Simultaneous Classification of Sets of Images Using Deep Learning and Clustering

Nellie Carleke och Hugo Sellerberg

Master's thesis
2020:E29

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Abstract

Classification of cell images is conventionally done manually in hematology laboratories by medical technologists. CellaVision aims to automate this work in order to make the analysis process faster, better and more flexible. The automatic classification is currently done by processing each individual cell image through a Convolutional Neural Network. This methodology does not exploit any correlations that might exist between cells from the same blood sample.

We suggest a method to first compress the images of a whole sample using a Convolutional Neural Network and a Variational Autoencoder, then cluster these compressed data points using DBSCAN clustering and Bayesian Optimization, and finally assign a cell class to each cluster using statistical tools such as Earth Mover's Distance. We used data from CellaVision's system DC-1 to train a Convolutional Neural Network with 90.68% accuracy on training data and 82.85% accuracy on test data. This was used both as a benchmark and as the foundation to our method. We managed to enhance the accuracies to 90.90% on training data and 83.13% on test data by applying our method.

We explored the feasibility of using our method on mixed cell data from different systems, but the results were not as good as on DC-1 data. Applying our method on images of handwritten digits from the MNIST dataset could be made advantageous by forming customized subsets of images. This indicates that our method is versatile enough to use on general image data, provided that correlations within the subsets exist.

Acknowledgements

We would like to thank CellaVision for providing us with the data, expertise and encouragement needed to carry out this work. Our time at the office in Lund has been a true joy and we appreciate the manifold rewarding conversation with each and every one of you. Specifically, we would like to thank Kent Stråhlen and Mattes Liebsch for guiding us through the project and for providing us with interesting ideas and comments continuously from the start to the very end of the work. Thank you Martin Almers for providing us with the godlike datasets as well as for sharing your enthusiasm and knowledge with us. Also, thank you Jesper Jönsson and Adam Ly for taking your time to proofread the report and give important feedback. A big thank you to the other master thesis students during our time at CellaVision. Namely, Anna Palmquist Sjövall, Oscar Odestål, Martin Carlberg, Oskar Klang, Justin Ma and Julia Fovaeus made the daily work a pleasure, and they also contributed to the implementation by providing countless of valuable discussions about our work.

We would also like to thank our supervisor Kalle Åström at the Department of Mathematics at LTH. His knowledge in the theory related to our work was an irreplaceable source of inspiration.

Contents

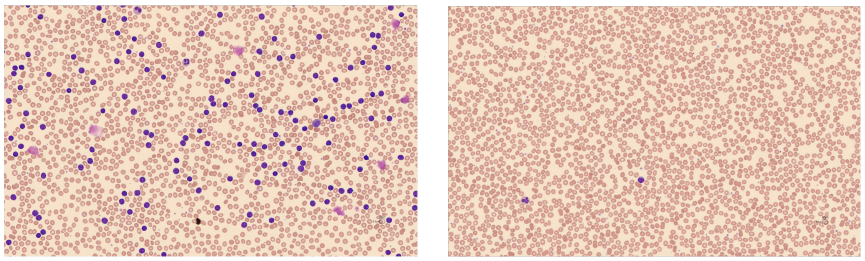
Abbreviations

- **Adam:** Adaptive Moment Estimation
- **AI:** Artificial Intelligence
- **ANN:** Artificial Neural Network
- **BN:** Batch Normalization
- **CNN:** Convolutional Neural Network
- **DBSCAN:** Density-Based Spatial Clustering of Applications with Noise
- **EMD** Earth Mover's Distance
- **ML:** Machine Learning
- **MNIST database:** Modified National Institute of Standards and Technology database
- **MSE:** Mean Squared Error
- **NN:** Neural Network
- **PLT:** Platelet
- **RBC:** Red Blood Cell
- **ReLU:** Rectified Linear Unit
- **VAE:** Variational Autoencoder
- **WBC:** White Blood Cell

1 Introduction

1.1 Motivation

Human blood consists of blood plasma and blood cells. There are three types of blood cells, namely red blood cells (RBCs), white blood cells (WBCs) and platelets (PLTs). The white blood cells, also known as leukocytes, make up for less than one percent of the blood volume but have several tasks in the body. For example, WBCs defend us against bacteria and parasites and inflect allergic inflammatory responses. By analyzing blood samples and the number of different WBCs, one can get information about a patient's health condition since the number of WBCs can be an indicator of several diseases. For instance, an increased number of lymphocytes, a type of WBCs, could mean that the patient has whooping cough. Similarly, a decreased number could suggest celiac disease or AIDS. Significantly increased number of WBCs could indicate leukemia [1]. An example of a blood sample from a patient with leukemia can be seen in figure 1, and a blood sample from a healthy person as a comparison.



(a) *(b)*

Figure 1: Blood smear from a patient with increased WBC concentration indicating leukemia (a), and from a healthy person with normal WBC concentration (b). WBCs are the purple dots in the smear [2].

Traditionally, a medical technologist is studying the blood smear though a conventional microscope, classifying and counting the different types of WBCs. CellaVision's products standardize the work by automation, digital imaging and artificial neural network technologies. Using a neural network, cells are classified, one cell image at a time. However, no information from the sample as a whole is used, i.e., the classification of one WBC is independent of the others of that sample. There may be some information from the other cells in the sample that the cell to be classified can benefit. For example, depending on the slide preparation, the whole sample might have certain characteristics, which can be exploited.

1.2 Aim

The main goal of this master's thesis is to explore the feasibility of classifying multiple cell images from the same blood smear simultaneously using state-of-the-art AI methods. For the investigation of this idea, we set a simpler subtask to simultaneously classify customized subsets of images of handwritten digits.

2 Background

As previously stated, this project mainly relates to analyzing images of WBCs. However, it is known that these images can be relatively hard to handle. We will therefore use another, simpler dataset as well, namely, the Modified National Institute of Standards and Technology database (MNIST database) [3].

2.1 MNIST Digits

The MNIST database is a set of images of handwritten digits, such as those in figure 2. The database is divided into two subsets: a training set of 60,000 images and a test set of 10,000 images. Each digit is a greyscale image that has been size-normalized to fit in a box of size 20x20 pixels and placed such that its center of mass is at the center of a 28x28 image [3]. The careful structure makes the MNIST database relatively simple to process and learn. As of April 2020, the current record classifier is a CNN with 99.84% accuracy on the test set [4].

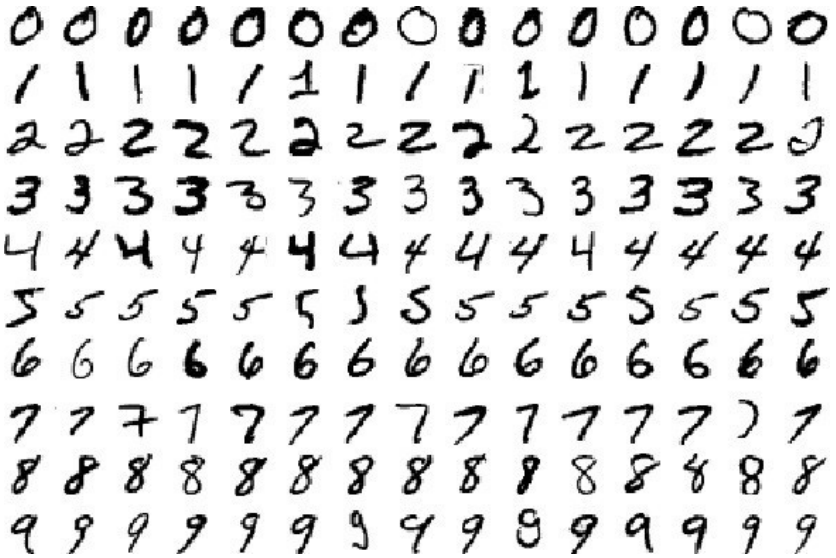


Figure 2: Sample images of the MNIST database. The image is taken from Medium [5].

2.2 White Blood Cells

White blood cells originate from stem cells in the bone marrow, circulate in the blood and end up in tissues and organs. In contrast to red blood cells and platelets, WBCs have a nucleus [1]. There are many ways to divide the WBCs into subgroups, but CellaVision has chosen to divide them into 19 different classes. The 19 classes can be seen in figure 3. CellaVision's machine gets a blood sample and classifies the cells into these classes.[1] The border between the different classes is not distinct. A WBC from one class can continuously

[1]In fact, CellaVision's machines only use 17 of these classes when classifying but there is information about 19 classes, so that is what we are going to use in the report.

evolve into another class. For example, a Promyelocyte continuously evolves to a Myelocyte, so these two classes can be extra hard to distinguish, especially during the transformation. It can therefore be ambiguous which class a WBC should be classified as.

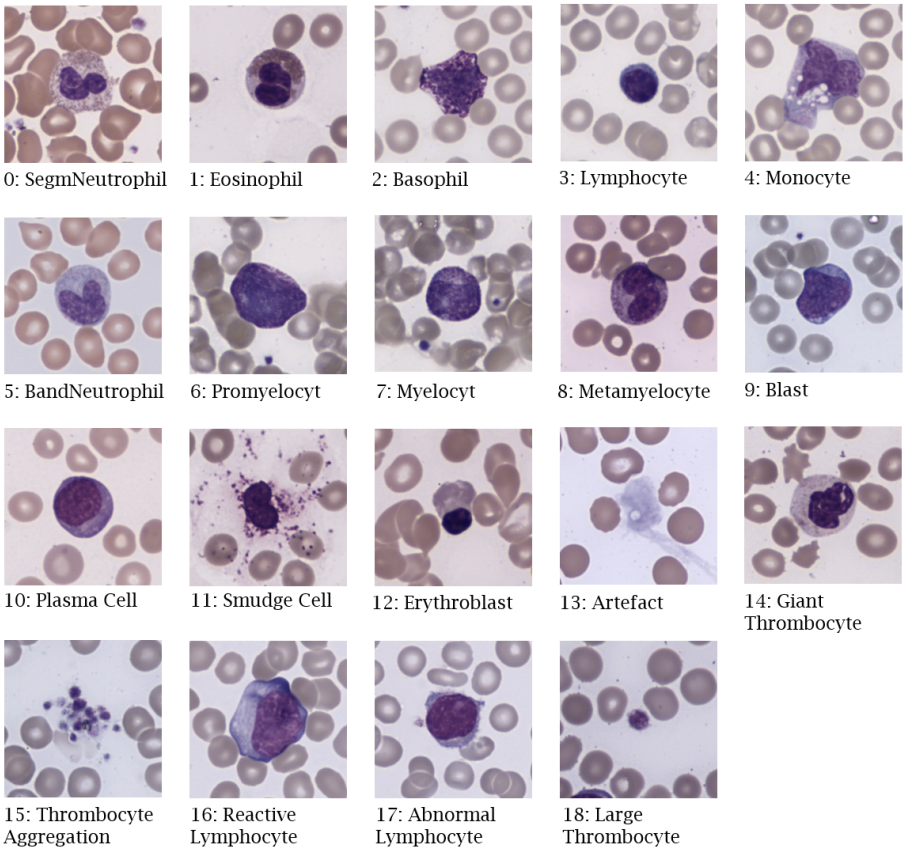


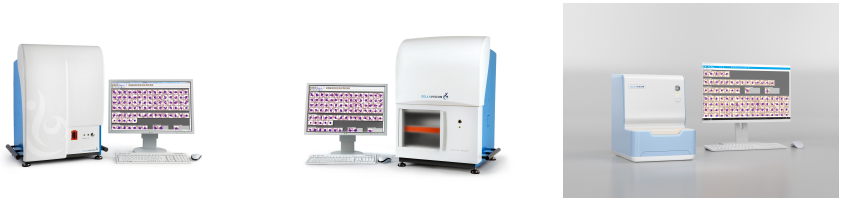
Figure 3: The 19 different classes of WBCs, defined by CellaVision [2].

2.3 CellaVision

CellaVision was founded in 1994 in Lund, Sweden [6]. CellaVision's aim is to automate the conventional microscope work performed in hematology laboratories and to make analysis process faster, better and more flexible [7]. To manually classify and count WBCs in a blood sample can be very time consuming. A lot of time can be saved by letting a machine do the job instead.

CellaVision currently provides four different machines, specialized to match different sized laboratories. These machines are named DM1200, DM9600, DC-1 and DI-60, three of which can be seen in figure 4. In this thesis, images and information from DC-1, DM1200 and DM96 (precursor to DM9600) will be used. When a slide from a blood sample is placed in the machine, the slide is automatically placed under the microscope. WBCs are located and high-resolution

cell images are captured. A cell image usually consists of one WBC and several RBCs. CellaVision provides a pre-classification based on the probability of the WBC belonging to a certain class. The class with the highest probability is selected but two other potential candidates are given. It is up to the expert to verify or correct the pre-classification [8].



(a) *(b)* *(c)*

Figure 4: Three of CellaVision's machines: DM1200, DM9600 and DC-1. The machines automatically capture cell images from blood samples and present pre-classifications [2].

2.4 Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational system inspired by the way the neurons in our brain transmit signals between each other. The building blocks of an ANN are called (artificial) neurons and are defined by Dreyfus [9] as a nonlinear, parameterized, bounded function $y = f(x_1, x_2, ...x_n; w_1, w_2, ..., w_p)$ where x_i are the variables and w_j are the parameters (or weights) of the neuron. Figure 5 shows a schematic illustration of a neuron.

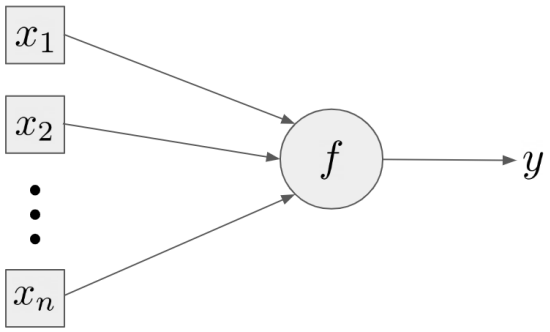


Figure 5: Illustration of an artificial neuron where $y = f(x_1, x_2, ...x_n; w_1, w_2, ..., w_p)$ is a nonlinear, parameterized, bounded function of the variables x_i and the parameters w_j.

Although different ANNs can be of greatly varying complexity, they are all simplified models of a biological neural network. Consider the simple composition of neurons as showed in figure 6. Here, two inputs are sent to two intermediate units, from which the outputs are sent forward to a single-valued output unit. At each neuron, a weighted sum of the inputs is sent through a function outputting the *activation* of that node. Formulated mathematically, this network is represented by the equation

$$y = \tilde{g}\left(w_0^{(2)} z_0 + w_1^{(2)} g\left(z_1\right) + w_2^{(2)} g\left(z_2\right)\right) \tag{1}$$
$$z_1 = w_{1,0}^{(1)} x_0 + w_{1,1}^{(1)} x_1 + w_{1,2}^{(1)} x_2$$
$$z_2 = w_{2,0}^{(1)} x_0 + w_{2,1}^{(1)} x_1 + w_{2,2}^{(1)} x_2$$

where x_1 and x_2 are the inputs and the w's are the weights. Here, $g(\cdot)$ and $\tilde{g}(\cdot)$ are the so called *activation functions* of the intermediate and output layer, respectively. We will discuss different types of common activation functions later on. Note that x_0 and z_0 are called *bias terms* and are commonly set equal to 1. The model discussed here is a simple form of a *feedforward neural network*, which is a composition of neurons grouped into layers. A more general form is illustrated in figure 7, showing a feedforward neural network with $n + 1$ input nodes, m outputs and two hidden layers. The network can of course be expanded to have even more hidden layers, and all layers may have different number of nodes. In a feedforward network, information only flows in one direction. In other words, starting at a specific node and moving along the flow of information, there is no way of getting back to the starting node [9]. There might however exist connections between nodes of nonadjacent layers.

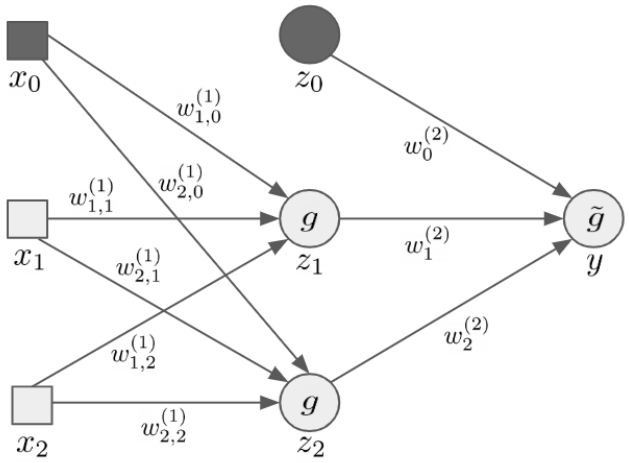


Figure 6: A simple example of a feedforward neural network. Here, x_0 and z_0 are the bias nodes of the input and intermediate layer, respectively.

Activation functions

The choice of activation function $g(\cdot)$ for a node is arbitrary, but there are a few common types of functions that are efficient for different tasks. The most simple activation function is of course the linear function $g(x) = x$. Introducing the modification of setting all negative values to zero, we get arguably the most successful and widely-used activation function [10] called the Rectified Linear Unit (ReLU), defined as

$$g(x) = \max(x, 0). \tag{2}$$

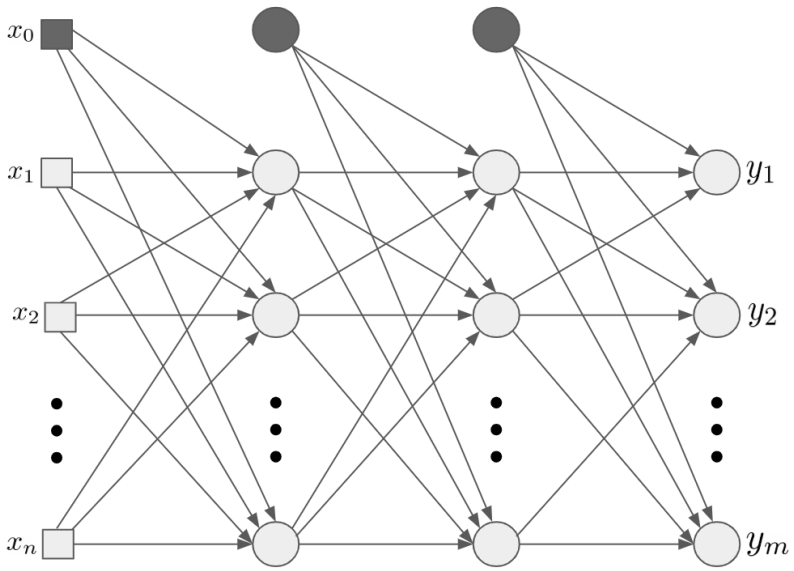


Figure 7: A feedforward neural network with $n+1$ input nodes $\{x_i\}$, m outputs $\{y_j\}$ and two hidden layers.

See figure 8a for a plot of the ReLU. While the simplicity and part-linearity of the ReLU makes it a convenient choice in many cases, it can often be advantageous to use a *threshold unit* such as the Heaviside step function, defined as

$$g(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{3}$$

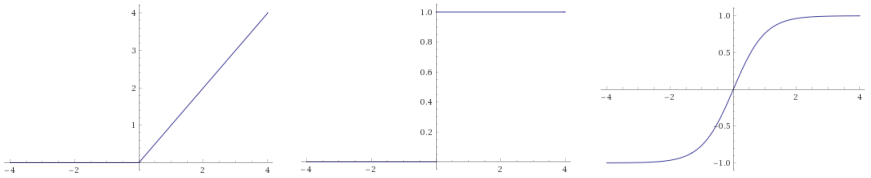
and plotted in figure 8b [11]. Continuous approximations of the threshold units are called *sigmoidals* [11]. A commonly used sigmoidal is the hyperbolic tangent (tanh), plotted in figure 8c and defined as

$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{4}$$

The differentiability of sigmoids often make them preferable over the discrete step function, as will be discussed later. Note that tanh can be made arbitrarily close to a step function between -1 and 1 by setting all the weights to very large values [11].

In the case of multiclass classification problems, it is common practice to use the so-called *softmax* activation function for the output nodes. Let a_k be the total input to node k of the last layer. Then the output y_k of that node will be

$$y_k = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})} \tag{5}$$



(a) *(b)* *(c)*

Figure 8: Three common activation functions: ReLU (a), Heaviside (b) and tanh (c).

where the summation is over all nodes of the last layer [11]. This will result in an output vector where the "winning" node, i.e., that with the highest value before applying the softmax function, will have a large value (ideally, close or equal to 1), while all other nodes will have a small positive value. The winning node will then be selected as the predicted class. Since the values must range between 0 and 1 and sum up to 1, they can be interpreted as probabilities.

2.5 Optimization

Cost function

Consider an ANN aiming to solve a binary classification, such as it aims to separate a class $\mathcal{C}_0$ from another class $\mathcal{C}_1$. Such a network will have a one dimensional output $y \in [0, 1]$ for each input vector $\mathbf{x}$, where the value of y should be interpreted as the predicted probability that the data belongs to $\mathcal{C}_1$, i.e., $P(\mathcal{C}_1|\mathbf{x}) = y$ (and equivalently, $P(\mathcal{C}_0|\mathbf{x}) = 1 - y$) [11]. For each input $\mathbf{x}$, there is also a target t representing the true class of the data, such that $t = 0$ in the case of $\mathcal{C}_0$ and $t = 1$ in the case of $\mathcal{C}_1$. The value of t can be either known or unknown, but considering supervised learning, the value of t is known. Combined into a single expression, the probability of observing either target value is

$$p(t|\mathbf{x}) = y^t(1 - y)^{1-t}. \tag{6}$$

Then, the likelihood of observing a complete dataset of many individual inputs $\mathbf{x}^n$, outputs y^n and targets t^n is given by

$$\prod_n (y^n)^{t^n}(1 - y^n)^{1-t^n}, \tag{7}$$

where the product is taken over all samples [11]. Maximizing this expression is advantageously done by instead minimizing the negative logarithm of it, leading to the *cross-entropy* error function

$$E = -\sum_n \{t^n \ln(y^n) + (1 - t^n)\ln(1 - y^n)\}, \tag{8}$$

which is to be minimized in order to optimize the performance of the binary classification network [11]. Segueing into the more general case of multiple

classes, consider a classification with m mutually exclusive classes. The output $\mathbf{y}^n$ of the ANN will then be m-dimensional, where each output y_k^n represents the predicted probability of the data belonging to the kth class, $\mathcal{C}_k$. The target vector $\mathbf{t}^n$ will then be a so-called *one-hot encoded* vector with as many elements as the number of classes, with a single 1 indicating the label, such that

$$t_k^n = \begin{cases} 1 & \text{if pattern } n \in \mathcal{C}_l \text{ and } k = l \\ 0 & \text{if pattern } n \in \mathcal{C}_l \text{ and } k \neq l. \end{cases} \tag{9}$$

The conditional distribution equivalent to equation (6) can be written as

$$p(\mathbf{t}^n|\mathbf{x}^n) = \prod_{k=1}^{m} (y_k^n)^{t_k^n}. \tag{10}$$

Forming the error function as the negative logarithm of the likelihood function as before results in

$$E = -\sum_n \sum_{k=1}^{m} t_k^n \ln(y_k^n) \equiv \sum_n E^n, \tag{11}$$

which should be minimized in order to optimize the network [11].

Gradient descent

Considering networks with differentiable activation functions, it is typically an easy task to evaluate the derivatives of the error function with respect to the weight parameters [11]. This paves the way for a variety of gradient-based optimization algorithms, one of the simplest being *gradient descent* [11]. By grouping all parameters of the network into a single vector $\mathbf{w}$, the error function can be expressed as $E = E(\mathbf{w})$. Assuming E is differentiable with respect to $\mathbf{w}$, gradient descent offers an efficient yet intuitive updating scheme for the weights. Starting with a (possibly random) initial guess for $\mathbf{w}$, it can then be updated iteratively by moving a small distance η (called *learning rate*) in the direction in which E decreases most rapidly. Formulated mathematically, we can generate a sequence of weight vectors $\mathbf{w}^{(\tau)}$ whose components are calculated as

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \frac{\partial E}{\partial w_{kj}} \bigg|_{\mathbf{w}^{(\tau)}}. \tag{12}$$

Using the error function depending on all the training examples for each update of the parameters as above is called *batch gradient descent* [12]. As is the case for equation (11), the error function can often be expressed as a sum of terms where each term depends only on one pattern from the training set, i.e.,

$$E(\mathbf{w}) = \sum_n E^n(\mathbf{w}). \tag{13}$$

Then, the updating scheme simplifies to the formula

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \frac{\partial E^n}{\partial w_{kj}}, \tag{14}$$

which is repeated many times by cycling through all patterns [11]. This method of updating the weights just one example at a time is called *stochastic gradient descent* and is a faster but less precise alternative to batch gradient descent [12]. Naturally, it is possible to form a combination of these alternatives, by using *mini batches* of *some* patterns for each weight update. This is called *mini batch gradient descent* [12]. Although there are more sophisticated optimization methods, gradient descent illustrates the logic of using derivatives of the error function to update the parameters of the network. Another common optimization method is called *Adam* and can be read about in detail in the original article by Kingma and Ba [13].

Error back-propagation

Calculating the gradients needed to perform gradient descent is advantageously handled by an algorithm called *back-propagation* [11]. The idea is to propagate errors backwards through the network, as we will now describe mathematically. The derivation of the algorithm is adapted from Bishop [11].

Each unit j of a general feed-forward network computes a weighted sum of its inputs of the form

$$a_j = \sum_i w_{ji} z_i, \tag{15}$$

where z_i is the input from a previous unit i and w_{ji} is the corresponding weight. The sum is then transformed by an activation function $g(\cdot)$ to form the activation z_j of unit j as

$$z_j = g(a_j). \tag{16}$$

Note that in the special case of the first layer of the network, the variables z_i are the inputs x_i to the network. Similarly, the unit j could be an output unit. In that case, its activation should be denoted by y_k. Note also that we don't need to handle the biases explicitly, as they can be treated as units with activation fixed as $+1$.

Consider an error function satisfying the condition in equation (13) and where each term E^n can be expressed as a differentiable function of the network output variables y_k. Suppose that we have calculated the activations of all units of a network, given a known input pattern, and say we want to evaluate the derivative of the corresponding term E^n with respect to some weight w_{ji}. Although the outputs of the various units will depend on the particular input pattern n, we will herein omit the supscript n from the input and activation variables to keep the notation uncluttered, as we shall consider one pattern at a time. It holds that E^n depends on the weight w_{ji} only via a_j. The chain rule therefore gives that

$$\frac{\partial E^n}{\partial w_{ji}} = \frac{\partial E^n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \equiv \delta_j \frac{\partial a^n}{\partial w_{ji}}, \tag{17}$$

where we have introduced the notation

$$\delta_j = \frac{\partial E^n}{\partial a_j}. \tag{18}$$

From equation (15) we get that

$$\frac{\partial a_j}{\partial w_{ji}} = z_i, \tag{19}$$

which together with equation (17) gives that

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j z_i. \tag{20}$$

Calculating the derivative of the error term with respect to a certain weight is thus reduced to calculating the corresponding δ and then applying equation (20). The task of calculating δ_k is relatively simple for output nodes. Indeed, using equation (16) with z_j denoted by y_k we get

$$\delta_k = \frac{\partial E^n}{\partial a_k} = g'(a_k)\frac{\partial E^n}{\partial y_k}. \tag{21}$$

For the hidden layers, we again make use of the chain rule to express

$$\delta_j = \frac{\partial E^n}{\partial a_j} = \sum_k \frac{\partial E^n}{\partial a_k}\frac{\partial a_k}{\partial a_j}, \tag{22}$$

where the sum runs over all units i to which unit j sends connections. We can now substitute the definition of δ given by equation (18) into equation (22) and use equation (15) and equation (16) to obtain the back-propagation formula

$$\delta_j = g'(a_j)\sum_k w_{kj}\delta_k. \tag{23}$$

Thus, we can evaluate δ_k for all the output units using equation (21) and then propagate backwards using equation (23) to obtain δ_j for each hidden unit in the network. We can then use equation (20) to get the derivatives needed for the iterative optimization of the network parameters.

Batch normalization

A conceivable issue with the optimization scheme during training of a network is that activations flowing from one layer to the next might have very different distribution between iterations. This variability makes it harder for the optimizer to find suitable parameters for the latter layer. As specified by Ioffe and Szegedy [14], one way to handle this and thus speeding up and stabilizing the training is *batch normalization* (BN). The idea is to normalize the activations with respect to their mean value and variance for the current iteration. Given a batch of activations $\mathcal{B} = \{x_i\}_{i=1,\dots,m}$ of size m, the mean and variance can be calculated as

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{24}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2, \tag{25}$$

and the batch is then normalized according to

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \tag{26}$$

where the small value ϵ is added to the variance for numerical stability [14]. The learnable parameters γ and β are introduced to form the output of the BN layer as

$$y_i = \gamma\hat{x}_i + \beta, \tag{27}$$

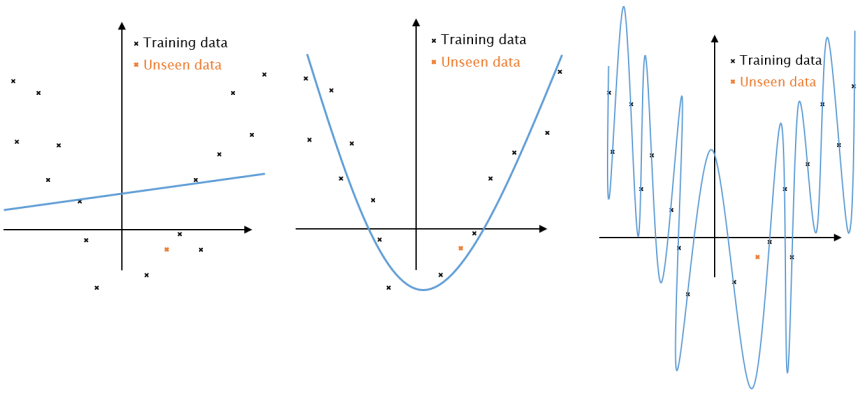
to allow the identity transform in case that is optimal [14].[2] Here, x_i and y_i denote the inputs and outputs of the batch normalization layer and not the whole network. In the case of multidimensional activations, BN is done componentwise and the model has two parameters $\gamma^{(k)}$ and $\beta^{(k)}$ for each dimension k in the activations entering the BN layer [14].

2.6 Generalization

Since a high number of hidden nodes in a network in general increases its ability to fit to the data that it trains on, it can be tempting to use as many hidden nodes as computationally possible when choosing the model architecture for a given problem. However, there is a substantial risk of fitting such a model too well to the training data, in the sense that it would be less likely to perform well on unseen data. This phenomenon is called *overfitting* and we say that the model's ability to *generalize* is reduced. A network with too many parameters will pick up information that is present in the training data but that is meaningless in terms of the underlying function that we aim to model [9]. By contrast, a network with too few parameters will not be complex enough to model the

[2] Indeed, $\gamma = \sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}$ and $\beta = \mu_{\mathcal{B}}$ may be used to recover the original activations.

structures of the data. A simple example of good and bad generalization is visualized in figure 9. A widely used method to detect overfitting is to divide the data into a training set, on which the model is trained, and a validation set, on which its generalization performance can be evaluated. As the training process of a network proceeds, the performance on the training data (called *training performance*) will in general increase indefinitely. However, the performance on the validation data (called *validation performance*) will only increase up to a certain point, and will then typically decline due to overfitting. This is illustrated in figure 10. There is a variety of techniques to handle this issue, one of which being *dropout*.



(a) *(b)* *(c)*

Figure 9: Basic example of underfitting (a), good generalization (b) and overfitting (c) of a regression model. The overfitted model can fit to the training data perfectly, but performs poorly on unseen data.

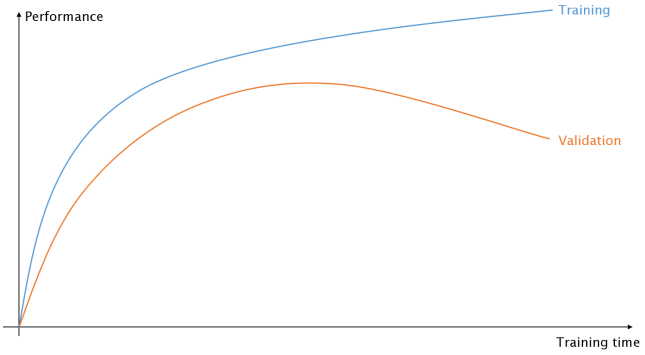
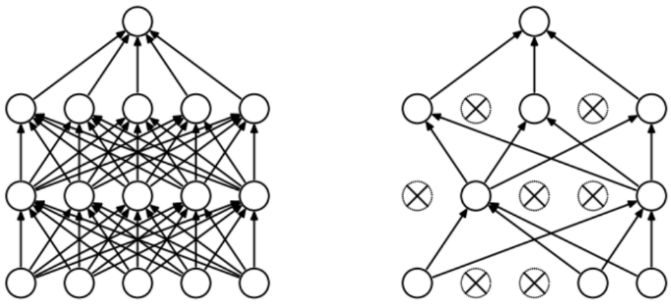


Figure 10: General plot of the typical behaviour of training and validation performance of a model during training. Overfitting occurs after a certain point.

Dropout

A straightforward way to enhance generalization is to train multiple networks independently, using different subsets for training and validation of the same data set, and then form an *ensemble* by averaging over these networks. Although

this will yield a more robust result, it's a computationally expensive method that quickly becomes impracticable as the number of subnetworks increases. Dropout provides an inexpensive approximation of this method by training an ensemble consisting of subnetworks that can be formed by removing nonoutput units from an underlying base network [15]. An example of such a subnetwork is illustrated in figure 11. Specifically, when applying dropout to the training process of a network, a subnetwork is constructed for each batch of data by removing each unit with a probability that is specified as a hyperparameter beforehand. The removal of a node is put into practice by multiplying the output from that node by zero. In contrast to conventional ensemble training, a tiny fraction of all possible subnetworks are each trained for a single step, and the weights of the base network are updated after each step. In the case of large models, it is impossible to train all possible subnetworks individually due to the enormous number of possible configurations. However, since all subnetworks share weights from a common base network, the training of just a fraction of the subnetworks will yield good settings for the remaining ones as well [15].



(a) *(b)*

Figure 11: Example of dropout applied on a base network (a) to construct a subnetwork (b). The images are taken from Medium [16].

2.7 Convolutional Neural Networks

A general *fully connected layer* with unique weights for all possible connections between the nodes of two adjacent layers in a network quickly infers a massive amount of parameters as the number of nodes increases. It is sometimes more efficient to let some connections share weights, provided that there is some structure in the data that allows for shared weights. Then, the number of *unique* weights will decrease, and the optimization will be faster. When dealing with image data, it makes sense to implement shared weights by using the *convolution* operation. A *convolutional neural network* (CNN) is a network with at least one such layer.

Mathematically, the discrete convolution of a two-dimensional image I and a two-dimensional kernel K is defined as

$$(I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n). \tag{28}$$

Note that the kernel is flipped in relation to the image, in the sense that as m and n increases, the index into the kernel decreases. The mathematical reason to use this definition is that it yields a commutative property. In the case of ANNs, this property is often not useful, and it is common to use the non-flipped version of the convolution operation. This is also known as the cross-correlation, and is defined as

$$(K * I)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n). \tag{29}$$

It is common to call this operation convolution as well, which we will do in this work from here on. The conceptual meaning of this is that the kernel will traverse the image and perform elementwise multiplication. This is visualized in figure 12 for an image of size 5×5 and a kernel of size 3×3. When applying the operation, it is possible to add a frame of zeros around the image. This is known as *zero-padding*. In the example in figure 12, zero-padding would result in an output of size 5×5 instead of 3×3. Another widely used parameter is the *stride*, which is one in this example. The stride determines how many pixels the kernel will move for each step of the convolution, so that with a stride of two, the kernel will skip every other position in the image. A stride of two in both dimensions would result in an output of size 2×2 in the example if no padding is used. In the case of RGB images, where each input has a depth of three channels, a 3×3 kernel will be applied in $3 \times 3 \times 3$ blocks, still mapping to only one output per step in the convolution. This is visualized in figure 13. A 1×1 kernel will *only* consider cross-channel correlations, and can be used for downsampling the number of channels.

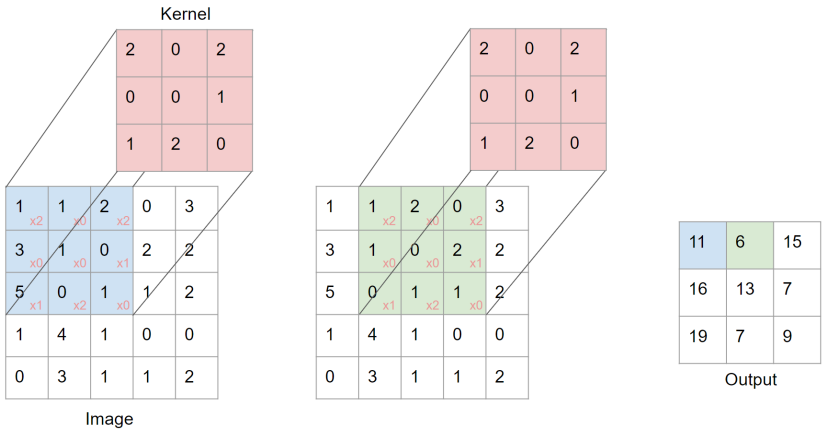


Figure 12: Example of the convolutional operation on a 5×5 image and a 3×3 kernel. Here, the convolution is performed with a stride of one and with no padding.

Following a convolutional layer, the output is often sent through an activation function followed by a *pooling* layer which modifies the output further. For example, *max pooling* calculates the maximum value of a rectangular neighbourhood, the size of which being a hyperparameter. Equivalently, *average pooling* does exactly what you think it does. Pooling serves to enhance the model's

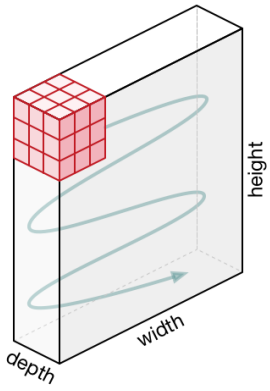


Figure 13: Movement of the kernel operating on an image with three channels. The image is taken from Saha [17].

robustness towards translations in the input, so that with a slightly translated input image, the values of most output nodes don't change.

Separable convolutions

When handling data with multiple channels, such as RGB images, it is possible to do the convolutional operation in two steps, where the cross-channel (pointwise) and spatial convolutions (depthwise) are done separately. This is called *separable convolutions* and can be interpreted as factorizing the conventional convolution into two parts: one handling cross-channel correlations and another handling spatial correlations, before being concatenated. This is illustrated in figure 14. An advantage in relation to ordinary convolutions is the potential decrease in number of weight needed to produce many output channels, which can speed up calculations and prevent overfitting.

Xception

There are numerous convolutional networks publicly available that have been proven to perform well on massive sets of image data. One such is the *Xception* network, developed by Chollet [19] in 2017 as an "extreme" version of the somewhat more well-known *Inception* network. A consistent theme of Inception is the usage of modules consisting of parallel convolutional layers in contrast to the ordinary series connection of layers. Then, the input will be split into separate branches and then concatenated again after a number of operations. An example of this is showed in figure 15a. The idea of Xception is to introduce a modification of the separable convolution that does the pointwise convolutions before the depthwise convolutions. Figure 15b shows an example of this. The input is first processed by a number of 1×1 kernels, each producing an output with depth one. These are then processed spatially by for example 3×3 filters. In relation to Inception, Chollet showed an increased performance of the Xception due to a more efficient use of model parameters. The full architecture of Xception is visualized in figure 16 and consists of three large parts that are repeated to form the full network.

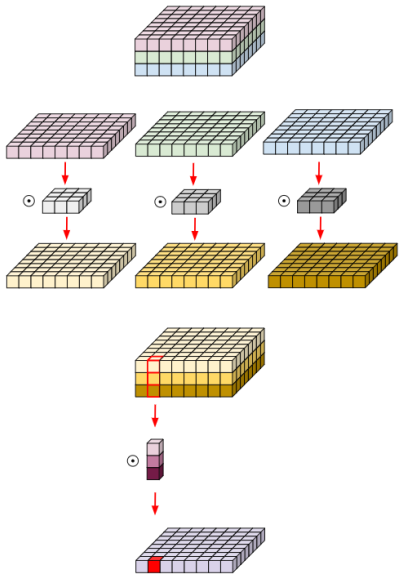
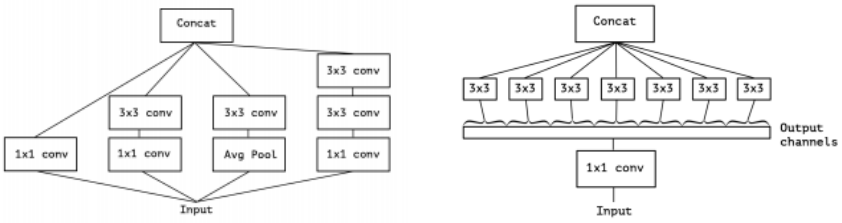


Figure 14: Depthwise convolution with a 3×3 kernel, followed by pointwise convolution with a 1 × 1 kernel. The image is taken from [18].



(a) *(b)*

Figure 15: Example of modules used in Inception (a) and Xception (b). The input is split into separate branches and then concatenated again after a number of operations. The examples are taken from Chollet [19].

2.8 Autoencoders

ANNs can be used for downsampling, by training it to copy its input to its output through a "bottleneck" of lower dimensionality. This type of network is called an *autoencoder* and can be viewed as consisting of a downsampling *encoder* and an upsampling *decoder* [15]. The encoder takes the input $\mathbf{x}$ and outputs a hidden layer $\mathbf{z} = f(\mathbf{x})$ of smaller dimension than the input. The decoder then takes the hidden layer and produces a reconstruction $\mathbf{r} = g(\mathbf{z})$, where $\mathbf{r}$ aims to be as close to $\mathbf{x}$ as possible. During training, $\mathbf{x}$ is therefore used both as input and as target to the autoencoder, and a mean squared error (MSE) cost function is typically used. The downsampling and upsampling structure makes it suitable to represent the autoencoder schematically as in figure 17a. The hidden layer is at the bottleneck (also known as *latent space*) of the model as it is the smallest layer in terms of dimensions. After training, the encoder may be used on its

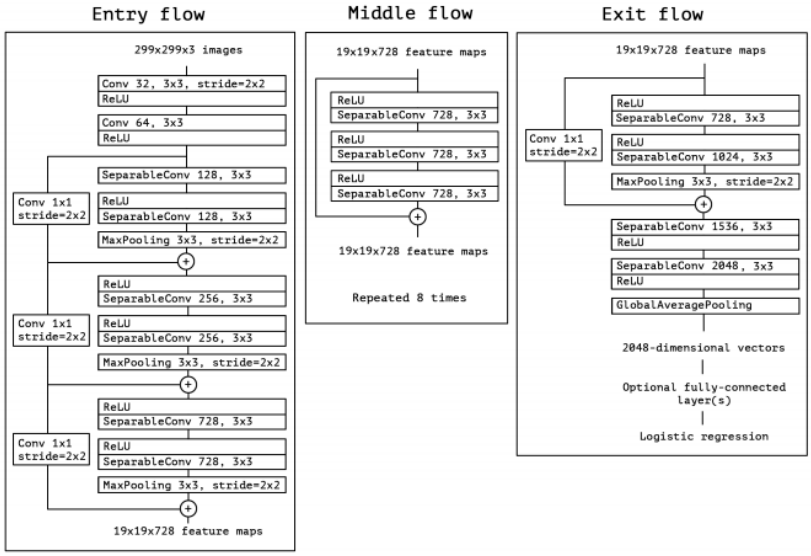


Figure 16: Architecture of the Xception network, produced by Chollet [19]. It consists of three large parts that are repeated to form the full network. The entry and exit flows are applied once, while the middle flow is repeated eight times.

own to downsample new data. If trained well, it will be able to encode data with marginal loss of information.

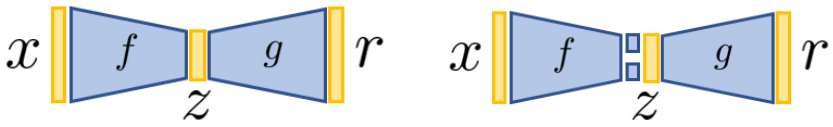


Figure 17: Basic schematic illustration of an autoencoder (a) and a VAE (b). The VAE learns two parameters of a distribution before sampling $\mathbf{z}$.

A probabilistic type of autoencoder is the *variational autoencoder* (VAE). The VAE is also built upon an encoding and a decoding part, but instead of encoding the input directly to the latent representation $\mathbf{z}$, it produces a *latent distribution* $p(\mathbf{z}|\mathbf{x})$ [20]. The representation $\mathbf{z}$ is then sampled and decoded to form the reconstruction $\mathbf{r} = g(\mathbf{z})$ as usual through the decoder. Specifically, the VAE learns the mean and variance of the latent distribution, as illustrated in figure 17b. While the ordinary autoencoder is solely trained to encode and decode with as little loss as possible, the stochasticity of the VAE forces it to organize the latent space logically, in the sense that even the parts of the latent space with no known data get meaningful. The result is, ideally, a latent space where the transitions between different classes of data are more smooth, and where the positions and distances between classes have visual meaning [20]. This is visualized in figure 18a, illustrating how unseen points from the latent space can generate meaningful outputs. In contrast, such structures would be irrelevant for the ordinary autoencoder to pick up, and it will therefore generate nonsense

as output as seen in figure 18b. The VAE is a powerful tool for generating data. If the VAE is used for downsampling only, it is in practice reduced to an ordinary autoencoder.

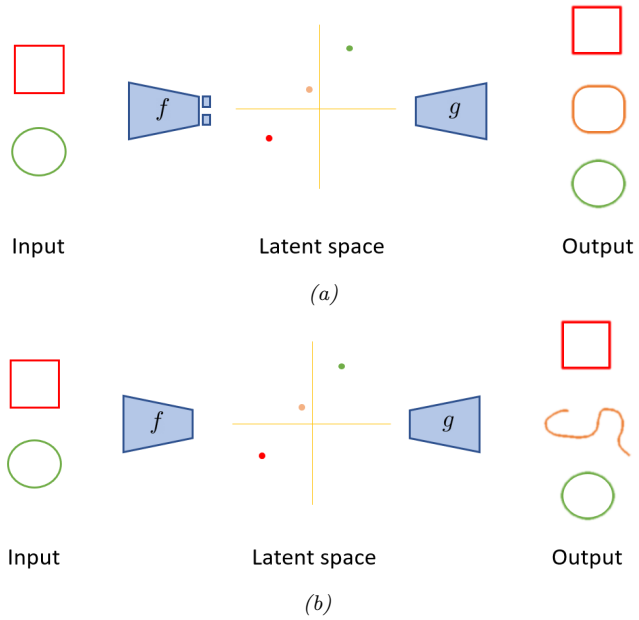


Figure 18: Illustration of the generative property of the VAE (a) that the ordinary autoencoder lacks (b). An unseen type of data (orange) is sampled from the latent space and decoded. The VAE outputs something that makes sense in relation to the seen data, while the autoencoder is highly confused.

2.9 DBSCAN Clustering

An important technique within machine learning is *clustering*, which aims to group data samples based on similarity. There are several clustering algorithms with different pros and cons, for many of which the number of wanted clusters needs to be specified. The amount of well-known algorithms quickly decreases if we don't want the number of cluster to be fixed, but one such is the *Density-based spatial clustering of applications with noise* (DBSCAN). For the sake of conceptual understanding, imagine a dataset $\mathcal{D}$ in a two dimensional space where the distance between two samples p and q, $dist(p, q)$, can be calculated using the ordinary Euclidean metric ($\mathcal{L}^2$ norm). Then, DBSCAN fundamentally depends on the ϵ-*neighbourhood* of each sample. For a certain sample p, the ϵ-*neighbourhood* of p, denoted $N(p)$, is the set of all samples in the same dataset which lie within a distance of ϵ from p. This is defined mathematically as

$$N(p) = \{q \in D | \operatorname{dist}(p, q) \leq \epsilon\}, \tag{30}$$

where the value of ϵ is set as a parameter in the algorithm [21]. For each sample, DBSCAN calculates its ϵ-*neighbourhood* and counts the number of samples in

that neighbourhood. If that number is larger than or equal to a certain number *min_samples* (which is the other parameter of the algorithm), p will be classified as a *core point* of a cluster. All other samples in the neighbourhood of a core point can either be a core point to the same cluster, or a *border point*, meaning that it is within the neighbourhood of a core point but is not one itself. Connected core points and border points are set to belong to the same cluster. Samples that are neither a core point nor a border point of any cluster are said to be *noise* [21]. These samples can be treated as outliers that don't belong to any cluster. The above logic is illustrated in figure 19. The same logic can naturally be applied in a space of arbitrarily many dimensions.

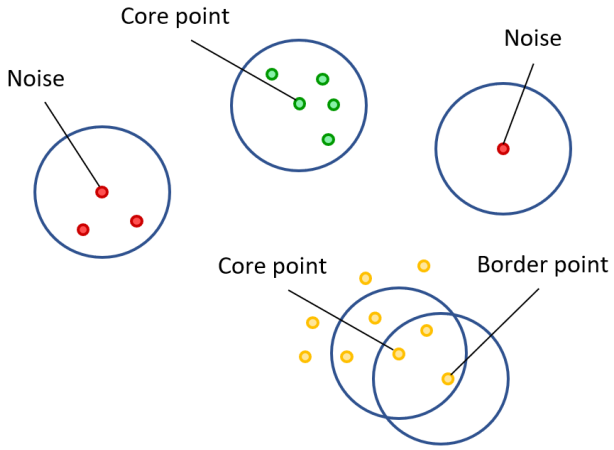


Figure 19: Example of how the DBSCAN algorithm clusters data. Here, min_samples = 5. The blue circles represent ϵ-neighbourhoods of some samples. Core points have at least five samples within their neighbourhood. Border points belong to the neighbourhood of a core point. Connecting core and border points are set to belong to the same cluster. In this example, DBSCAN produces two clusters (yellow and green). All samples which are neither a core point nor a border point are classified as noise (red).

As already mentioned, a benefit of the DBSCAN algorithm is that the number of clusters is not fixed. Conceivable drawbacks include the need for decently constant densities of different types of data. Furthermore, the parameters ϵ and *min_samples* need to be specified by the user, which might be hard without vast knowledge about the behaviour of the data.

2.10 Earth Mover's Distance

The *Earth Mover's Distance* (EMD), also know as *Wasserstein distance*, is a statistical measure of the distance between two probability distributions [22]. The name stems from the analogy to moving piles of dirt of certain weights into holes of certain sizes using as little work as possible. Similarly, EMD is described in [23] as proportional to the minimum amount of work required to convert one distribution into another, where 1 unit of work is the amount of work necessary to move one unit of weight by one unit of distance. Consider a set of m points $P = \{p_i | i \in [1, m]\}$ ("holes"), each with weight w_i, and another

set of n points $Q = \{q_j | j \in [1, n]\}$ ("piles"), each with weight u_j.[3] Let d_{ij} be the distance between p_i and q_j. We want to find the optimal *flow* represented by an $m \times n$ matrix F with elements f_{ij} corresponding to the amount of weight at p_i matched to q_j, such that the minimum amount of work is needed. By mathematical formulation, we want to find the minimum work

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij}, \tag{31}$$

subject to the constrains

$$\begin{cases} f_{ij} \geq 0 \quad \text{where} \quad 1 \leq i \leq m, 1 \leq j \leq n, \\ \sum_{j=1}^{n} f_{ij} \leq w_i \quad \text{where} \quad 1 \leq i \leq m, \\ \sum_{i=1}^{m} f_{ij} \leq u_j \quad \text{where} \quad 1 \leq j \leq n, \\ \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} = \min \left\{ \sum_{i=1}^{m} w_i, \sum_{j=1}^{n} u_j \right\}. \end{cases} \tag{32}$$

The EMD is then defined as the work normalized by the total flow,

$$\text{EMD}(P, Q) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij}}, \tag{33}$$

where the minimum work corresponds to the optimal EMD [24]. Figure 20 provides a visualization of optimal and non-optimal flow and calculation of the EMD for a simple example.

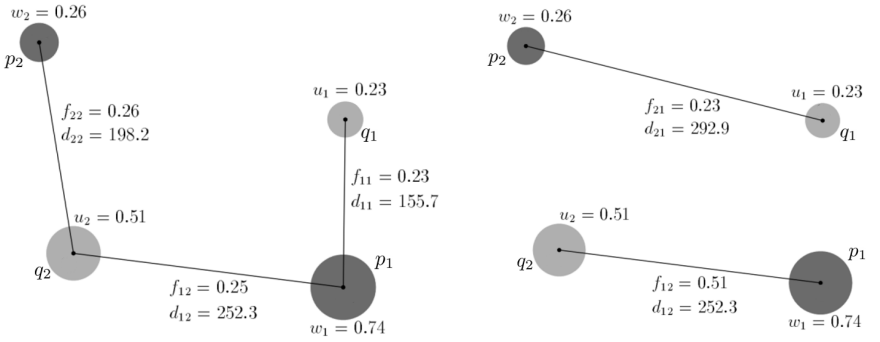


Figure 20: Example of optimal flow (a) and non-optimal flow (b). The total weight of the lighter distribution (Q) is 0.74*, so* EMD $= 150.4/0.74 = 203.3$ *for the optimal flow. The figure is inspired by Example 4 in [23].*

[3] In the analogy to moving dirt, if the sum of weights in the two distributions differ, think of Q as the lighter distribution. Mathematically, in the opposite event, the problem becomes a matter of instead moving the holes to the piles.

2.11 Bayesian Optimization

Bayesian optimization is a method for finding the extrema of functions. The algorithm constructs a posterior Gaussian process that describes the (unknown) objection function by iteratively obtaining observations of this function at sampled values [25]. It is particularly powerful when the objection function is expensive to evaluate, as it uses an *acquisition function* at each step to sample efficiently [26]. The decision of where to sample next boils down to a trade-off between *exploration* and *exploitation*. Specifically, the algorithm aims to utilize its prior knowledge to sample close to expected extrema (exploitation) while at the same time obtain more information about areas it knows little about (exploration). The name stems from the well-known *Bayes' theorem*, which states that the posterior probability of a model M given evidence E is proportional to the likelihood of E given M multiplied by the *prior* probability of M:

$$P(M|E) \propto P(E|M)P(M). \tag{34}$$

In the case of Bayesian optimization, let $\mathbf{x}_i$ denote the ith sample, yielding the evaluation $f(\mathbf{x}_i)$ of the objective function f. As we accumulate observations $\mathcal{D}_{1:t} = \{\mathbf{x}_{1:t}, f(\mathbf{x}_{1:t})\}$, up to time t, Bayes' theorem tells us that the posterior distribution of f given all observations is proportional to the prior distribution multiplied by the likelihood function $P(\mathcal{D}_{1:t}|f)$:

$$P(f|\mathcal{D}_{1:t}) \propto P(\mathcal{D}_{1:t}|f)\, P(f)[26]. \tag{35}$$

The prior $P(f)$ represents our belief about the space of possible objective functions [26]. This relation is used in the algorithm to guess the underlying function based on observations. A simple example of how Bayesian optimization works is illustrated in figure 21, taken from [26].

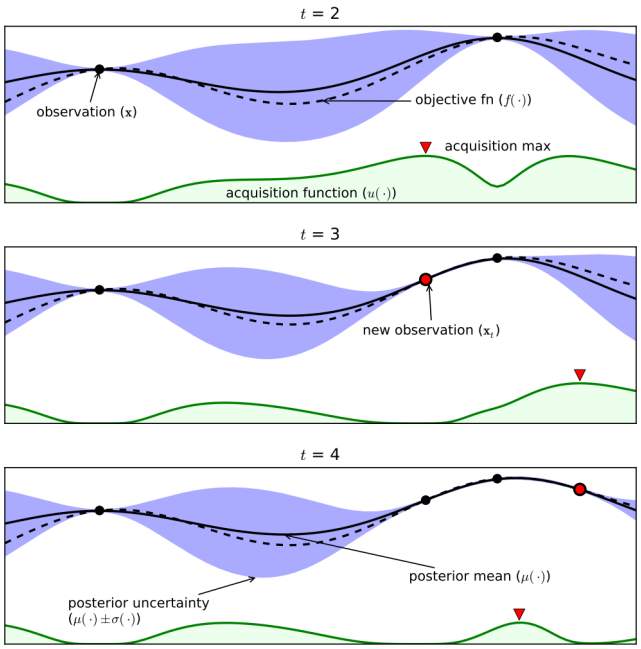


Figure 21: Example of Bayesian optimization applied on a simple problem in one dimension. At each step, the maximum of the acquisition function gives the point of the next sample, and the evaluation at that point updates the posterior approximation and uncertainty of the unknown objective function. The image is taken from [26].

3 Methodology

Although the main goal of this work is related to WBC data provided by CellaVision, we start off with the simpler MNIST database. We will construct personalized subsets of MNIST data that are customized to fit our methods. These subsets of multiple digits are then represented by blood samples of multiple cell images in the WBC case. We hope that the personal correlations within a blood sample can be simulated by subsets of hand-picked MNIST digits exhibiting visual similarities. The algorithms used for both problems follow the same overall structure. The pipeline consists of a CNN and an encoder, on whose output the DBSCAN clustering method is applied to form clusters that finally are classified by minimizing the EMD. The pipeline is visualized in figure 22 and will be described in detail in the following sections.

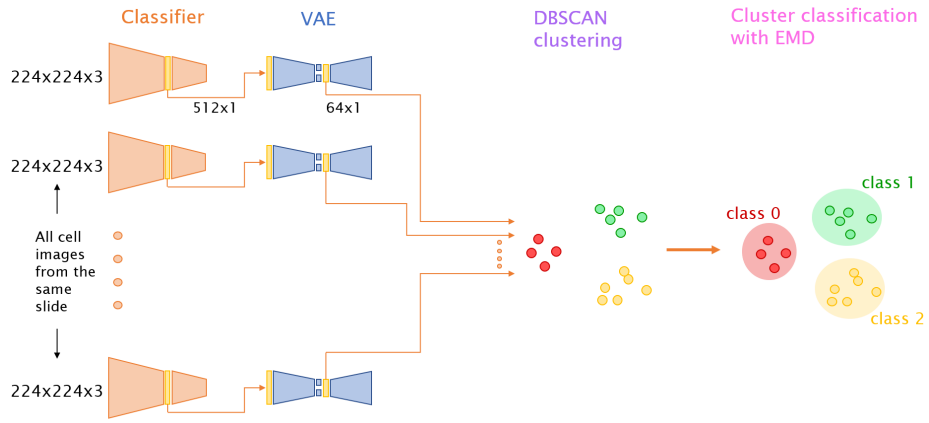


Figure 22: Pipeline of the methods used in this work. Observe that the clustering can be done in arbitrarily many dimensions and not only in 2 dimensions as illustrated in the figure. The numbers displayed in the image correspond to the WBC data, but the overall structure is the same for MNIST data.

3.1 MNIST Digits

Dataset

We initiated the research by downloading the MNIST database of a large training set of 60,000 images and a smaller test set of 10,000 images, and then trained a classifying CNN on the training set. The images were normalized by dividing the image matrices by 255, yielding a value between 0 and 1 for each pixel.

Classification

Throughout this work, Keras together with TensorFlow version 2.0 was used as deep learning library. A CNN, structure taken from Keras website [27], was built to classify the MNIST digits. An illustration of the CNN network can be seen in figure 23. As there are ten classes of the MNIST data, we used ten output channels. We used the Adam optimizer with the default parameters:

learning rate=0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The model was trained for 12 epochs with a batch size of 128 samples. The test accuracy of this network is important to note, as this is the benchmark that we wish to reach, or even exceed, using the techniques introduced in this work.

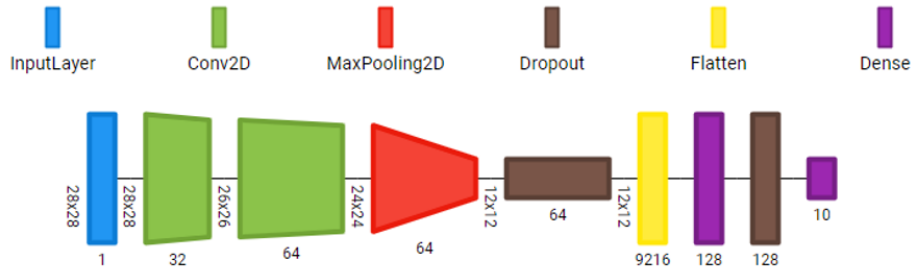


Figure 23: The CNN network used to classify the MNIST digits. The illustration is made with the tool Net2Vis [28].

Dimensionality Reduction

Proceeding, we extracted the outputs from the intermediate dense layer of the network in figure 23, such that a vector of length 128 was extracted for each input image (herein called "128-vectors"). The training set of these vectors (i.e., those generated from the training images) was then used to train a relatively simple VAE of two encoding layers, two decoding layers and a latent space of eight dimensions. The architecture of the VAE is visualized in figure 24.

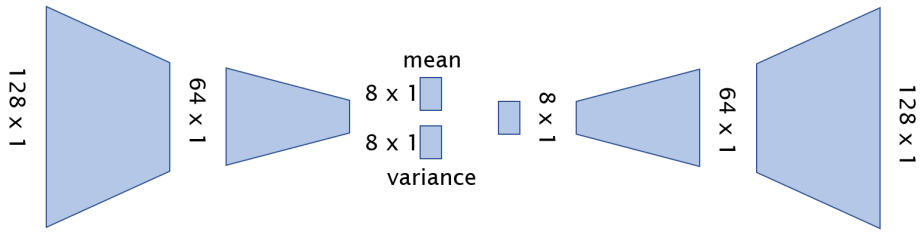


Figure 24: The VAE used to reduce the dimension of the 128-vectors further.

To evaluate the VAE, the VAE output was fed into the last part of the CNN network, where the input vectors were conceived before. From the last part, the resulting softmax vector and a classification were received. This whole procedure, from the 28×28 input image, through the first part of the CNN, downsampling to 8 dimensions by the VAE, upsampling and lastly classification by the last part of CNN, can be seen in figure 25.

Before proceeding to the main part of the algorithm, we wanted to get an intuition of the data by studying the structures of the latent space. For visualization purposes, we picked out the first three dimensions of each sample in the latent space and plotted the whole training set labeled by the target class as shown

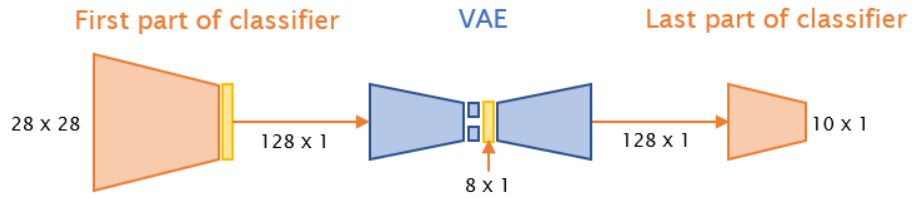
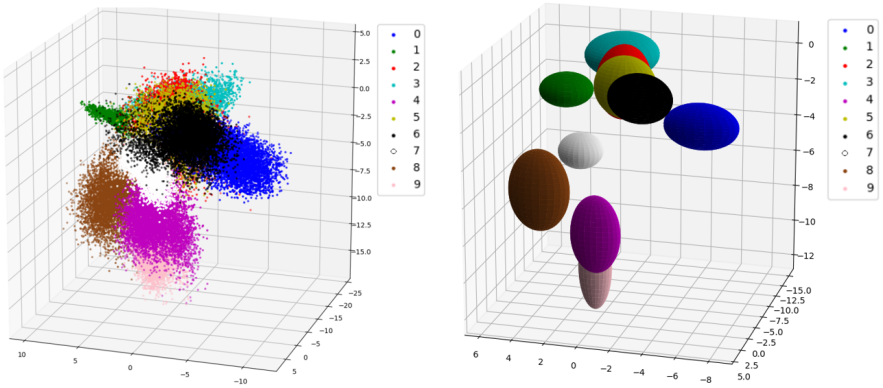


Figure 25: The pipeline for evaluating VAE.

in figure 26a. Even though far from linearly separable, some distinction of the classes can be seen. Indeed, if we approximate the data of each class by a normal distribution (in three dimensions) with the same mean value and variance as the set of samples, as plotted[4] in figure 26b, the separation of the classes is more visibly distinct.



(a) *(b)*

Figure 26: The first three dimensions of the latent space for each sample in the training set plotted figure (a). A color corresponds to a label. Figure (b) shows the data of each class approximated as a normal distribution. The separation of the classes is more visibly distinct in figure (b).

Slide generation

Let us focus on the data corresponding to class one and seven (corresponding, naturally, to the digits 1 and 7) plotted in figure 27. An overlap between these classes is clearly visible, which is reasonable as there presumably exist different types of handwriting that draw ones and sevens almost identically. It is obvious that a conventional classifier, however well-trained, will have a hard time distinguishing these overlapping data points from each other. Consider, as a basic example, a person A that writes the digits 1 and 7 according to figure 28a and another person B that writes them according to figure 28b. Notice the undeniable similarity between person A's ones and person B's sevens. These digits on their own are in practice indistinguishable, but provided with the information

[4]The normal distributions are plotted as spheroids centered at the mean value and with semi-axis corresponding to the standard deviation in each dimension.

about A's sevens and B's ones, one could possibly draw the accurate conclusion that A writes unsure ones (that an observer might mistake for a seven) and sure sevens (clearly distinguishable from any other digit) while B writes sure ones and unsure sevens. The same logic should be applied in our algorithms in order to enhance the accuracy on uncertain samples.

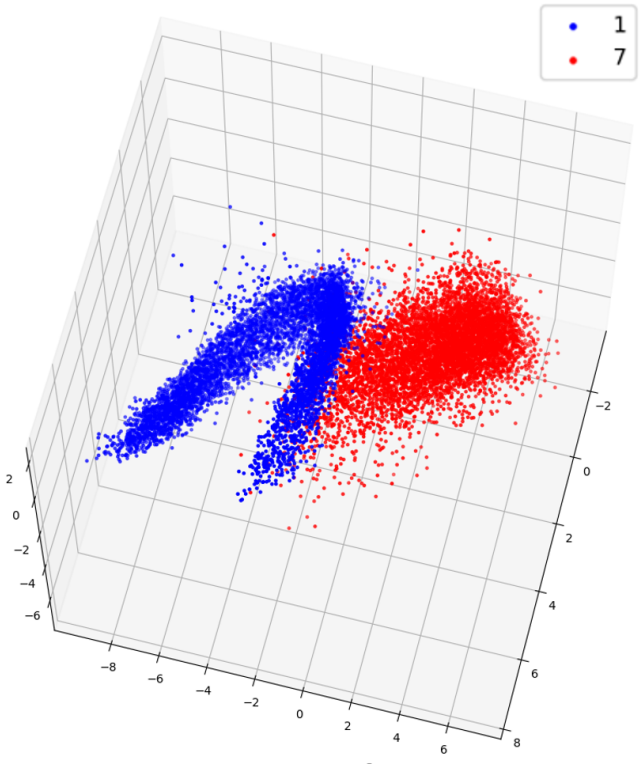


Figure 27: The first three dimensions of the latent space plotted for digit 1 and 7. An overlap between these classes is clearly visible.

(a) *(b)*

Figure 28: Toy example of two persons with different handwriting. Note the visual similarities between person A's ones and person B's sevens. The example points out the potential of using knowledge about all the images from the slide for classification.

Before proceeding, we must act upon the fact that the MNIST database doesn't contain any information about personal connections. Put differently, we don't know which digits were written by the same person. To solve this, we applied the brute force method of creating this information on our own. Thus, the

results of this part of the work must only be considered proof of concept, as they don't solve any real world problem. We customized these groups (herein called "slides", inspired by the biomedical term) by hand-picking samples from the training data that the CNN found particularly difficult to classify and pairing these with carefully selected sets of sure digits to make it as easy as possible for our algorithms to perform well. Some subsets of sure and unsure images are visualized in figure 29. We ended up with 22 manually organized slides of 1226 images in total. The composition of these slides are summed up in figure 30. To generalize, we also programmed a stochastic slide generator that sampled sets of sure and unsure digits of different classes and paired them together according to the above logic. Each slide was restricted to have unsure samples from at most one class. This way, we generated 1,500 slides with the number of samples in each slide ranging from 20 to 100. The random generation was done with replacement and resulted in a dataset of 89,079 images in total.

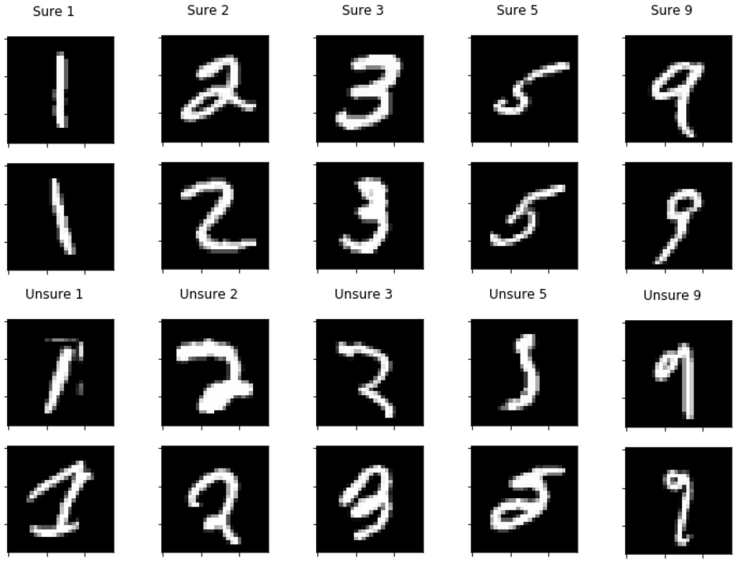


Figure 29: Examples of subsets of MNIST data used to organize customized slides.

Evaluation

Now that slides had been generated, we used these to evaluate our algorithm. Given a slide of n images, we run these through the first part of the CNN and then process the corresponding 128-vectors through the encoder of the VAE, resulting in n vectors in the eight-dimensional latent space. These vectors were then clustered according to the DBSCAN method described in section 2.9, and the clusters were classified to minimize the EMD[5] between the clusters and predefined reference distributions as described in section 2.10. The reference distribution for a specific class is made up by the latent representation of all training samples of that class. Outliers were handled separately and classified

[5]In contrast to the EMD example in section 2.10, the clusters and reference clusters are not allowed to be divided into differently distributed parts. In other words, a whole cluster must be assigned to one reference cluster.

Slide	Number of 0	# of unsure 1	# of 1 type 1	# of 1 type 2	Number of 2	Number of 3	Number of 4	Number of 5	Number of 6	Number of 7	Number of 8	Number of 9
slide_0	10		10							19		
slide_1	10			20	10	10		10		10	60	10
slide_2					15	15	21	15		10		11
slide_3	20						10		30	10		10
slide_4		10								40		
slide_5					10		10	10	10	10	10	10
slide_6	5			5	5	5	5	5	5	5	5	5
slide_7	5			20	10					6		
slide_8						6	26		6			10
slide_9						30	20					50
slide_10		11				5				10	10	10
slide_11							1			10	20	5
slide_12	1				1		1		1	1		1
slide_13				10			7			11		7
slide_14			10				10		20			
slide_15			20				10	20		10		15
slide_16			10					9	10			10
slide_17					9					9		30
slide_18					13	45						
slide_19			10		20	10				7		
slide_20				20				20			20	
slide_21	17	7								7		

Figure 30: Summation of the 22 customized slides of MNIST data. The numbers represent the amount of a certain type of digit that is present in a certain slide. Each digit can be either sure (green) or unsure (red). Note that the sure digit 1 has been split into two types depending on the handwriting, where type one is tilting and type two is vertical. For example, the first slide "slide_0" contains ten sure zeros, ten sure ones of the first type, and 19 unsure sevens. The idea is that the presence of sure ones should make it easier for our algorithm to identify the unsure sevens as sevens and not misclassify them as ones.

according to the closest distribution that hadn't already been assigned to a cluster. This was done for all slides, and we performed a grid search over ϵ and *min_samples* to find a well performing configuration over the whole set. We did this for the 22 manually generated slides and the 1,500 stochastically generated slides separately. The results are discussed thoroughly in later sections of the report.

3.2 White Blood Cells

Datasets

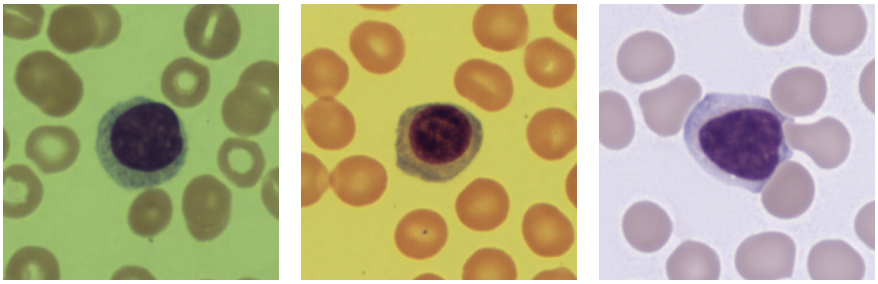
Cell images and relevant information were obtained from CellaVision's databases. For each cell image, the associated cell class and information about which slide it belonged to are received. Each cell is manually classified by up to 15 experts, which is used as "ground truth". If all experts vote on the same class, the cell is assigned that class. If the experts disagree, only the cells with a majority vote are used. This is to reduce the uncertainty of what is going to be used as ground truth.

From the beginning, a total of 423,274 images were received: 45,382 from DM1200, 28,783 from DM96 and 349,109 from DC-1. One dataset was created by only using the newer DC-1 machine, and one dataset with mixed data from both the new DC-1 and the older machines, DM1200 and DM96. The DC-1 dataset consisted of 349,109 cell images and the mixed dataset (named "ALL") of 423,274. After removing images where no slide information was available or where there was no majority vote, 307,139 images remained in the DC-1 dataset and 377,213 in the ALL dataset. As can be seen in table 1, most of the images belong to the DC-1 machine.

Table 1: Data distribution of the cell images for the new DC-1 machine and the older machines. Updated images are the images left after removal of images without sufficient information or no majority vote.

	# images	% images	# updated	% updated
DM1200 & DM96	74,165	17.52%	70,074	18.58%
DC-1	349,109	82.48%	307,139	81.42%
Total (ALL dataset)	423,274		377,213	

The cell images from the three different machines do not look exactly the same. The background of the cell images differs a lot due to different illumination sources and different cameras. A cell image from each machine is shown in figure 31.



(a) *(b)* *(c)*

Figure 31: One cell image from each of the three machines: DM96, DM1200 and DC-1.

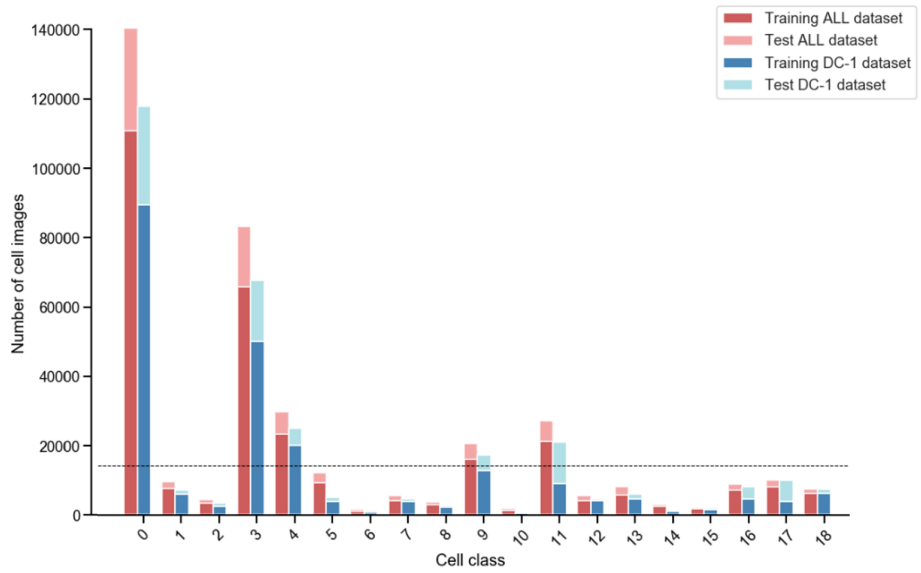
The two created datasets, DC-1 and ALL, were then each divided into a training and a test set. The division for the two datasets can be seen in table 2. The percentage split is not the same for the two datasets since the DC-1 dataset was already divided into training and test when receiving it from CellaVision. The division of the ALL dataset was made by us.

Table 2: Training and test distribution of the cell images for the two datasets.

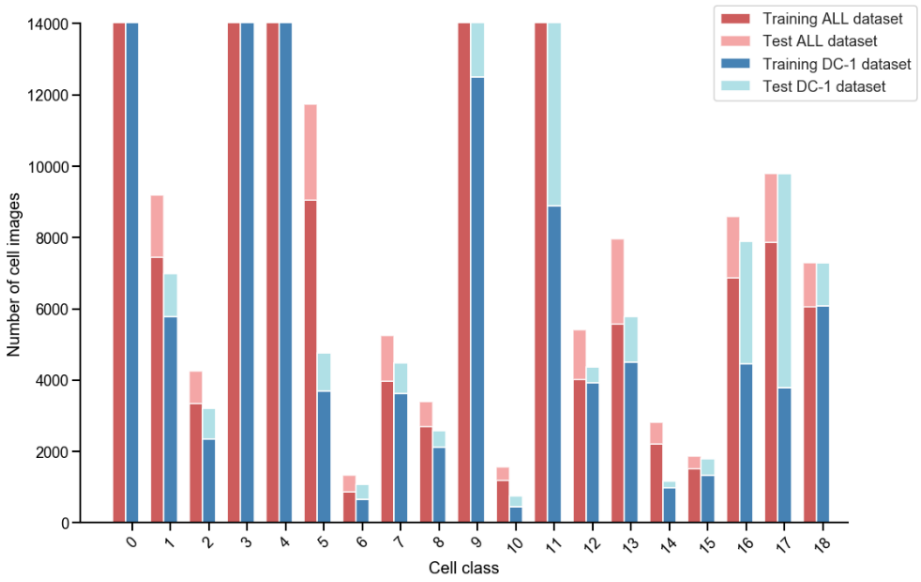
	# train imgs	% train imgs	# test imgs	% test imgs
DC-1 dataset	222,520	72.88%	82,809	27.12%
ALL dataset	296,835	78.69%	80,378	21.31%

The bar chart in figure 32a shows the distribution of the 19 cell classes divided into training and test sets for both datasets. It is clear that the distribution is uneven since the number of images for each class varies a lot. For example, class 0, SegmNeutrophil, has over 100,000 images, while class 10, Plasma Cell, has less than 2,000 images. Since the difference in images is large, it is hard to see the bars for the classes with fewer images in figure 32a. Therefore, a bar chart with cut y axis is shown in figure 32b. The y axis in figure 32b ranges to 14,000 instead of 140,000, as in figure 32a. The y axis limit of figure 32b is shown as a dashed line in figure 32a.

Each cell image has information of which slide it belongs to. A slide contains



(a)

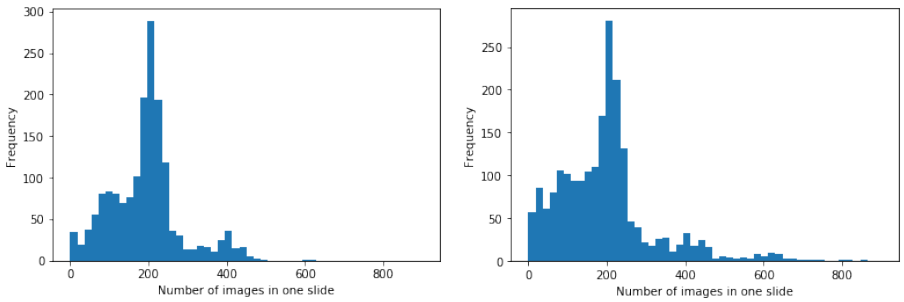


(b)

Figure 32: Class distribution for both datasets and their training and test sets. Figure a) shows the whole scale while figure b) is cut to be able to see the small bars. The dashed line in a) demonstrates where the cut in figure b) is made.

several cell images, all from the same blood sample, hence the same person. The number of cell images belonging to a slide varies a lot, see figure 33.

Before the images are used for training, they are resized from $360 \times 360 \times 3$ to $224 \times 224 \times 3$ to decrease memory usage. The images are then normalized. The normalization is done by dividing each channel in the image matrix by its



(a) DC-1 dataset *(b) ALL dataset*

Figure 33: Histogram showing the number of cell images in slides.

maximum value, so each pixel in the image matrix is between 0 and 1.

Classification

To prevent overfitting, the data is augmented before it is fed into the network. By augmenting the data, the classifier trains on a more diverse dataset. The augmentation used is rotation and zooming. Before sending the image through the classifier, it will be rotated a random number of degrees between -90 and 90. It will also zoom the image a factor between 0.9 and 1.1. This setting of augmentation is used because of beneficial outcomes of previous investigations of augmentation settings on CellaVision data done by other master thesis workers [29].

The images were then fed into a modified version of the Xception network, described in section 2.7. The architecture of the network is visualized in figure 34. Instead of repeating the middle layer eight times, as in the original, it was repeated three times. Dropout layers (with hyperparameter 0.2) were added to the original architecture to enhance the generalization of the model, as can be seen in figure 34. The training dataset was divided into a training set, 75% of the original training dataset, and a validation set of 25%. The optimizer used was Adam with the default parameters: learning rate=0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The model was trained for 50 epochs with a batch size of 32 samples. The training of the modified Xception was accomplished in about three days when performed on our GPU.

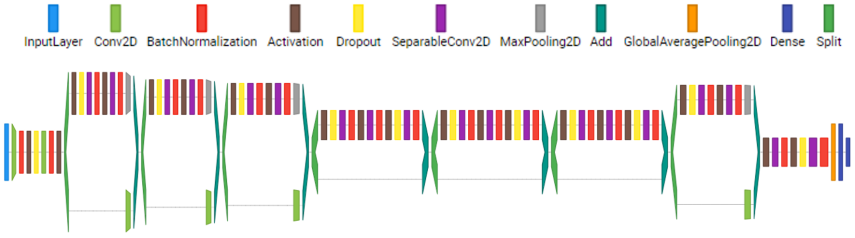


Figure 34: The modified Xception network used to classify the WBCs. The illustration is made with the tool Net2Vis [28].

The same training was done using both the DC-1 and the ALL dataset. The weights were saved for the epoch with lowest validation loss.

Dimensionality Reduction

The output of the Xception network is a softmax vector of dimension 19. To be able to perform another classification than the CNN, we broke the network before the output. Instead of taking the 19 dimensional softmax vector, a feature vector of 512 dimensions was obtained by breaking the network after the intermediate dense layer. If the network would have been broken earlier in the architecture, the resulting vector would be too large. To break one layer earlier would have resulted in a vector of size $12,544 \times 1$. The memory of our computers could not handle $377,213$ (the size of the larger ALL dataset) $12,544 \times 1$ vectors. To move the breakpoint closer to the output would produce vectors that rely to much on the classification of the network. This is discussed in greater detail later in the report.

The Xception network reduces the size of each sample from $224 \times 224 \times 3$ to 512×1. To reduce the dimensionality further, a Variational Autorencoder was used. Due to time constraints, only two different VAEs were constructed and tested: one with dimension 3×1 in the bottleneck layer, and one with dimension 64×1. Both VAEs are constructed in the same way: a dense layer followed by a batch normalization layer, followed by a layer with ReLU activation function, followed by a dropout layer with parameter 0.2. This four layer setup is then repeated three (or four) times when the bottleneck layer is of dimension 64 (or 3, respectively). To upsample, the block of layers are then repeated in reverse order. The architecture of the 64-dimensional VAE is visualized in figure 35.

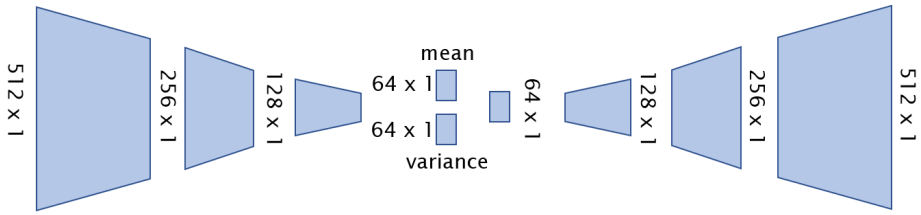


Figure 35: The VAE used to reduce the dimension of the 512-vectors further. The VAE visualized here has 64 latent dimensions.

The VAEs were first trained using the feature vectors derived from the Xception network, for both the DC-1 and the ALL dataset. The full training sets defined in table 2 were used to train the VAEs. The goal for the VAE is to get the output as similar as possible to the input. We want to keep as much information as possible when downsampling the input to the bottleneck dimension. To evaluate the VAEs, the VAE output was fed into the last part of the Xception network, where the input vectors were conceived before. From the last part, the resulting softmax vector and a classification were received. This whole procedure, from the $224 \times 224 \times 3$ input image, though the first part of the CNN, downsampling to 3 or 64 dimensions by the VAE, upsampling and lastly classification by the last part of CNN, can be seen in figure 36.

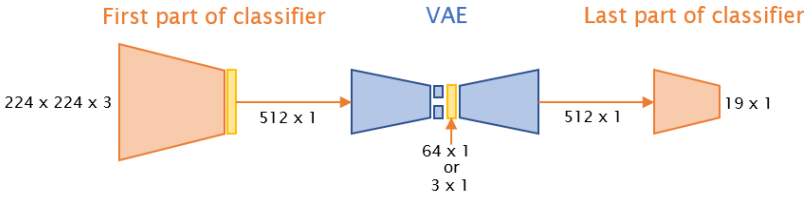
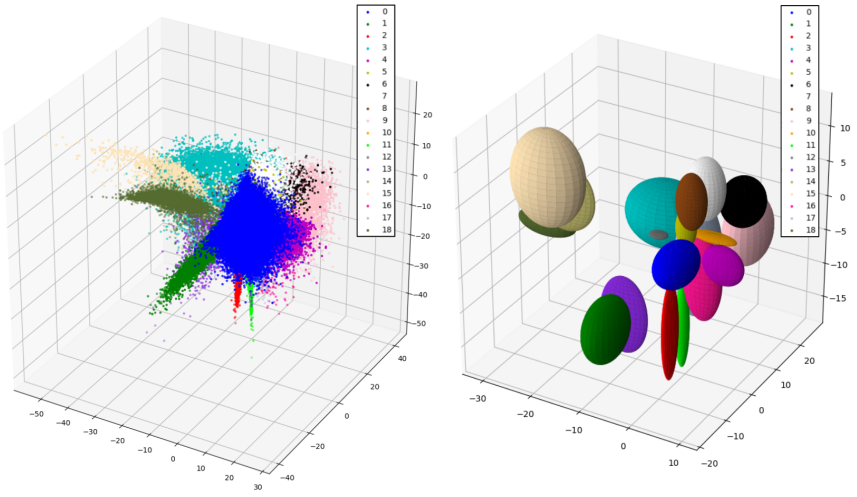


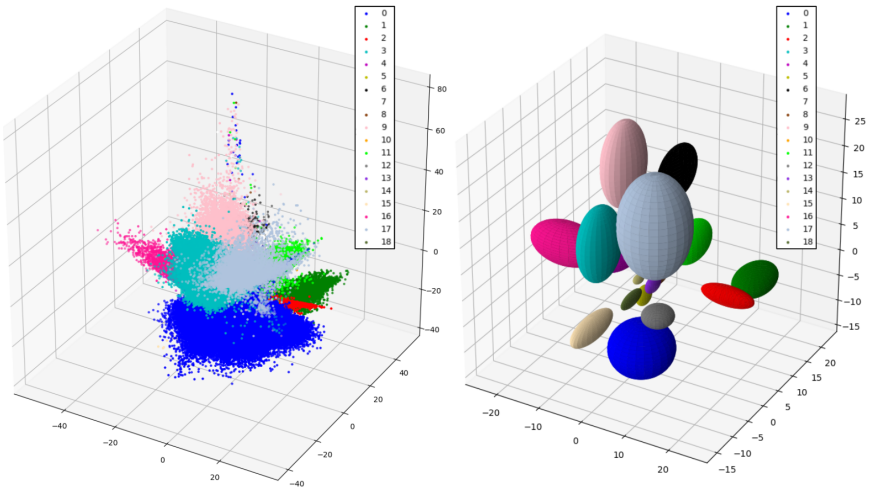
Figure 36: The pipeline for evaluating the VAE.

By comparing the accuracy from the classification done after this whole pipeline and the accuracy from a regular classification using just the CNN, we got a measurement of how good the VAE had restored the information when down-sampling. The optimizer used was Adam with the default parameters: learning rate=0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The model was trained for 50 epochs with a batch size of 32 samples and MSE as loss. The training of the VAE was accomplished in about one hour when performed on our GPU. For visualization purposes, the training images in the three dimension latent space are plotted in figure 37a for the DC-1 data and in figure 38a for the ALL data. To make the classwise relations clearer, approximate normal distributions have been plotted in figure 37b for the DC-1 data and in figure 38b for the ALL data. Hereafter, only the 64-dimensional VAE was used as it showed to yield better results than the 3-dimensional VAE.



(a) *(b)*

Figure 37: The latent representation (a) and approximate normal distributions (b) for the DC-1 training data in the latent space of the three-dimensional VAE, labeled by class. The normal distributions are plotted as spheroids centered at the mean value and with semi-axis corresponding to the standard deviation in each dimension.



(a) *(b)*

Figure 38: The latent representation (a) and approximate normal distributions (b) for the ALL training data in the latent space of the three-dimensional VAE, labeled by class. The normal distributions are plotted as spheroids centered at the mean value and with semi-axis corresponding to the standard deviation in each dimension.

Clustering

After training the VAEs, the goal is to cluster the vectors from the bottleneck layer for each slide. One slide contains several cell images. The used clustering method, DBSCAN, is described in section 2.9. Two different methods to find the two DBSCAN parameters ϵ and *min_samples* were tested. First, an ordinary grid search was applied for the grid in figure 39. For every slide, DBSCAN clustering was performed several times using a grid of parameter configurations. The configuration yielding the lowest total EMD over all clusters was used. Secondly, the parameters were derived using Bayesian optimization, described in section 2.11. Bayesian optimization was implemented using code from [25].

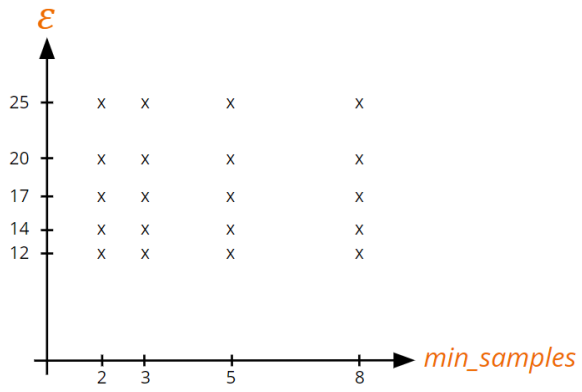


Figure 39: The configurations of the DBSCAN parameters ϵ and min_samples tested in the grid search.

Class Assignment

A class label should be assigned to each cluster in a slide. To pair a cluster with a class label, the reference clusters plotted in figures 37a and 38a were used. To get these reference clusters, all data was fed into the first part of the network, then into the encoder of the VAE to get the latent vectors. All vectors with label 0 were then used as reference cluster for class 0. The same was done for the rest of the classes, so reference clusters for every class were obtained. Applying the linear sum assignment problem [30] on the EMD described in section 2.10, we got a label for each cluster. In the assignment problem, the rows are the clusters from a slide, that should be matched with a reference cluster, stated in the columns. The label of the reference cluster is then the label of the cells in the matched cluster. Outliers were classified according to the CNN. We also introduced a cutoff parameter to decide the largest acceptable *disagreement* between our algorithm and the CNN, defined as the number of differently classified images in a slide. This means that for a single slide, if the predicted labels by our algorithm and the CNN differ on more images than the cutoff value, we choose the CNN's prediction and discard our algorithm for that slide. This is motivated by the empirical observation that our algorithm is more likely to fail completely, in contrast to the CNN that is relatively stable.

4 Results

4.1 MNIST Digits

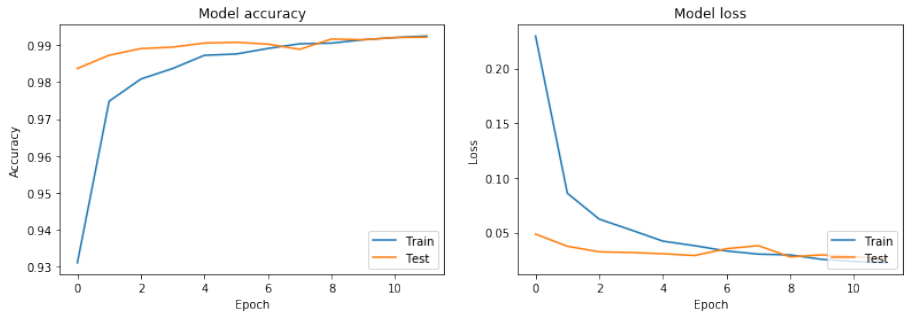
Classification

The training and test accuracy is shown in table 3.

Table 3: Training and test accuracy for the MNIST classification CNN.

	training	test
Accuracy:	99.31%	99.23%

Plots of accuracy and loss over the number of training epochs, for both training and test, are shown in figure 40.



(a) *(b)*

Figure 40: Accuracy (a) and loss (b) for both training and test over the number of training epochs for a CNN trained on MNIST data.

Dimensionality Reduction

The resulting accuracy for the VAE with a latent space dimension of 8, and the regular CNN, can be seen in table 4. Classifying using the VAE means that the MNIST images are fed through the first part of the CNN, down- and upsampled by the VAE, and then classified by the last part of the CNN.

Table 4: Training and test accuracy for the VAE and the CNN.

	training	test
VAE:	99.79%	99.18%
CNN:	99.31%	99.23%

Training and test loss as a function of epochs for the VAE can be seen in figure 41.

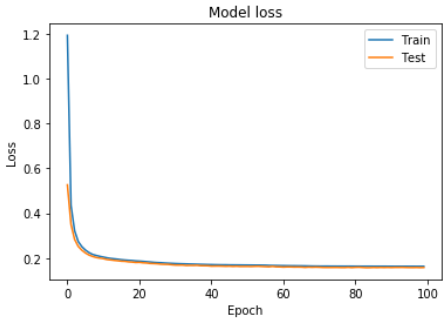


Figure 41: Training and test loss as a function of epochs for the VAE.

Cluster Classification

The errors (i.e., falsely classified digits) for different DBSCAN parameters ϵ and $min\_samples$ are shown in table 5. The best configuration proved to be ϵ= 5 and $min\_samples$ = 2 or 3. The result of letting our clustering method, with the parameters $\epsilon = 5$ ans $min\_samples = 3$, classify the 22 manually generated slides can be seen in table 6. The result of the stochastically generated slides is shown in table 7. The results for the CNN and VAE are shown as comparison.

Table 5: The number of errors when classifying the 22 manually generated slides with our method for different values of the DBSCAN parameters ϵ and $min\_samples$. The best configuration is ϵ= 5 and $min\_samples$ = 2 or 3. 11 errors out of 1226 samples is equivalent to an accuracy of 99.10%

$\epsilon \backslash min\_samples$	1	2	3	4	5	6	7
3	176	179	79	76	75	77	138
4	130	65	33	28	27	22	38
5	38	11	11	13	13	25	36
6	82	81	81	81	81	81	71
7	140	139	139	139	139	139	139

Table 6: The number of errors and accuracy of classifying the 22 manually generated slides using our method with optimal DBSCAN settings, the CNN and the VAE. Classifying using VAE means that the MNIST images are fed through the first part of the CNN, down- and upsampled by the VAE, into the last part of the CNN and then classified.

	Number of errors:	Accuracy:
Our method:	11	99.10%
CNN:	49	96.00%
VAE:	54	95.60%

Table 7: The number of errors and accuracy of classifying the 1,500 stochastically generated slides using our method with optimal DBSCAN settings, the CNN and the VAE. Classifying using VAE means that the MNIST images are fed through the first part of the CNN, down- and upsampled by the VAE, and then classified by the last part of the CNN. This set contains 89,079 samples.

	Number of errors:	Accuracy:
Our method:	2,464	97.23%
CNN:	2,576	97.11%
VAE:	2,243	97.48%

4.2 White Blood Cells

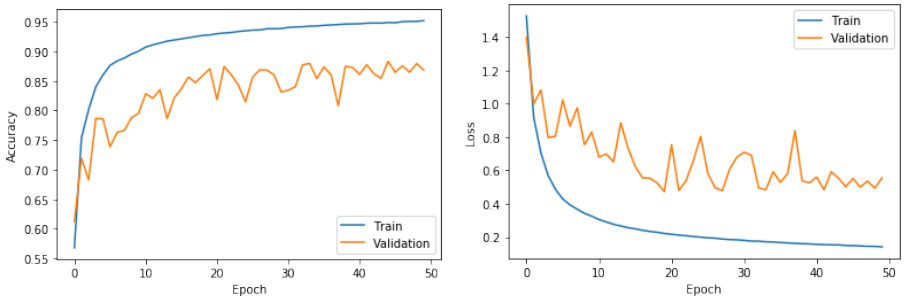
Classification

The training, validation and test accuracies for the modified Xception are shown in table 8.

Table 8: Training, validation and test accuracy for the two datasets.

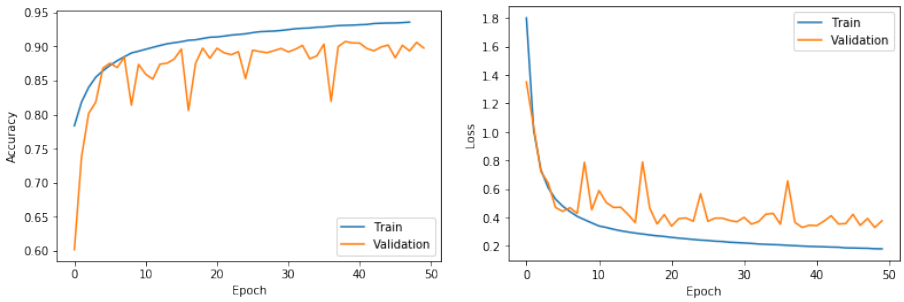
Accuracy for:	training	validation	test
DC-1 dataset	92.78%	87.03%	82.85%
ALL dataset	93.50%	90.60%	89.82%

The training and validation loss and accuracy for DC-1 dataset can be seen in figure 42. The training is done for 50 epochs but the model is saved at epoch 20 since the validation loss does not improve after that. The training and validation loss and accuracy for the ALL dataset can be seen in figure 43. The training is done for 50 epochs but the model is saved at epoch 49.



(a) *(b)*

Figure 42: Training and validation accuracy (a) and loss (b) as functions of epochs for the DC-1 dataset.



(a) *(b)*

Figure 43: Training and validation accuracy as a function of epochs for the ALL dataset in figure (a). Training and test loss in figure (b).

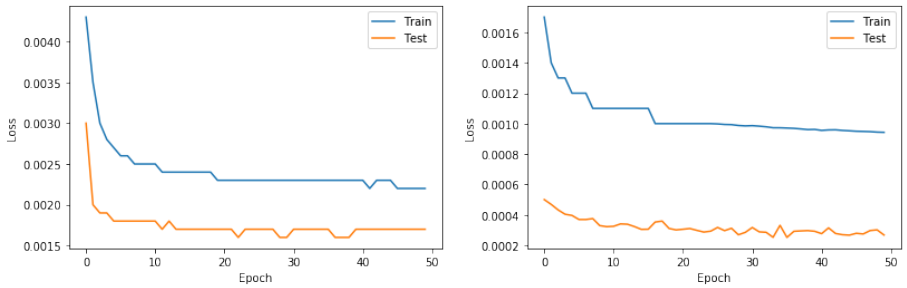
Dimensionality Reduction

The resulting accuracies for the VAEs with a latent space dimension of 3 respective 64, and the regular modified Xception network, can be seen in table 9. The accuracies of the VAEs are calculated by running the output vector through the final part of the classifier, and comparing the result to the known targets. The training accuracies for the CNN are not the same as the results in table 8. In table 8, the accuracies for training and validation are divided into two separate results. In table 9, these results are merged into one accuracy for the whole set of data used for training and validation.

Table 9: Accuracy for the two different VAEs and classification using the regular CNN for both DC-1 dataset and ALL dataset.

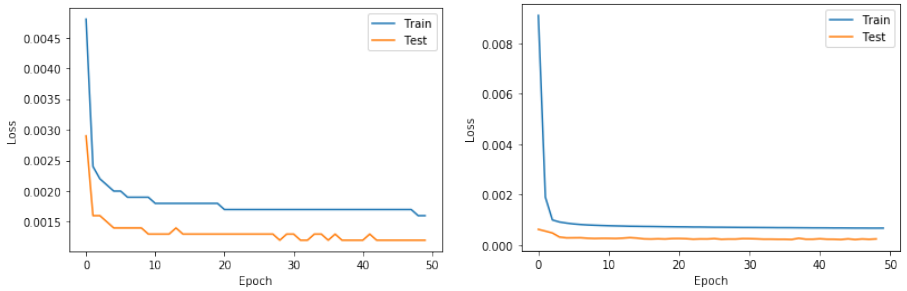
Accuracy for:	DC-1 dataset		ALL dataset	
	training	test	training	test
VAE 3	90.07%	82.05%	92.31%	89.35%
VAE 64	91.01%	83.04%	93.10%	89.69%
CNN	90.68%	82.85%	93.29%	89.82%

The loss of the two VAEs using 3 and 64 dimensions as functions of epochs can be seen in figure 44 for DC-1 dataset and in figure 45 for the ALL dataset.



(a) *(b)*

Figure 44: Training and test loss as a function of epochs for the DC-1 dataset. Figure (a) shows the loss for the VAE with three dimensional latent space, and figure (b) the VAE with 64 dimensions.



(a) *(b)*

Figure 45: Training and test loss as a function of epochs for the ALL dataset. Figure (a) shows the loss for the VAE with three dimensional latent space, and figure (b) the VAE with 64 dimensions.

Cluster Classification

All results from here on correspond to the VAE with 64 latent dimensions. Figure 46 shows how the cutoff parameter for disagreement between our methods and the CNN affects the number of errors for our Grid and Bayesian methods.[6] Both DC-1 plots have a minimum at cutoff 25. Table 10 shows the percentage of all slides that were not discarded when using a cutoff value of 25.

Table 10: The percentage of all slides that were not discarded when using a cutoff value of 25 on the different configurations of data and methods.

	DC-1:	ALL:
Grid:	44%	44%
Bayesian:	44%	27%

[6]Recall that a cutoff parameter of 25 means that for a single slide, if the predicted labels by our algorithm and the CNN differ on more than 25 images, we choose the CNN's prediction and discard our algorithm for that slide.

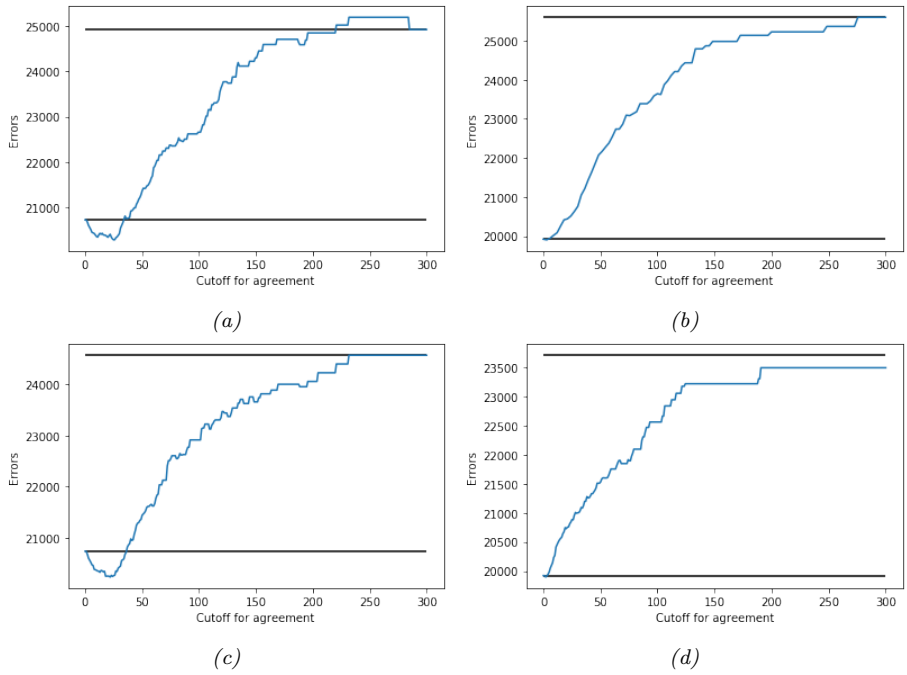


Figure 46: How the cutoff parameter for disagreement between our methods and the CNN affects the number of errors for our Grid method on DC-1 data (a) and ALL data (b) and our Bayesian method on DC-1 data (c) and ALL data (d). In all cases, the upper line represents our method without cutoff and the lower line represents the CNN performance. A low number of errors is of course sought-after. Both DC-1 plots have a minimum at cutoff 25.

Table 11 shows a summary of the final results for our methods and the classifier. To get these values, an agreement cutoff of 25 was applied in the Grid and Bayesian methods.

Table 11: Training and test accuracy for different methods. Grid, Bayesian and Optimal are using our method with different choices of the DBSCAN parameters. Grid tests a fixed set of ϵ and min_samlpe and chooses the configuration that yields the lowest loss. Bayesian chooses the configuration by utilizing Bayesian Optimization. Optimal tests the same grid of parameters as Grid, but chooses the one with the lowest number of errors. The CNN columns just show the accuracy of the modified Xception network.

		DC-1:	ALL:
Grid:	Train:	90.88%	92.68%
	Test:	83.16%	89.25%
Bayesian:	Train:	90.90%	92.96%
	Test:	83.13%	89.50%
CNN:	Train:	90.68%	93.29%
	Test:	82.85%	89.82%
Optimal:	Train:	92.91%	94.25%
	Test:	86.57%	92.06%

5 Discussion

5.1 MNIST Digits

Classification

The CNN used to classify the MNIST digits achieved satisfactory results. Minimal overfittning occurs since the lines for training and test in figure 40 seem to converge and the accuracies in table 3 are very close. Both accuracy and loss seem to have stagnated after 12 epochs. While the achieved test accuracy of 99.23% is quite far from the world record of 99.84%, it is still considered high enough for our problem.

Dimensionality Reduction

We broke the CNN network and extracted the feature vectors at a layer in the end of the network where the dimension was 128×1. Trying to break earlier would result in vectors of size $9,216 \times 1$. Handling $60,000$ such vectors would be more demanding. Breaking the network later would give us the softmax vector of dimension 10×1. Using these vectors, we would not have been able to perform a classification differing from the CNN results.

Different structures of the VAE were tested until this current model was found. Since this model produced good result, even better than the CNN, the investigation stopped. The goal of the VAE is to down- and upsample the input vector, without losing information. The fact that our VAE has better training accuracy than the CNN is an unexpected result. It may be that the VAE filters out noise in the data, but it could also just be a coincidence. Also, the test accuracy for VAE is a bit worse than CNN's accuracy. Even if the accuracies differ a bit, they still are very similar, so the goal of the VAE is fulfilled. By looking at figure 41, it is clear that there is no overfitting since curves for the training and test losses coincide. The curves are flattened, so training for 100 epochs seems sufficient. The choice to set the dimension of the latent space to eight proved to provide a good compromise between computational effort and the algorithm being able to derive a sufficient number of output clusters.

Cluster Classification

By studying table 5, one can see that our method is sensitive to the choice of DBSCAN parameters as the number of errors varies a lot when changing the configurations. There is a minimum around $\epsilon = 5$ and $min\_sample = 2$ or 3. Table 5 was generated using the 22 manually generated slides. However, the same set of parameters seem to be the optimal choice for the 1,500 stochastically generated slides as well. The results in table 7 are achieved with the optimal configuration chosen from the manually generated slides, but different configurations were also tested. The configuration from the manually generated slides was best in this case too.

The results from table 6 state that our method performs best of the three methods. It shows that the concept of our method works and can exploit that it is easier to classify unsure digits with more information about how the same person writes other digits. The 22 manually generated slides are customized

to fit our method. For example, if unsure sevens are present in a slide, we have made sure that sure ones also are in the slide so our method can use this information to classify the unsure sevens. Also, to get unsure digits, we have used the digits the CNN has difficulties to classify, which decreases the accuracy of the CNN on this dataset compared to the original CNN accuracy for all digits. Furthermore, the parameters used in our method were chosen according to this seen data, so it is expected that our method performs well. The CNN has a slightly higher accuracy than the VAE has, but the difference is not large. It is expected since the VAE and the CNN have very similar training and test accuracies on all digit data.

The 1,500 stochastically generated slides are not as customized as the manually generated ones. There are also a lot more images classified, so the results are somewhat more reliable. For these slides, the VAE is the best, followed by our method and at last the CNN. However, the differences are very small. Again, it is expected that the VAE and the CNN perform similar. The reason why our method performs worse at these 1,500 slides than the 22 slides is probably because the 1,500 slides are not as customized. For example, if there are unsure ones in a stochastically generated slide, there may not be sure sevens to distinguish the unsure ones from sevens.

5.2 White Blood Cells

Classification

As can be seen in table 8, the classification performance on WBC data is significantly reduced compared to MNIST data. This is expected, as the whole point of working with MNIST data was to start with a simple dataset. The WBC models were trained for 50 epochs, which appears to be enough as the validation performances plotted in figures 42 and 43 seem to have leveled off by that point. Taking a close look at the results on the DC-1 dataset, we notice that some overfitting seems to be present as the accuracy is much lower on the validation and test sets than on the training set. This effect is also visible in figure 42, where the validation accuracy and loss quickly separates from the training equivalent. This issue should not be disregarded as generalization is something to always strive for when training models. Adding dropout layers to the ordinary Xception network was meant to enhance generalization, but the effect was only marginal. Ideally, we would have wanted a bit less distinct discrepancy between the training, validation and test data performance. On the other hand, the model that is trained on the ALL dataset exhibits much improved generalization. This is clearly visible in both table 8 and figure 43. The performance is still better on the training set than on the other data, but the difference is smaller compared to the DC-1 dataset and is considered acceptable. The enhanced generalization performance on ALL data in relation to DC-1 data is expected, as this dataset is both larger and, evidently, more diverse. Adding images from other systems should make the model more robust to variations in for example color, due to the natural color differences visible in figure 31. It is still an interesting finding that the accuracy is higher on ALL data than on DC-1 data. This wasn't known beforehand, and is arguably a noteworthy result for CellaVision. Overall, we conclude for the classification CNNs that the performances are high enough to ensure some trustworthiness in the forthcoming results, and low enough to leave

a substantial room for improvement.

Dimensionality Reduction

As input (and target) data to the VAEs, we used the so-called 512-vectors of size 512×1 that were extracted from an intermediate layer of the classifier (recall the pipeline illustrated in figure 36). While the decision to split the classifier at this specific point was thought over, this is probably something to investigate further in order to optimize our algorithms. Breaking the CNN at an earlier point in the architecture would infer some difficulties in the handling of the data, but should still be considered a possible extension of our work. Breaking at a latter point would likely lead to data that is too much weighted towards the expert classification,[7] but could also be worth exploring if time allows.

We have consistently been focusing on VAEs in this work, rather than ordinary autoencoders. It was rather unclear at the start of the work exactly which techniques were going to be used, and if we would have wanted to generate new data at any point, the generative property of the VAE would come in handy. As it turned out, we used the VAEs for dimensionality reduction only, so in practice, the VAEs were reduced to ordinary autoencoders. The decision to use two kinds of VAEs - one with three dimensions and one with 64 dimensions - can be debated. We wanted to have one substantial VAE (producing a latent vector of few dimensions) and another less industrious VAE to compare with. Another reason to specifically choose 3D for the former was to be able to plot the encoded data points for visualization purposes (see figures 37 and 38). Very little experimentation were made to explore the dimensionality reduction's effect on the final result of our work. What we can conclude, however, is that the reduction is made without major loss of information for both VAEs. Figure 45 shows that the loss curves for all configurations appear to level off before the training of the VAEs end, indicating that they were trained for enough epochs. The loss is constantly higher on the training data than on the test data for all VAEs, due to the dropout layers that are activated only during the training. Table 9 clearly shows that the drop in performance is very small, especially for the 64-dimensional VAE and when applying it on the DC-1 dataset. In fact, that particular configuration yielded higher accuracy on both the training and the test set in relation to the CNN. Since this effect was present also for the MNIST data, we have reasons to doubt that it would be a mere coincidence. As discussed previously, one explanation could be that the VAE creates a subspace that has less irrelevent information and in fact does improve the performance. This is something that potentially could be very useful and well worth exploring further. However, as it falls outside the scope of this work it is not investigated here. We conclude that while both VAEs seem to work okay, the 64-dimensional VAE is a reasonable choice to use in the continuation of the work.

[7] As our method aims to exploit visual similarities in the data, and many of these similarities are supposed to be identified in early layers in the network, it was deemed logical to extract our vectors as early in the classifier as manageable (which is still very close to the end of the network).

Cluster Classification

Let us now dive into the main results of this work. Table 11 shows how our methods (named Grid and Bayesian) perform on different data compared to the classifying CNN. The table also presents results using the optimal parameters with hindsight (named Optimal), which can be considered an indicator for the potential of our algorithms with respect to the search for the DBSCAN parameters. Let us first focus on Grid and Bayesian. Recall that the former uses a fixed grid of 20 specific parameter configurations to try, while the latter draws ten random configurations (within a predefined region) and performs a Bayesian optimization search to test another ten configurations. While the results are quite similar for the two methods, Bayesian performs slightly better than Grid on both the DC-1 data and the ALL data. This motivates the usage of Bayesian optimization. The most important result is arguably how this method performs in relation to the CNN. We are happy to conclude that our Bayesian method performs slightly better than the CNN on DC-1 data. The difference is marginal, but is considered trustworthy as it holds for both training and test data. On the ALL data, our Bayesian method is close to the performance of the CNN but doesn't manage to exceed it. One possible explanation could be that our work is meant to pick up visual irregularities in the data, such as unintended color shifts on a whole slide of samples, and a CNN trained on the ALL data might already have learnt to deal with such irregularities. After all, the CNN trained on ALL data has a significantly higher accuracy than that trained on DC-1 data. It is also worth noticing that the discrepancy in performance between training and test data for our Bayesian method on both datasets is of the same magnitude as for the CNN, meaning that our method doesn't contribute to any drop in generalization other than that already present for the CNN.

Both our Grid and Bayesian methods utilize a cutoff at 25 for the largest acceptable disagreement with the CNN, meaning that any slide where our method and the CNN differs by more than that are deemed too risky to use our method on. On those slides, our method fully adapts the CNN prediction. The reason to introduce this cutoff is the empirical observation that our methods are less robust than the CNN. This is confirmed by figure 46, showing that cutoff 25 is a wise choice for DC-1 data. The ALL data appears more cumbersome to handle. There is no cutoff value that produces a satisfying result - the best we can do is to simply reproduce the CNN predictions totally. As this would imply a rather boring algorithm, we chose to use cutoff 25 for the ALL data as well. We conclude that our algorithm, in its current shape, works better on the DC-1 data than on the ALL data. For the data and model configurations explored, the cutoff at 25 corresponds to discarding more than half of the slides, as can be seen in table 10. This limits our method from diverging too far from the CNN. Being able to increase the cutoff value (i.e., decrease its effect) without aggravating the performance would enlarge the potential outcome of our method.

Going back to table 11, we will now discuss the Optimal method. This method uses the optimal parameters with hindsight, and should not be considered to be anything other than an indicator on the potential of our methods. A significant improvement in relation to the CNN accuracy can be observed, meaning that it might be worth to continue developing our algorithms. Making our Grid and

Bayesian methods more like the Optimal method all boils down to finding a better way to find suitable parameters for the DBSCAN clustering. A more sophisticated search could, evidently, yield results that beat the ordinary classifier much more significantly. This thought leaves us in an optimistic state of mind and smoothly conveys us to the next section.

5.3 Future Work

Training CNNs and VAEs is very time consuming, so just a few possible settings and structures have been tried out. Further improvements could include exploring other types of CNNs and VAEs. The chosen modified Xception network gave good results but further investigation could improve the test accuracy further. Also, the structure of the VAE and its latent space dimensions were rather arbitrary. Testing other dimensions of the latent space could result in better clustering and therefore better results of our method. It may also be possible to skip the CNN and just use a VAE to compress the data, or to break the CNN at an earlier stage in the architecture. Of course, testing different hyper parameters, such as batch size, number of epochs, learning rate and so on, could also improve the results.

In this work, DBSCAN was used as clustering technique. Further work could include to test other clustering techniques. The set of parameters that need to be specified in DBSCAN clustering, ϵ and $min\_samples$, and how to choose them should be explored more. We tried a grid technique and Bayesian Optimization, but there might be other techniques to try out. The loss, used to choose which of the tested configurations to proceed with, could also be explored more. If it is possible to find a perfect loss, i.e., the loss that always finds the configuration yielding the lowest number of errors, the accuracy would improve a lot.

6 Conclusion

This work demonstrates a way to classify multiple images from the same subset simultaneously. Using the method introduced in this report on cell image data from the DC-1 system, we were able to beat a regular CNN by a marginal but noteworthy amount on both training and test data. We also explored the possibility of using our method on mixed data from different systems, but the results were not as good. Applying our method on the MNIST dataset could be made advantageous by forming customized subsets of images. This indicates that our method is versatile enough to use on general image data, provided that correlations within the subsets exist.

References

[1] Nationalencyklopedin. *Blod*. http://www.ne.se.ludwig.lub.lu.se/uppslagsverk/encyklopedi/lång/blod (Accessed 2020-04-01)

[2] Pictures provides by CellaVision.

[3] LeCun, Yann and Cortes, Corinna and Burges, Christopher J.C. *MNIST handwritten digit database* http://yann.lecun.com/exdb/mnist/ (Accessed 2020-02-10)

[4] GitHub. Matuzas77. *MNIST-0.17*. 2020-01-28. https://github.com/Matuzas77/MNIST-0.17 (Accessed 2020-02-10)

[5] Medium. Furkan Özbek, Abdullah. *How to Train a Model with MNIST dataset*. 2019-01-28. https://medium.com/@afozbek_/how-to-train-a-model-with-mnist-dataset-d79f8123ba84 (Accessed 2020-04-24)

[6] Allabolag. *CellaVision AB*. https://www.allabolag.se/5565000998/cellavision-ab (Accessed 2020-04-09)

[7] CellaVision. *About CellaVision*. https://www.cellavision.com/en/about-us (Accessed 2020-04-09)

[8] CellaVision. *Introducing CellaVision DM1200*. https://www.cellavision.com/en/our-products/products/cellavision-dm1200 (Accessed 2020-04-09)

[9] Dreyfus, Gérard. *Neural Networks - Methodology and Applications*. Berlin Heidelberg: Springer-Verlag, 2005. E-book.

[10] Ramachandran, Prajit and Zoph, Barret and Le, Quoc V. *Searching for Activation Functions*. arXiv:1710.05941v2 [cs.NE] 2017-10-27 https://arxiv.org/pdf/1710.05941.pdf

[11] Bishop, Christopher M. *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press, 1995.

[12] Patrikar, Sushant. Batch, Mini Batch Stochastic Gradient Descent. *Towards Data Science*. 2019-10-01. https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a (Accessed 2020-04-13)

[13] Kingma, Diederik P. and Ba, Jimmy. *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980 [cs.LG] 2017-01-30 https://arxiv.org/pdf/1412.6980.pdf

[14] Ioffe, Sergey and Szegedy, Christian. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv:1502.03167 [cs.LG] 2015-03-02 https://arxiv.org/pdf/1502.03167.pdf

[15] Goodfellow, Ian and Bengio, Yoshua and Courville, Aaron. *Deep Learning*. Cambridge, MA: MIT Press, 2017. E-book.

[16] Budhiraja, Amar. Dropout in (Deep) Machine learning. *Medium*. 2016-12-15. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5 (Accessed 2020-04-24)

[17] Saha, Sumit. A comprehensive guide to convolutional neural networks - the eli5 way. *Towards Data Science*. 2018-12-15. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (Accessed 2020-04-13)

[18] Bendersky, Eli. *Depthwise separable convolutions for machine learning*. 2018-04-04. https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning.html (Accessed 2020-04-27)

[19] Chollet, François. *Xception: Deep Learning with Depthwise Separable Convolutions*. arXiv:1610.02357v3 [cs.CV] 2017-04-04 https://arxiv.org/pdf/1610.02357.pdf

[20] Rocca, Joseph. Understanding Variational Autoencoders (VAEs). *Towards Data Science*. 2019-09-24. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 (Accessed 2020-04-13)

[21] Bhattacharyya, Saptashwa. DBSCAN Algorithm: Complete Guide and Application with Python Scikit-Learn. *Towards Data Science*. 2019-06-09 https://towardsdatascience.com/dbscan-algorithm-complete-guide-and-application-with-python-scikit-learn-d690cbae4c5d (Accessed 2020-04-15)

[22] Treleaven, Kyle and Frazzoli, Emilio. *An Explicit Formulation of the Earth Mover's Distance with Continuous Road Map Distances*. arXiv:1309.7098v2 [stat.CO] 2013-10-14 https://arxiv.org/pdf/1309.7098.pdf

[23] Berasategi, Ane. Earth mover's distance - A semantic measure for document similarity in semantic search. *Towards Data Science*. 2019-04-08. https://towardsdatascience.com/earth-movers-distance-68fff0363ef2 (Accessed 2020-04-15)

[24] Rubner, Yossi and Tomasi, Carlo and Guibas, Leonidas J. *The Earth Mover's Distance as a Metric for Image Retrieval*. International Journal of Computer Vision 40(2), 99–121, 2000. http://robotics.stanford.edu/∼rubner/papers/rubnerIjcv00.pdf

[25] Nogueira, Fernando. *Bayesian Optimization: Open source constrained global optimization tool for Python*. 2014. https://github.com/fmfn/BayesianOptimization (Accessed 2020-04-27)

[26] Brochu, Eric and Cora, Vlad M. and de Freitas, Nando. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*. 2010. arXiv:1012.2599v1 [cs.LG] 12 Dec 2010-12-12 https://arxiv.org/pdf/1012.2599v1.pdf

[27] Keras. *Trains a simple convnet on the MNIST dataset*. https://keras.io/examples/mnist_cnn/ (Accessed 2020-02-11)

[28] Net2Vis. https://viscom.net2vis.uni-ulm.de/hNMyFzuLcbhY1wvp1Zbep3aIONyYEKf0pvjZuUVao1rvhfypie (Accessed 2020-04-27)

[29] Palmqvist Sjövall, Anna and Odestål, Oscar. *Adaptive Reference Images For Blood Cells Using Variational Autoencoders And Self-Organizing Maps.* Master's Thesis. 2020. Lund University.

[30] Scipy. *scipy.optimize.linear_sum_assignment.* https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.linear_sum_assignment.html (Accessed 2020-04-29)

Master's Theses in Mathematical Sciences 2020:E29
ISSN 1404-6342

LUTFMA-3410-2020

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/