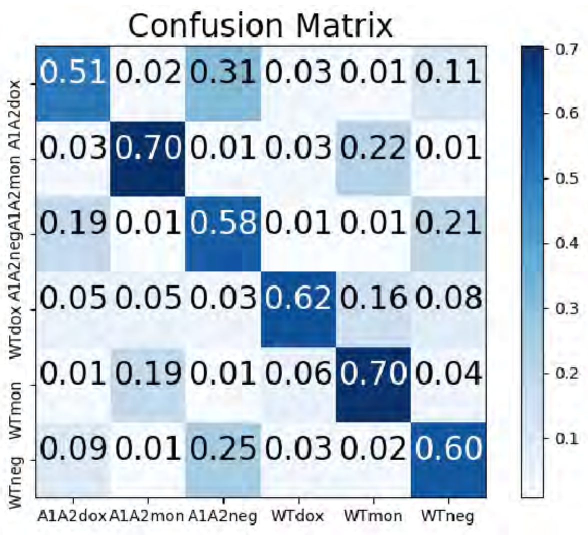


Figure 5.3: Confusion matrix for the discriminative model trained on the second channel.

5.2.3 Results training on both channels

The final test accuracy was 66% for the 6 classes using both channels, outperforming both classifiers that only trained on a single channel, indicating that there is information in the first channel that is not in the second channel. Figure 5.4 shows the confusion matrix for the classifier.

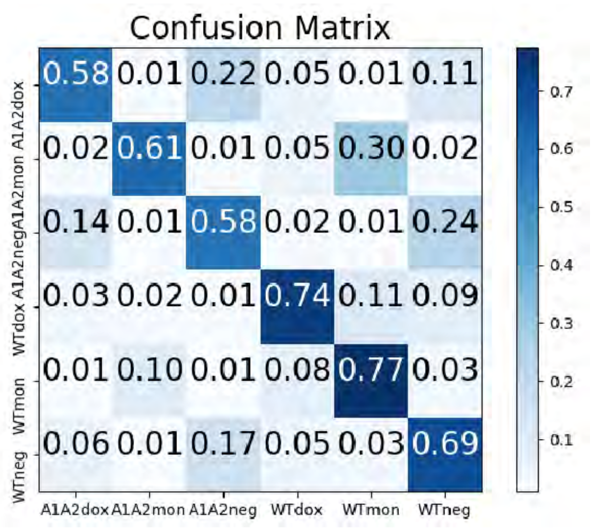


Figure 5.4: Confusion matrix for the discriminative model trained on both channels

5.2.4 Comparison and discussion

The results above show us that a CNN is able to find distinguishable features inside the cells with regards to processes happening inside of them. The confusion matrices for the classifiers are very similar with regards to misclassification, only differing in accuracy.

By analysing the confusion matrices, a number of interesting things about the cells can be determined. First of all, the mutant cells seemed to respond completely differently to doxorubicin than the wild-type cells did. We expected components to localise and form dots similarly to WTdox, but with lower number of dots. However, the algorithm seems to mainly misclassify these cells as negative, indicating that the components inside the cells did not localise as expected. Secondly, we can see that the algorithm did a very good job of classifying the monencin cells, both for the mutants and the wildtypes. It mainly misclassified A1A2mon as WTmon and vice versa, indicating that the mutant behaves similarly to the drug. Thirdly, we can see that for both mutants and wild-types, the algorithm seems to recognise negative cells well, mainly confusing them with A1A2dox, further supporting our statement that components inside A1A2dox did not localise as expected.

An interesting result from biological perspective was to see that the classifier was able to find information using only the first channel, since this channel is usually not considered for autophagy analyses. However, this should not be to surprising, as the second channel should be a lossy function of the first to some extent, and what is more interesting is to analyse how much information there is in the first channel that is not in the second channel. The fact that using both channels yields better accuracy than using only the second channel indicates that there is some additional information in the first channel. An interesting experiment would be to compute the mutual information between the two channels, in order to see how much information there is in the first channel that is not in the second. Another interesting experiment to address this question of mutual information from another angle is to train a variational auto encoder with the U-net architecture [23], to translate from the first channel to the second channel. However, due to time constraints, we leave this to future work.

Chapter 6

Generative Modelling

This chapter describes the two generative latent variable models that were trained on the cell images, $\beta-$VAE and infoGAN, and the experiments conducted using these models. If we can model a disentangled latent space by training on cell images where different processes are happening, latent dimensions should represent features that are different for cells which have different processes happening in them. For example, it is known that the localisation of components inside cells is different for cells where autophagy is happening and for those where it is not. Therefore, a dimension that represents this localisation would contain valuable information about our cells, and the latent space mapping of that dimension should be different for different cell types. Modelling this latent space allows us to discover interesting features of the cells, gives us a robust way of quantifying these features, and most importantly has the potential to help researchers better understand processes like autophagy. For all the models, the same dataset was used as in Chapter 5.

6.1 Model Architectures

Scaling GANs using CNN architectures that are common in the field of computer vision often leads to difficulties, resulting in unstable training. Radford et al. [22] proposed a solution to this with the DCGAN, where they came up with 5 simple architecture guidelines for GANs using deep CNNs:

1. For the discriminator, use convolution with a stride larger than 1 instead of pooling layers, and for the generator, use transposed convolution.
2. Use batch normalisation after every convolutional layer for both the generator and the discriminator.
3. Remove all fully connected layers.
4. In the generator, use RELU non-linearity, except for the output layer.
5. In the discriminator, use leaky-RELU, except for the output layer.

These guidelines were followed when designing the infoGAN architecture; see Figures 6.1 (generative network) and 6.2 (discriminative network). The auxiliary distribution Q(c|x) was designed to share all layers with the discriminative network, except for a fully connected layer at the end, which generated the prediction of the latent code c.

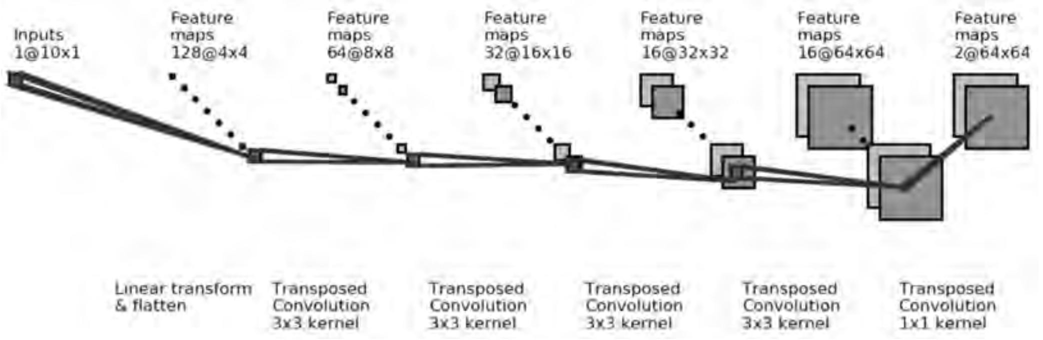

Inputs 1@10x1
Feature maps 128@4x4
Feature maps 64@8x8
Feature maps 32@16x16
Feature maps 16@32x32
Feature maps 16@64x64
Feature maps 2@64x64
Linear transform & flatten
Transposed Convolution 3x3 kernel
Transposed Convolution 3x3 kernel
Transposed Convolution 3x3 kernel
Transposed Convolution 3x3 kernel
Transposed Convolution 1x1 kernel

Figure 6.1: Explanation of the CNN architecture used for the generator.

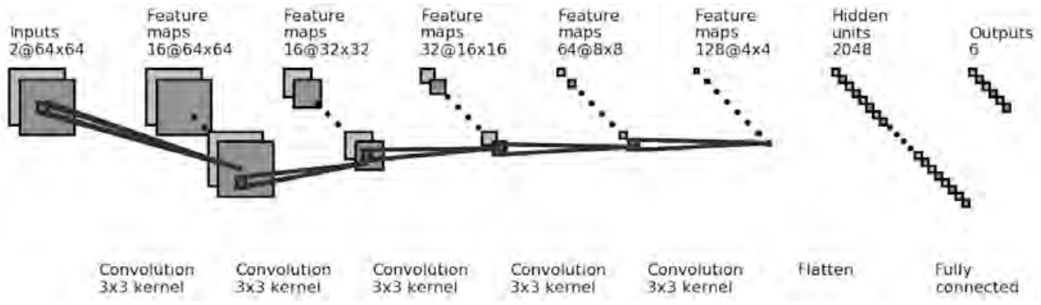


Figure 6.2: Explanation of the CNN architecture used for the discriminator.

For fair comparison, and since the GAN was reconstructing the images effectively, a similar architecture was used for the $\beta-$VAE, with the encoder using the same architecture as the discriminator and the decoder using the same architecture as the generator.

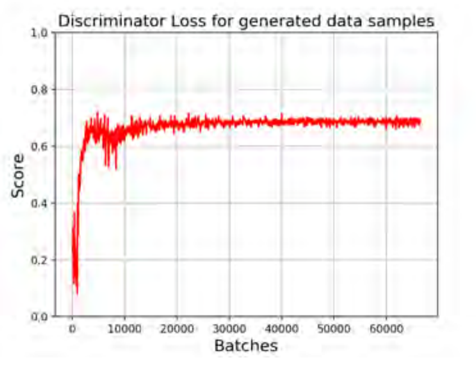
Tuning the hyperparameters for these models is challenging, since quantifying disentanglement is a difficult and ongoing research problem. Furthermore, these models are computationally expensive and take a long time to train, making it difficult to carry out many experiments. For the InfoGAN, all reported experiments used a 10 dimensional latent vector, while the noise vector was set to be 1 dimensional. This forced the network to generate images mainly from latent values that contain information about the generative features. However, as the dimension of the noise vector is reduced, the outputs become more deterministic, limiting stochastic behaviour in the generating process, and if we excluded noise, the GAN would fail to model any distribution except for the delta function [12]. For our experiments, we found that using a 1 dimensional noise vector worked well. All latent variables were set to take continuous, uniformly distributed values, due to the fact that discrete latent features would not be interesting for our problem. For the $\beta-$VAE, the dimension of the latent vector and the value of β were tuned together using grid search, since the optimal value of β depends on the size of the latent vector. Best results were obtained using 32 dimensional noise vector for all experiments, but the value of β varied.

For both models, all experiments were conducted using the Adam optimisation algorithm to optimise the objective function, with a learning rate of $2e^{-4}$ and a batch size of 100. For the fully connected layer in the auxiliary distribution of the InfoGAN, a keep probability of 0.75 was used for dropout. The number of epochs was set to 150 for the InfoGAN, and 80 for the $\beta-$VAE.

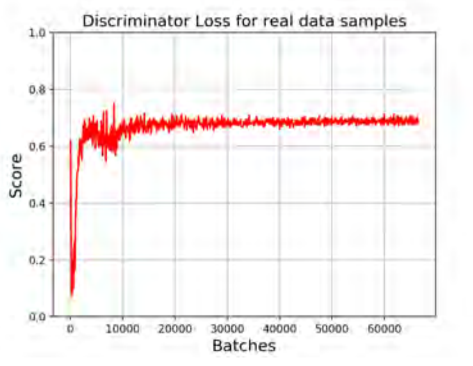
6.2 InfoGAN - Results and discussion

In this section, we present results from the InfoGAN model when trained on only the first channel, only the second channel, and both channels. First, we present figures that show the different training losses, plotted against number of batches, demonstrating that the algorithm trained successfully. Secondly, randomly generated images from the models are compared with the original images, to validate that the model has captured the data distribution, and generates realistic images. Thirdly, to ensure that the latent space contains informative generating features, images are mapped to their latent vector using Q(c|x), before reconstructing them using the generative model and comparing them. Finally, we compare how different cell types are mapped to the latent space by computing kernel density plots (Gaussian kernel) of all cell types, for each latent dimension. These plots confirm that some of the generative features that the latent space represents are connected to the cell type, and that the InfoGAN has found distinguishable features and patterns for cells where different processes were happening. The section is then concluded with discussion and comparison of these three different models. It should be noted that pixel values of all second channel images shown are log values.

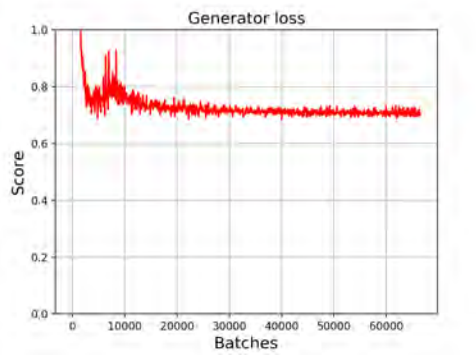
6.2.1 Training on first channel



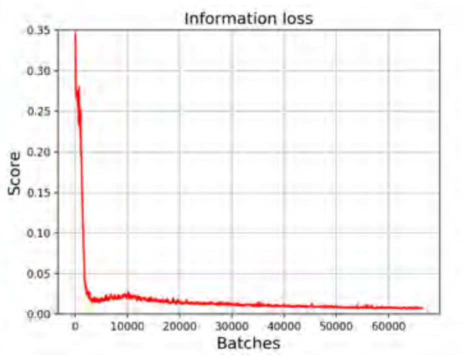
(a) Discriminator loss - fake images



(b) Discriminator loss - real images

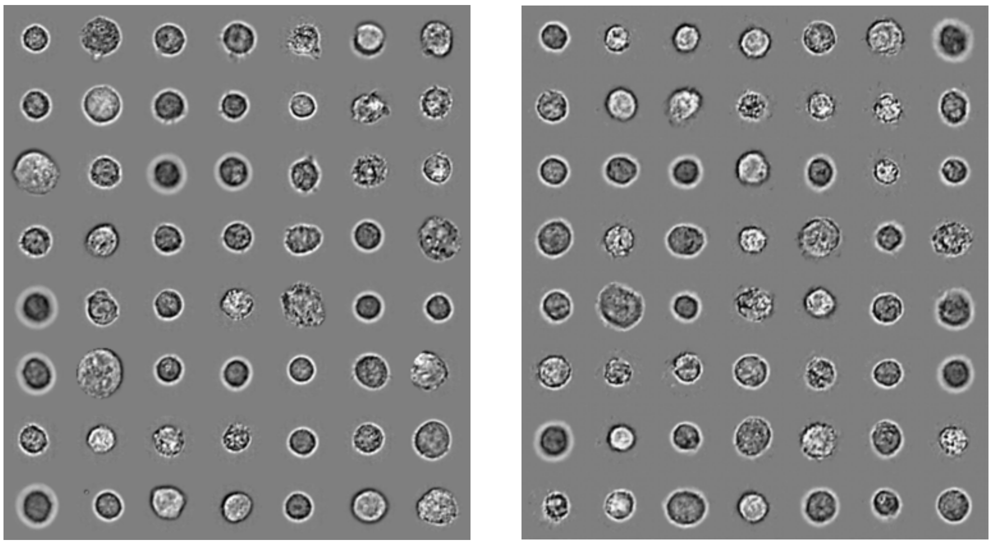


(c) Generator loss



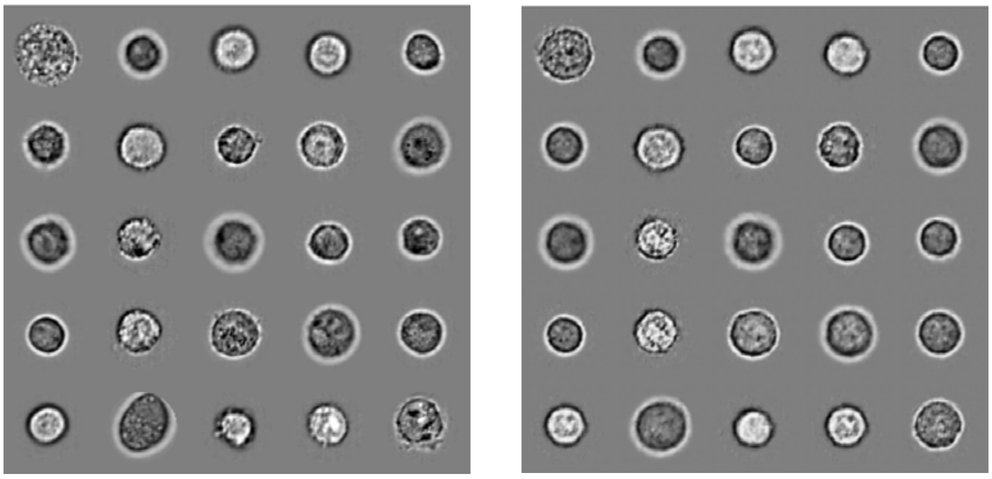
(d) Information loss

Figure 6.3: First channel - Training losses for the InfoGAN plotted agains number of batches.



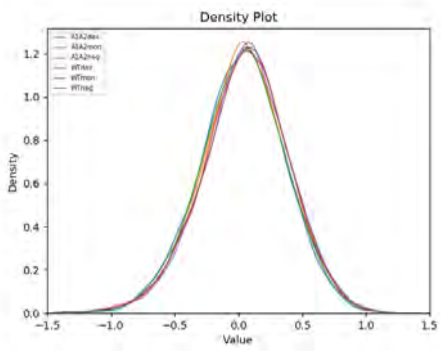
(a) Original images (b) Generated images

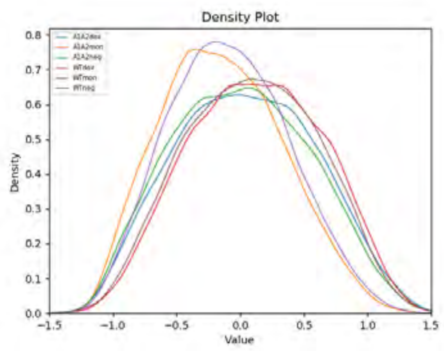
Figure 6.4: Comparison between original and generated images for the Info-GAN, trained on the first channel.

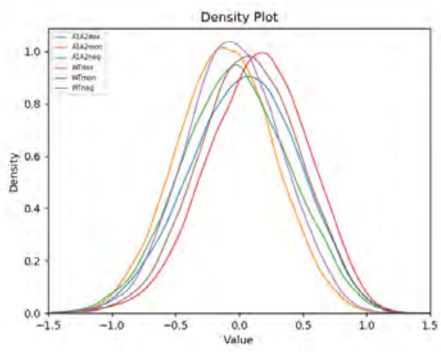


(a) Original images (b) Reconstructed images

Figure 6.5: Comparison between original images and their corresponding reconstruction for the InfoGAN, trained on the first channel.


(a) Dimension 4


(b) Dimension 5


(c) Dimension 6

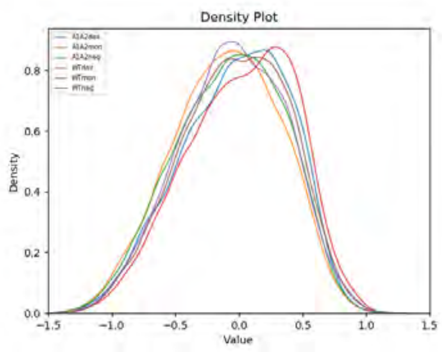
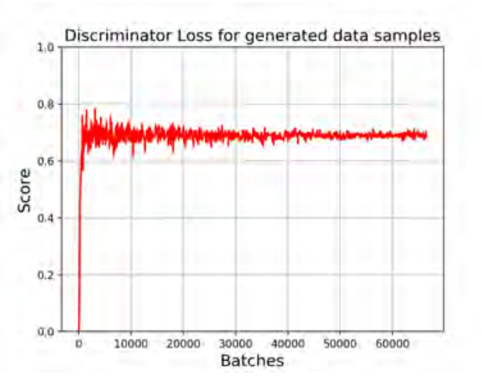
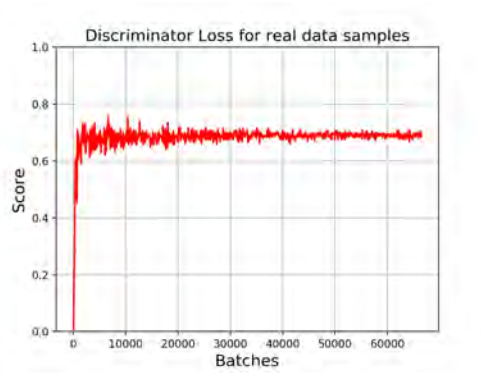

(d) Dimension 9

Figure 6.6: Comparison of density plots for different dimensions of the latent space modelled by the InfoGAN, trained on the first channel.

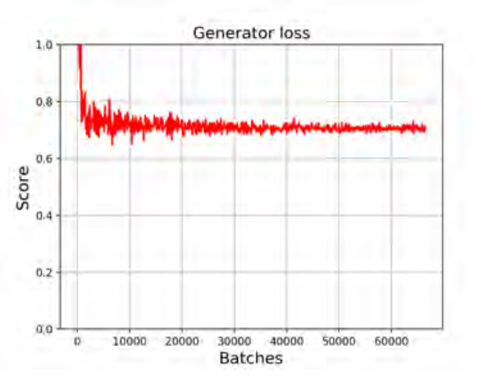
6.2.2 Training on second channel



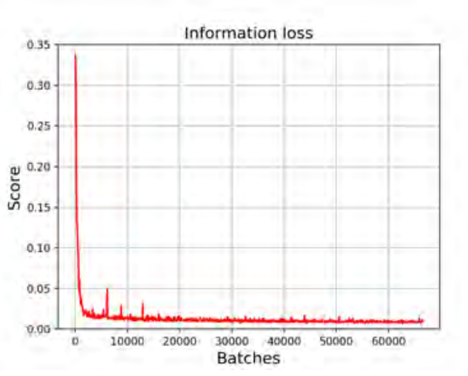
(a) Discriminator loss - fake images



(b) Discriminator loss - real images

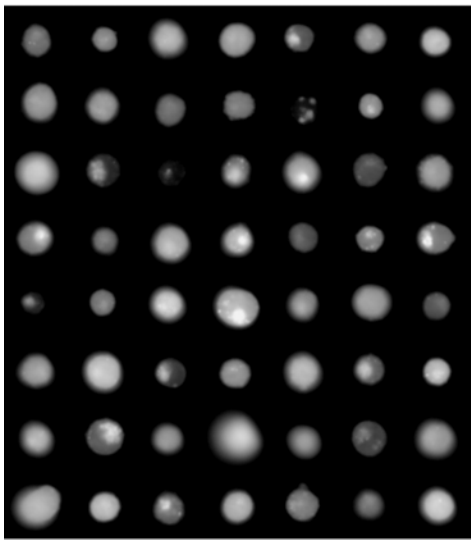


(c) Generator loss



(d) Information loss

Figure 6.7: Second channel - Training losses for the InfoGAN plotted against number of batches.


(a) Original images

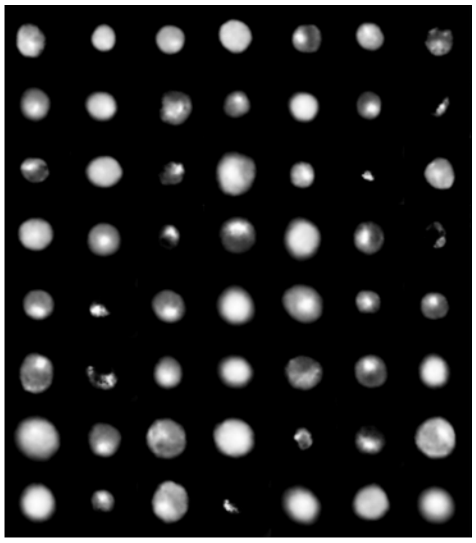
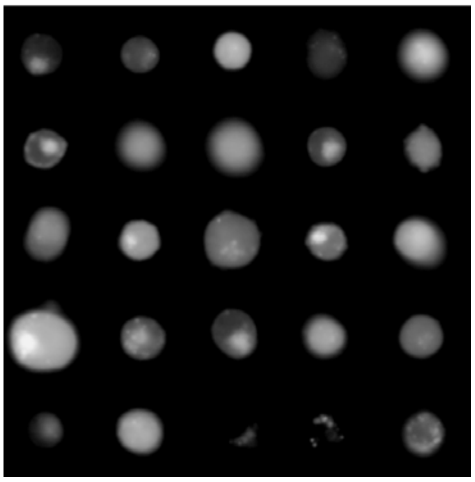

(b) Generated images

Figure 6.8: Comparison between original and generated images for the InfoGAN, trained on the second channel.


(a) Original images

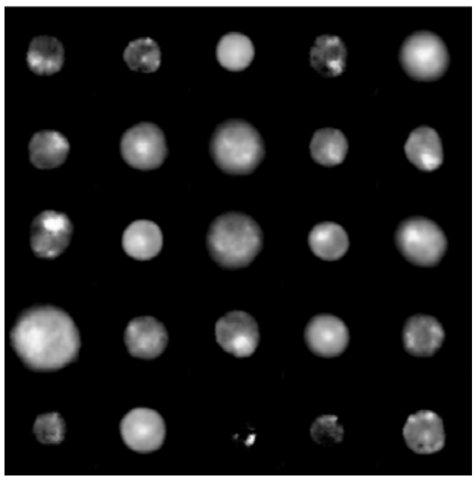
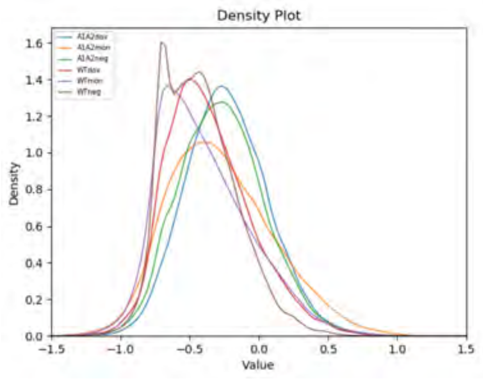
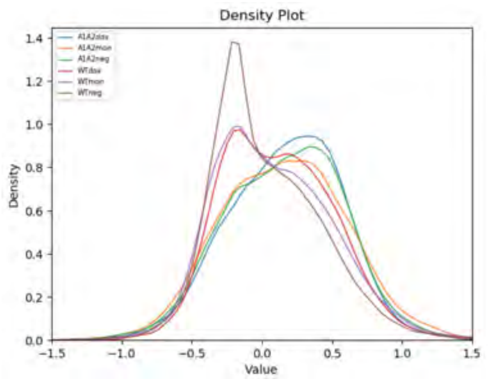

(b) Reconstructed images

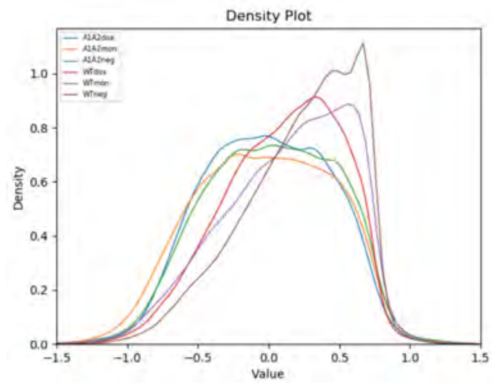
Figure 6.9: Comparison between original images and their corresponding reconstruction for the InfoGAN, trained on the second channel



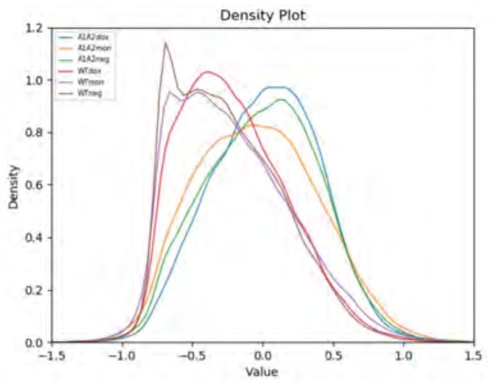
(a) Dimension 1



(b) Dimension 2



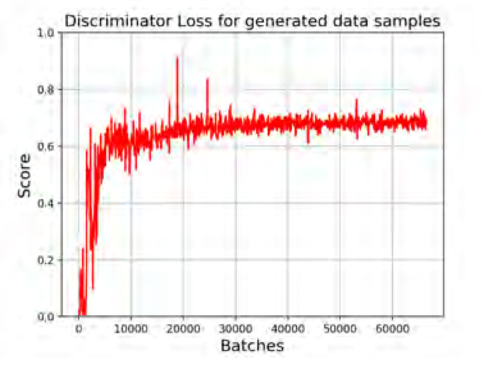
(c) Dimension 3



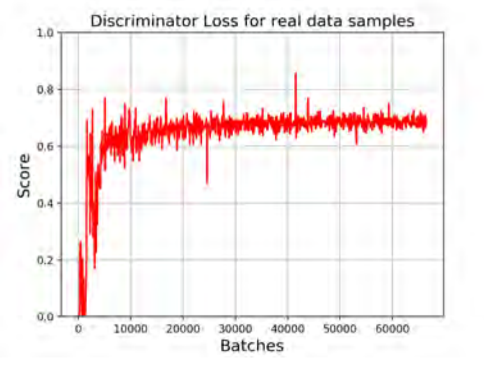
(d) Dimension 4

Figure 6.10: Comparison of density plots for different dimensions of the latent space modelled by the InfoGAN, trained on the second channel.

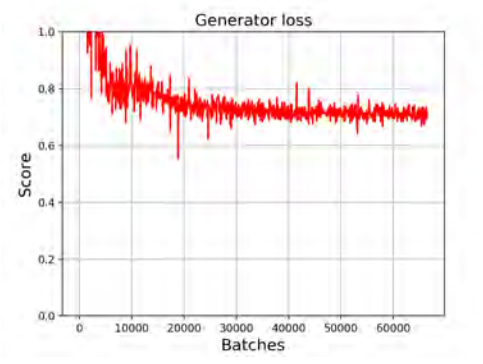
6.2.3 Training on both channels



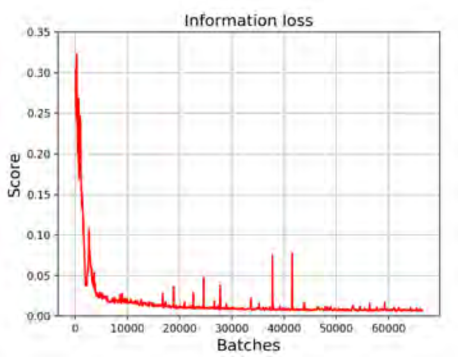
(a) Discriminator loss - fake images



(b) Discriminator loss - real images

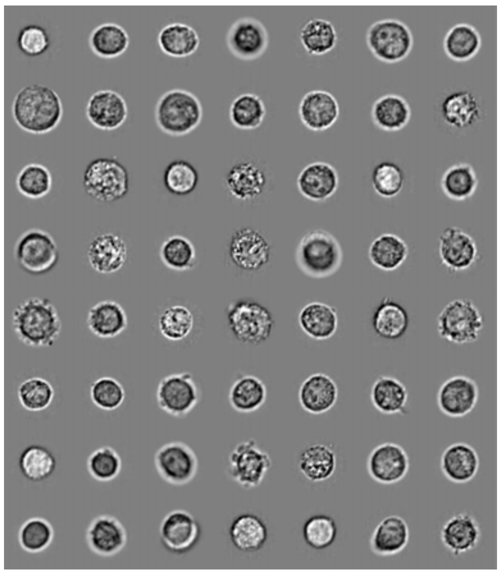


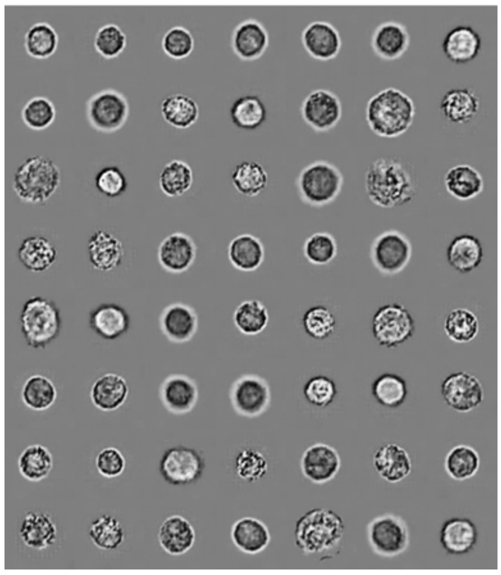
(c) Generator loss

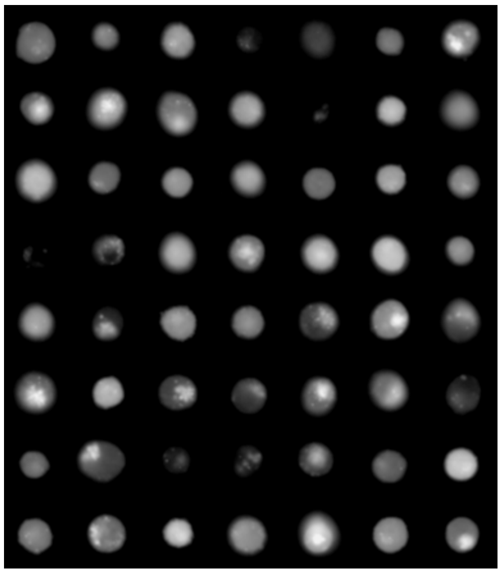


(d) Information loss

Figure 6.11: Both channels - Training losses for the InfoGAN plotted against number of batches.


(a) Original images


(b) Generated images


(c) Original images

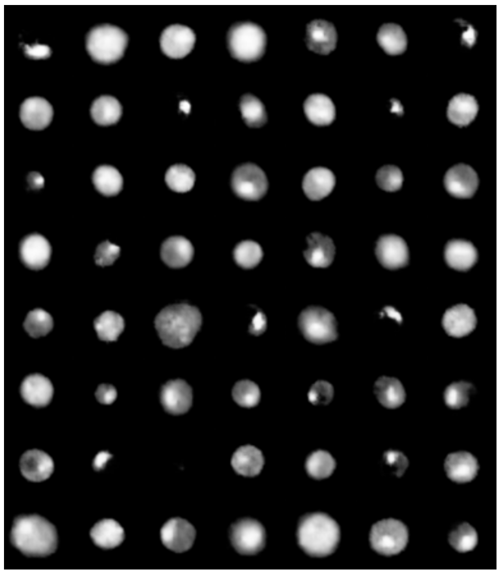
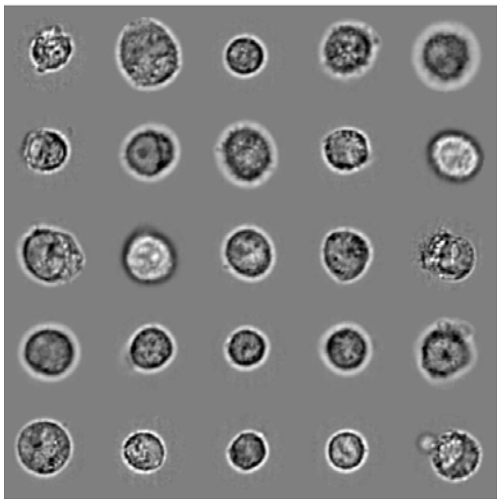
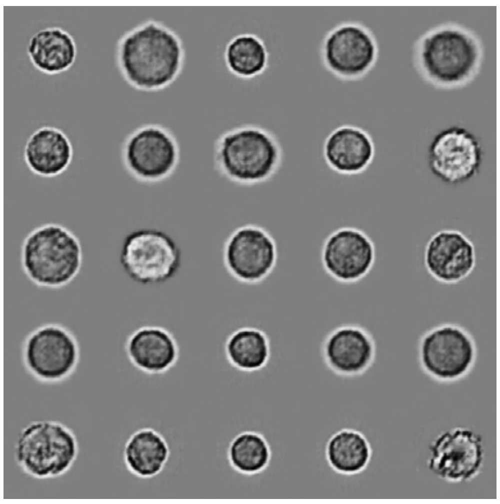
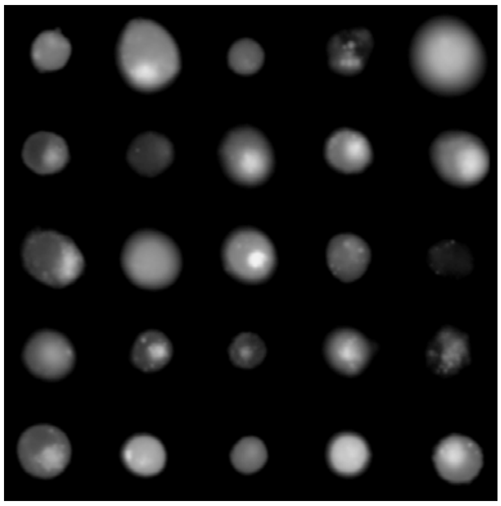

(d) Generated images

Figure 6.12: Comparison between original images and generated images for the InfoGAN, trained on both channels.


(a) Original images


(b) Reconstructed images


(c) Original images

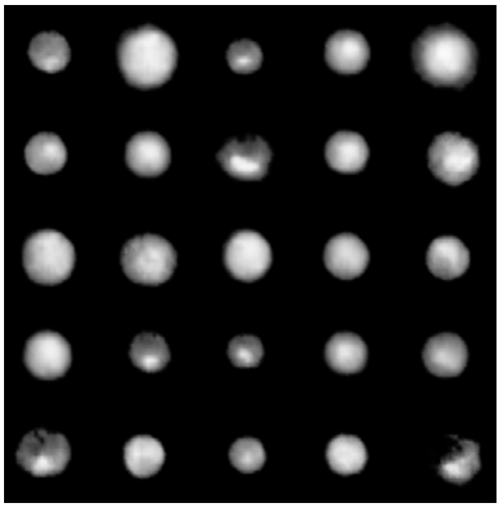
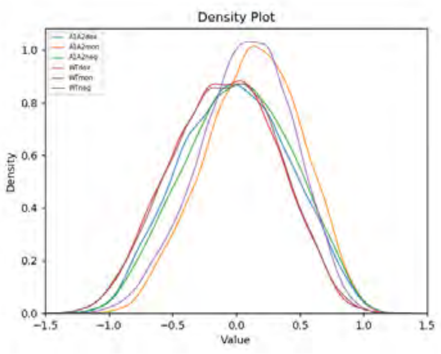
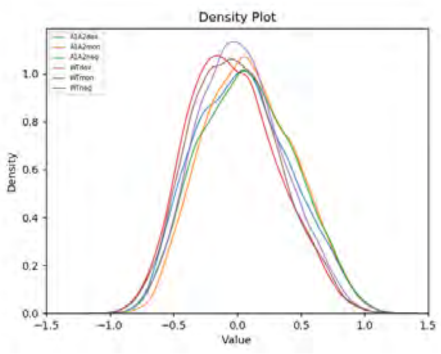

(d) Reconstructed images

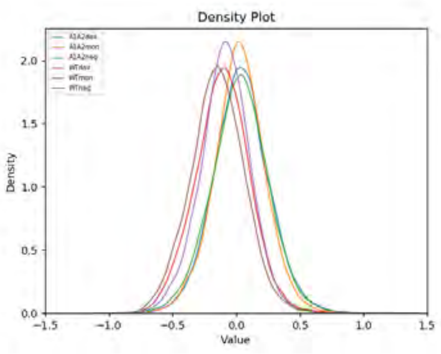
Figure 6.13: Comparison between original images and their corresponding reconstruction for the InfoGAN, trained on both channels.



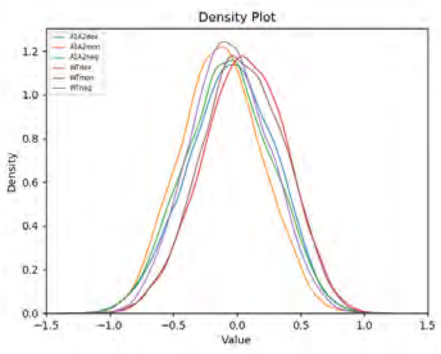
(a) Dimension 1



(b) Dimension 5



(c) Dimension 7



(d) Dimension 9

Figure 6.14: Comparison of density plots for different dimensions of the latent space modelled by the InfoGAN, trained on both channels.

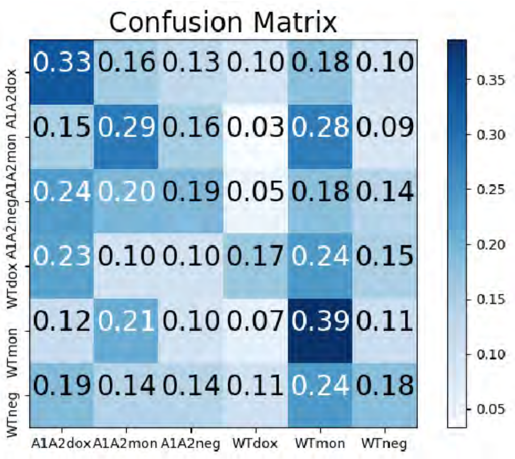
6.2.4 Discussion and comparison

Firstly, the InfoGAN achieves impressive performance in generating realistic and diverse images for all three models, indicating that it did well in modelling the underlying data distribution. Comparing the images for the first channel, it is hard to find any obvious artifacts in the generated images that distinguish them from the original. For the second channel, it produces images where the distribution of dots varies between the images, with some of them spread widely over the cell while others were localised to certain areas, which is exactly the behaviour we were looking for.

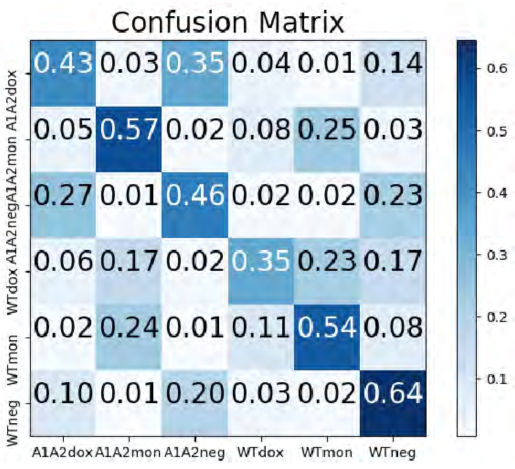
Secondly, the latent space for all models contains informative generating features, as it does a good job in mapping the images to their latent space and then reconstructing them. However, we note that reconstruction for the model using both channels is considerably worse than for the models trained on a single channel. A probable explanation is that all models are built using the same architecture and hyperparameters, and in order to encode the generative features for both channels, the dimension of the latent space needs to be increased, or the capacity of the network.

Finally, analysing the density plots for the models, we see evidence that the modelled generative features differ for different cell types. For the first channel, certain dimensions do not differ for different cell types, as they represent generative features linked to the appearance of cells, and not the processes happening inside of them;*e.g.* size of the cell. However, other dimensions do contain some information with regard to the cell type, with the most obvious difference being with cells that have been given the drug monencin. This is consistent with the results from the discriminative model, where it did considerably better in correctly identifying monencin cell types for first channel images. For the second channel, we notice that the latent mappings differ considerably between mutants and wild-types. Looking at the wild-types and mutants separately, we also see a clear difference in some dimensions based on the drug the cells were given. As an example, we see that for mutants, the doxorubicin and negative cells behave similarly compared to the monencin cells. For the wild-type cells, this is not the case, and we see more difference between all cell types. This is constant with our results from the discriminative model, which concluded that the mutants respond differently to the drugs than wild-types. For both channels, we see same differences in the latent mappings, but not as clearly distinguishable as for the second channel only. Overall, these results show that the InfoGAN has managed to model a latent space containing some disentanglement, finding generative features of the cell images that are consistent with the results from the discriminative model.

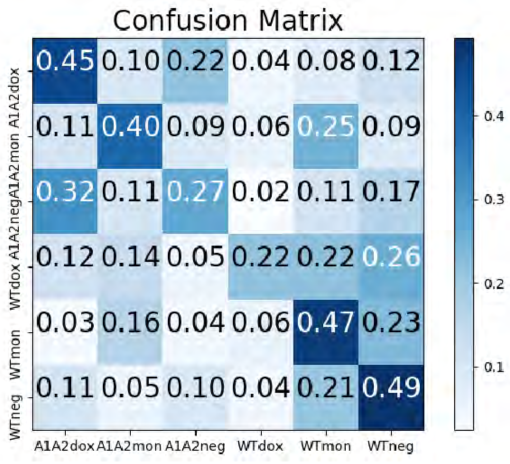
Looking at the dimensions independently is interesting to confirm that the model has found disentangled generative features, but it does not tell the whole story of how the models are encoding different cell types. The reasons for this are that the latent space is not fully disentangled, and even if it was, the cell type is not conditionally independent given one generative feature, as it depends on a combination of these features. In order to quantify how differently the models encode cell images of different type, a simple MLP with two hidden layers was trained to distinguish between the cell types, using their latent code as an input. Comparison of confusion matrices for the three models can be seen in figure 6.15. For the first channel, the classifier achieves 26% accuracy, mainly finding distinguishable patterns for monencin cells. The second channel does well, achieving 50% accuracy, with similar misclassifications as the discriminative model. Using both channels, the classifier achieves accuracy of 38%, which is worse than for only the second channel. This is not consistent with the discriminative model, that was able to use information from both channels to improve the accuracy. Again, the reason for this is most likely that the model is only using 10 dimensional latent space to encode represent features for both first and second channel, which means that it is not able to properly capture all important features, and therefore looses important information with regards to the cell type.



(a) First channel



(b) Second channel



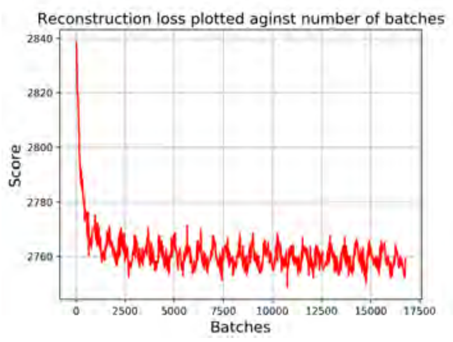
(c) Both channels

Figure 6.15: Comparison of confusion matrices for the MLP trained to distinguish between cell types, using the cell images latent vectors modelled by the InfoGAN as input.

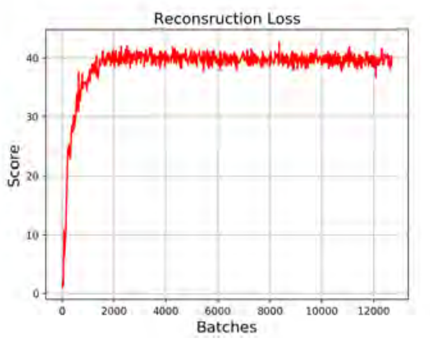
6.3 $\beta-$VAE - Results and discussion

In this section, we present results from the $\beta-$VAE model when trained only on the first channel, only on the second channel and on both channels. First, we present Figures that show the reconstruction loss and KL divergence plotted against number of batches, demonstrating that the algorithm trained successfully. Secondly, we sample randomly from the prior and generate new images, comparing them with original images to validate that the model has captured the data distribution, and generates realistic images. Thirdly, to ensure that the latent space contains informative generative features, images were mapped to their latent vector using the probabilistic decoder, $q_\phi(z|x)$, and then reconstructed using the probabilistic encoder, $p_\theta(x|z)$, before comparing the images. Finally, we compare how different cell types are mapped to the latent space by computing kernel density plots (Gaussian kernel) of all cell types, for each latent dimension. These plots confirm that some of the generative features that the latent space represents are connected to the cell type, and that the $\beta-$VAE found distinguishable features and patterns for cells where different processes were happening. The section is then concluded with discussion and comparison of these three different models. It should be noted that pixel values of all second channel images shown are log values.

6.3.1 Training on first channel

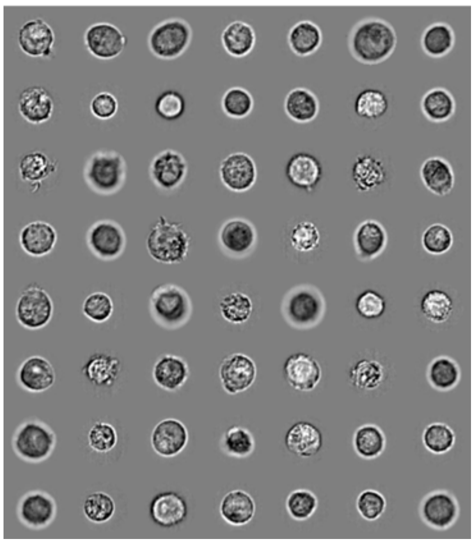


(a) Reconstruction loss



(b) KL divergence

Figure 6.16: First channel - Training losses for the $\beta-$VAE plotted against number of bathces.


(a) Original images

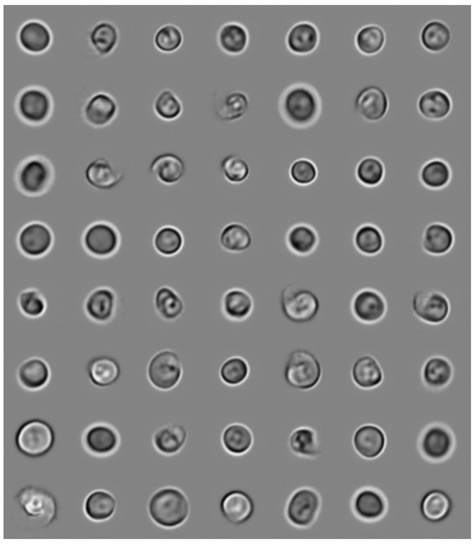
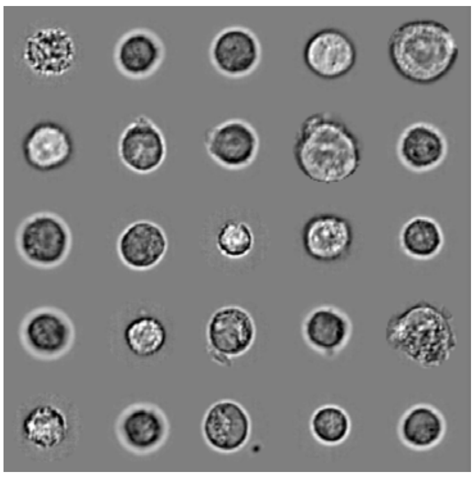

(b) Generated images

Figure 6.17: Comparison between original images and generated images for the $\beta-$VAE, trained on the first channel.


(a) Original images

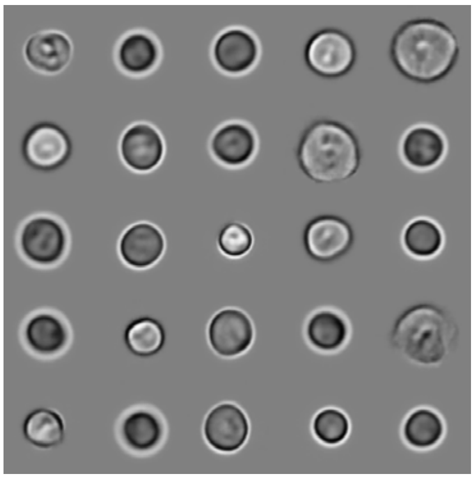
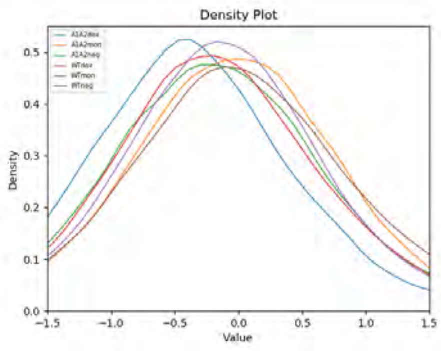
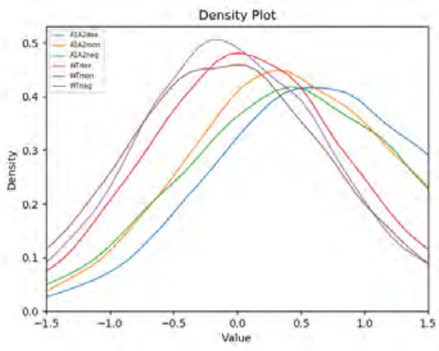

(b) Reconstructed images

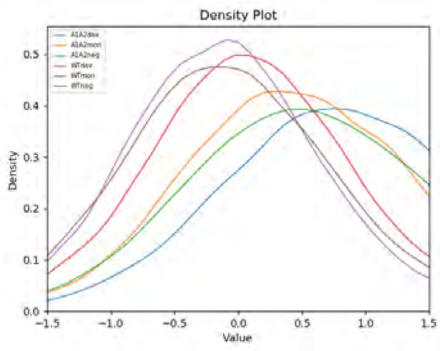
Figure 6.18: Comparison between original images and their corresponding reconstruction for the $\beta-$VAE, trained on the first channel.



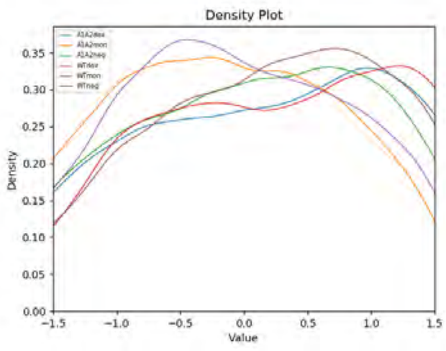
(a) Dimension 1



(b) Dimension 8



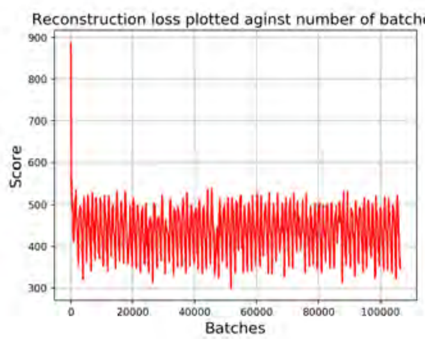
(c) Dimension 10



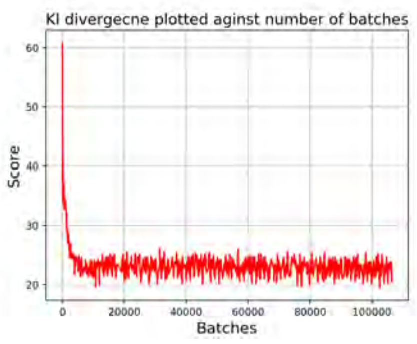
(d) Dimension 18

Figure 6.19: Comparison of density plots for different dimensions of the latent space modelled by the $\beta-$VAE, trained on the first channel

6.3.2 Training on second channel

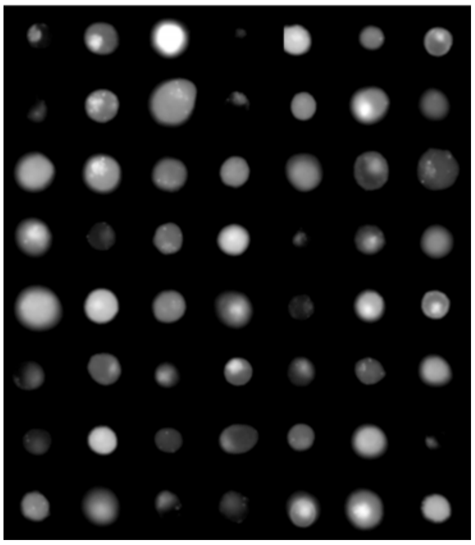


(a) Reconstruction loss



(b) KL divergence

Figure 6.20: Second channel - Training losses for the $\beta-$VAE plotted against number of batches.


(a) Original images

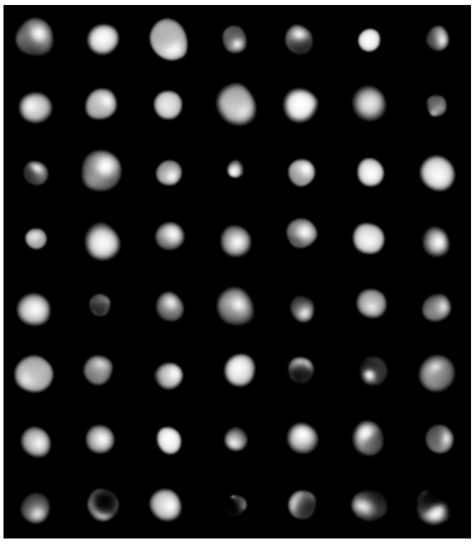
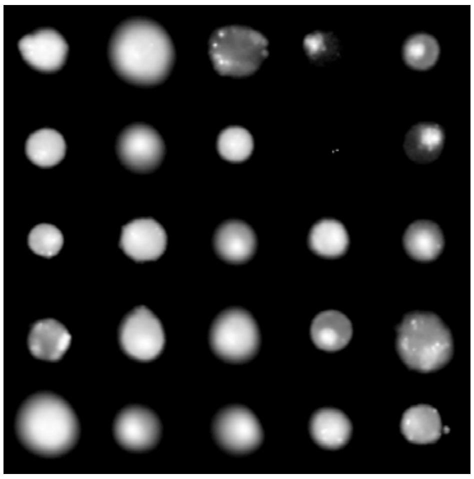

(b) Generated images

Figure 6.21: Comparison between original images and generated images for the $\beta-$VAE, trained on the second channel.


(a) Original images

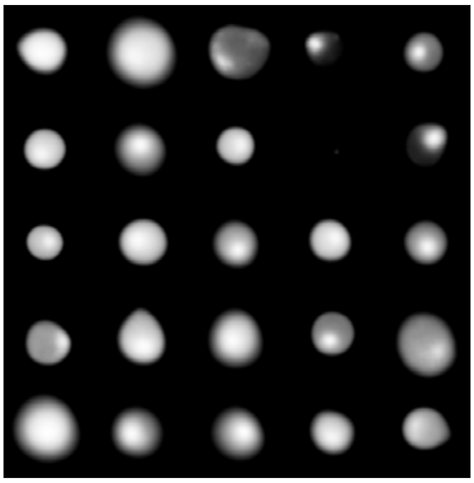
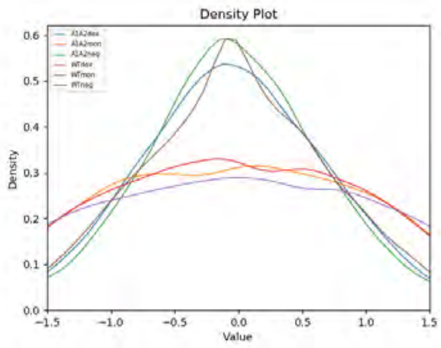
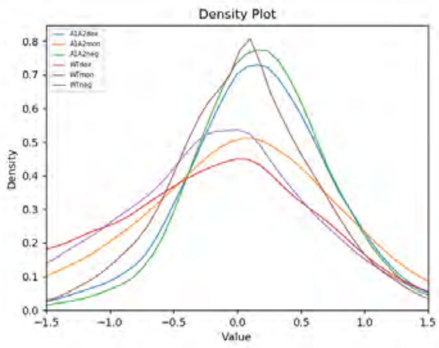

(b) Reconstructed images

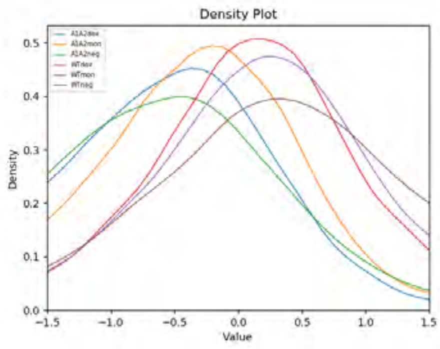
Figure 6.22: Comparison between original images and their corresponding reconstruction for the $\beta-$VAE, trained on the second channel.



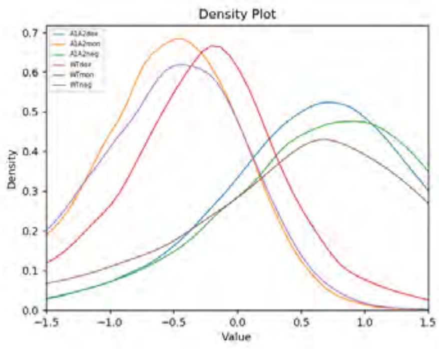
(a) Dimension 3



(b) Dimension 5



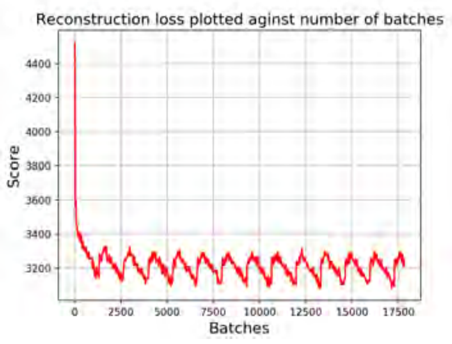
(c) Dimension 8



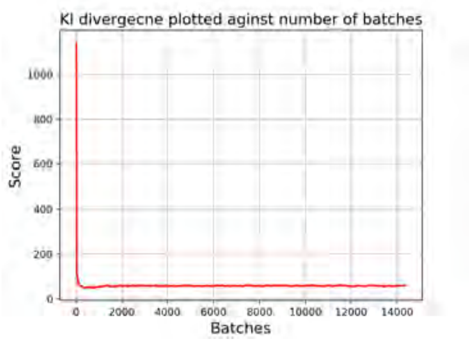
(d) Dimension 9

Figure 6.23: Comparison of density plots for different dimensions of the latent space modelled by the $\beta-$VAE, trained on the second channel.

6.3.3 Training on both channels

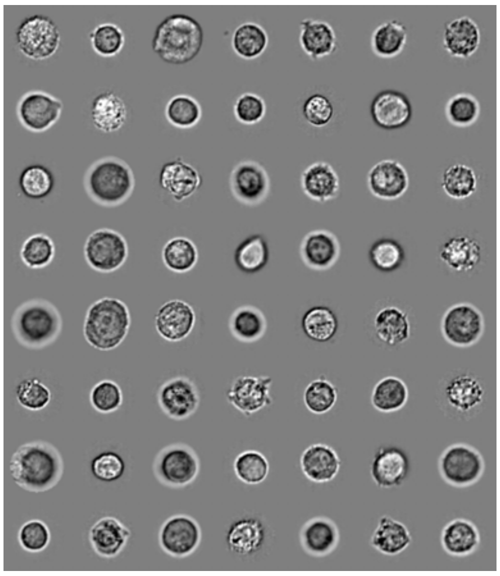


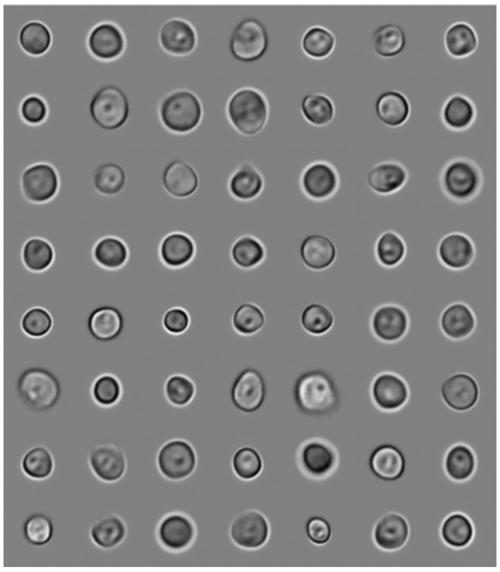
(a) Reconstruction loss

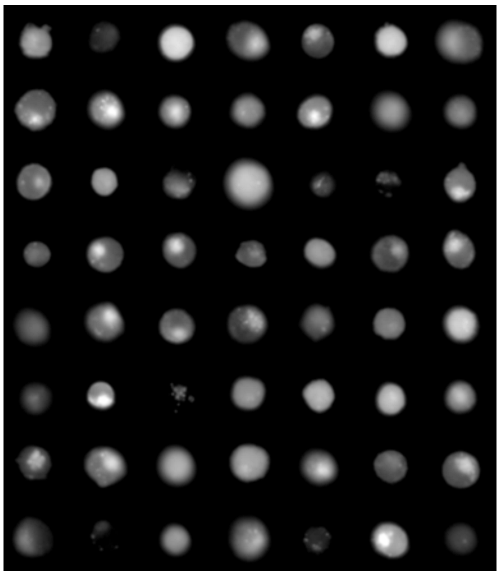


(b) KL divergence

Figure 6.24: Both channels - Training losses for the $\beta-$VAE plotted against number of batches.


(a) Original images


(b) Generated images


(c) Original images

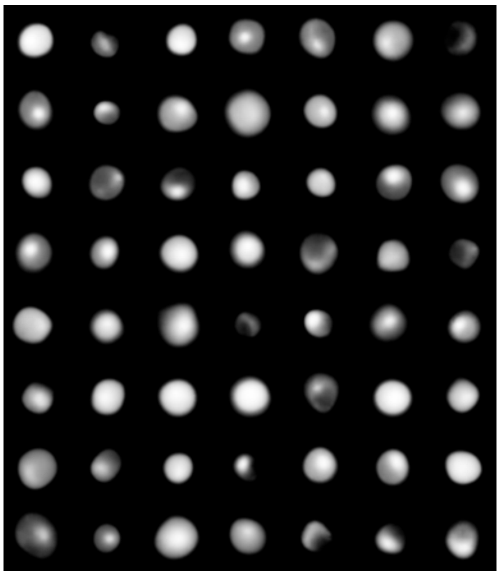
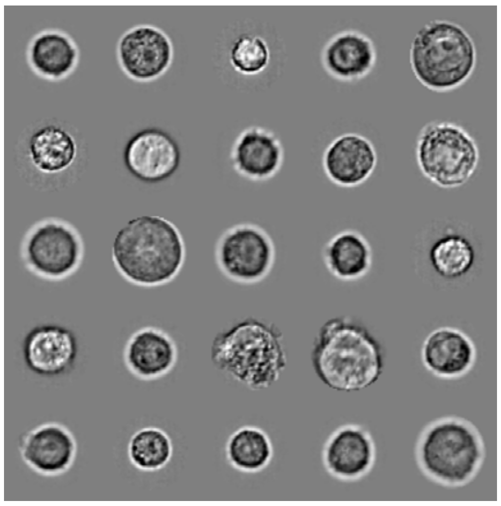
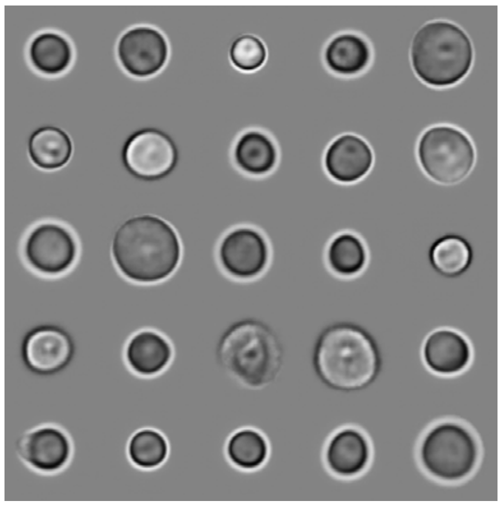
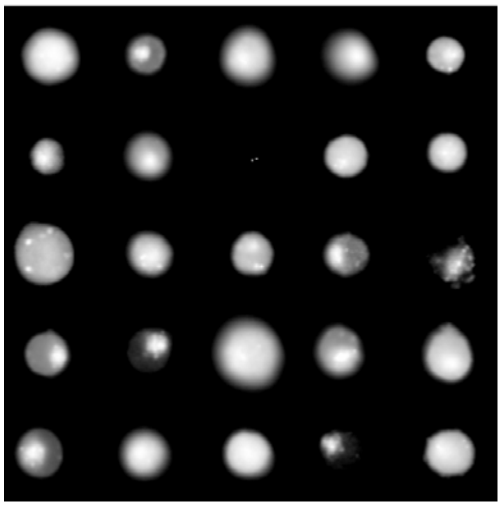

(d) Generated images

Figure 6.25: Comparison between original images and generated images for the $\beta-$VAE, trained on both channels.


(a) Original images


(b) Reconstructed images


(c) Original images

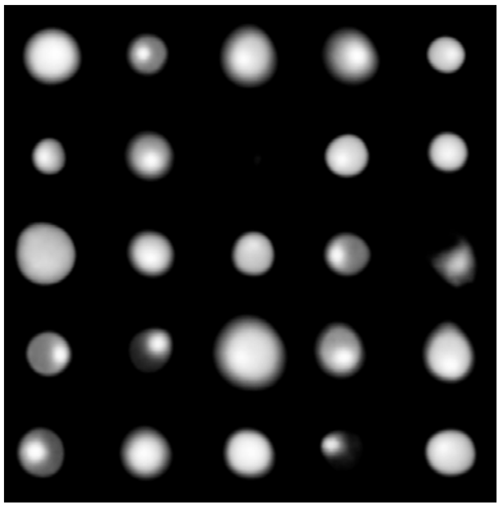
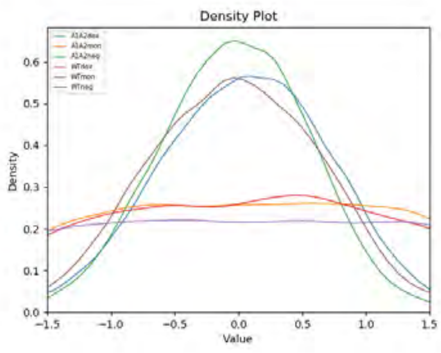
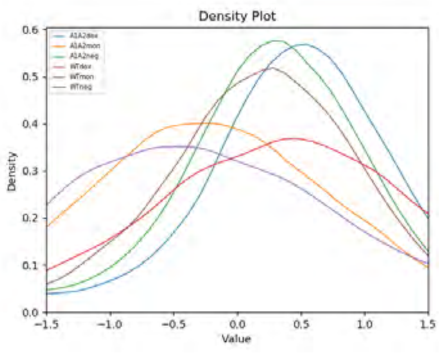

(d) Reconstructed images

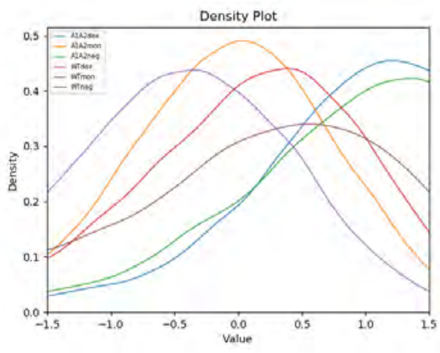
Figure 6.26: Comparison between original images and their corresponding reconstruction for the $\beta-$VAE, trained on both channels.



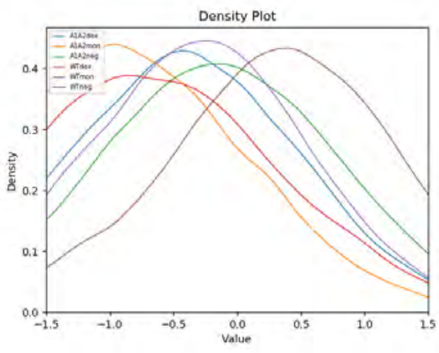
(a) Dimension 2



(b) Dimension 5



(c) Dimension 7



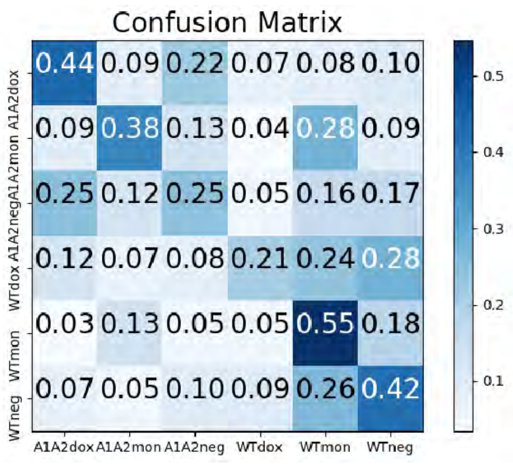
(d) Dimension 8

Figure 6.27: Comparison of density plots for different dimensions of the latent space modelled by the $\beta-$VAE, trained on both channels

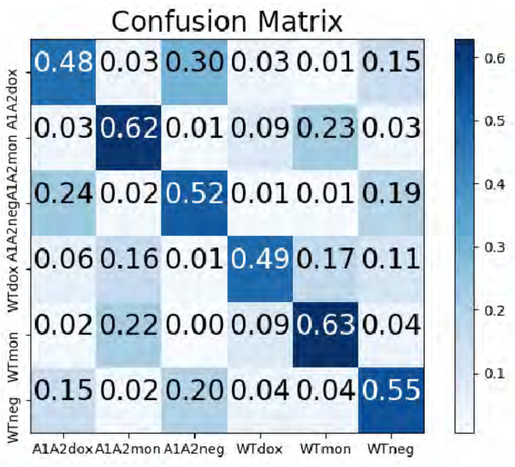
6.3.4 Discussion and comparison

Firstly, we see that the $\beta-$VAE has captured some information about the underlying data distribution, but the images do contain obvious artifacts that distinguish them from the original cell images, the most obvious one being how blurry the images are. This is a known issue with VAEs, which is caused by an over simplified approximation of the posterior distribution, and the loss functions associated with them [28]. Also, the model uses a high value for β to encourage disentanglement, which leads to worse reconstruction. However, it does capture important generative features, both for the first channel, and the second channel, where the distribution of dots varies between the images, which is important for our task. Secondly, it does a good job in mapping the images to their latent space and then reconstructing them. The reconstructions are blurry, but all other features seem to be well encoded into the latent space for all models. Finally, analysing the density plot for the models, we see that the modelled generative features differ for different cell types. For the first channel, the difference is not as clear, which is no surprise since we know that there is less information in that channel with regards to the cell type. For the models that use the second channel and both channels, we see clear differences between cell types, with cell types that are known from previous experiments to behave similarly being mapped to similar values in the latent space.

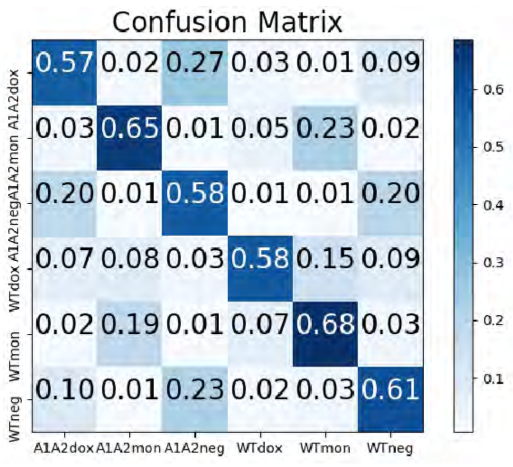
Again, looking the dimensions independently does not tell us the whole story of how the models are encoding different cell types. In order to quantify how differently the models encode cell images of different type, the same MLP as in section 6.2.4 was trained to distinguish between the cell types, using their latent code as an input. Comparison of confusion matrices for the three models can be seen in figure 6.28. For the first channel, the classifier achieves 38% accuracy, for the second channel, the accuracy is 55%, and for both channels it is 61%. This classification accuracy is similar to the discriminative model, showing that the $\beta-$VAE does a very good job in modelling the features that distinguish different cell types.



(a) First channel



(b) Second channel



(c) Both channels

Figure 6.28: Comparison of confusion matrices for the MLP trained to distinguish between cell types, using the cell images latent vectors modelled by the $\beta-$VAE as input.

6.4 $\beta-$VAE and InfoGAN - Comparison

	First channel	Second Channel	Both channels
InfoGAN	26%	50%	38%
$\beta-$VAE	38%	55%	61%

Table 6.1: Comparison of accuracies for the MLP trained to distinguish between cell types, using the cell images latent vectors as input.

For both methods, we were able to model a latent space with certain disentanglement, containing generative features of the data related to the cell type. However, we see that both methods have their strengths and weaknesses. With regards to image quality, the InfoGAN outperforms the $\beta-$VAE. This was expected, as VAEs are known to output blurry images, while GANs have shown great empirical results in generating realistic images. The reason for this is that GANs use complicated discriminative networks that are trained to discriminate between fake and real images as a loss function, while VAEs use simple distance measures like the L2 distance. Using this measure, the VAE would need to properly encode every single pixel in order to perfectly reconstruct the image. Also, two images containing same object at different positions are very different according to the L2 distance. Therefore, in order to minimise the L2 distance, the VAE learns to output a rough average of the images, resulting in blurry images.

On the other hand, the $\beta-$VAE models a latent space that contains more informative generative features with regards to the cell type, encoding cells of same type more similarly. This can be seen clearly in table 6.1, that contains the accuracy of a simple MLP that was trained to distinguish between different cell types using the latent vectors as an input. However, it should be noted that this comparison is not entirely fair, as the latent space modelled by the $\beta-$VAE was 32 dimensional while the InfoGAN was only 10 dimensional. How this constraint hurts the performance of the InfoGAN becomes very clear when we compare the models trained on both channels, as 10 dimensions are simply not sufficient to capture the features for both channels.

Chapter 7

Summary and Conclusions

In this thesis, we have successfully demonstrated how machine learning can be used to improve understanding of autophagy. To begin with, we showed that discriminative models are able to find distinguishable patterns and features inside cells where different processes with regards to autophagy are happening. Furthermore, we showed that generative latent variable models can successfully model the latent space of the cell images, where dimensions in this space represent distinct generative features for different cell types. We implemented two different generative models, the InfoGAN and $\beta-$VAE, highlighting their strengths and weaknesses. Our findings lead to two interesting results from the biological perspective. First, we showed that mutant cells respond differently to the drug doxorubicin than wild-type cells do. Secondly, we demonstrated that the first channel of the cell images contains information with regards to processes happening inside of them, but currently this channel is not used for autophagy research.

We also developed robust and vigorous methods to preprocess the cell images so they can be used for machine learning research, and to filter cells that are suitable for research from cells that are not. To our best knowledge, our method outperforms all methods that are currently being used to filter the cells.

The work presented in this thesis shows promising results using machine learning to better understand autophagy. However, there are many things left to investigate, and we hope that our work offers a stepping stone to further research in this area. Related to the work presented in this thesis, future work includes:

1. Determine how much information there is present in the first channel that is not in the second channel. We propose doing this by training a variational auto encoder with a U-net structure, to translate from the first channel to the second channel.

2. Develop a segmentation algorithm that takes in images containing multiple cells, and find every cell in that image. Methods presented in this thesis can then be used to return quantitative information about the processes in these cells and group cells with similar features together.

3. Further improve the algorithms introduced in this thesis; *e.g.* improve reconstruction quality of the variational auto encoder, experiment with different architectures and hyperparameter settings, implement other methods known to encourage disentanglement.

4. Train a separate generative latent variable model for each cell type to better understand what is happening inside of them.

An interesting continuation of this thesis would then be to make use of video data that shows the movement of components (lysosomes) inside the cells, and how they localise. These movements are known to follow complex patterns, which most likely contains valuable information about processes happening inside of them.

Bibliography

[1] D. Michael Ando, Cory McLean, and Marc Berndl. "Improving Phenotypic Measurements in High-Content Imaging Screens". In: *bioRxiv* (2017). DOI: 10.1101/161422. eprint: https://www.biorxiv.org/content/early/2017/07/10/161422.full.pdf. URL: https://www.biorxiv.org/content/early/2017/07/10/161422.

[2] M. Arjovsky and L. Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". In: *ArXiv e-prints* (Jan. 2017). arXiv: 1701.04862 [stat.ML].

[3] C. P. Burgess et al. "Understanding disentangling in β-VAE". In: *ArXiv e-prints* (Apr. 2018). arXiv: 1804.03599 [stat.ML].

[4] Peter D. Caie et al. "High-Content Phenotypic Profiling of Drug Response Signatures across Distinct Cancer Cells". In: *Molecular Cancer Therapeutics* 9.6 (2010), pp. 1913–1926. ISSN: 1535-7163. DOI: 10.1158/1535-7163.MCT-09-1148. eprint: http://mct.aacrjournals.org/content/9/6/1913.full.pdf. URL: http://mct.aacrjournals.org/content/9/6/1913.

[5] T. Q. Chen et al. "Isolating Sources of Disentanglement in Variational Autoencoders". In: *ArXiv e-prints* (Feb. 2018). arXiv: 1802.04942 [cs.LG].

[6] X. Chen et al. "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets". In: *ArXiv e-prints* (June 2016). arXiv: 1606.03657 [cs.LG].

[7] V. Dumoulin and F. Visin. "A guide to convolution arithmetic for deep learning". In: *ArXiv e-prints* (Mar. 2016). arXiv: 1603.07285 [stat.ML].

[8] Peter Goldsborough et al. "CytoGAN: Generative Modeling of Cell Images". In: *bioRxiv* (2017). DOI: 10.1101/227645. eprint: https://www.biorxiv.org/content/early/2017/12/02/227645.full.pdf. URL: https://www.biorxiv.org/content/early/2017/12/02/227645.

[9] I. J. Goodfellow et al. "Generative Adversarial Networks". In: *ArXiv e-prints* (June 2014). arXiv: 1406.2661 [stat.ML].

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[11] K. He et al. "Deep Residual Learning for Image Recognition". In: *ArXiv e-prints* (Dec. 2015). arXiv: 1512.03385 [cs.CV].

[12] P. Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". In: *ArXiv e-prints* (Nov. 2016). arXiv: 1611.07004 [cs.CV].

[13] Michael I. Jordan, ed. *Learning in Graphical Models*. Cambridge, MA, USA: MIT Press, 1999. ISBN: 0-262-60032-3.

[14] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *ArXiv e-prints* (Dec. 2014). arXiv: 1412.6980 [cs.LG].

[15] D. P Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *ArXiv e-prints* (Dec. 2013). arXiv: 1312.6114 [stat.ML].

[16] O. Z. Kraus, L. J. Ba, and B. Frey. "Classifying and Segmenting Microscopy Images Using Convolutional Multiple Instance Learning". In: *ArXiv e-prints* (Nov. 2015). arXiv: 1511.05286 [cs.CV].

[17] Yann LeCun et al. "Efficient BackProp". In: *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK, UK: Springer-Verlag, 1998, pp. 9–50. ISBN: 3-540-65311-2. URL: http://dl.acm.org/citation.cfm?id=645754.668382.

[18] Yann LeCun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE*. Vol. 86. 11. 1998, pp. 2278–2324. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665.

[19] A. Osokin et al. "GANs for Biological Image Synthesis". In: *ArXiv e-prints* (Aug. 2017). arXiv: 1708.04692 [cs.CV].

[20] J. Paisley, D. Blei, and M. Jordan. "Variational Bayesian Inference with Stochastic Search". In: *ArXiv e-prints* (June 2012). arXiv: 1206.6430 [cs.LG].

[21] Nick Pawlowski et al. "Automating Morphological Profiling with Generic Deep Convolutional Networks". In: *bioRxiv* (2016). DOI: `10.1101/085118`. eprint: `https://www.biorxiv.org/content/early/2016/11/02/085118.full.pdf`. URL: `https://www.biorxiv.org/content/early/2016/11/02/085118`.

[22] A. Radford, L. Metz, and S. Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *ArXiv e-prints* (Nov. 2015). arXiv: `1511.06434 [cs.LG]`.

[23] O. Ronneberger, P. Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *ArXiv e-prints* (May 2015). arXiv: `1505.04597 [cs.CV]`.

[24] K. Roth et al. "Stabilizing Training of Generative Adversarial Networks through Regularization". In: *ArXiv e-prints* (May 2017). arXiv: `1705.09367 [cs.LG]`.

[25] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (Oct. 1986), pp. 533–. URL: `http://dx.doi.org/10.1038/323533a0`.

[26] L. Zamparo and Z. Zhang. "Deep Autoencoders for Dimensionality Reduction of High-Content Screening Data". In: *ArXiv e-prints* (Jan. 2015). arXiv: `1501.01348 [cs.LG]`.

[27] M. D Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks". In: *ArXiv e-prints* (Nov. 2013). arXiv: `1311.2901 [cs.CV]`.

[28] S. Zhao, J. Song, and S. Ermon. "Towards Deeper Understanding of Variational Autoencoding Models". In: *ArXiv e-prints* (Feb. 2017). arXiv: `1702.08658 [cs.LG]`.