Adaptive Reference Images for Blood Cells using Variational Autoencoders and Self-Organizing Maps

Generering av adaptiva referensbilder till vita blodkroppar med variativ auto-kodare och självorganiserade avbildningar

Anna Palmqvist Sjövall
Oscar Odestål

April 2, 2020

Supervisors
Martin Almers, CellaVision
Sven Hedlund, CellaVision
Ida Arvidsson, Lund University
Niels Christian Overgaard, Lund University
Examinator
Magnus Oskarsson, Lund University

Abstract

CellaVision develops automated microscopy for blood analysis. Their products can pre-classify 19 different types of white blood cells and support the medical technologist performing the final classification. CellaVision provides *reference cells*. The reference cells are a fixed set of cells handpicked by a technologist, chosen to be typical for that specific class. The reference cells are static, i.e. the same for each cell image currently being classified and could be improved. We propose *adaptive reference cells*.

Using a combination of machine learning techniques, we develop a pipeline consisting of an Xception classifier, a Variational Autoencoder (VAE) and a Self-Organizing Map (SOM). This pipeline is used to produce the adaptive reference cells which are specific for each cell image. The medical technologist classifying an image is thus presented with cells that are the most visually similar to that image. The adaptive reference cells are of particular use for cells that are hard to classify.

The result consists of adaptive reference cells using both the output from the VAE and the SOM. The adaptive reference cells using the VAE are found superior. Also, cluster visualization using SOMs is presented together with proposed measurements for the robustness of the classifier and accuracy of the SOM.

Acknowledgements

We would like to give CellaVision credit for providing us with an innovative and friendly atmosphere which gave us multiple interesting conversations throughout our time there. Specifically, thank you to the employees who made us feel at home. A big thank you to Martin Almers for his guidance, enthusiasm and ideas during our time at CellaVision. Also, to Sven Hedlund for his vast knowledge, insightful reflections and feedback on the report. Jesper Jönsson, thank you for helping with both the work and the report throughout. And thank you Ida Wagnström for taking the time to perform the SVOMEN test.

Also, thank you to our supervisors Ida Arvidsson and Niels Christian Overgaard at the Department of Mathematics. They both had insightful feedback and ideas concerning both the report and thesis in general.

Finally, we would like to thank Oskar Klang, Martin Carlberg, Hugo Sellerberg and Nellie Carleke - the other master thesis students during our time at CellaVision. Oskar and Martin have given us countless of valuable discussions about our thesis, and the cool experiments overlapping both projects. They, as well as Hugo and Nellie, have contributed to making our time at CellaVision a pure joy.

Contents

Abbreviations

- **ANN:** Artificial Neural Network.
- **Adam:** Adaptive Moment Estimation.
- **CNN:** Convolutional Neural Network.
- **DSOM:** Deep Self-Organizing Map.
- **DW:** Depthwise Convolution.
- **DWS:** Depthwise Separable Convolution.
- **KL:** Kullback-Liebler.
- **MSE:** Mean Squared Error.
- **NN:** Neural Network.
- **Outlier:** Outlying Observation.
- **PCA:** Principal Component Analysis.
- **PW:** Pointwise Convolution.
- **RBC:** Red Blood Cell.
- **RGB:** Red, Green and Blue.
- **RI:** Middle Image in the SVOMEN test.
- `ROTMES`**:** Rotation Measurement created in this paper.
- **ReLU:** Rectified Linear Unit.
- **SOM:** Self-Organizing Map.
- **SVD:** Singular Value Decomposition.
- `SVOMEN`**:** Evaluation metric created in this paper.
- **TLU:** Threshold Logical Unit.
- **U-matrix:** Unified Distance Matrix.
- **VAE:** Variational Autoencoder.
- **WBC:** White Blood Cell.
- **Xception:** Extreme Inception.

1 Introduction

1.1 Motivation

White blood cells (WBCs) circulate the human body and act as our defense mechanism against diseases. An abnormal blood sample can indicate that the patient might suffer from a disease such as leukemia or lymphoma. Such an example can be seen in Figure 2 and 3, where the left smear is from a healthy person and the right from a person suffering from leukemia. To correctly identify these diseases, the WBCs need to be classified consistently. The WBCs are typically classified into 19 classes unequally divided into five categories (one category is nucleated red blood cells). Within these, one cell starts as one class and continuously develops into others. Figure 1 shows one of these categories and the different developing stages. The rest of the categories can be found in the Appendix in Figure 49-52. In Figure 1, we see that a Band could develop into a Neutrophil. During this development, the cell class will be ambiguous and the classification differs depending on the technician classifying the cell.

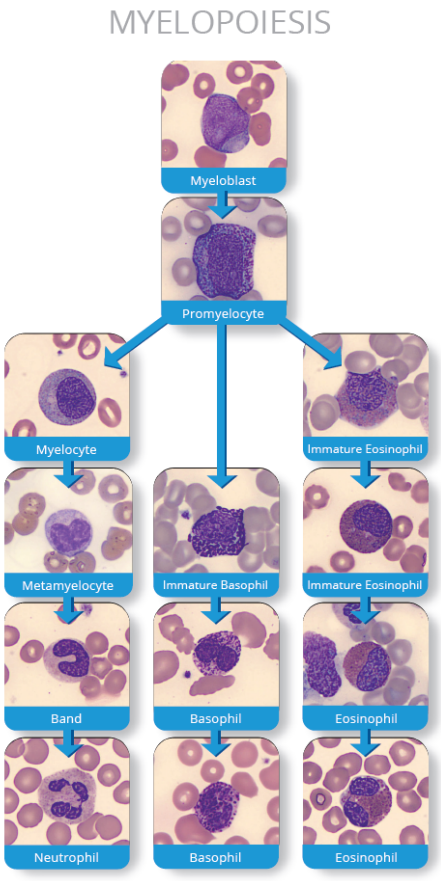

Figure 1: Myeloblast transformation to Neutrophil, Basophil and Eosinophil, within the category Myelopoiesis.

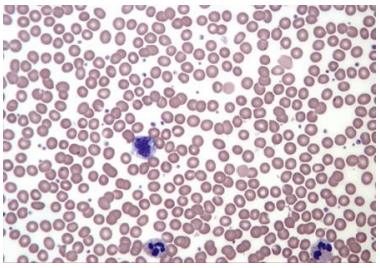

Figure 2: Blood smear from a healthy person.

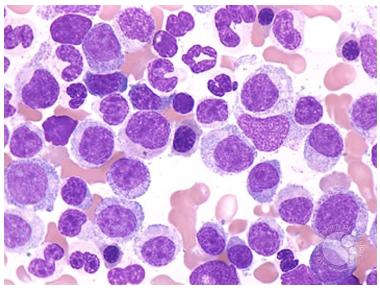

Figure 3: Blood smear from a person with leukemia.

The medical technologist tasked with classifying and finding abnormalities in WBCs is assisted by CellaVision's systems. The output from CellaVision's microscope is an image of a WBC. The systems can pre-classify the cell image, i.e. give a suggestion of which class a WBC image belongs to. Also, CellaVision shows six example cells in each class, *static reference cells*, handpicked to be typical for that class by a technician. The technician can use both the pre-classification and its results graphically presented, as well as static reference cells before making the final classification themselves. These static reference cells do not account for the specific image or system at hand. So, the reference cells are not the most visually similar and can be from systems with different optics.

1.2 Aim

This thesis aims to produce *adaptive reference cells*, i.e. reference cells that are different depending on the cell image being classified. Possibly these adaptive reference cells can also be based on cells previously collected by the same lab/system. These reference cells should be the cells that are the most visually similar to the cell image being classified. Also, visualization methods will assist the technicians in finding patterns in the data. These patterns could distinguish a cutoff between the classes for consistent classification. These aims can be concretized into two subtasks:

- A cell that is of interest to an expert should be shown together with visually similar images (adaptive reference cells) to make a more nuanced classification.
- Use visualization techniques to find cell similarity.

2 Background

2.1 White Blood Cells

White blood cells are a part of the human immune system and unlike a red blood cell (RBC), they contain a nucleus. A typical cell image consists of one WBC and none or multiple RBCs. Every WBC derives from stem cells in the bone marrow. Once developed into a WBC, they are often categorized into 10-20 classes depending on who makes the classification. The classes and proportions in Table 1 are not all WBCs but they are found in human blood. These statistics are obtained by counting all the nucleated elements, both mature and immature present in cellular bone marrow films [1]. A large shift in this distribution could be a sign of a serious disease such as Adult Acute Myeloid Leukemia (AML) in which the number of Blast cells (Myeloblasts) increases rapidly. Myeloblasts in AML are abnormal and will not develop into healthy blood cells as they are intended to do [2].

Table 1: Blood cell classes proposed by d'Onofrio and Zini in *Morphology of the blood* [1].

Cell class	**Percentage of total cells**
Blasts (myeloblasts)	1-3%
Promyelocytes	1-7%
Neutrophil myelocytes	5-20%
Neutrophil metamyelocytes	10-30%
Segmented and band neutrophils	5-25%
Eosinophil series	1-4%
Proerythroblasts	1-5%
Basophilic erythroblasts	2-8%
Polychromatic erythroblasts	2-20%
Orthochromatic erythroblasts	2-10%
Monocytes	0-3%
Lymphocytes	3-15%
Plasma cells	0-3%
Macrophages, reticulum cells	$<2\%$

2.2 CellaVision

CellaVision addresses the global shortage of qualified staff and the increasing pressure to reduce costs in hematology laboratories by delivering automated processes with a more reliable result. During the 18 years in the business, CellaVision has claimed a market-leading position with an 18% share of their target market [3]. Their newest product, which is the machine that has captured the images used in this report, is called DC-1 and is found in Figure 4.

When studying the blood, microscopy is an important part to identify and count WBCs. This is a labor-intensive job and dependent on medical technologists. However, this task is necessary in order to identify and, in extension, prevent and cure diseases. CellaVision provides systems that automatically analyze blood smears and present its result to a technician for further investigation.

Figure 4: The DC-1 machine and its software interface from CellaVision.

The result from a CellaVision system can be seen in Figure 5, with the collected cells to the left and the reference cells to the right.

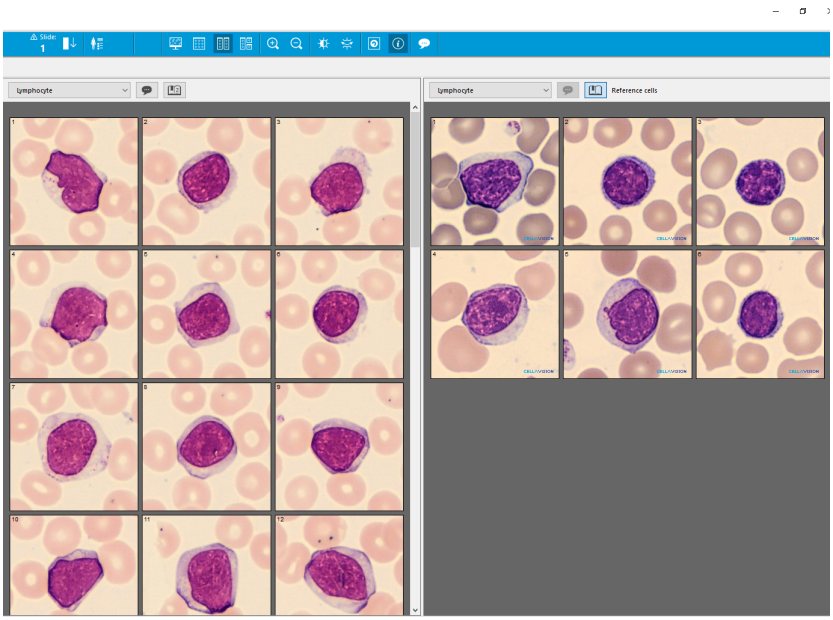

Figure 5: From CellaVisions interface: Collected lymphocyte cells to the left and lymphocyte reference cells to the right.

The system works in the following way. First, a blood smear is fed to the machine which starts to analyze the sample. Secondly, a microscope hovers over a certain part of the smear and captures a picture of each WBC. Finally, the images are classified by the machine and the result is presented as in Figure 5. The classes used by CellaVision are shown in Figure 6. Classes like *Artefact* and *Smudge Cell* are created by CellaVision to describe objects unrelated to blood and broken cells respectively.

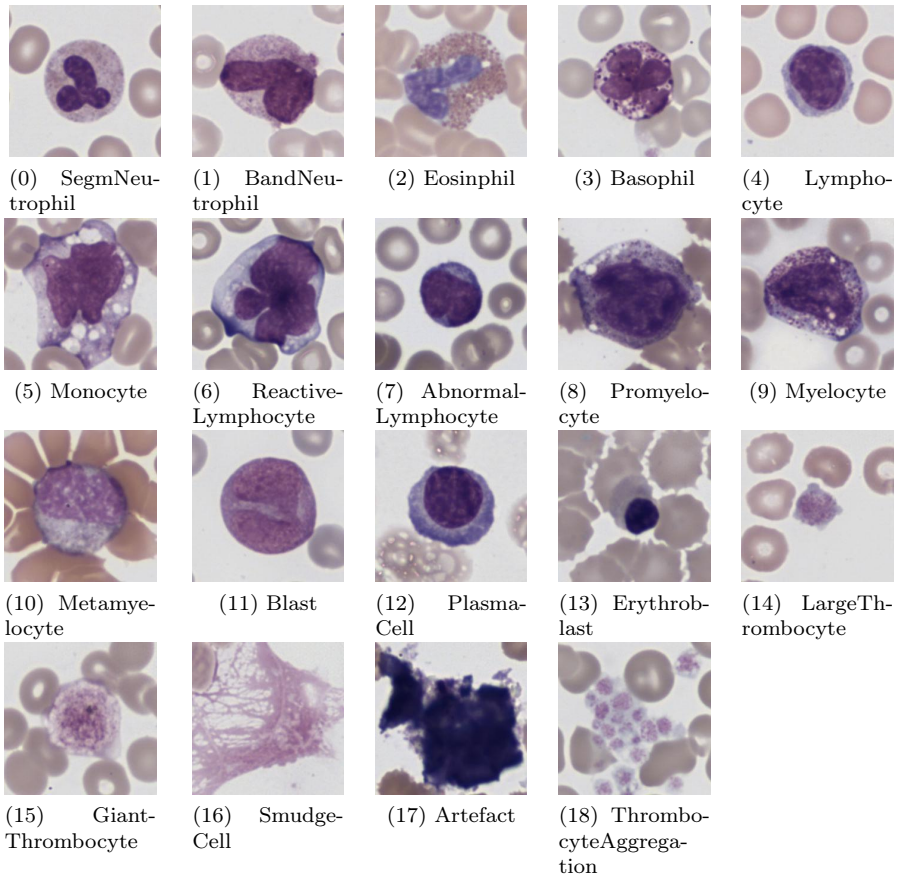

Figure 6: The cell classes used by CellaVision, with their given class number.

2.3 Artificial Neural Networks

The human brain has biological neurons that are connected in a vast network. It is also called a biological neural network. This is the inspiration of the machine learning models known as neural networks (NN) [4]. An artificial neural network (ANN), in turn, is called so when a neural network is implemented in software as the neurons are not physical objects (such as in the brain) [5]. The first ANN architecture was created in 1943 by Warren McCulloch and Walter Pitts [6]. After decades of varying waves of interest, ANNs have gained traction steadily since the 1990s. Amongst other things, this is due to the increase in computational power which allows for both greater data sets and models together with faster computations [4].

McCulloch and Pitts' model was later developed into a perceptron in 1957 by Frank Rosenblatt. The perceptron is an ANN architecture and is based on the artificial neuron called a threshold logical unit (TLU) shown in Figure 7. The neuron receives multiple inputs and if they together exceed a given threshold, the neuron is excited and sends the signal forward. The TLUs inputs, x_i, are each associated with a weight, w_i. The linear combination of the inputs and the weights are summed, $z = \sum_i w_i x_i$, and sent through a step/activation function

$h_w(x) = f(z)$ that outputs the result. The step function could, for example, be chosen to be the Heaviside step function, $\theta(z)$ [4].

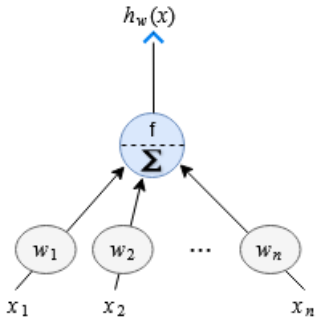

Figure 7: A threshold logical unit (TLU), the base of a perceptron/multilayer perceptron.

By then creating a network of TLUs, we have a neural network (more specifically a perceptron/multilayer perceptron). An example of such a network is shown in Figure 8. The nodes are neurons and the edges are the flow of information. The information is fed into the input layer, flows through one or more hidden layers of neurons and finally into the output layer. This output could be binary, categorical, numbers, etc. As described by Gérard Dreyfus book *Neural Networks*, a neural network without hidden layers and with a linear activation function is a linear system, but ANNs are particularly useful for non-linear relations in the data [5].

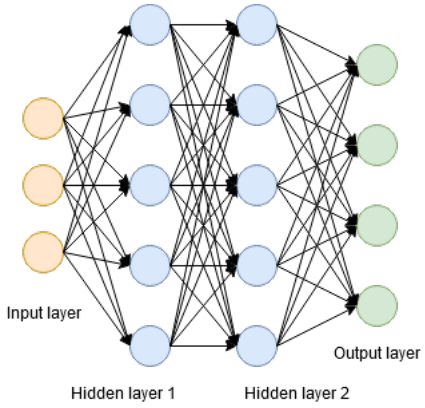

Figure 8: A simple fully-connected neural network with two hidden layers.

The ANN needs to be trained to provide sensible and relevant output. This training can either be *supervised* or *unsupervised*. Supervised training is when the input has known corresponding outputs. Unsupervised training is without known outputs and is often the case in visualization techniques such as clustering or self-organizing maps [5].

To train an ANN means to optimize the weights (parameters) of the network. This optimization is usually done with gradient descent which will minimize

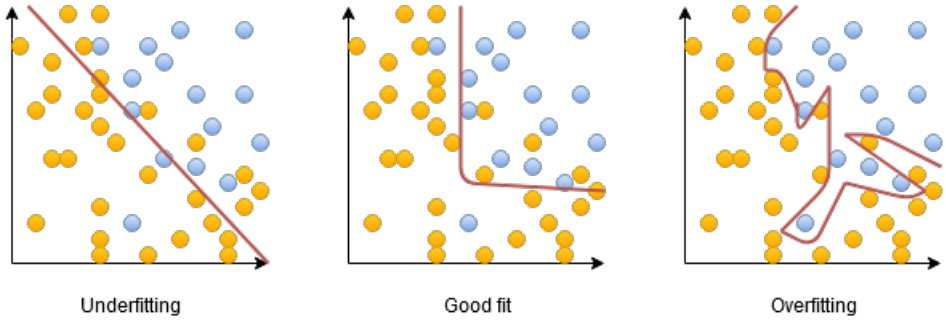

Figure 9: An example of underfitting and overfitting compared to the optimal one in the middle. The yellow and blue dots are different classes and the line is how the classifier divides them.

a *cost function* (also known as loss function). The cost function is a scalar defined specifically for the task at hand and describes the performance of the network. In supervised tasks, this cost function could, for example, be the mean squared error (MSE) between expected output and actual output [4]. Gradient descent gives us a way to minimize the cost function $f(x)$ by moving x in small steps using derivatives. How big the steps are is determined by the learning rate, α. To calculate these derivatives, means to calculate the *gradient*, a multi-variable derivative of f. Computing the gradient in an ANN is done with the *back-propagation* algorithm. The algorithm propagates backward the cost information through the network. Backpropagation turns a computationally expensive operation, calculating the gradient numerically, into an inexpensive task [7].

The data set is first divided into a train, validation and test set before training. The training data is used to train the model as described. The validation data is used to measure the performance of the model while training. Finally, the test set is used after training to evaluate the model.

When designing the ANN model, i.e. deciding how many and which layers to include, there are two key aspects to consider. The first one is when the network has too many parameters so it can very accurately fit both the training data and noise/outliers but cannot generalize and do well on new data. This is also known as overfitting, see the example in Figure 9. During training, a high training accuracy but low validation/test accuracy is a typical sign of overfitting. The other aspect is when we have too few parameters. This means we will underfit the data and not be able to accurately learn the training data [5]. An example of underfitting is also shown in Figure 9.

The training can either be done by batch (nonadaptive) or online (adaptive) learning. Batch learning means that training is done on all the data at once. It requires a lot of computational resources which can make it impossible for huge data sets. Online learning, on the other hand, means that the learning is performed incrementally by feeding the network mini-batches (or individual) instances of data. It is useful when the data set cannot fit in the computer's main memory as it will run training on some data and then discard it before loading more [4].

Parameters

There are four key parameters to set when training a neural network. They are the batch size, learning rate, number of iterations and choice of optimizer.

For online training, we can set the size of the mini-batches (batch size). Setting it as high as the GPU can manage could increase performance. However, a large batch size will make the training unstable and could lead to less generalization [4]. Dominic Masters and Carlo Luschi's paper from 2018 claims that the batch size should be between 2 and 32 [8]. While a paper from Elad Hoffer et al. in 2017 showed that very large batch sizes can perform well, but then other adjustments need to be made to the learning rate [9].

An important part of online learning is setting the learning rate. The learning rate decides how fast the system should adjust to changes in data. Too high of a learning rate results in a system not being influenced enough by previous data. A learning rate that is too small will conversely result in a system learning much slower, but also being less sensitive to outliers [4].

Manually setting the number of iterations can be replaced by adding `early stopping` as a callback in Keras which is a deep learning library for Python. Keras implements machine learning programs faster and easier. Without using `early stopping`, the number of iterations must be chosen large enough for the model to learn. But not too large such that we keep iterating even though the model is not learning any more information, that will simply waste computational resources and risk overfitting. Another consideration is that selecting the number of iterations will also impact the learning rate, a lower learning rate will need more iterations as it will converge slower [5].

Different optimizers can be used for different problems. One is Adam(adaptive moment estimation) and was introduced by Diederik P. Kingma and Jimmy Ba in 2014. It is an adaptive learning rate algorithm, meaning that less tuning of the learning rate is required. The adaptive learning rate will decrease over time as the cost function moves closer to a minimum [10]. The default starting value in Adam (learning rate = 0.001) is often a good choice [4].

3 Theory

3.1 Convolutional Neural Networks

Convolutional neural networks (CNN) are a type of ANNs specifically used when the input is an image or video data. As the book *Deep Learning* defines it: *"Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers."* [7]. Again, the inspiration comes from the human brain, specifically the visual cortex [10]. CNNs have been highly successful, even outperforming humans on certain tasks [4]. As input, ANNs can, for example, take certain image features that the designer can choose. For cells, this might mean the color intensity of the nucleus, cell membrane, background and number of red blood cells [11]. This, however, requires a lot of preprocessing and it is hard to choose the right features. By instead using a CNN with many hidden layers (deep learning) we can send in the whole image and the CNN will itself learn the different features.

This input image is represented as Figure 10 in the computer. A square image is typically represented using a matrix with dimensions $M \times M \times 3$. The size of the third dimension depends on whether the image is grayscale or in color. A grayscale image would have dimensions $M \times M \times 1$ but a color image would have $M \times M \times 3$ where the third dimension here represents red, green and blue (RGB) values. Images can also be represented by more than 3 channels and a rectangular matrix, but this is not as common. The pixel values can be represented in different ways, but generally low numbers are darker and higher are brighter.

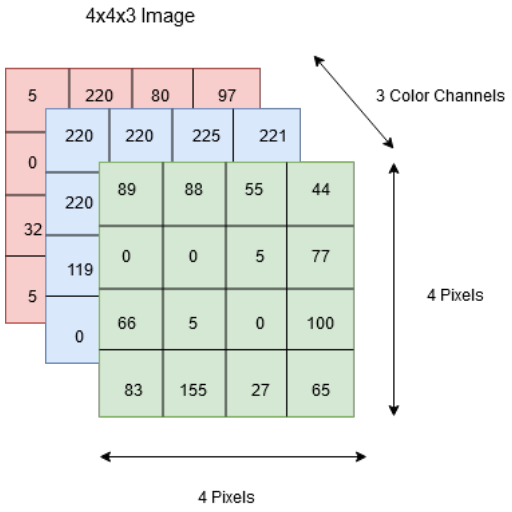

Figure 10: An image as represented in the computer's memory. The different color channels give the red, blue and green intensities.

Besides convolutional layers, CNNs can consist of different layers combined in different ways. Commonly, they consist of some combination of convolution, activation and pooling [12]. The final output after those layers is then flattened to a one-dimensional array and sent as input through a fully-connected layer(s). Fully connected layers are known as dense layers and can be connected to an

output layer used for example in classification [13]. An example of a CNN classifier from image to classification is shown in Figure 11.

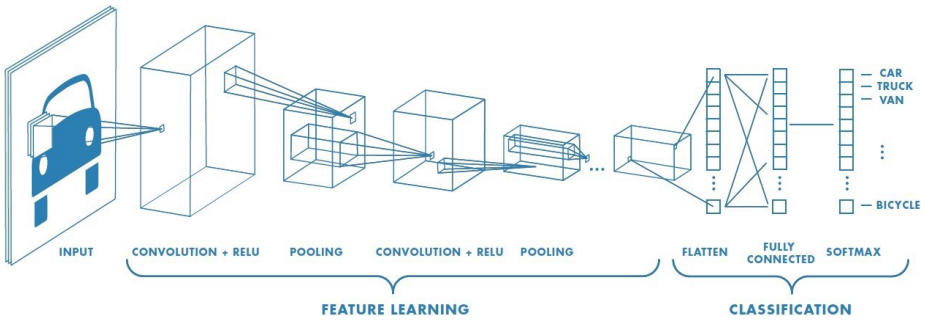

Figure 11: A CNN classifier that takes an image as input, goes through a combination of convolution, activation and pooling layer, sends it through a fully-connected neural network and finally gives the class as output. The image is taken from [14].

Convolution

The operation *Convolution* is denoted by an asterisk. The discrete 2D convolution is defined in Equation (1), with input image , f, and a $(2a+1) \times (2b+1)$ kernel, w. The feature map (output image) is represented by g.

$$g(x,y) = (w * f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t). \tag{1}$$

In the convolutional layer, each node is a filter. So, it receives an input, filters it and sends the filtered image through. Below, the calculations are based on the input layer. For black and white images $M \times M \times 1$, the filter is a $N \times N \times 1$ kernel which convolves (slides) across the image and takes the dot product of the two matrices. This process is shown in Figure 12 with a $3 \times 3 \times 1$ kernel ($N = 3$). In color images which has 3 channels (RGB), $M \times M \times 3$, this filter will be $N \times N \times 3$, i.e. a 3 dimensional kernel (cubic kernel) [7]. The output will then be of dimensions $(M - N + 1) \times (M - N + 1) \times 1$. Convolution with a cubic kernel is seen in Figure 13.

The following is done to compute the output from the convolutional layer given the neuron. So the output $z_{i,j,k}$ of a neuron in row i, column j in feature map k in a convolutional layer l with three channels is calculated with this equation:

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f'_n-1} x_{i',j',k'} \cdot w_{u,v,k',k}, \quad with \quad \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

where $x_{i',j',k'}$ is the output of the neuron in layer $l-1$, $w_{u,v,k',k}$ is the connection weight, $s_h, s_w, f_h, f_w, f_{n'}, b_k$ are the vertical and horizontal strides, the height and width of the receptive field, number of feature maps in layer $l-1$ and the bias term [4].

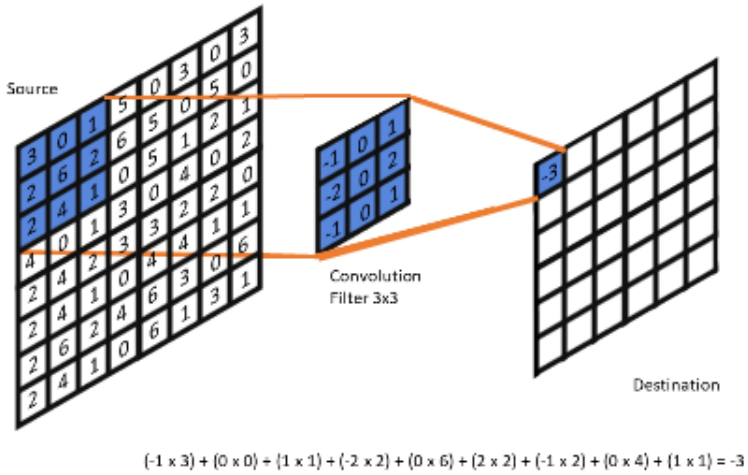

Figure 12: The source is convolved with the kernel to produce the pixel value of -3 in the destination.

In Figure 14 we see an example of strided convolution and the corresponding convolution and downsampling. In the left image, the stride is set to two. The networks will result in the same output but using a strided convolution instead of a combination of convolution and downsampling will decrease the number of computations. Hence, using strided convolution will decrease memory usage and computational power requirements [7].

An advantage of using CNN is the network's sparse interactions, not all nodes are connected to all other nodes (as a fully-connected layer). We achieve this by selecting a kernel that is smaller than the input which reduces the number of weights. This leads to fewer parameters in the computer's memory, thus reducing memory requirements [7]. We see an example of this sparse connectivity in Figure 14. Taking an RGB image of size $100 \times 100 \times 3$ and using 100 convolution layers with $3 \times 3 \times 3$ filters and the same output size, we would need 2 800 parameters ($(3 \cdot 3 \cdot 3 + 1) \cdot 100 = 2800$). If we instead take an ANN with one fully connected layer, that would be over 3 million parameters ($100^2 \cdot 100^2 \cdot 3 = 3030000$, a layer with 100×100 neurons) [4].

Activation

Usually, following each convolutional layer in a CNN is a nonlinear activation layer [16]. The most common one is the Rectified Linear Unit (ReLU), with the activation function $f(x) = max(0, x)$ [13].

Pooling

Pooling helps the model become robust against smaller translations in the input image. That is, by shifting the image by a small amount, the output will remain the same. Pooling can also be combined with down-sampling or as described, strided convolutions. We then space the pooling units N pixels apart instead of

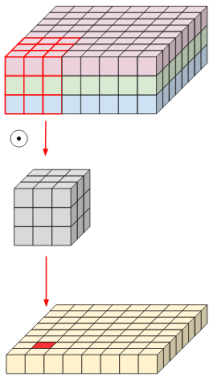

Figure 13: Regular convolution in 3D [15].

one pixel. There are different types of pooling, such as average, max and global pooling. See max pooling in Figure 15, which outputs the highest value of its input. Global average pooling will, however, take the average of the input, and instead of a smaller pooling size, it will do this on the full input.

Besides helping with computational efficiency, max-pooling will also extract the dominant features [13]. The paper *Flexible, High Performance Convolutional Neural Networks for Image Classification* by Ciresan et al. found that *"max-pooling can lead to faster convergence, select superior invariant features, and improve generalization"* [12]. The use of convolution and pooling can, however, cause underfitting if, for example, invariance is not desired for a problem where an exact position is relevant [7].

Depthwise Separable Convolution

Depthwise Separable Convolution (DWS) was first used in network design 2014 and has gained more popularity since it was included in machine learning library TensorFlow [17]. DWS is a method used in models to significantly reduce the number of parameters in the model while achieving the same results as a convolution layer [18]. DWC is used in multiple model architectures for image processing, such as Xception, GoogleLeNet and MobileNets [19]. Given an image with three channels, the ordinary convolution kernel will be a three-dimensional cube as seen in Figure 13. The kernel convolves over all three channels at once and produces a single value in the final output. DWS, on the other hand, splits this method into two steps: a depthwise convolution (DW) and a pointwise convolution (PW). These steps are visualized in Figure 16. First, the depthwise convolution is performed by separating the channels and doing the convolution separately with a $N \times N \times 1$ kernel. Secondly, the result from the DW is stacked and we do a pointwise $1 \times 1 \times 1$ convolution across the channels. The DW focuses on spatial relationship modeling and PW is for the cross-channel relationships [20]. By splitting the convolution into these two steps, we can drastically decrease the memory and computation usage (fewer parameters and operations) [18]. Possible drawbacks of using DWCs instead of convolutions is when the model is too small as the decrease in parameters will then underfit the data.

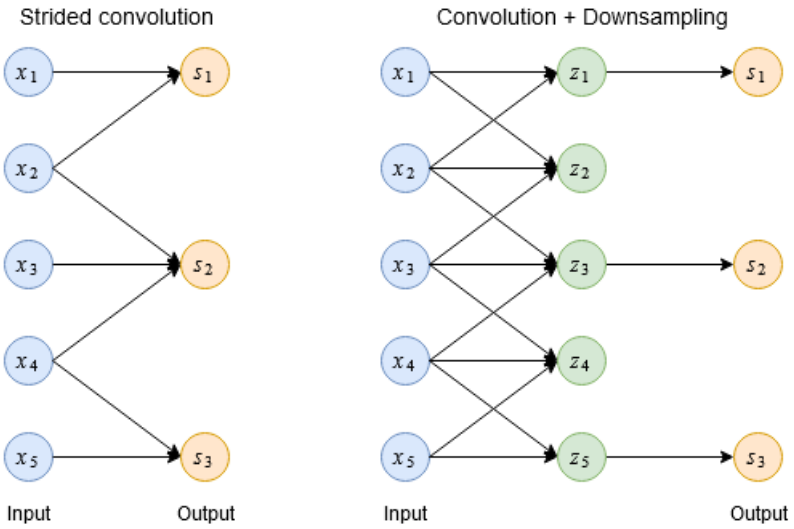

Figure 14: Strided convolution vs convolution and down sampling. Both methods lead to the same output.

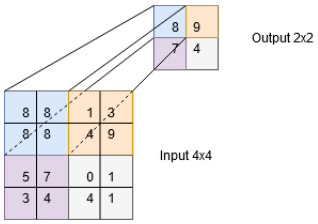

Figure 15: Max pooling the input with stride 2 resulting in output.

3.2 Xception Model

A classifier proposed by Google inc. in 2016 is *Xception* which stands for Extreme Inception. The Xception model has been superior to other current state of the art classification models such as Inception V3, ResNet-152 and VGG-16. It modifies the depthwise separable convolution operation by shifting the order of the convolutions, i.e. first the pointwise convolution and then the depthwise convolution. The creator of Xception, François Chollet, claims this change will not make a difference as the steps are stacked. In Xception, unlike Inception, this operation is not followed by an intermediate ReLU non-linearity [17].

The architecture of Xception can be seen in Figure 17. It consists of 36 convolutional layers. The layers called separable sonvolution are the modified depthwise separable convolution. The input data goes through the entry flow, repeats the middle flow eight times before exiting through the exit flow. All Convolution and separable sonvolution layers are followed by batch normalization [17]. Batch normalization layers are a way to make normalization a part of the model architecture and will thus normalize each mini-batch. The layers also allows higher learning rates [21].

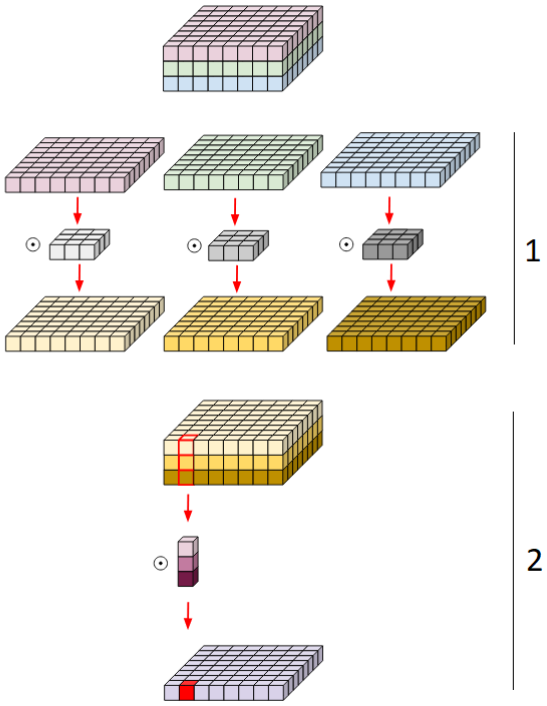

Figure 16: Depthwise convolution followed by pointwise convolution over three channels with filters sized 3×3. Together they form depthwise separable convolution [15].

3.3 Principal component analysis

Handling large data sets requires a lot of computational power combined with large memory usage. Several methods aim to reduce data dimensionality with the goal to keep as much information as possible. Dimensionality reduction is often used in image analysis when the image needs to be compressed but still retain as much of the information contained in the full-size image as possible. There are different ways to perform dimensionality reduction. Some common techniques are principal component analysis (PCA), autoencoders and variational autoencoders (VAE).

PCA is one of the simplest, yet effective, dimensionality reduction methods. This reduction is done by finding the correlation between features in the data. Features that highly correlate can be removed to reduce the overall dimensionality of the data [22].

PCA is performed by first normalizing the data, i.e. by transforming each data point to 0 mean and unit variance. Secondly, the covariance matrix is computed using singular value decomposition (SVD). Given an arbitrary matrix X with dimensions $m \times n$, X can be expressed using SVD as:

$$X_{m \times n} = U_{m \times m} S_{m \times n} V^T_{n \times n}, \tag{2}$$

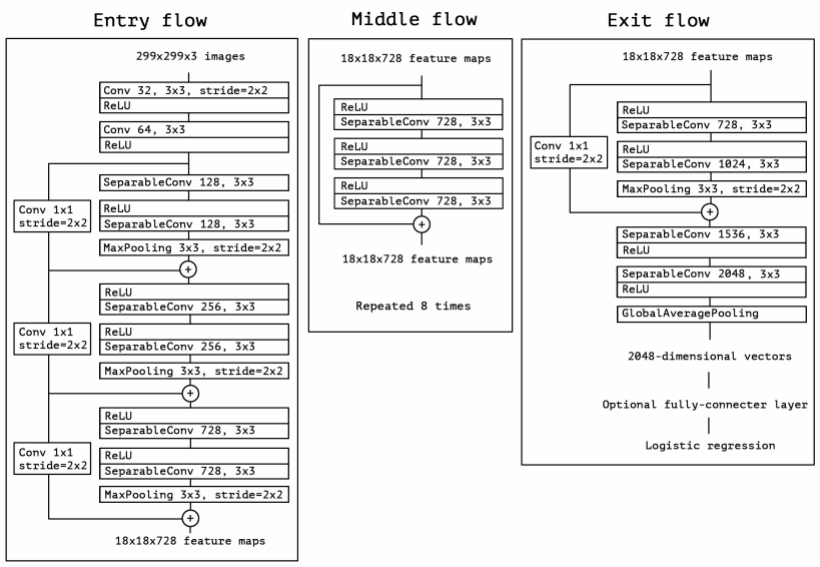

Figure 17: Xception architecture. Image taken from [17].

where

1. U, V are orthogonal, i.e $U^T U = I_{m \times m}$ and $V^T V = I_{n \times n}$.
2. S is a diagonal matrix.

The result of this factorization is the matrix U containing the left-singular vectors, S containing the corresponding eigenvalues and V^T containing the right-singular vectors. Here, S is a bi-linear operator between the two space spanned by U and V^T. This means that S can map a vector in U to V^T by $v^T = u^T S$ and a vector in V^T to U by $u = Sv$.

From Equation (2) we get the covariance matrix:

$$C = X^T X = V S^T U^T U S V^T = V S^2 V^T.$$

We can then find the eigenvalues in S and the corresponding eigenvectors in V. Since the goal of this method is to reduce the dimensionality, we can reconstruct X using the most important data in S and V, by multiplying X with the L first columns in V. This results in the lower dimension matrix P.

$$P_{m \times L} = X_{m \times L} V_{n \times L}. \tag{3}$$

Choosing a higher L will result in a P which becomes more similar to X but the trade-off is that P also becomes bigger with a bigger L. Sometimes L is chosen for P with a specific size. It could also be selected to get a P which contains a specific percentage of the information contained in X [23].

3.4 Autoencoder

An autoencoder can be divided into two parts; an encoder and a decoder. Both the encoder and the decoder are set up as ANNs, see an example of this structure

in Figure 18. Each time data is fed to the encoder-decoder model, the output is compared to the original input and the error is computed using MSE and the gradients are computed using backpropagation. Then the weights of both the encoder and decoder are updated to provide better reconstructions of future data [7].

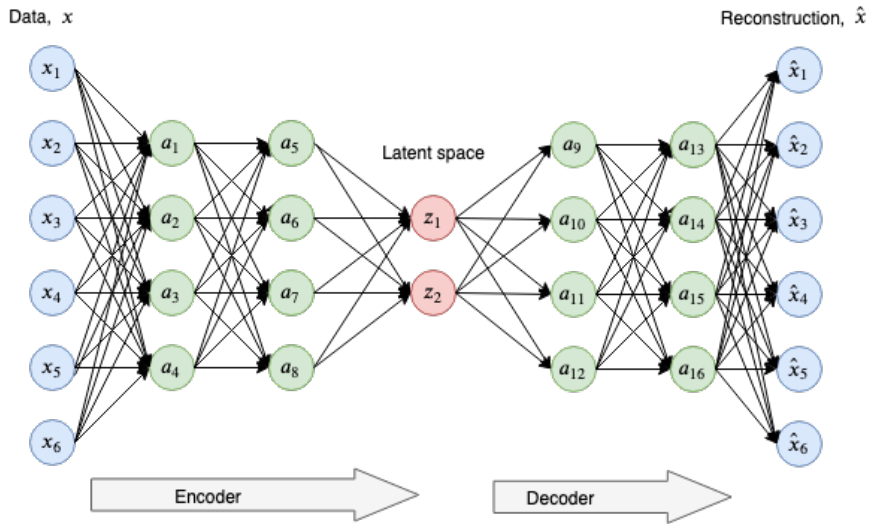

Figure 18: A possible architecture of an autoencoder with latent dimension = 2. The inputs x_i are encoded to the latent representation z_1,z_2 and then decoded to the reconstruction $\hat{x_i}$.

The input data $X = \{x^i\}_{i=1}^N$ is compressed into an L-dimensional latent vector $z = \{z_i\}_{i=1}^L$ by the encoder. This is done by either dense layers or convolution layers which reduces the dimensionality of the data. The vector z is then fed to the decoder which will try to reconstruct X with its estimate $\hat{X}$. The goal is to minimize the difference between X and $\hat{X}$, i.e. to minimize the function:

$$\text{loss} = \|X - \hat{X}\|^2. \tag{4}$$

With a larger sized latent vector z, more information about the input is passed to the decoder which makes it easier to reconstruct the original input. Several factors affect the loss, but the two key ones are the architecture of the network with the size, L, of the latent vector z. A bigger and better network is more likely to produce a better latent vector and thus a smaller loss.

3.5 Variational Autoencoder

The main difference between a variational autoencoder (VAE) compared to an ordinary autoencoder is that instead of the input being mapped to a latent vector z, it is being mapped to a set of distributions. The architecture of a VAE is illustrated in Figure 19. VAE is often used as a generative model, i.e. to create new, unseen data. The model can capture dependencies in any input data following some distribution and create output data which approximates the same

distribution as the input. This also makes VAEs suitable for dimensionality reduction since the lower dimension data will approximately follow the same distribution as the original data.

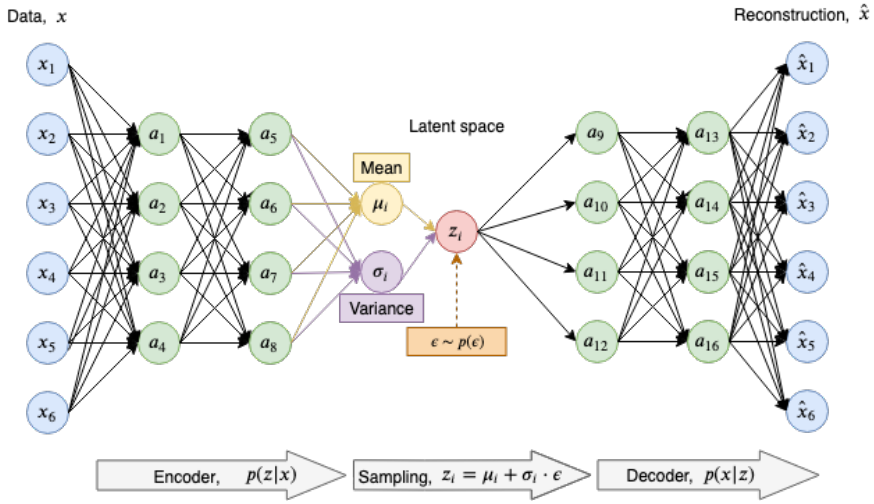

Figure 19: A possible architecture of a variational autoencoder. The inputs x_i are encoded to a mean μ_i and a variance σ_i. Then, the vector z_i is sampled using $z_i = \mu_i + \sigma_i \cdot \epsilon$ and is decoded to the reconstruction $\hat{x_i}$.

Given inputs $x = \{x_i\}_{i=1}^N$, where x_i is from the distribution $x_i \sim d(X|z)$ and a random variable z, X is encoded through several layers. The encoding is a function that maps the input to a lower-dimensional representation. In a VAE, the data is encoded into distributions approximating the input data's features. The encoder and decoder can be described in the following way:

- Encoder: $q_\phi(z|x)$, the distribution of z, given x as input.
- Decoder: $p_\theta(x|z)$, the distribution of x, given the encoded z as input.

ϕ and θ are parameters for the encoder and decoder respectively. The parameter ϕ contains the so called *variational parameters* which are parameters learned by the encoder [24].

It is difficult to calculate the true posterior $p_\theta(z|x)$ when using a VAE, i.e. given x, what is the probability of a particular z? This can be expressed using Bayes Theorem [25]:

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)} = \frac{p_\theta(x,z)}{\int p_\theta(x|u)p_\theta(u)du}. \tag{5}$$

The problem is that calculating the posterior will involve the intractable denominator $p(x)$. Instead, we try to approximate $p_\theta(z|x)$ using the tractable (often Gaussian) distribution $q_\phi(z|x)$. The VAE works by tuning the parameters of $q_\phi(z|x)$ to make it as similar to $p_\theta(z|x)$ as possible. The two distributions are compared to each other using the Kullback-Liebler divergence term.

Kullback-Liebler divergence term

Given two probability distributions q and p, the Kullback-Leibler (KL) divergence measures how well q approximates p. The KL divergence is defined as [26]:

$$D_{KL}(p(x)||q(x)) = \sum_{i=1}^{N} p(x_i) \cdot \log \frac{p(x_i)}{q(x_i)}.$$

This is a term used to measure how much information is lost when approximating the distribution $p(z|x)$ with the approximation $q_\phi(z|x)$. We also use the KL divergence to approximate the variational parameters ϕ^*, i.e. by minimizing [27]:

$$\phi^* = \arg\min_{\phi} D_{KL}[q_\phi(z|x)||p_\theta(z|x)].$$

Loss

The loss function of a VAE depends on two parts:

1. Reconstruction term: $E_{z\sim q_\phi(z|x)}\left[\log\ p_\theta(x|z)\right]$
2. Regularization term: $D_{KL}[q_\phi(z|x)||p_\theta(z|x)]$

The idea of the reconstruction term is to make the reconstruction as good as possible, i.e to make $x = \hat{x}$. The regularization term is there to regularize the latent space in order to make the approximated $q_\phi(z|x)$ as close to the true posterior $p_\theta(z|x)$ as possible [27].

The VAE tries to learn one distribution for each feature. A latent dimension $L = 50$ requires the VAE to fit 50 feature distributions inside $p_\theta(z|x)$. A high KL divergence will lead to a non-continuous latent space which increases the risk of sampling data outside a latent distribution resulting in ambiguous decoding. However, by only use the KL loss, the distributions would become overlapping and result in a high reconstruction loss. This process is visualized in Figure 20.

3.6 Self-Organizing Maps

The Self-Organizing Map (SOM) was created in 1982 by Teuvo Kohonen and is a way to visualize multidimensional data in lower dimensions, usually two dimensions. This provides a way to classify and cluster data in an unsupervised manner. The SOM both compresses data whilst also producing abstractions based on the key topological and metric relationships in the original input. Kohonen describes the SOM formally as a: *"nonlinear, ordered, smooth mapping of high-dimensional input data manifolds onto the elements of a regular, low-dimensional array"* (p.106) [28].

An example of a SOM is shown in Figure 21. It was built with the `MNIST handwritten digit database` which contains images of handwritten digits as seen in Figure 22 [29]. There are 720 images of numbers between 0 and 3. We can then see how digits 0, 1, 2 and 3 are assigned a coordinate and some coordinates contain multiple different digits in Figure 21.

The SOM is an ANN with a feed-forward structure where the input layer is fully-connected to the grid of nodes (the SOM), as seen in Figure 23. The

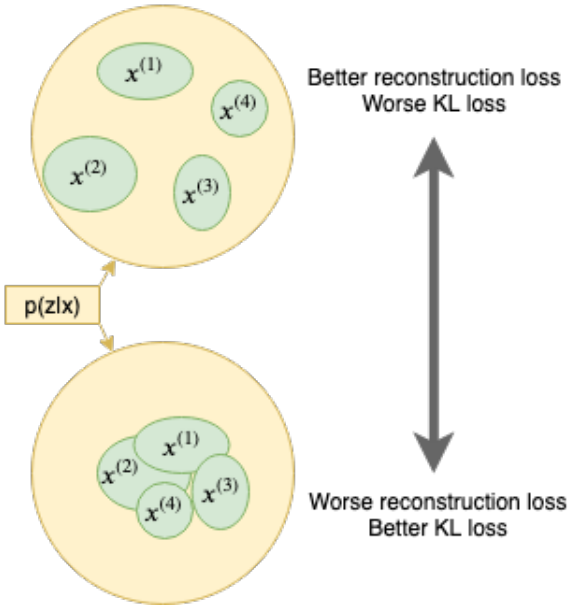

Figure 20: Tradeoff when training the VAE. The reconstruction term tries to separate the feature distributions, $x^{(i)}$, resulting in less overlapping and easier reconstruction. The KL term tries to regularize $q_\phi(z|x)$ to be closer to the true posterior $p_\theta(z|x)$. This could create more overlapping distributions and a more difficult reconstruction.

SOM uses competitive learning where different neurons in the network specialize in different input types. That means, for each input x, only one neuron will win. There is also an ordering between the neurons, they are placed on a discrete lattice, the SOM, which allows the winning neuron to also affect its neighboring neurons. Thus, neighborhoods will eventually learn to specialize in different types of input which will lead to the representations being ordered on the map [30].

The eventual visualization of the 2D SOM is a grid with nodes, m (also known as neurons and units). When training the SOM, the nodes that are (topographically) close to each other (up to a certain distance) will learn from the same input x by activating each other. The learning process can be described by the following equation:

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)], \tag{6}$$

where t is the coordinate ($t = 0, 1, 2...$) and h stands for the neighborhood function and acts as a smoothing kernel. One example of a simple rectangular topological neighborhood is given in Figure 24. We denote the nodes in the neighborhood of node c as N_c. We then have the following neighborhood function, where $0 < \alpha < 1$ is a learning rate factor.:

$$h_{ci}(t) = \begin{cases} \alpha(t), & \text{if } i \in N_c \\ 0, & \text{if } i \notin N_c. \end{cases} \tag{7}$$

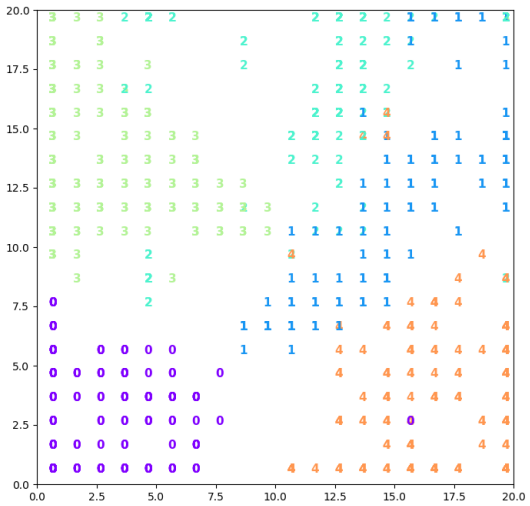

Figure 21: A 20×20 SOM built using 720 images from the `MNIST` data set. The numbers represent the original label of the digit image.

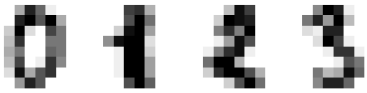

Figure 22: Four different handwritten digits from MNIST.

Both α and the radius of N_c, σ, decrease as time progresses [28]. σ starts larger to ensure a global ordering of the map [30].

The basic SOM algorithm has two steps, described by [32], which are repeated for each input in the training phase.

1. Assign a winning node to the input. The index of the winning node m_c is given by $c = \arg\min_i \{d(x, m_i)\}$, where $d(x, m_i)$ is usually the Euclidean distance between input x and node m_i, i.e. $d(x, m_i) = \|x - m_i\|^2$.

2. Modify the nodes according to Equation (6), such that only neighbors are affected.

This stochastic version differs slightly from the batch algorithm. The batch algorithm instead processes all data in step 1 and then updates in step 2 based on a generalized mean [33]. The batch algorithm can converge faster than the stochastic one, but it is also critical to select the correct initial values. Therefore it is not certain that the batch algorithm is faster, due to the need for repeating experiments to find these parameters [32]. The stochastic algorithm will result in a better topology preservation [33].

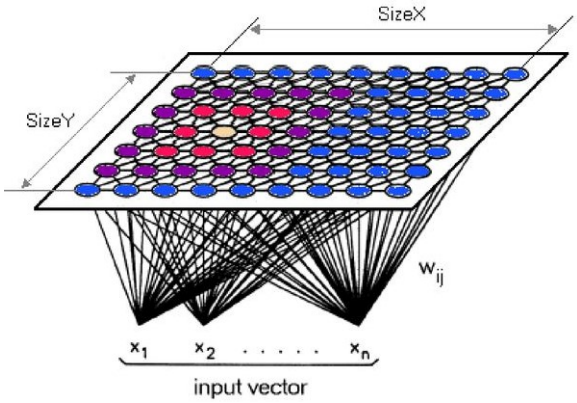

Figure 23: The structure of a SOM, with the input layer being fully connected to the lattice of neurons and each connection is given a trainable weight. Image taken from [31].

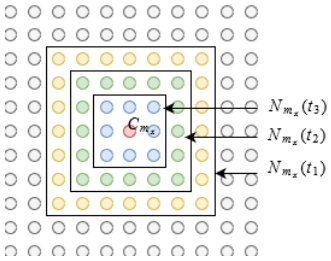

Figure 24: A rectangular neighborhood function on the SOM lattice. If a neuron is within N_{m_x}, it will be impacted when C_{m_x} is the winning neuron.

MiniSom

The package `MiniSom` is available on GitHub and created by Giuseppe Vettigli. Included is both an implementation of a self-organizing map in Python 3.6.2 using NumPy and some visualization functions. Vettigli describes it as a minimalistic implementation of a SOM [34]. `MiniSom` does not contain optimization that suits larger data sets and as such it is quite slow compared to other implementations.

Parameters

The parameters to select when building a SOM with `MiniSom` is the size, number of iterations, radius of the neighborhood function (σ) and learning rate (α).

The size of the `MiniSom` is the side length of its visualization grid. Selecting 100 will result in a 100×100 map. If the size is too large, the map will take more time, computational power and memory to train. Too small of a map will generalize too broadly and each coordinate will contain many images. A 3×3 map for 10 000 images will have about 1000 images at each coordinate. Samuel Kaski describes in *Data Exploration using Self-Organizing Maps* that the size of the SOM should be as large as the computational resources will allow, given that

there is an unlimited number of training data. If the training data is limited, a rule of thumb proposed is to set the number of nodes to the size of the input data [30].

A general rule for setting the number of iterations is that it should be at least 500 times the number of units [28]. So, the least number of iterations for a 3×3 map is $500 \cdot 3 \cdot 3 = 4500$. Again, it is also a question of computational resources.

Another parameter to set is the radius of the neighborhood function, σ. To produce a map that is globally ordered, σ needs to be very wide initially before it decreases. Kohonen writes that *"[t]he initial radius of N_c can even be more than half the diameter of the network!"*, where the referred radius is the value of σ (p.112) [28].

For randomly initialized values for the SOM, Kohonen claims the learning rate should initially be reasonably high. He, however, continues to note that to minimize the learning time, the learning rate value is crucial [28].

U-matrix

The *U-matrix* (unified distance matrix) is a way to visualize clusters in the SOM. It shows the average distance of a neighboring m_i, large distances are lighter (white) and small ones are darker (black) [28]. The white parts thus represent the borders of the clusters. The black parts, on the other hand, represent low distances and is thus a part of a cluster. These darker parts thus reflect the density of the input space. Looking at the U-matrix in Figure 25, we see some clearly defined clusters like 1 and 2. On the other hand, 3 and 4 have a weaker line between them. This gives a good measurement of where we have the best-separated clusters. It also differs from traditional clustering methods like K-means where a shape needs to be assumed. As Samuel Kaski points out, the SOM is equivalent to a regular clustering technique if the neighborhood function is always zero [30].

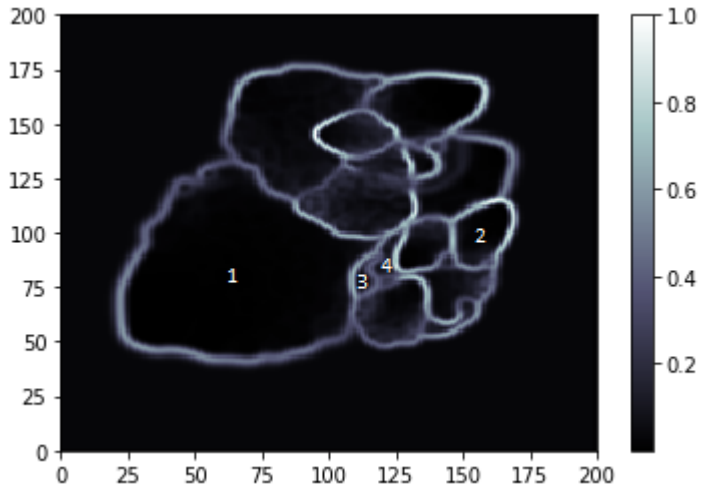

Figure 25: A U-matrix, which shows the average distance of a neighboring unit. Light colors mean large distances.

Outlier

Outliers (outlying observations) are so-called because they seem to differ significantly from the other samples in the data set. They can occur due to errors in data collection in which case they are outliers that should be removed. Outliers can also occur as an extreme value caused by the data's inherently random nature. This type of outlier should, however, be retained in the data set [35].

With the SOM, outliers only affect one node and its neighborhood. Hence, the outliers will not affect the global ordering of the map and therefore not affect the integrity of the other data. Also, the outliers can be detected with the U-matrix as there will be few representations in those neighborhoods due to outliers differing nature [30].

3.7 Average Image Hashing

Average image hashing is a method to hash an image to, for example, compare images to each other. Dr.Neal Krawetz describes the algorithm of average hashing as the following [36]:

1. Downscale the image to 8×8. Using Figure 26a as the original image will result in Figure 26b.
2. Convert the image to grayscale using the *ITU-R 601-2 luma transform* (`PIL.Image.Image.convert('L')`) [37], see Figure 26c.
3. Calculate the mean intensity of the image.
4. Binarize the image, one if the pixel value is higher than the mean, zero if it is lower. See Figure 26d.
5. Calculate the 64-bit hash based on these 8×8 binary numbers.

These steps are also shown graphically in Figure 26a-26d from original image to bit image. The resulting average hash from the original image is `fbfd040627ecfc7a`. If we rotate the original image with one degree, we get the average hash `dbfd040627acfc7a`. This hash differs by two bits (see the colored ones), so they have a *hamming difference* of 2.

(a) Original image. (b) Step 1. (c) Step 2. (d) Step 4.

Figure 26: Step 1,2 and 4 from the average image hashing algorithm performed on the original image.

4 Methodology

The method consists of building and evaluating three parts of a pipeline. These are shown visually in Figure 27. Before entering the pipeline, the image is transformed and resized. First, that image is sent through the classifier. The classifier outputs the probability of the cell image being one of 19 classes. By breaking the classifying model earlier, we can instead get a feature vector for each image. Secondly, that feature vector of size 12544×1 will be the input for the VAE. By sampling from the latent space, we get a smaller vector of size $L \times 1$ ($L < 12544$). Finally, that vector is assigned a neuron by the SOM and placed on the map in two dimensions. To verify the results, different evaluation techniques are used. A further description of these parts will follow later in this section.

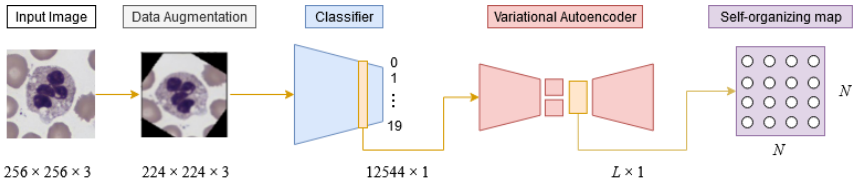

Figure 27: A summary of the techniques used in this papers pipeline.

4.1 System Overview

We had access to two stationary computers with NVIDIA GeForce GTX 1660 Ti having 6 GB memory each. Also, another computer having two NVIDIA 2080 Ti was sometimes available. This computer was used for the more computational heavy training, whereas the two stationary computers were used for day to day work. The GPU's used CUDA toolkit 10.0 which is a GPU acceleration application together with NVIDIA Deep Neural Network library 7.6 (cuDNN). As a deep learning library, Keras together with TensorFlow 2.0 and Python version 3.7 was used on all computers.

4.2 Data

CellaVision has assigned each WBC class the index shown in Figure 6. The images used to train the various networks are cell images captured by CellaVision's products. The images have been normalized to have the same background intensity. The original size of an image is $640 \times 480 \times 3$, but is cropped to $256 \times 256 \times 3$ pixels centered around the nucleus. Each image is captured with a lens with $100\times$ magnification resulting in each pixel being 0.1μm. Some of the cell images appear multiple times in the data set as CellaVision's machines have slightly different optics. Different machines of the same model can produce slightly different images and some cells are captured more than once by the same machine. This means that the same cell image can reoccur with only some small difference (e.g. in color and translation) compared to the other samples of the same image. These are so similar that they are viewed as *duplicates*. See Figure 28 for two duplicate cells.

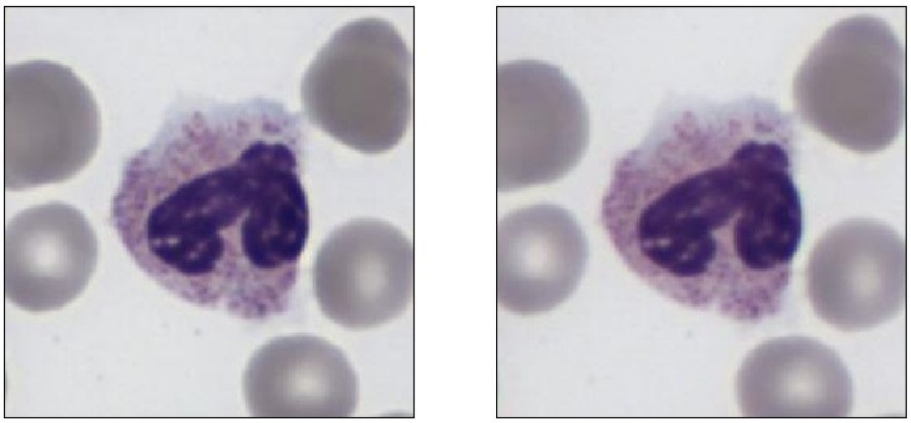

Figure 28: Example of an image and its duplicate.

For the adaptive reference cells, it is important to remove these duplicates from the data set. As these duplicates have some translation and color differences, it is not enough to compare them pixel by pixel. Thus, the average image hashing from the package `imagehash` [38] was used. The images in the same class are then compared by taking the average hash of both images and comparing the Hamming difference (bit-wise difference). If this difference is lower than two, they are believed to be duplicates. If the difference is three, the images were manually controlled.

The graph in Figure 29 shows the number of cells per class and the data sets' division into a test, train and validation (val) set. Duplicates were not removed from the train set as it does not directly impact the adaptive reference cells. Thus creating five different subcategories of images: test, test without duplicates, val, val without duplicates and train. The distribution can be found in Table 2. There are duplicates between the test and validation set. Therefore, the images in the test set that had a duplicate in the validation were removed from the validation set. The validation set is hence smaller than the test set. We will refer to the test set without duplicates as the *trainSOM* set and the validation set without duplicates as the *testSOM* set.

Table 2: Data distribution of the cell images provided by CellaVision.

	Percentage (%)	# images	# images without duplicates
Train	60	242 109	-
Test	20	80 744	61 995 (trainSOM)
Val	20	80 764	35 533 (testSOM)
Total:	100	403 617	97 535

Normalization

The system should weigh all features equally. A non-scaled feature vector can have some feature values consistently much higher than others. This will cause the machine learning algorithms to weigh greater values higher and smaller

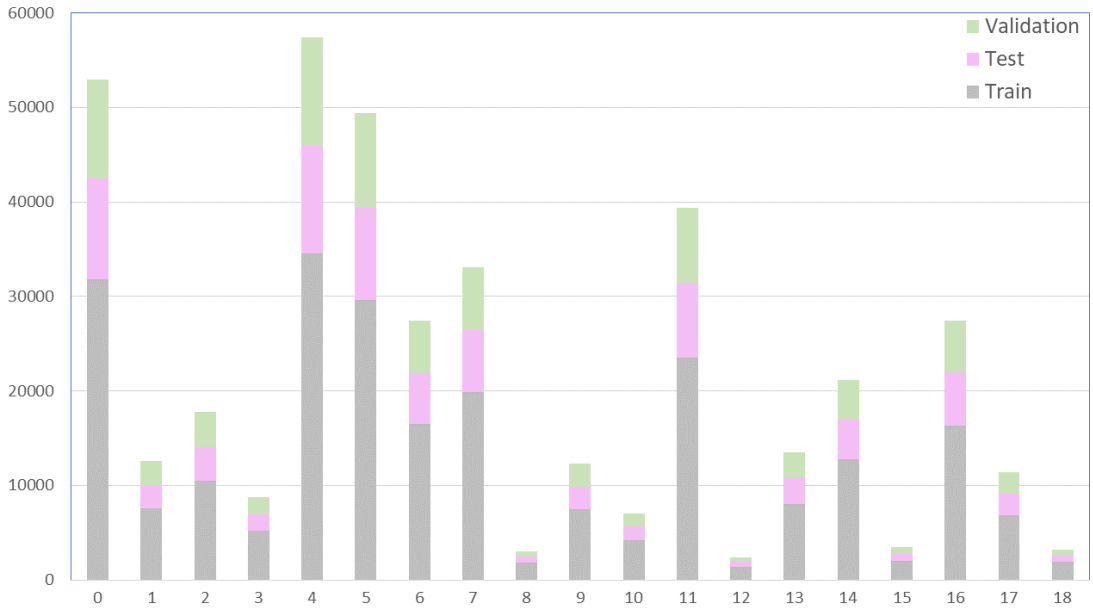

Figure 29: The distribution of the data set in each class. The graph also shows the distribution when the data is split into test, train and validation.

values lower. Also, as the data consists of images where each pixel value is a number ranging from 0 to 255, each image also needs to be scaled. The images are scaled such that each pixel instead ranges from 0 to 1.

4.3 Classification

Data Augmentation

An important part of the system is the classification. Before sending the data through the classifier, Keras is used to augment some of the data. The augmentation is performed to train the classifier on a more diverse data set. Several types of augmentation were relevant for the training. Three different types of augmentation applied to the same model were evaluated: A, B and C. They will be referred to as model A, model B and model C. The augmentations used for each model is seen in Table 3.

Table 3: Augmentations for model A, B and C.

	Model A	Model B	Model C
rescale	1./255	1./255	1./255
zoom_range	0.1	0.1	
rotation_range	90	90	
fill_mode	constant	constant	
hotizontal_flip	True		
vertical_flip	True		
width_shift_range	0.1		
height_shift_range	0.1		

The `rotation_range` setting will rotate the image a random number of degrees between `-rotation_range` and `rotation_range`. This will make the model more invariant to a smaller rotation of the cell image. The `zoom_range`

will zoom the image a factor between $[1 - \texttt{zoom\_range}, 1 + \texttt{zoom\_range}]$. When rotating the image, blank areas occur. The blank areas are filled with black pixels when `fill_mode` is set to "constant" [39]. These areas could be filled with any color since the idea is that the classifier should learn that these areas do not affect the final classification and therefore ignore them. An example of an image from CellaVision's data set with those random transformations applied is shown in Figure 30.

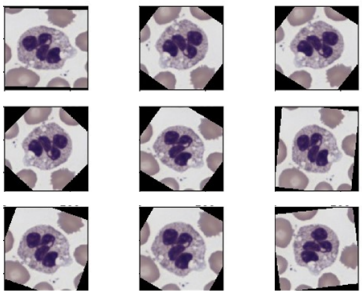

Figure 30: A cell image with 9 random combinations of transformations from Table 3 applied.

Model Architecture

The model used in this paper is a modified version of the Xception model described in Section 3.2 and seen in Figure 17. The middle flow is repeated three times instead of eight. The adjustment is due to previous successful work within CellaVision and our training time restrictions. The network is trained using the training set containing 242 109 images and the augmentations described. Other settings being used are listed in Table 4.

The goal is to break this model at an appropriate layer to send a feature vector through that is not only dependent on the technologists' classification nor the information surrounding the WBC. That is, we want feature vectors that contain information about visual similarity and not just class information. Also, we want feature vectors that focus on the WBC and thus ignores the RBCs and miscellaneous artefacts.

Table 4: Settings used when training the modified Xception network. Note that the `target size` rescales and does not crop the image from 256×256 to 224×224.

Settings	Value
`optimizer`	Adam(default learning_rate=0.001)
`batch size`	32
`epochs`	50
`target size`	(224,224)

The classification focuses on the WBCs, but the cell images also contain RBCs, other cells and sometimes different artifacts. This surrounding information is supposed to be marginalized out by the classifier. Towards the end of

the classifier, the model hones in on the specific features of the WBC (RBCs e.g. does not impact a WBCs classification). But selecting a layer that is close to the output resulted in a feature vector highly weighted against the final classification.

The breakpoint seen by the red line in Figure 31 was chosen, which is the sixth last layer. It is a Batch Normalization layer with an output size of $256 \times 7 \times 7$. This layer's output for a given cell image can be seen in Figure 32. These feature vectors will primarily contain information about how the input image looks visually and not rely heavily on the final classification. After breaking the model and extracting features from this Batch Normalization layer, the features are flattened such that an image becomes a vector of length 1×12544. Going further up (closer to the input) in the model and extracting feature vectors would produce feature vectors containing more general information such as borders, size, color and rotation. However, this information does not address internal structures in the WBC, which makes it irrelevant for finding visually similar WBCs. Figure 33 shows an example of the output from such an early layer in the model. Additionally, a layer earlier in the model resulted in vectors too big for the computers working memory to handle.

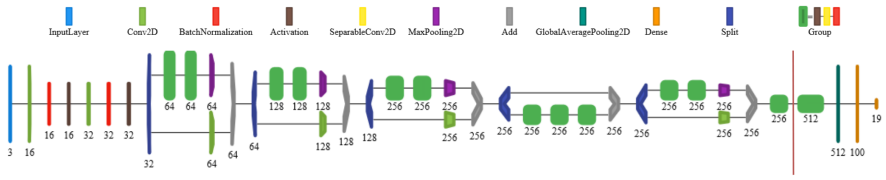

Figure 31: The modified Xception model. The red line shows the break point. Created with the tool `Net2Vis` [40].

This results in an output set with dimensions 242109×12544. Hence, each image is reduced from $256 \times 256 \times 3$ to 1×12544. This is still quite a large representation and is hard to cluster and compute with. Therefore, the next step is to reduce the size of each feature vector but still keep as much of its information as possible. For this purpose, a VAE is used.

4.4 Variational Autoencoder

To select which method to use for dimensionality reduction, both PCA and a VAE were tested. Performing a PCA on the 1×12544 feature vectors and keeping just the 50 most relevant components, i.e. $L = 50$ in Equation (3), resulted in a data loss of 44%. This number indicates that PCA does not work well on our data as almost half of the information will be lost.

The VAE architecture proposed by Brian Keng, Adjunct Professor of Data Science at the Rotman School of Management, University of Toronto, was used [41]. The architecture contains multiple layers and given the large, rather complex data set, this was preferred. First, the VAE is trained using the feature vectors derived from the train data set. Secondly, it is used to produce new and smaller feature vectors using its encoder-decoder structure. The VAE's architecture can be seen in Figure 56 and 57 in the Appendix. Different output sizes of the VAE were tested and evaluated by the SVOMEN measurement (described

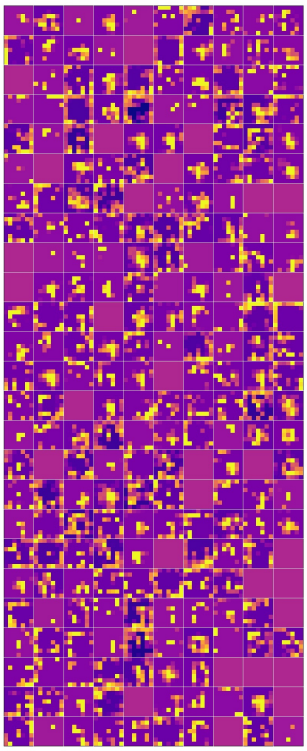

Figure 32: Output from the 6th last layer for a given cell image.

in Section 4.7). Since the VAE is not used in a generative matter, the mean of each input's feature distribution is used as the resulting feature vector.

Having smaller feature vectors allows for faster computations and less required memory. Feature vectors can thus be compared to each other, an integral part of how the adaptive reference cells are computed. In order to visualize how the feature vectors are related to each other, they are passed to the SOM.

4.5 Self-Organizing Maps

A new implementation of the SOM was not written as Kohonen recommends using a previously implemented SOM in practical applications due to the many pitfalls in building it without expert knowledge [28]. It is possible to select how to initialize the SOM weights. The SOMs weight initialization can be selected randomly, or by spanning the first two principal components. By selecting the second method, the weights should converge faster on linear data. Random initialization is preferred on non-linear data. We thus chose random initialization.

The input to the SOM is 61710×50. The first dimension is the number of feature vectors deriving from the trainSOM, as presented in Table 2. The size of a feature vector is dependent on the latent space size of the VAE. Each input vector is normalized by subtracting the mean and dividing by the standard deviation. It is not crucial to normalize the data, but according to Kohonen, it may improve the numerical accuracy [28]. Batch learning, i.e. all data is

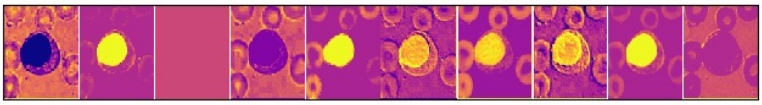

Figure 33: The first activation layer for a given cell image in the trained model for class 4.

read at once sequentially, is selected for performance reasons. After the SOM is trained, the testSOM is plotted on the SOM for visualization of the input. The map is also made interactive such that the user can press a coordinate and a folder containing the images placed at that coordinate are shown. This enables exploration and visualization of the data.

Parameters

To set the parameters of the SOM, the parameters recommended by Kohonen were considered. The input data (trainSOM) is ∼62 000 vectors, thus the map should be about 248×248. The number of iterations should then be $500 \cdot 248 \cdot 248 = 30752000$, i.e. over 30 million. Next, the initial value of σ can be more than half the networks diameter, hence $\sigma = 124$ should be reasonable. The learning rate should be high, but no specific value is recommended as it depends on the other values. Using `MiniSom`, training a SOM with the recommended parameters would take over 22 days. Therefore, the first maps were built with significantly fewer iterations which in turn required σ and the learning rate to have an initial smaller value.

4.6 Reference Cells

To obtain reference cells that are adaptive to any given cell image, the whole pipeline is used. See the grid in Figure 34 for an example of the adaptive reference cells chosen for one cell image, each reference image has its class label in bold. The cell image is from the testSOM set, and its possible neighbours are selected from the trainSOM set. If a reference cell has a duplicate it is redundant to classify. This is a scenario that will not occur at the laboratories using CellaVision's system.

The neighbouring cells are either selected based on the VAE or SOM. For the VAE, the feature vectors that are sampled from the latent space are compared to the feature vector of the cell image. The eight cells with the smallest MSE are selected and chosen to be the reference cells. For the SOM, the neighbouring cells are instead based on the closest images based on the coordinates on the SOM. The images in the same coordinate are primarily selected, but if there are more than eight neighbours in the same coordinate, the ones with the smallest quantization error to the node are selected. If there are less than eight neighbours in that coordinate, neighbours from the coordinates closest, by Euclidean distance, are sampled for additional neighbours.

As an evaluation method, adaptive reference cells were produced using both the SOM and the VAE. Both correctly classified and misclassified images by the

modified Xception network were tested. The correctly classified images determined which method, using the output from the SOM or the VAE, produced the best output. This best method was then used with the misclassified images to show an expert.

There were 2362 cells that the modified Xception model misclassified from the testSOM set. These images are interesting since they are hard to automatically classify. Therefore the misclassified images are those primarily helped by using the adaptive reference cells. By selecting an even distribution of 10 images of each cell class, a set of 186 images were obtained. Class 13 only had six images misclassified which are why the set is not complete with 190 images. Two test people, Anna and Oscar, without knowledge of blood morphology classified the cell image using only its adaptive reference cells. Also, a medical technologist classified the same cells with adaptive reference cells to get a more accurate classification as the ground truth. Some cells in the data set can be misclassified by the expert which would lead to some of our current labels being wrong. Having a medical technologist at CellaVision performing a thorough classification will verify the images and it will be classified with only one expert. The test persons' classification is then compared to the technologist's and the classifier's prediction.

4.7 Evaluation

`ROTMES`

The rotation measurement (`ROTMES`) is created to make sure the VAE is invariant to different transformations in the data. An image X is read and sent through parts of the pipeline from Figure 27 together with three transformed versions of itself $x_{\text{rot}}, x_{\text{hor\_flip}}, x_{\text{vert\_flip}}$. The first augmentation x_{rot}, comes from a rotated version of the input image, the second one $x_{\text{hor\_flip}}$ from a horizontally flipped and the third $x_{\text{vert\_flip}}$ from a vertically flipped version. The augmented images' feature vectors f_1, f_2, f_3 are compared to the feature vector of the original image f_X using MSE resulting in the errors $e_1 = \text{mse}(f_1, f_X)$, $e_2 = \text{mse}(f_2, f_X)$ and $e_3 = \text{mse}(f_3, f_X)$. Finally, `ROTMES` is computed by taking the mean of these 3 errors, i.e.

$$\texttt{ROTMES} = \text{mean}(e_1, e_2, e_3),$$

which tells us how well the system performs on augmented data. `ROTMES` is performed on 100 images from each class and is used to evaluate how robust the models are.

`SVOMEN`

To evaluate the performance of the pipeline and to be able to hyper-tune parameters, a measurement was created to measure the similarity of two cells. It was primarily developed to evaluate the SOM and called `SVOMEN`.

The test was performed on ten people that went through images manually, such as those in Figures 35-37. Two images were given, alternative 1 and 2, to compare the middle image (RI) with. If alternative 1 was the most similar, the key 1 was pressed. The same goes for key 2. There was also a third option, key 0 when no cell was deemed particularly similar. An even distribution of 100 images of each of the 19 classes was selected and picked at random for the

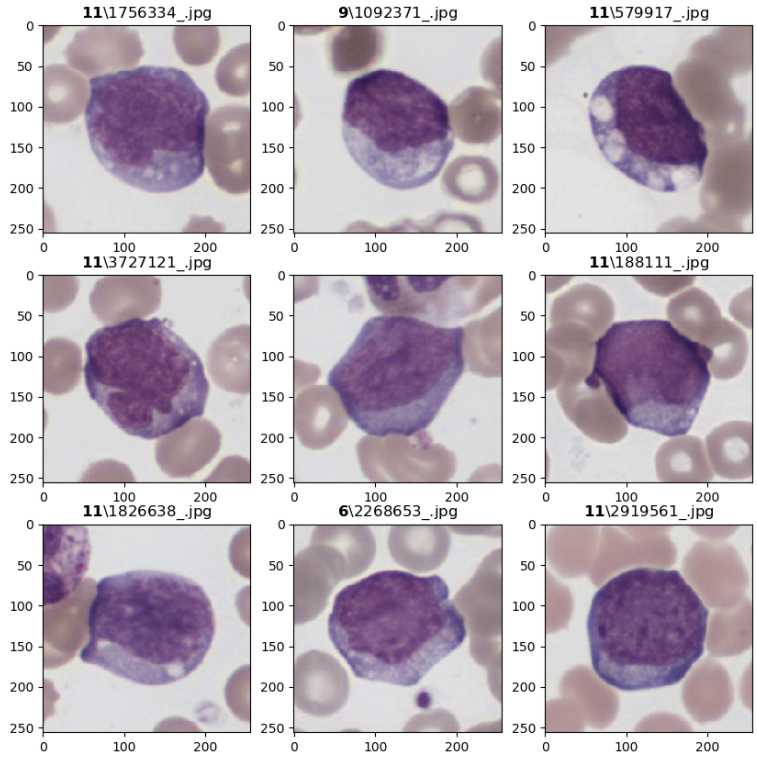

Figure 34: The cells surrounding the middle image is its adaptive reference cells. The number in bold represent the cell class. The middle image is classified as 11. The reference cells are found using the VAE's output data.

RI, totalling 1900 images. The first alternative was always selected by finding the eight nearest neighbours (smallest MSE) and selecting one at random. The other image, however, was chosen from three methods, selected at random.

- close-to-close: Selecting another image from the 8 as described, see Figure 35.
- close-to-class: A random image from the same class, see Figure 36.
- close-to-random: A completely random image, see Figure 37.

As can be seen in Figure 35, the choice is especially difficult when both alternatives are very similar. Besides presenting the options and to select 0 when they are unsure, no further instructions were specified. Ten people participated. Master thesis students with limited cell knowledge: Anna, Oscar, Oskar, Martin C, Hugo and Nellie. And employees at CellaVision: Martin, Sven, Jesper and Ida. The employees have more knowledge in the field but are not specialized

Evaluation test, 3 example images

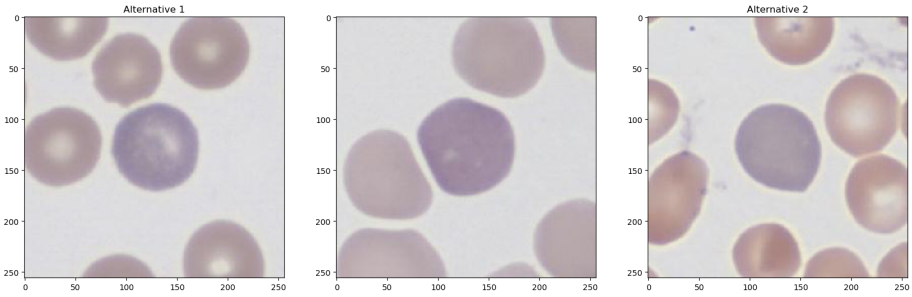

Figure 35: close-to-close: Both alternative 1 and 2 are in the RI eight nearest neighbours (smallest MSE).

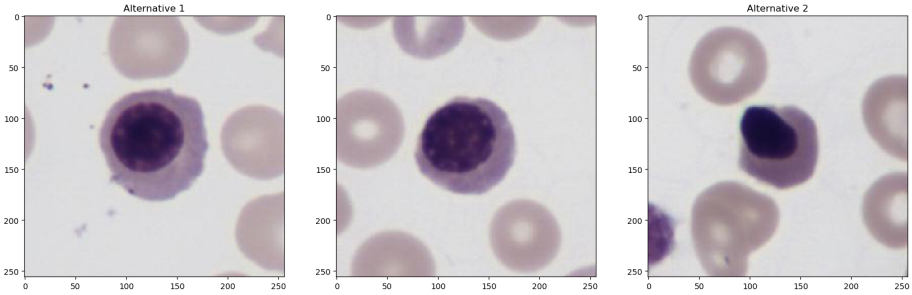

Figure 36: close-to-class: All images are in class 13, Alternative 1 is one of the RI neighbours.

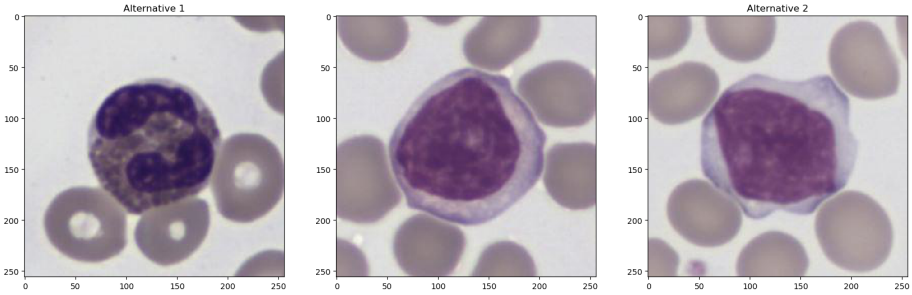

Figure 37: close-to-random: The RI and Alternative 2 are both class seven. Alternative 1, however, is class 2.

in cell classification. Table 5 shows each participants number of selections of alternative 0, 1 and 2. We see that alternative 0 is commonly used by Ida with 12.5% versus Hugo with 3.0%.

An entry was valid if eight participants selected either alternative 1 or 2. Some participants described how they on accident selected the wrong one, which is why not all ten had to pick the same alternative. Of the 1900 RI, 1262 remained. Figure 38 shows the remaining number of RI in each class.

When applying the `SVOMEN` to a SOM, each RI and its alternatives are placed on the map. One point is awarded if the image that is closest by Euclidean distance is also closest according to `SVOMEN`. The perfect score is thus the number

Table 5: Each participant in creating the gold standard for SVOMEN and their selections for the total 1900 images.

Name	Alternative 0	Alternative 1	Alternative 2
Anna	170	860	870
Oscar	149	871	880
Oskar	105	923	872
Martin C	204	828	868
Hugo	58	919	923
Nellie	79	885	936
Martin	106	871	923
Jesper	197	874	829
Ida	238	800	862
Sven	92	884	924

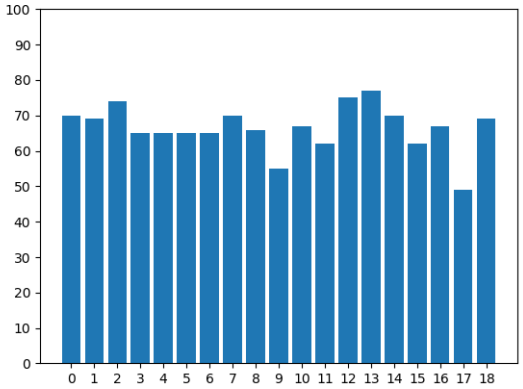

Figure 38: The number of valid images in each class for the SVOMEN measurement. There were originally 100 images in each class.

of valid RI, i.e. 1262. In the same way, applying SVOMEN to the VAE means to award a point if the image that is the most visually similar also has the smallest MSE.

5 Results

5.1 Classification

Confusion matrices from the models in Table 3 are found in Figure 53, 54 and 55 in the Appendix respectively. The ROTMES result of the models are found in Table 6. By evaluating the models with ROTMES and the confusion matrix, model B is chosen as the superior model. It has a ROTMES score which is higher than model A but smaller than model C. However, model A's confusion matrix shows that the model has lower accuracy on some classes than model B. The compromise of performing well on rotation invariance and achieving a good accuracy across the classes is thus model B. Therefore, all references to a modified Xception model from here on refers to model B.

Table 6: ROTMES for the models: A, B and C

Model A	Model B	Model C
0.424	0.808	1.197

The results achieved by the modified Xception classifier can be found in Table 7, and the training and validation loss in Figure 39 and 40 respectively. The high accuracy shows that the model has been trained to the extent that it can correctly classify a never before seen CellaVision cell image more than nine out of ten times.

Table 7: Xception model training results.

	Accuracy	Loss
Train	0.950	0.139
Validation	0.937	0.193

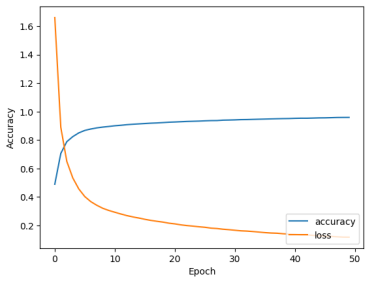

Figure 39: Xception train accuracy and loss.

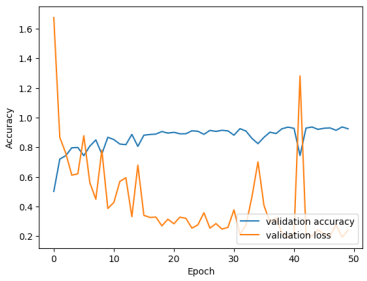

Figure 40: Xception validation accuracy and loss.

5.2 Dimensionality reduction

Shown in Table 8 are the different sizes of latent space L in the VAE and its respective loss and SVOMEN score. The lowest loss is when L is the largest (100). The highest SVOMEN score, on the other hand, occurs when $L = 50$. Since $L = 50$

also gave a low loss, this was selected as the best L and is here on referred to as the output size of the VAE. A visualization of the latent space for the VAE using a 2-dimensional latent space is found in Figure 41.

Latent space size (L)	Test loss	SVOMEN(%)
2	3097	44.4
10	2337	79.0
25	2378	80.4
50	**2279**	**82.4**
75	2299	81.2
100	2278	81.2

Table 8: The loss and SVOMEN score based on the size of the VAEs latent space size, L.

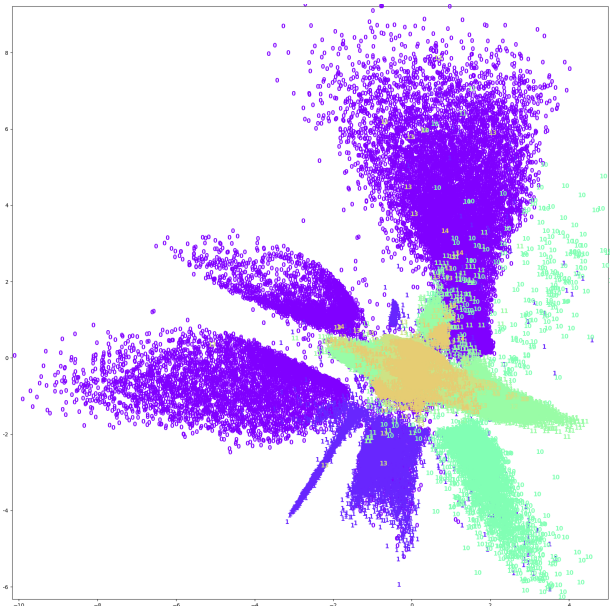

Figure 41: Two dimensional latent space of the VAE using the test data as input. Each cell class is represented by its index and color.

Training the VAE given a latent space size of 50 resulted in the loss plot shown in Figure 42. Notice that in the plot, the validation loss is lower than the training loss. It is caused by the dropout layers that are inactivated during the validation of the VAE. This was confirmed by experimentation. The number of iterations/epochs was 20 and batch size 32.

5.3 SOM

Different settings of the SOM were evaluated to find the parameters resulting in the best SOM considering the SVOMEN score. These settings and scores are shown in Table 9. The best SVOMEN score was given settings size 244×244, one million iterations, learning rate (α) 0.5 and neighbourhood radius (σ) 20. The

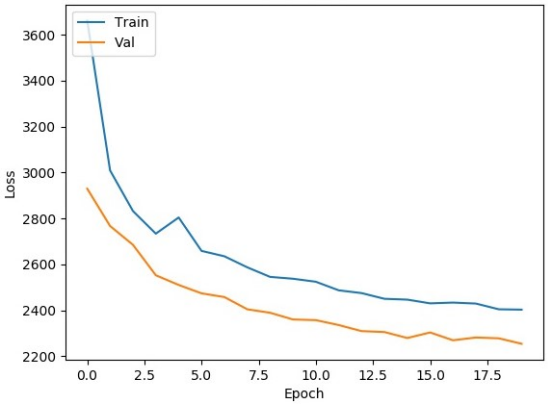

Figure 42: VAE loss as a function of the number of epochs. The blue line is the loss on the training data and the orange is on the validation data.

SVOMEN score was 75.0%, which is 946 correct ones. This will be referred to as the SOM.

Table 9: The settings of the SOM and its SVOMEN score.

Size	Iterations	α	σ	SVOMEN(%)
100	1000000	0.5	30	68.1
244	1000000	0.25	20	74.9
244	1000000	0.5	70	67.6
244	1000000	0.5	50	69.5
244	2000000	0.5	50	71.1
244	1000000	0.5	10	73.0
244	**1000000**	**0.5**	**20**	**75.0**
244	1000000	0.5	30	72.7
244	1000000	1	20	71.9

The SOM that was the best is shown in Figure 43. Each cell from the testSOM set is plotted at the winning SOM coordinate. The image is placed as a number, representing its class number labeled by the technologists, and a different color. Class 0 and 1 are in the same category, so they should be close in the SOM, which they are. Also, we can see that class 16 and 17 are close, which makes sense as they are both 'non-cells' (artifact and smudge). We also see clusters that are well defined and separated. However, there are areas with a large mix of different classes which is probably due to the cell images being an outliers and cannot be clustered with the other cells from that class. Cells that are out of focus could also be in these areas. The resulting U-matrix is shown in Figure 44.

Accompanying the SOM is a built visualization tool that allows the user to click a coordinate and see the cells placed there. This visualization is shown as an overlay of the SOM digits over the U-matrix. See the SOM overlay in Figure 58 in the Appendix. Figures 59-77 in the Appendix show the overlay of SOM digits on the U-matrix for each cell class. Also, the overlays show which cells were misclassified by the modified Xception network as these are in red.

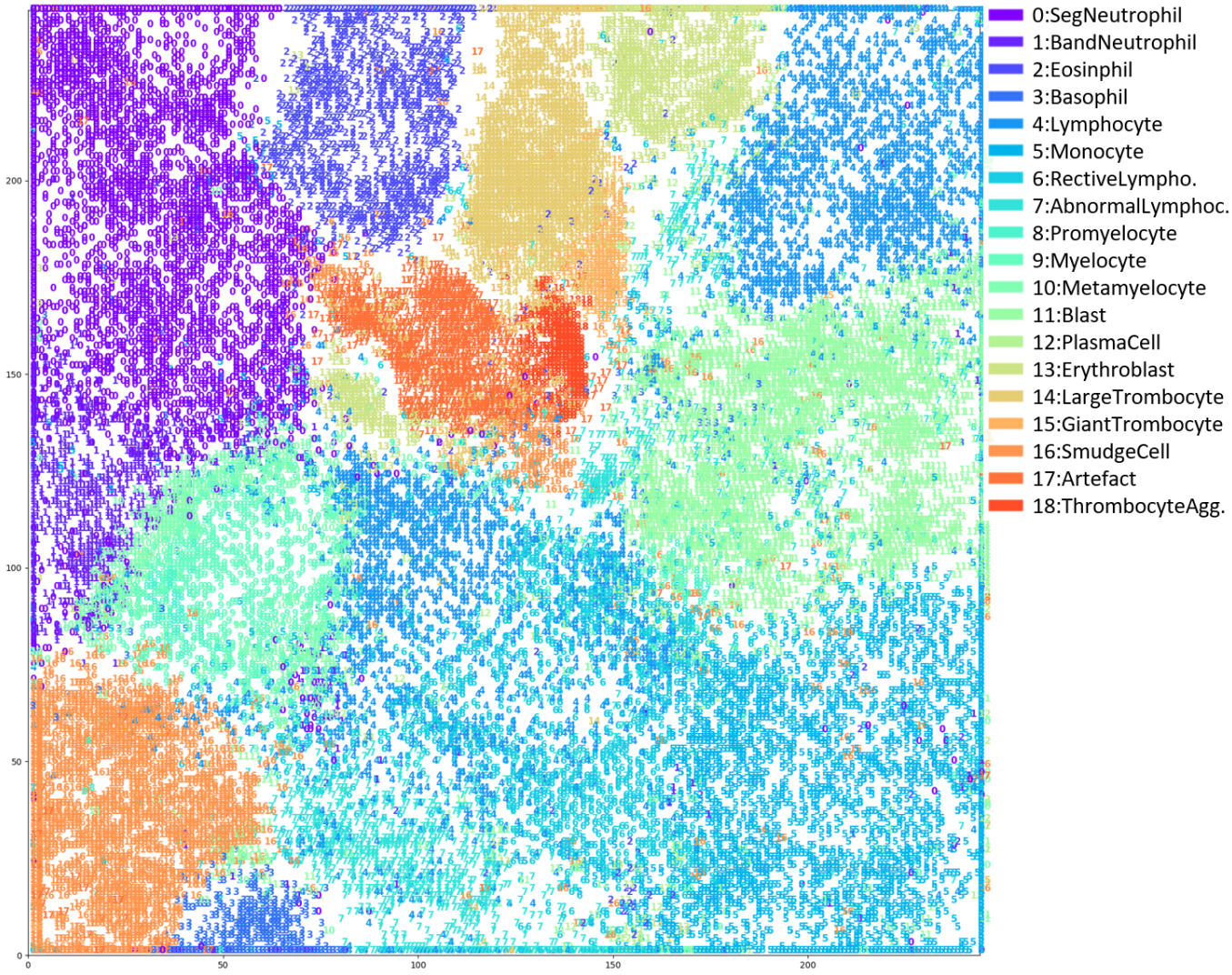

Figure 43: The SOM with the highest `SVOMEN` score. Each cell class has its own number and its own color.

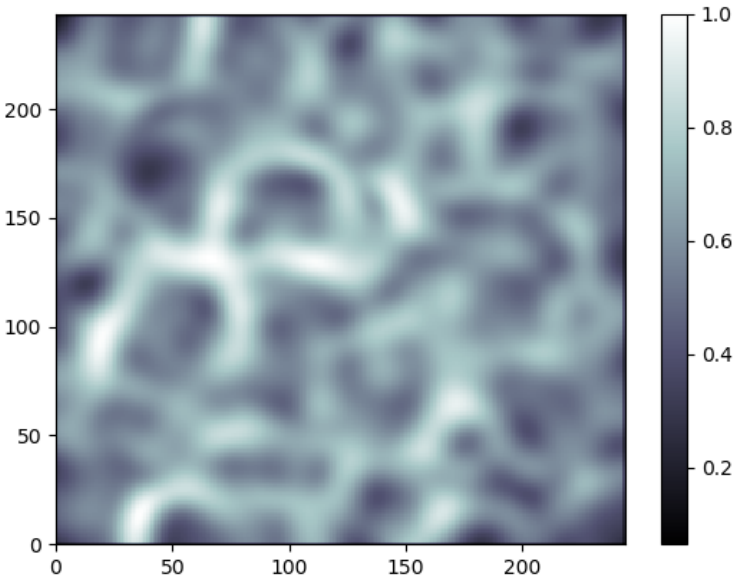

Figure 44: U-matrix for the SOM with the highest `SVOMEN` score. White areas indicate a large distance between the nodes.

5.4 Reference cells

Figure 45 and 46 show the adaptive reference cells for the same image using the output from the SOM and the VAE respectively. The middle image was correctly classified by the modified Xception network. In Figure 47 and 48 the middle image was incorrectly classified by the Xception model. Looking through the generated images, the VAE had consistently more visually similar reference cells. This is confirmed by the result in Table 10, which shows that both Anna and Oscar were better at classifying cells when using the VAEs adaptive reference cells. Out of 100 cells, Anna and Oscar correctly classified about 50% of the 19 possible options using the VAEs reference cells.

Table 10: Results from the classification performed on, by the Xception, correctly classified images. The test was performed by Oscar(O) and Anna(A) using 100 adaptive reference cells.

	A SOM	O SOM	A VAE	O VAE
Current Labels	35	39	48	51

Besides looking at the adaptive reference cells that were correctly classified by Xception, those that were misclassified were also tested. Table 11 shows the result of Anna, Oscar and a technologist classifying 186 of the misclassified images. Again, the VAE was slightly better than the SOM for the adaptive reference cells. The table also shows that classifying these cells was more difficult,

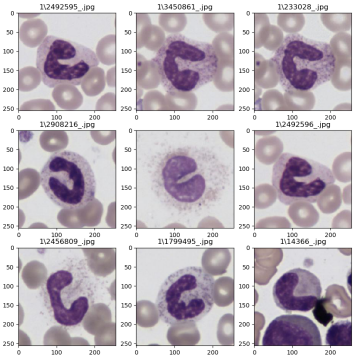

Figure 45: The reference cells for the middle image from class 1. All the reference cells are also class 1. The cell was **correctly** classified by the Xception network and used the output from the **SOM**.

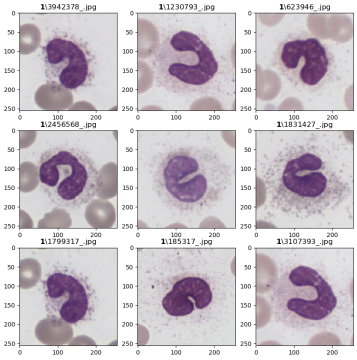

Figure 46: The reference cells for the middle image from class 1. All the reference cells are also class 1. The cell was **correctly** classified by the Xception network and used the output from the **VAE**.

which is also confirmed by the technologist asked from CellaVision. The technologist only classified 80 out of the 186 images in the same way as the current label of the cell image. The current label is determined by up to five medical technologists' classification and only those with a majority are in the data set.

Table 11: Results from the classification performed by Oscar(O), Anna(A) and a technologist using the adaptive reference cells. For example, the cell value of 95 means that Anna, using adaptive reference cells from the SOM, classified 95 out of the 186 images the same as CellaVision's technologist.

	A SOM	O SOM	A VAE	O VAE	Technologist
Technologist	95	94	104	88	-
Xception's Predictions	75	84	88	90	80
Current Labels	65	57	75	67	80

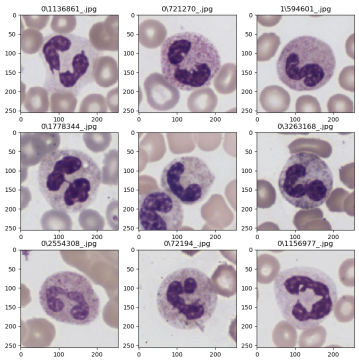

Figure 47: The reference cells for the middle image from class 1. Seven reference cells are classed as 0 and one as class 1. The cell was **incorrectly** classified by the Xception network and used the output from the **SOM**.

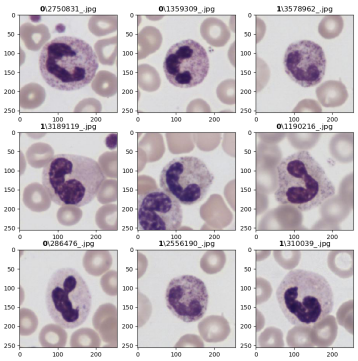

Figure 48: The reference cells for the middle image from class 1. Four reference cells are classed as 0 and four as class 1. The cell was **incorrectly** classified by the Xception network and used the output from the **VAE**.

6 Discussion

Classifying cell images can be a critical part when diagnosing a patient. CellaVision's products try to improve this work through different technical solutions. As a tool for the medical technologists classifying cell images, we propose adaptive reference cells. They are obtained by finding visually similar cells to the cell being classified and presenting them to the analyst as support for the final decision. Also, we propose a self-organizing map mainly as a tool for further research. The class separation done by the SOM could be used by a medical technologist as a method to distinguish one class from another, better than they can today.

6.1 Classification

The classifier achieved a satisfactory result. The classification experiments were primarily based around different data augmentations. The overall accuracies of the models were similar, but the less populated classes' accuracies fluctuated between the models. Resizing is done on all models' input as there are memory issues when the input is too large. The models are trained on the two stationary computers, so the images could probably be bigger if it was trained on the larger system. The other important augmentation is the `rotation range`. Without this augmentation, the model is not invariant to rotations. Thus the classifier could predict one image and its rotation differently. The resulting 1×12544 feature vector for two visually similar, but rotated, images would not have similar feature vectors. That means the following reference cells and placement on the SOM would also be variant to rotations. For future work, we propose even further augmentation of the data. Rotation augmentation from -180 to 180 degrees instead of from -90 to 90 is recommended. This would make the feature vectors even more invariant to rotations of the input images.

Feature vectors are extracted from the sixth last layer and are of size 1×12544. Breaking the model too late results in features highly weighted against the medical technologists' classification. This means we cannot find new patterns and from those, evaluate the existing classes. However, breaking the model too early results in features not focusing enough on the WBC in the image. WBC images are visually similar when the WBC, not the surrounding RBCs, are similar. Looking at the produced adaptive reference cells, the choice of layer works as intended. The reference cells focus on the WBC. This is confirmed by the fact that the reference cells with similar WBC but with very different RBC still end up close to each other in the SOM and are paired as reference cells by the VAE. Larger feature vectors were not investigated as it was not computationally possible. Having more computational power and more memory would enable us to test another layer than the one used.

To deal with the uneven data distribution, either upsampling, downsampling or increase of data is suggested. However, these methods may result in unwanted side effects where the classifier gains unwanted information. Also, downsampling and upsampling lead to a decrease in data available and overfitting. These could be further experimented with.

The data used to test the final bits of the pipeline (SOM and SVOMEN) comes from a subset of the validation set. As the validation set was used for validating both the classifier and the VAE, the data set is tainted. It was used

to select the hyperparameters of the model. Thus, a subset of the validation set is not appropriate for testing the outcome from the pipeline. For future work, the testSOM should be from completely unseen data.

CellaVision has machines from different generations with differing specifications. These all have different optics and the resulting cell images differ quite a bit. One way to increase the data set would be to include images from the other systems. Rebuilding the classifier using *domain-adversarial* training (DANN) of the ANN, the classifier should become invariant to the specific system the cell images are from [42]. The increased data set would increase the computational power needed but should hopefully also make the resulting reference cells invariant to the varying optics. It is costly for CellaVision to make sure each system has the same optics, thus the DANN could save the company money by relaxing this requirement. This would be an intriguing way to improve the current pipeline.

6.2 Dimensionality Reduction

The loss in the VAE is almost constant for different sizes of the latent space. The loss consists of both the reconstruction term and the KL term. The VAE tries to find both a good reconstruction and KL loss. The tradeoff is that too high of a reconstruction loss will result in a low KL divergence and a low reconstruction loss results in a high KL divergence. This is further visualized in Figure 20 and is the reason for the almost constant loss throughout the experiments. The VAE's loss function is not an ordinary loss function as in standard ANNs. Instead, the VAE tries to minimize the KL divergence and to maximize the marginal likelihood, $p_\theta(z)$. When increasing the latent space size, the reconstruction loss will become smaller and the KL divergence will become bigger, leading to a nearly unchanged total loss. The KL divergence becomes bigger since the encoder needs to fit more distributions $x^{(i)}$ inside $p(z|x)$ in Figure 20, i.e. one for each feature.

The PCA did not work well on our data as 44% of the information was lost during the transformation from the 1×12544 feature vector to a 1×50 feature vector. The reason for this is probably that PCA only works with linear transformations. A VAE, on the other hand, is based on ANNs which can use non-linear activation functions and thus find patterns in the data that the PCA cannot.

Notice that using the 2-dimensional latent space found in Figure 41 we can still see some patterns. The VAE can, using only two features, separate some of the data. That only classes 0, 1, 10 and 11 can be distinguished indicates that these classes, compared to the rest, are the easiest to represent using only two features. However, the low `SVOMEN` score and the fact that the separation of the other 15 classes is non-existing require us to use a higher dimensional latent space together with the SOM.

VAEs are often used in a generative matter. However, here it is only used for data compression. The VAE was used instead of an ordinary autoencoder as suggested by the supervisors at CellaVision. However, an ordinary autoencoder might produce a similar result. This was not investigated. Future work would include trying an ordinary autoencoder and compare that to the VAE. Also, testing a different architecture of the VAE. For example, the disentangled β-VAE proposed by [43] would be interesting to further explore. The main advantage

would be the ability to generate samples where the individual features can be modified independently of the others.

One possible modification to the pipeline would be to skip the classifier and only use the VAE and the SOM, i.e. the input to the VAE would be the full cell image. This would likely lead to a good compression of the data. Also, removing the classifier and sending the images directly to the VAE enables its full potential by taking advantage of the generative process. This means that the VAE could be used to generate new, fake images to extend the current data set. However, there are issues with disregarding the classifier. First, the surrounding information (such as RBCs) would be included in the latent space. Secondly, the VAE would only be trained on finding visual similarities and disregarding the medical technologist's knowledge. This kind of information is what the feature vector from the classifier contains. As the adaptive reference cells are helping humans, the human classifications should not be disregarded. Therefore, using images as input to the VAE would require segmenting the WBC to only include this information in the latent space. This could potentially show unknown patterns in the data by completely disregarding today's classifications.

6.3 SOM

Using the implementation `MiniSom` of a SOM has its pros and cons. It was used due to time restrictions and limited knowledge about SOMs when starting the experiments. Other implementations in Python exist but were not selected as they did not provide the same visualizations and documentation nor did they have the same community of contributors and users. `MiniSom` is a minimalistic implementation of a SOM which means the implementation is not optimal for large data sets such as ours. `MiniSom` does not support GPU training, hence the training is all done on the CPU. The long training process made it harder to perform all the experiments desired. There are other implementations of SOMs which support GPU training, but most were written in a different language than Python and lead to compatibility issues.

The radius of the neighborhood function and the learning rate were decreased from the values originally considered. The number of iterations would then need to be much higher for the ordering to become local enough and not just order the data globally. Had the SOM implementation supported training on the GPU, the number of iterations could be higher which should produce a more globally ordered map than the resulting one.

Training SOMs with size 244×244 instead of the suggested 248×248 was due to a rough calculation (based on input size 60 000) when starting the experiments. We did not increase the size of the map afterward as the difference was small enough not to have a big impact and the time it would take to retrain the maps was deemed too high.

The SOM can be used by the medical technologist to analyze what separates one class from another. Today, it is hard for them to specify the difference between some classes. However, using our SOM, they can visualize similar images from two classes and from those get a better understanding of the cell classes. Such visualization can find differences that are meaningful for the classification but could also find differences that are not. For example, let us say that the SOM separates the original class 1 into two subclasses and let us call them 1.1 and 1.2. Successful separation is one that results in something meaningful in the

sense that it makes it easier for the analysts to classify cell images. However, the new subclasses could also be irrelevant for the classification. Say the difference between 1.1 and 1.2 is a slight shift in the color intensity. The separation on the map is reasonable as it represents the information in the data, however, it is not meaningful for the final classification.

The idea of separating the cells in different classes is something that can be further explored with the SOM. By plotting the categories by themselves, the continuation and cutoff between the classes could be visualized. Also, by plotting the classes by themselves could provide further information. The tool that allows the user to see the cells placed at the different coordinates, could include options for these types of visualization. As the tool stands, the script is run with Python. The tools at CellaVision are written in C#, so rewriting the interactive SOM to a web application in this language would make it more useful for further experimentation.

The classification could also be based on the SOM, by labeling the cells after the majority of its neighbors. The current classification has a high accuracy, so this was not tried. This would also provide information about outliers in the data set. The cells that have a different label than those around them can be assumed to be misclassified by the technologist. Removing them from the data set will increase the accuracy of the pipeline.

Our new U-matrix from Figure 44 differs a lot from the original U-matrix generated from the medical technologist's current classification. The old one is based on a manual classification that labeled each cell image into one of the 19 classes from Table 6 which results in the sharp U-matrix in Figure 25. However, our new U-matrix tries to move away from the predetermined classes and instead cluster the cell images based on visual similarity. The result of this is the new, more blurry, U-matrix in Figure 44. It is clear that our clusters do not directly correlate with the old ones even if looking at the SOM in Figure 43 shows that some images from the same class have been clustered together.

The SOM was selected to visualize the cell images in two dimensions. Other techniques, such as density-based clustering [44] and k-means clustering [45] were considered. For both methods, the data needs to be in two dimensions to visualize the clusters in two dimensions. Thus, the VAE would need to compress the data such that the latent space is of size two. Rebuilding a 1×12544 vector from only two variables will introduce a lot of data loss. As the data then is in two dimensions and it is labeled the clustering techniques are not needed for visualization reasons. There is also no need to use a clustering algorithm on the SOM as it already shows the data in reasonable clusters. Using SOMs, on the other hand, will maintain both the spatial and topological data whilst also visualizing the data in an easy format.

Finally, there are many possible ways to explore the data using the SOM. Perhaps showing the classes of a person's blood smear from when they are healthy and sick. This might show how the cells move during this transition. Plotting different demographics, diseases and such could also provide further insight.

6.4 Reference cells

When classifying the cells using the adaptive reference cells it is clear that the VAE is preferred over the SOM, as shown in Table 10. As seen Table 11, the

asked technologist's answers differed from the current label. This makes it hard to trust the current classification and creates uncertainties in whether to use CellaVision's medical technologist's classification or the current label as the ground truth. This is an ongoing problem, the different experts will have different opinions. When classifying cells from one blood smear, medical technologists usually first look at the full sample to get an overview before classifying them individually. This overview calibrates their classification. Perhaps each image should thus be shown with an overview of the blood smear as well. Or a step needs to be added to the pipeline such that the images are adjusted based on each individual's blood smear.

The technologist classifying with the adaptive reference cells complained that the resolution of the images was too poor to make certain crucial distinctions. The images presented are however of the same resolution as the technologist is used to see them with CellaVision's products. The key difference, however, is in the normalization. The images we show are normalized according to the background. CellaVision performs multiple other types of image processing before showing cell images to the technologists. For example, the nucleus is brightened to show key details that are otherwise drowned by the dark purple color.

Another way of improving the adaptive reference cells is to also provide some statistics for each image. Perhaps showing a distribution of the closest 100 reference cells. This distribution could be weighted to account for which cells are the closest. This would give the technologist yet another tool and a better overview of its peers' classifications for such an image.

For the adaptive reference cells to be used in a specific laboratory, some work is required. Since each CellaVision machine has some varieties in the optics, the adaptive reference cells first need to be adapted to that laboratory's setup. This means that they would need to use our pipeline to produce their reference cells and use them to find their adaptive reference cells. This is not optimal and could be solved using, for example, a DANN. It would enable the customers to train their network using images from multiple systems leading to better adaptive reference cells, invariant to different laboratory setups.

6.5 Evaluation

The `SVOMEN` measurement was developed to create some evaluation method for the SOM. It is meant to quantify cell similarity. The goal of the SOM is to place visually similar images close to each other and not similar images far apart. The `SVOMEN` score tries to capture how well the SOM accomplished this for our cell data. A disadvantage of `SVOMEN` is that it is based on the test person's subjective opinion. Since it is humans performing the test, it will never be invariant to rotation or color intensities. Performing the test on more test persons would reduce these side effects but they will never disappear.

The `ROTMES` measurement worked as intended but could be further developed. It was developed since we saw that images in an area of the SOM all had the same rotation. This was not desired since we want the SOM to be invariant to the rotation angle of the cell images. This measurement made us add the `rotation_range` augmentation to the classifier which solved the rotation problem. However, `ROTMES` has further potential. It would be useful to evaluate the pipeline using the duplicates in the original data set. Sending an image and

its duplicate through the pipeline should result in their feature vectors ending up close to each other in the SOM, preferably on the same coordinate. Here, `ROTMES` could be used to evaluate how good (similar) feature vectors from the VAE become when passing duplicates as input to the pipeline. Also, more fields could be added to `ROTMES` beyond the current ones, for example `zoom_range` and different `shift` augmentations. However, this kind of augmentation is already used in the classifier which should result in a low `ROTMES` score on these fields.

7 Final Thoughts

The major takeaway from this thesis regards the ground truth. For supervised machine learning techniques to work as intended, there must be a well defined and trustworthy true label. If this is not the case, it does not matter how well the system performs since the results consequently become untrustworthy. In the case of cell image classification, this is a problem since the classification comes down to one or several medical technologists' subjective opinions. Methods that automatically segments the WBC and in an unsupervised matter cluster the data would be preferable to solve the problem. This approach could lead to a new standardization of cell classes and has the preconditions to be more consistent than today's solution. The problem with the approach, as other unsupervised methods, is the evaluation. Using new clusters standardized by a computer requires large parts of the medical field to accept it and to change their work methods accordingly which is infeasible at this time.

References

[1] G. d'Onofrio, *Morphology of the blood.* Oxford Boston: Butterworth-Heinemann, 1998.

[2] "Adult acute myeloid leukemia treatment (pdq®)–patient version," US National Cancer Institute, [Accessed: 2020-03-05]. [Online]. Available: https://www.cancer.gov/types/leukemia/patient/adult-aml-treatment-pdq

[3] "About cellavision," CellaVision, 2020, [Accessed: 2019-12-05]. [Online]. Available: https://www.cellavision.com/en/about-us

[4] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, Inc., 2019. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=cat07147a&AN=lub.6373158&site=eds-live&scope=site

[5] G. Dreyfus, *Neural Networks: Methodology and Applications*, 01 2005.

[6] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity." *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, p. 115, 1943. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=72219868&site=eds-live&scope=site

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[8] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *CoRR*, vol. abs/1804.07612, 2018. [Online]. Available: http://arxiv.org/abs/1804.07612

[9] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks." 2017. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=edsarx&AN=edsarx.1705.08741&site=eds-live&scope=site

[10] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex." *The Journal of physiology*, vol. 148, pp. 574–91, 1959.

[11] S. Younghak and I. Balasingham, "Comparison of hand-craft feature based svm and cnn based deep learning framework for automatic polyp classification." *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Engineering in Medicine and Biology Society (EMBC), 2017 39th Annual International Conference of the IEEE*, pp. 3277 – 3280, 2017. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=edseee&AN=edseee.8037556&site=eds-live&scope=site

[12] D. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification." 07 2011, pp. 1237–1242.

[13] R. Yamashita, M. Nishio, R. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, 06 2018.

[14] S. Saha, "A comprehensive guide to convolutional neural networks - the eli5 way," Towards Data Science, Dec 2018, [Accessed: 2020-03-05]. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[15] E. Benderskys, "Depthwise separable convolutions for machine learning," Apr 2018, [Accessed: 2020-03-05]. [Online]. Available: https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/

[16] G. Zhao, Z. Zhang, H. Guan, P. Tang, and J. Wang, "Rethink relu to training better cnns." 2017. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=edsarx&AN=edsarx.1709.06247&site=eds-live&scope=site

[17] F. Chollet, "Xception: Deep learning with depthwise separable convolutions." 2016. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=edsarx&AN=edsarx.1610.02357&site=eds-live&scope=site

[18] L. Yadan, H. Zhenqi, X. Haoyu, L. Lizhuang, L. Xiaoqiang, and Z. Keke, "Yolov3-lite: A lightweight crack detection network for aircraft structure based on depthwise separable convolutions." *Applied Sciences*, no. 18, p. 3781, 2019. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=edsdoj&AN=edsdoj.346b9d957cbb4d6b834d739ae0e8d902&site=eds-live&scope=site

[19] K. Drossos, S. I. Mimilakis, S. Gharib, Y. Li, and T. Virtanen, "Sound event detection with depthwise separable and dilated convolutions." 2020. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=edsarx&AN=edsarx.2002.00476&site=eds-live&scope=site

[20] J. Guo, Y. Li, W. Lin, Y. Chen, and J. Li, "Network decoupling: From regular to depthwise separable convolutions." 2018. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=edsarx&AN=edsarx.1808.05517&site=eds-live&scope=site

[21] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift." 2015. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=edsarx&AN=edsarx.1502.03167&site=eds-live&scope=site

[22] I. T. Jolliffe, *Principal component analysis.*, ser. Springer series in statistics. Springer, 2002. [Online]. Available: http://ludwig.lub.lu.

se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=cat07147a&AN=lub.1683380&site=eds-live&scope=site

[23] A. Datta, S. Ghosh, and A. Ghosh, *PCA, Kernel PCA and Dimensionality Reduction in Hyperspectral Images*. Singapore: Springer Singapore, 2018, pp. 19–46. [Online]. Available: https://doi.org/10.1007/978-981-10-6704-4_2

[24] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, "Understanding disentangling in β-vae," 2018.

[25] J. Joyce, "Bayes' theorem," in *The Stanford Encyclopedia of Philosophy*, spring 2019 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2019.

[26] G. Wu, H. Zhang, Y. He, X. Bao, L. Li, and X. Hu, "Learning kullback-leibler divergence-based gaussian model for multivariate time series classification," *IEEE Access*, vol. 7, pp. 139 580–139 591, 2019. [Online]. Available: https://doi.org/10.1109/access.2019.2943474

[27] K. P. Murphy, *Machine learning : a probabilistic perspective.*, ser. Adaptive computation and machine learning series. MIT Press, 2012. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=cat07147a&AN=lub.2503144&site=eds-live&scope=site

[28] T. Kohonen, *Self-Organizing Maps, Third Edition*, ser. Springer Series in Information Sciences. Springer, 2001. [Online]. Available: https://doi.org/10.1007/978-3-642-56927-2

[29] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *ATT Labs*, vol. 2, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[30] S. Kaski, *Data Exploration Using Self-organizing Maps*, ser. Acta polytechnica Scandinavica. Finnish Academy of Technology, 1997. [Online]. Available: https://books.google.se/books?id=_1AEMQAACAAJ

[31] "Self organizing maps." [Online]. Available: https://miro.medium.com/max/983/1*QG7afWQKjY3IpezhNQMzBg.png

[32] J. . Fort, M. Cottrell, and P. Letremy, "Stochastic on-line algorithm versus batch algorithm for quantization and self organizing maps," in *Neural Networks for Signal Processing XI: Proceedings of the 2001 IEEE Signal Processing Society Workshop (IEEE Cat. No.01TH8584)*, Sep. 2001, pp. 43–52.

[33] M. Olteanu and N. Vialaneix, "Using SOMbrero for clustering and visualizing graphs," *Journal de la Societe Française de Statistique*, vol. 156, no. 3, pp. 95–119, Nov. 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01232672

[34] G. Vettigli, "Justglowing/minisom," GitHub, Jan 2020, [Accessed: 2020-03-05]. [Online]. Available: https://github.com/JustGlowing/minisom

[35] F. E. Grubbs, "Procedures for detecting outlying observations in samples." *Technometrics*, vol. 11, no. 1, p. 1, 1969. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=5987709&site=eds-live&scope=site

[36] "Looks like it - the hacker factor blog." [Online]. Available: http://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html

[37] F. Lundh and A. Clark, "Image module," [Accessed: 2020-03-18]. [Online]. Available: https://pillow.readthedocs.io/en/3.1.x/reference/Image.html#PIL.Image.Image

[38] "Johannesbuchner/imagehash." [Online]. Available: https://github.com/JohannesBuchner/imagehash/tree/master/imagehash

[39] "Image preprocessing - keras documentation." [Online]. Available: https://keras.io/preprocessing/image/

[40] A. Bäuerle and T. Ropinski, "Net2vis: Transforming deep convolutional networks into publication-ready visualizations," *CoRR*, vol. abs/1902.04394, 2019. [Online]. Available: http://arxiv.org/abs/1902.04394

[41] B. Keng, "Variational autoencoder," GitHub, [Accessed: 2019-12-05]. [Online]. Available: https://github.com/bjlkeng/sandbox

[42] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks." 2015. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.1505.07818&site=eds-live&scope=site

[43] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, "Understanding disentangling in β-vae." 2018. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=edsarx&AN=edsarx.1804.03599&site=eds-live&scope=site

[44] H. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, p. 231, 2011. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edo&AN=ejs23541340&site=eds-live&scope=site

[45] J. Wu and s. SpringerLink (Online, *Advances in K-means Clustering. [Elektronisk resurs] A Data Mining Thinking.*, ser. Springer Theses, Recognizing Outstanding Ph.D. Research. Springer Berlin Heidelberg, 2012. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search-ebscohost-com.ludwig.lub.lu.se/login.aspx?direct=true&db=cat07147a&AN=lub.5957597&site=eds-live&scope=site

Appendix

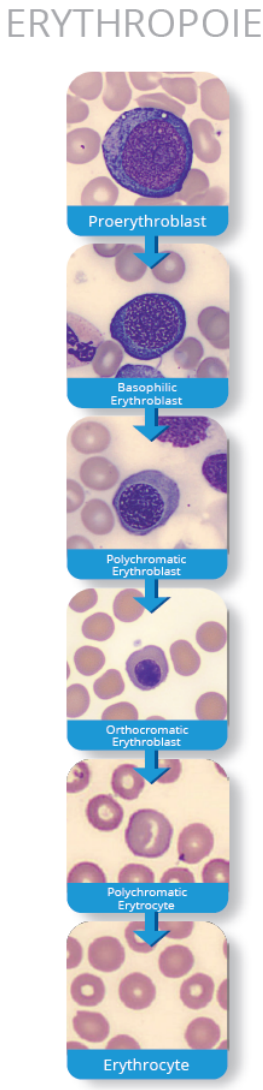

Figure 49: The category Erythropoiesis, nucleated RBCs found in some animals (not mammals) and its cell classes.

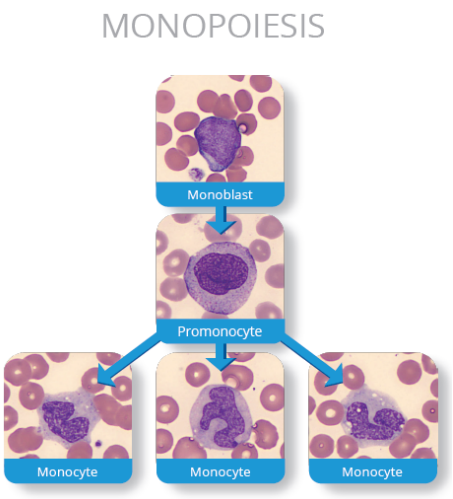

Figure 50: The WBC category Monopoiesis and its cell classes.

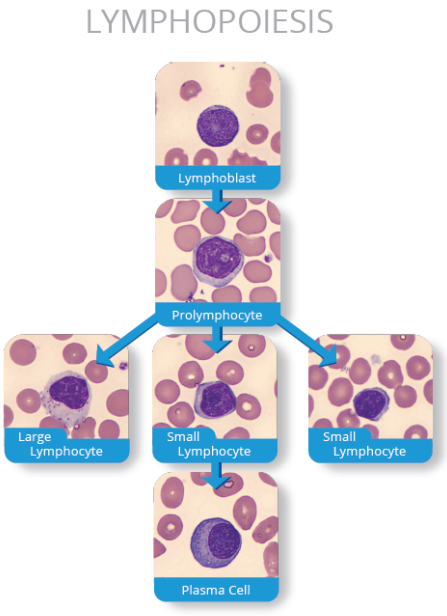

Figure 51: The WBC category Lymphopoiesis and its cell classes.

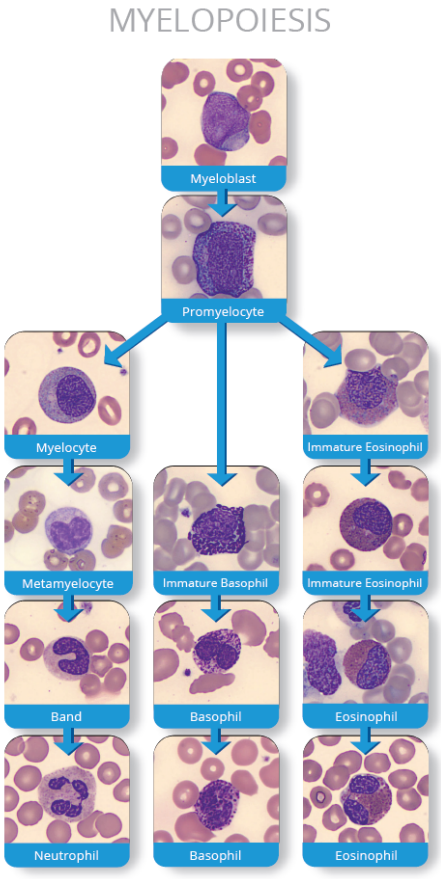

Figure 52: The WBC category Myelopoiesis and its cell classes.

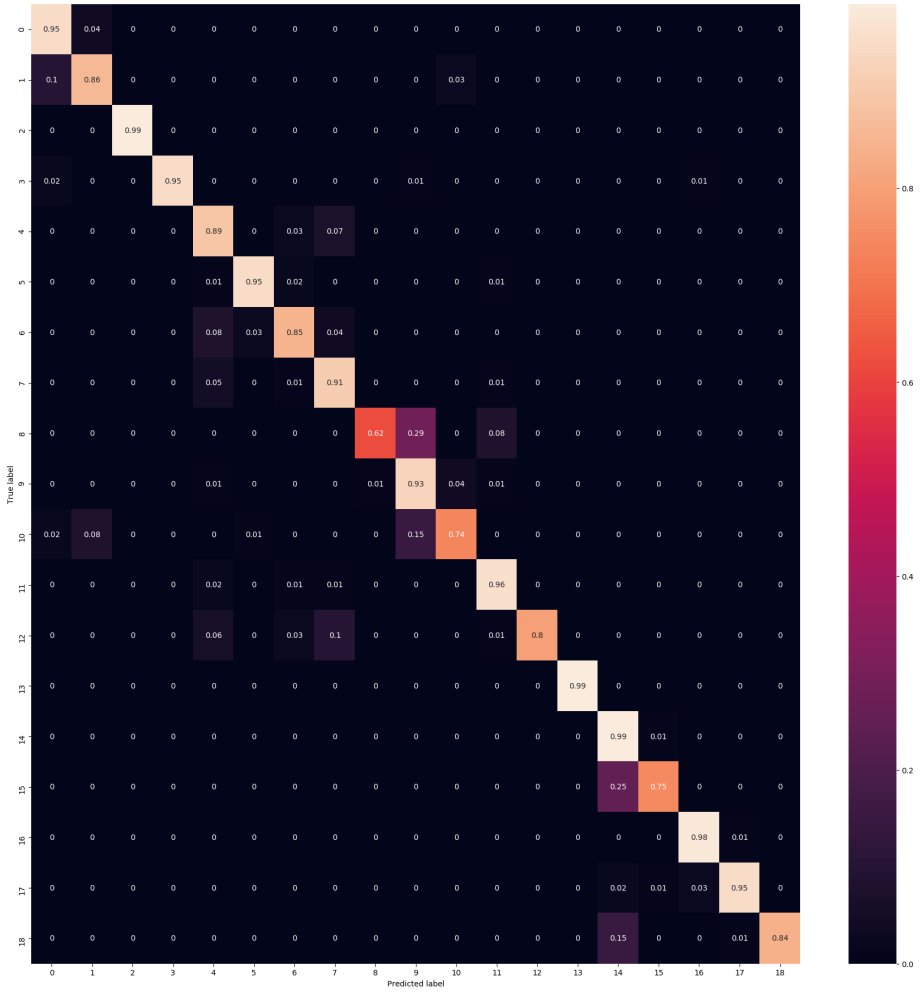

Figure 53: Confusion matrix of the accuracy of model A.

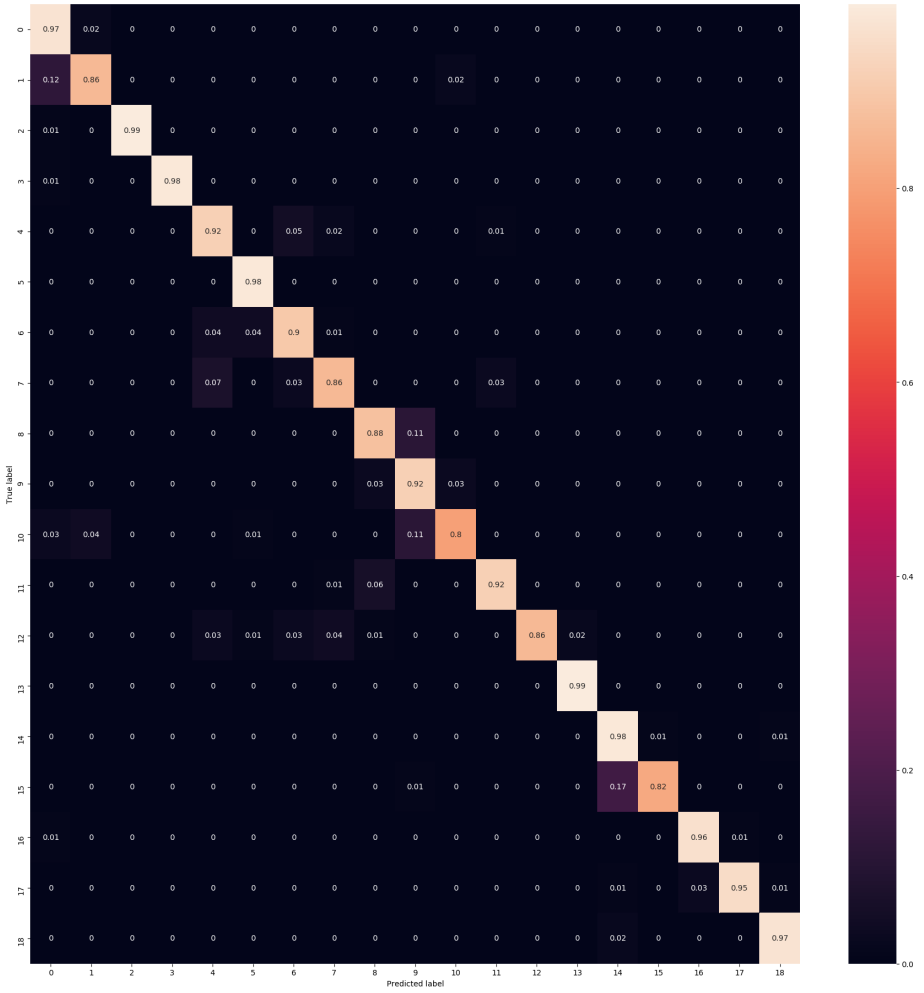

Figure 54: Confusion matrix of the accuracy of model B.

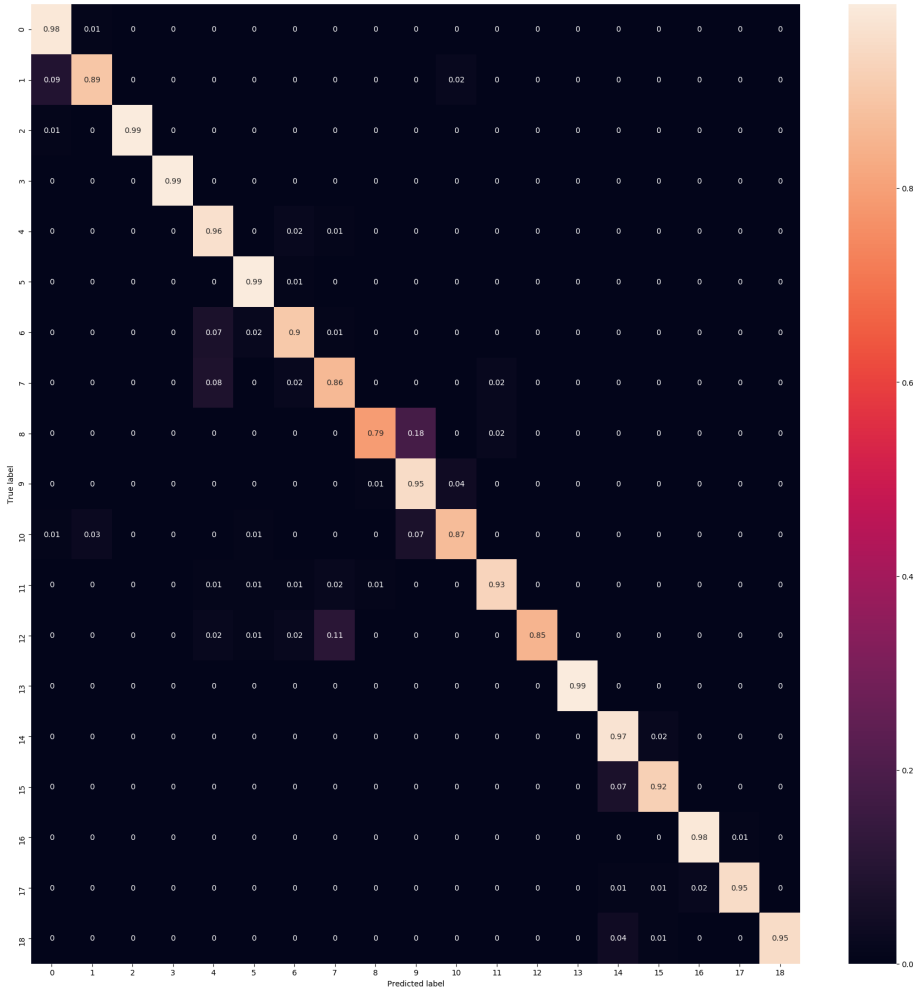

Figure 55: Confusion matrix of the accuracy of model C.

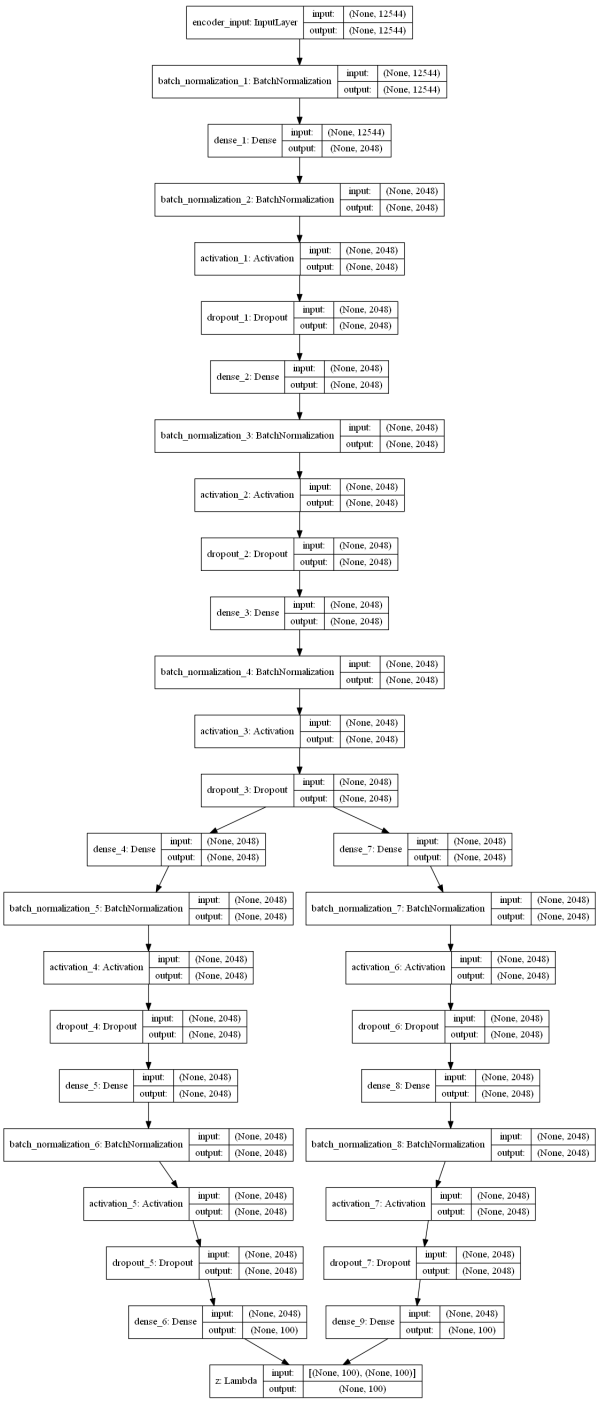

Figure 56: Encoder part of the VAE.

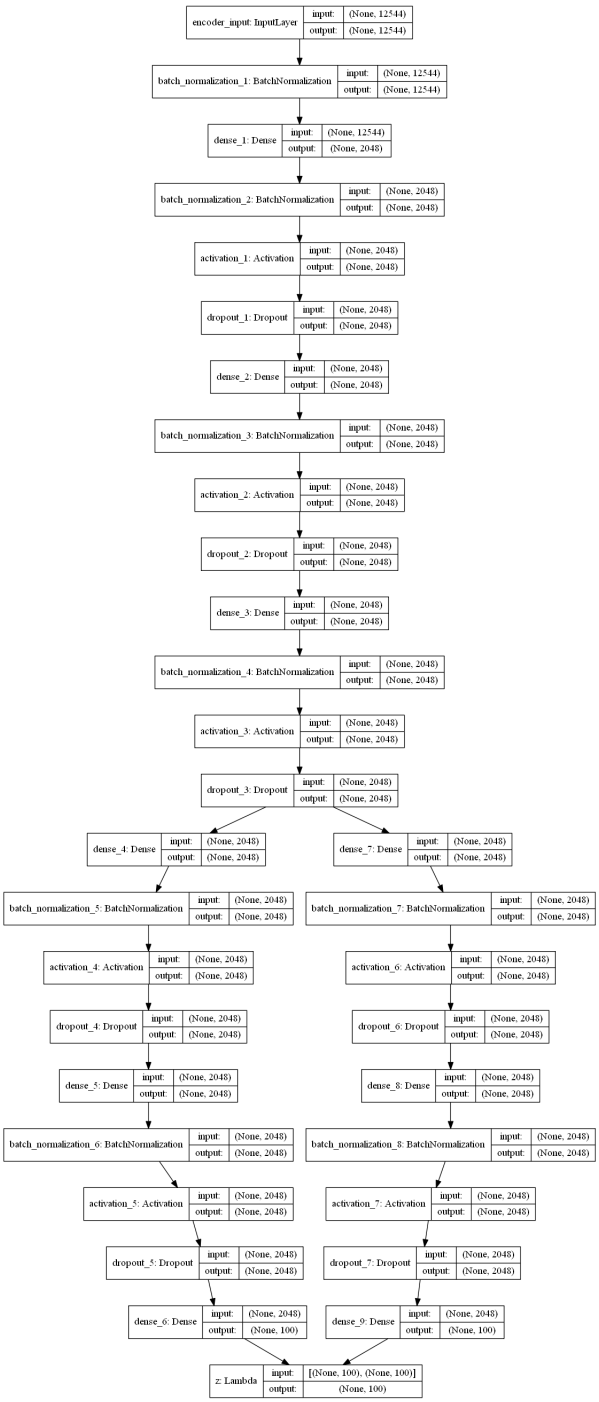

Figure 57: Decoder part of the VAE.

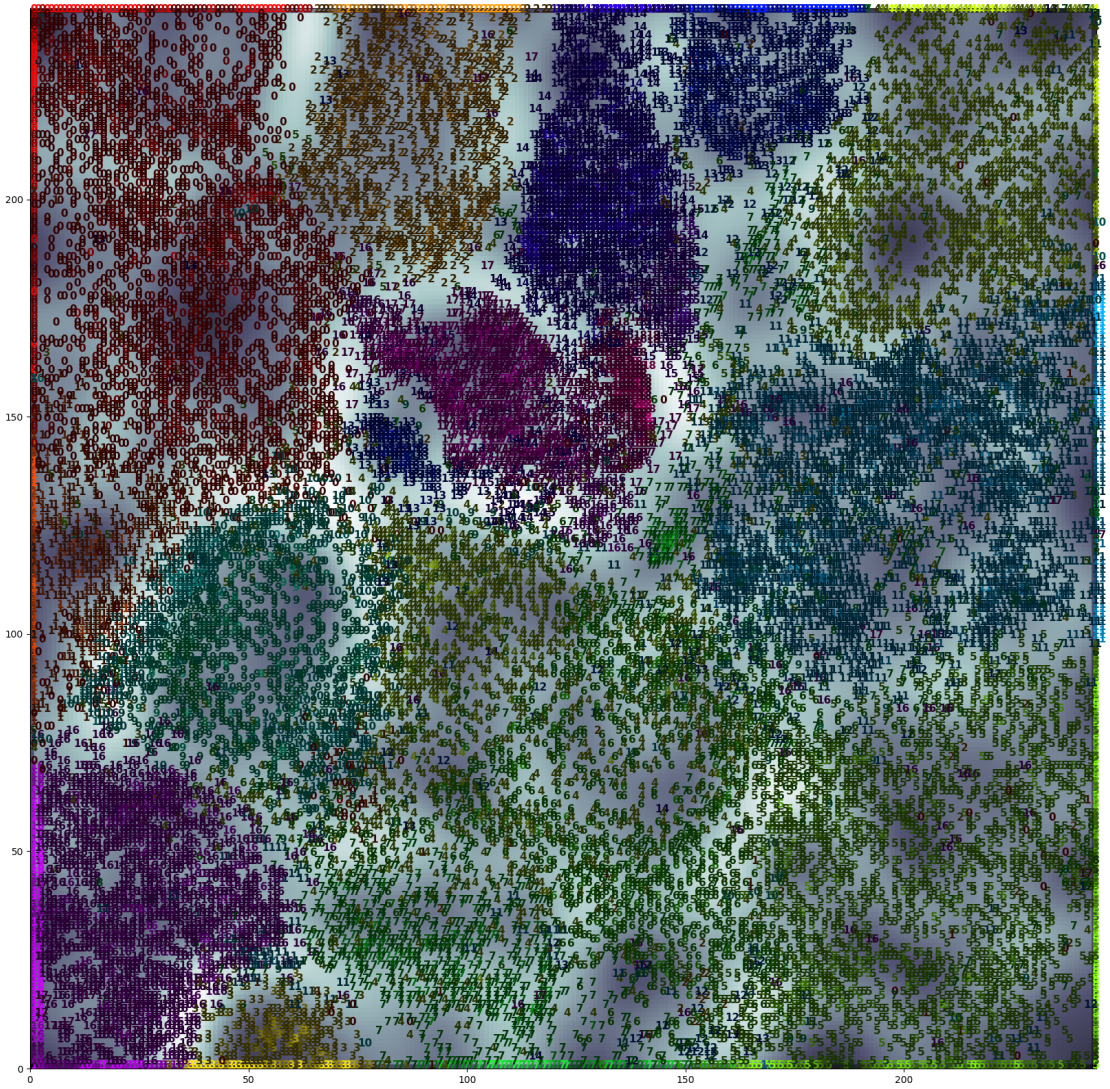

Figure 58: SOM overlay for the SOM with the highest SVOMEN score.

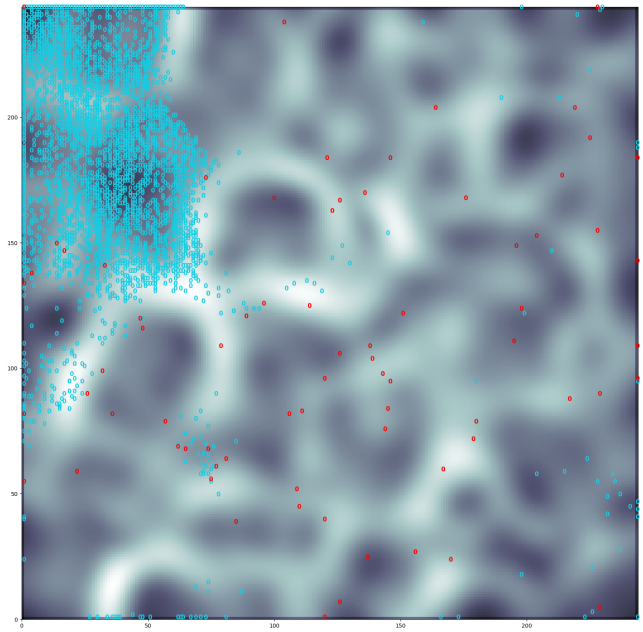

Figure 59: The cell images from class 0 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

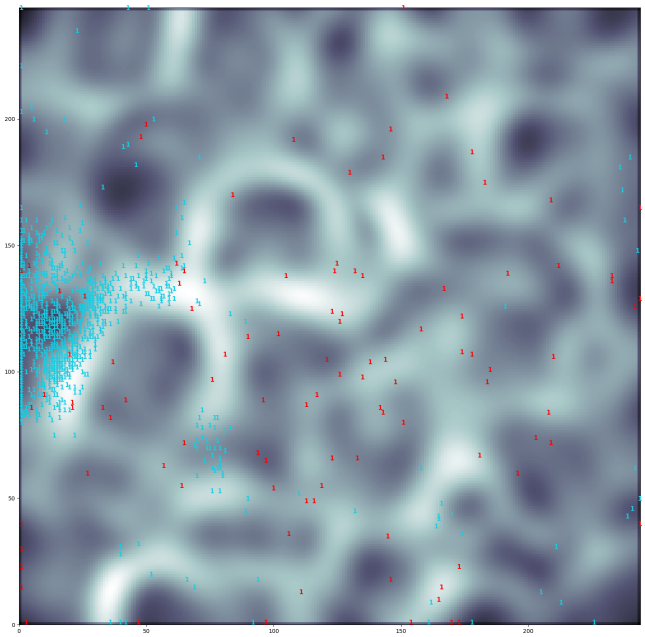

Figure 60: The cell images from class 1 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

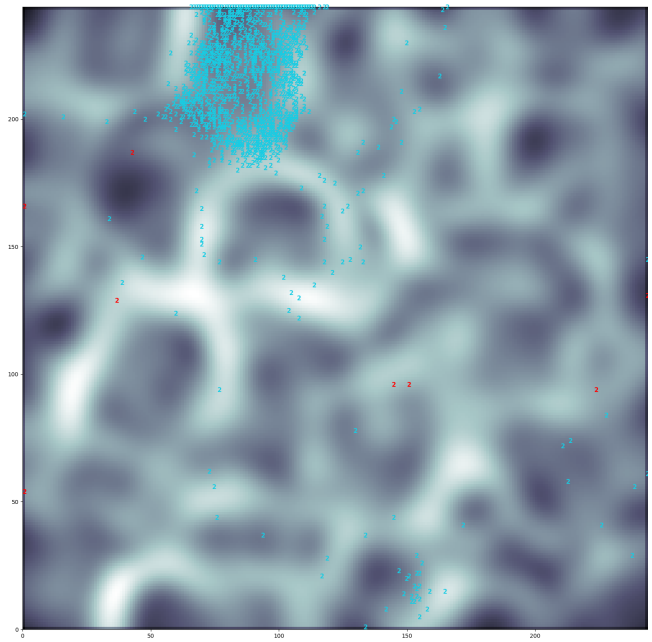

Figure 61: The cell images from class 2 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

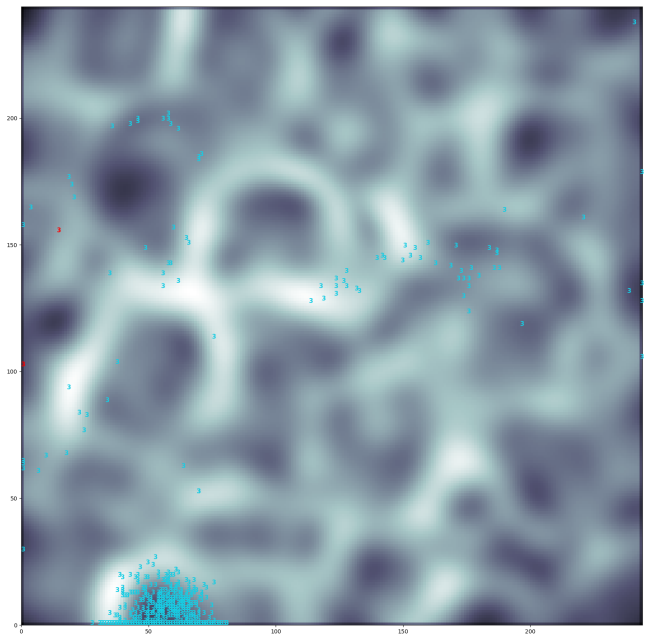

Figure 62: The cell images from class 3 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

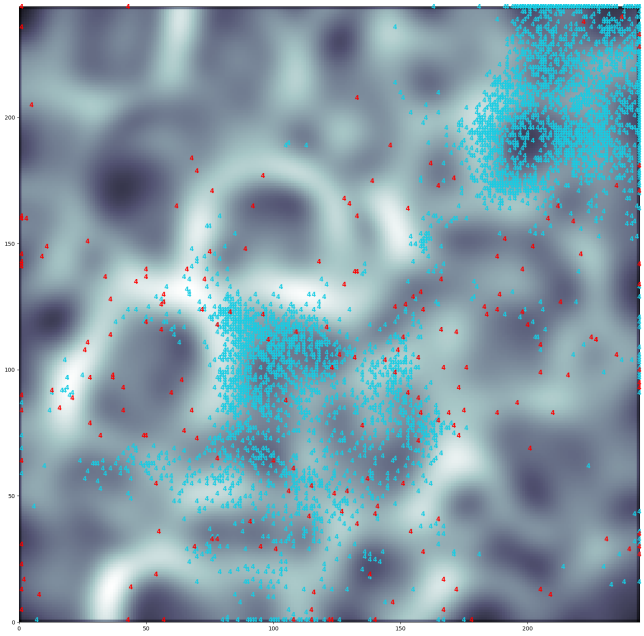

Figure 63: The cell images from class 4 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

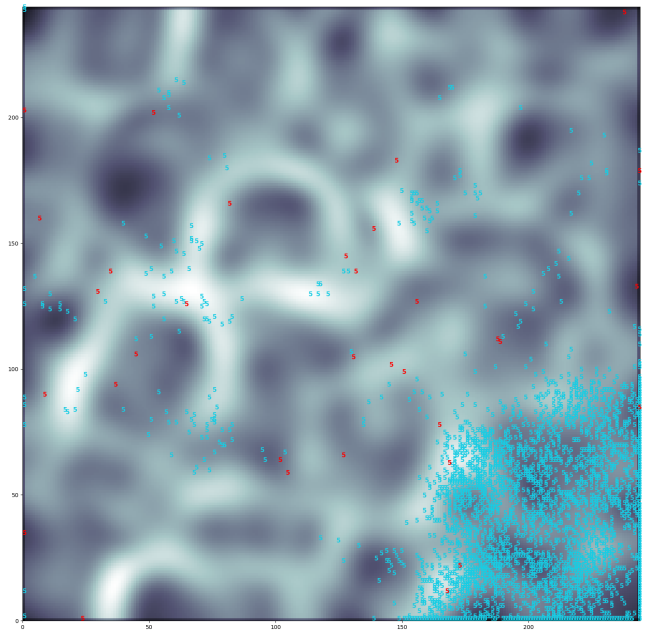

Figure 64: The cell images from class 5 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

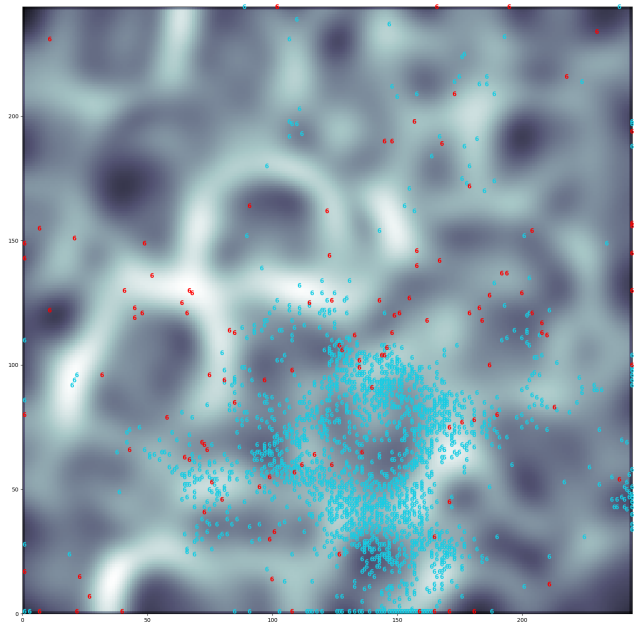

Figure 65: The cell images from class 6 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

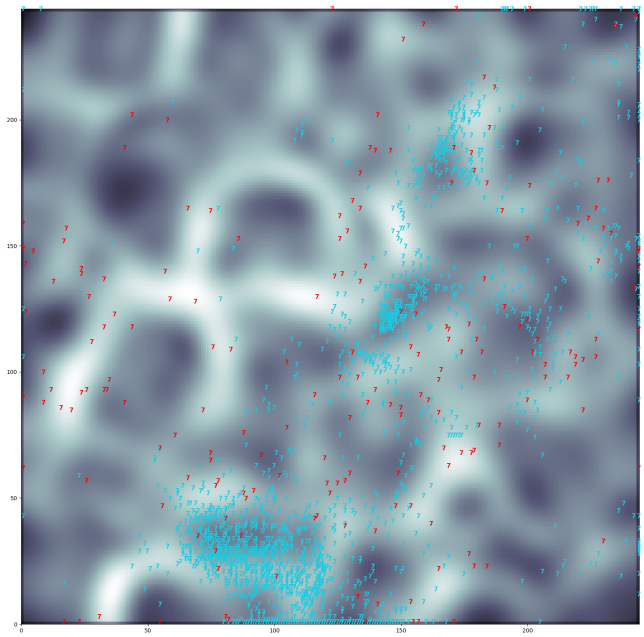

Figure 66: The cell images from class 7 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

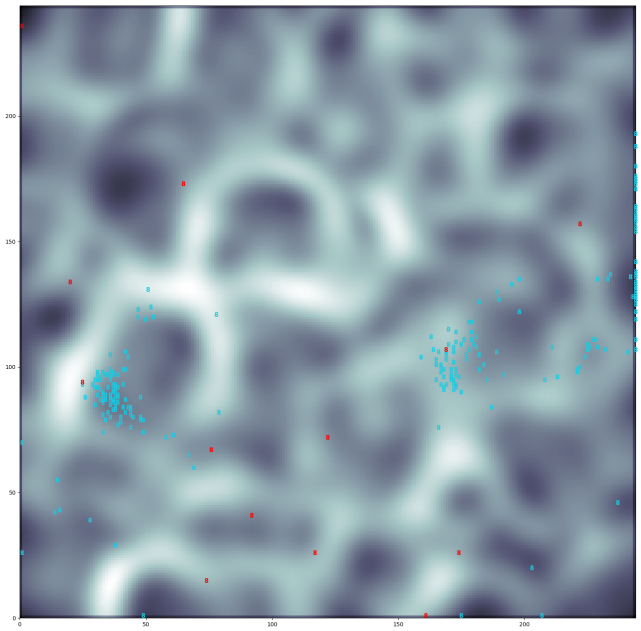

Figure 67: The cell images from class 8 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

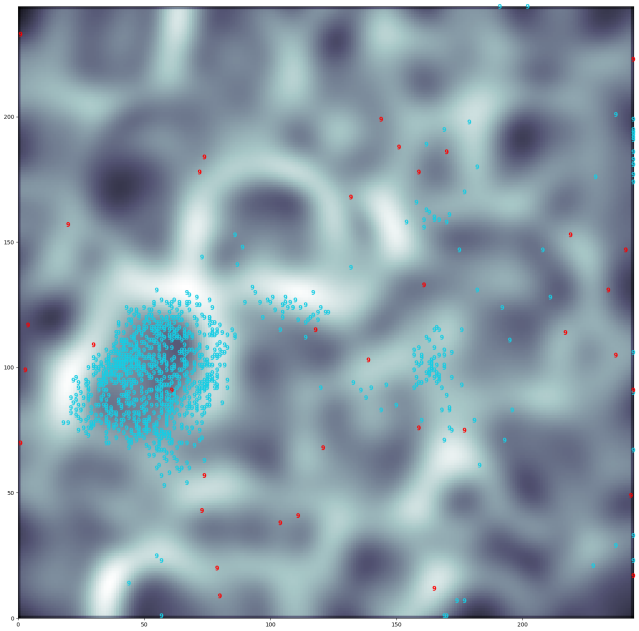

Figure 68: The cell images from class 9 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

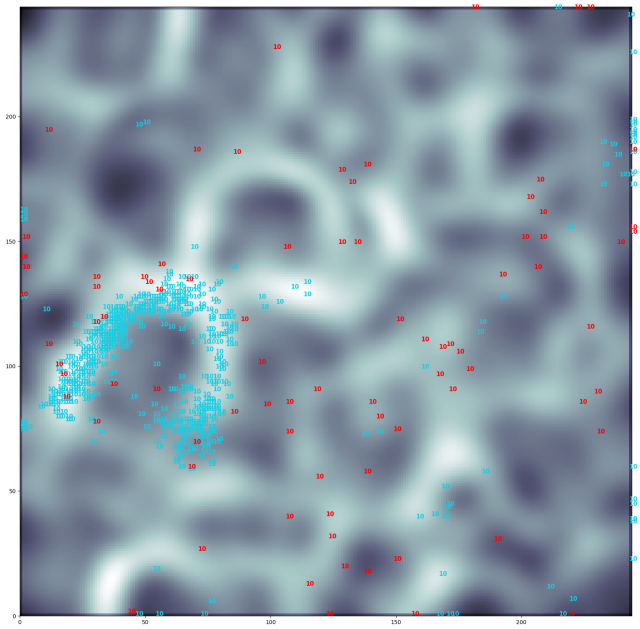

Figure 69: The cell images from class 10 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

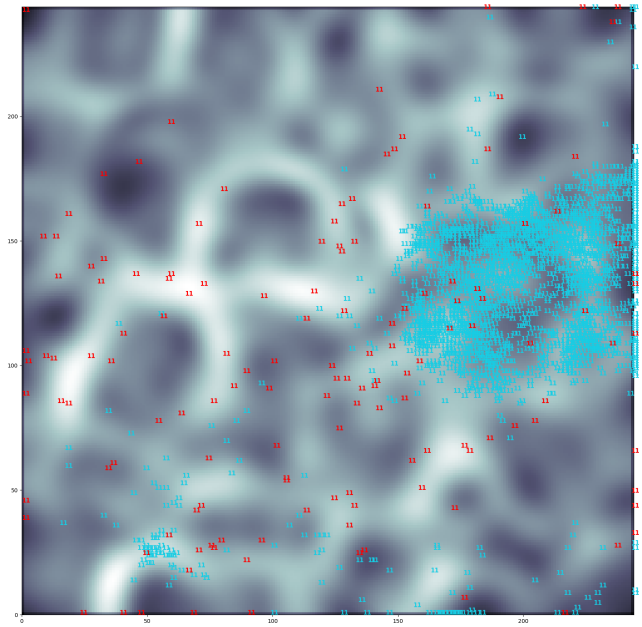

Figure 70: The cell images from class 11 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

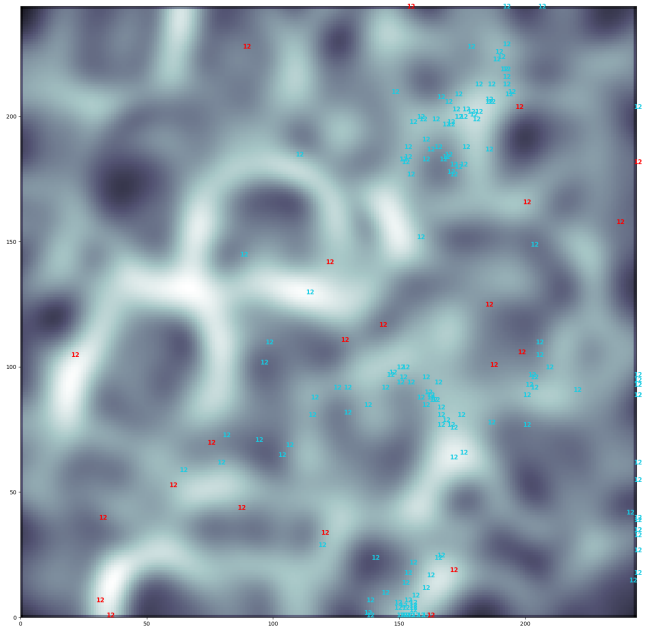

Figure 71: The cell images from class 12 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

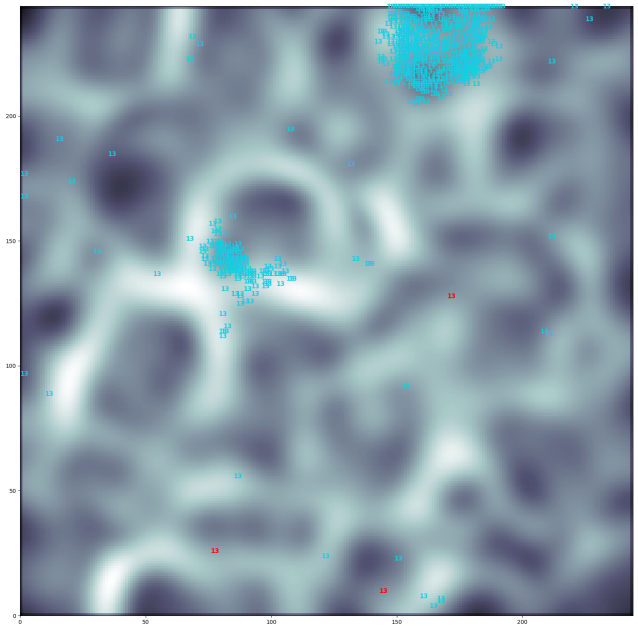

Figure 72: The cell images from class 13 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

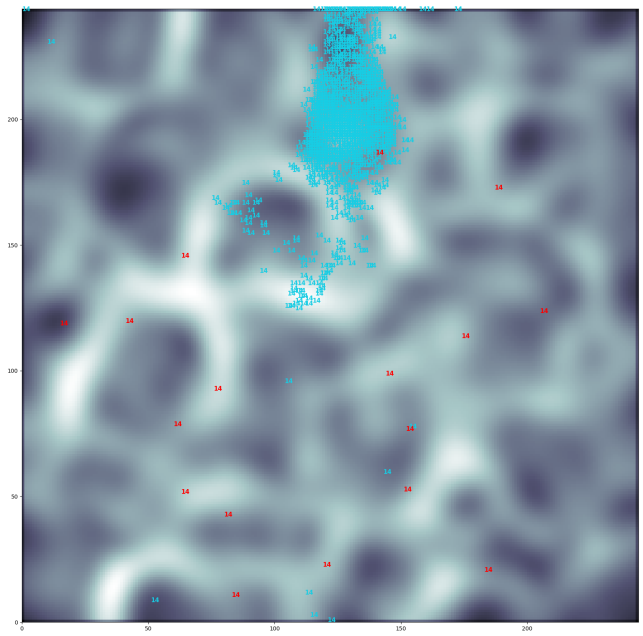

Figure 73: The cell images from class 14 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

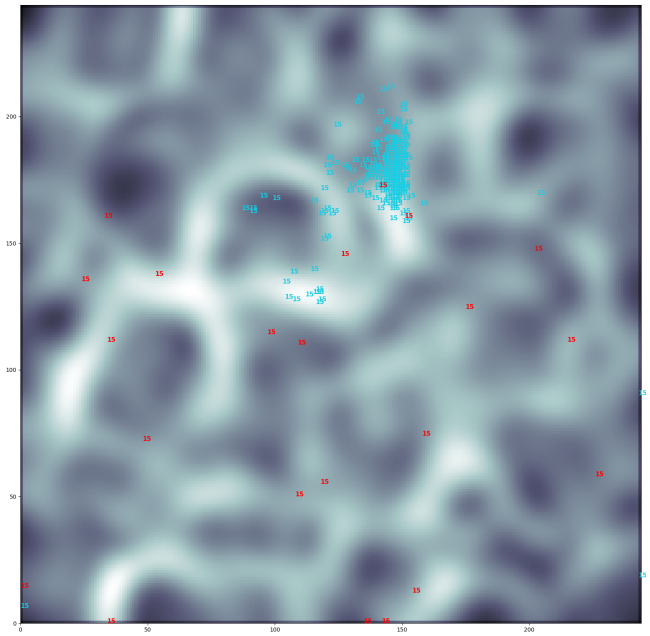

Figure 74: The cell images from class 15 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

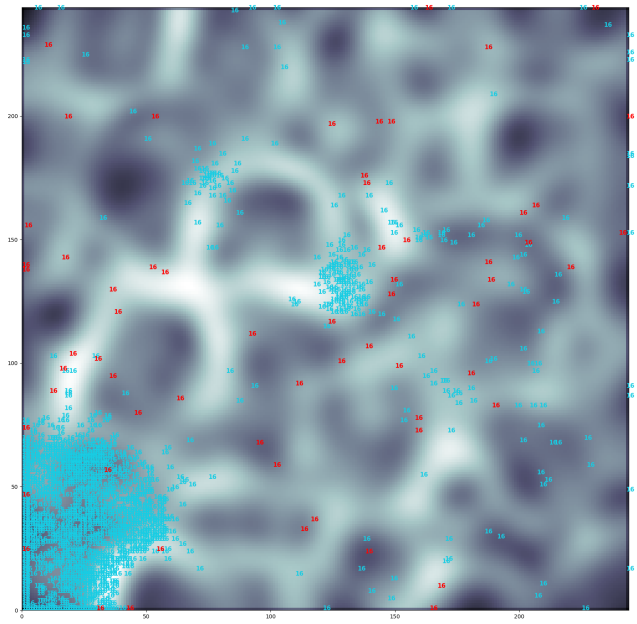

Figure 75: The cell images from class 16 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

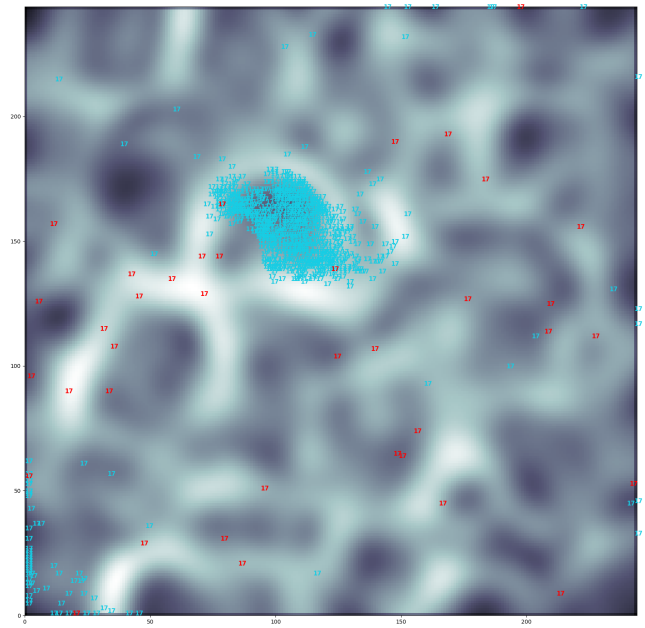

Figure 76: The cell images from class 17 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.

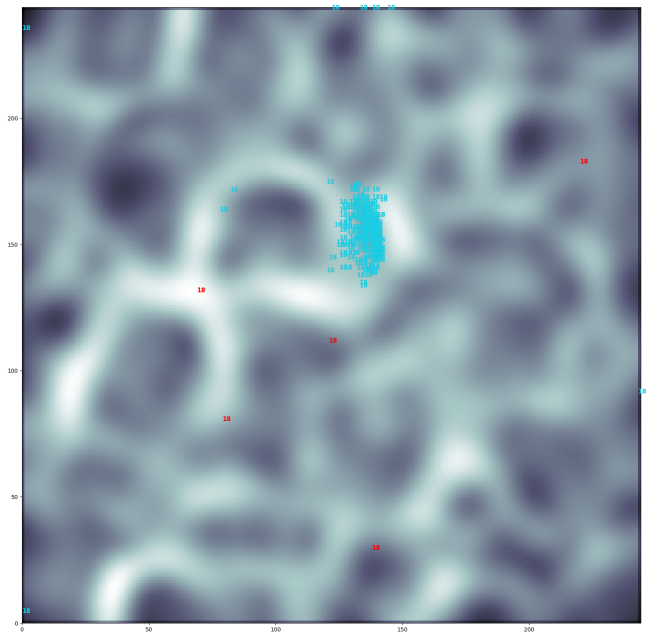

Figure 77: The cell images from class 18 in the testSOM set are plotted over the U-matrix. The red numbers are misclassified by the Xception network.