Generation of Synthetic White Blood Cell Images Using Denoising Diffusion

Fanny Nilsson
Louise Zettergren

Supervisors
Anders Heyden (Lund University)
Gert-Ola Carlsson (CellaVision)
Karolina Lind (CellaVision)
Examinator
Niels Christian Overgaard (Lund University)

LUND UNIVERSITY

Abstract

CellaVision's digital hematology systems are designed to analyze blood and pre-classify different types of blood cells. Some abnormal white blood cells are rare, which can cause imbalanced datasets. This can lead to a decrease in pre-classification performance and a need to carry out more time-consuming data gathering. The aim of this thesis is to investigate the possibility of using deep learning to generate synthetic images of white blood cells with abnormalities, in order to augment the training dataset of the pre-classifier.

Denoising diffusion is a new cutting edge method to generate synthetic data and has been shown to be able to generate state-of-the-art images. A diffusion model works by adding noise to training images and learning to remove the noise. The diffusion model of this thesis was created by first training a base model on images with and without abnormalities and then fine-tuning it for three different types of abnormalities: hypersegmentation, Döhle bodies and hypergranulation. A Generative Adversarial Network (GAN) was trained and its performance was compared to the performance of the diffusion model.

To evaluate the generated images, the performance of a classifier trained on a dataset augmented by generated images was compared to a classifier trained only on real cell images. It is uncertain whether adding generated images to the training dataset resulted in an improved classifier performance. For two of the abnormalities, an increase in accuracy was seen for the abnormal class but in the other cases there was a decrease in accuracy.

Moreover, a medical expert and an experienced CellaVision employee were both given a set of 100 cell images, whereof 50 were synthetic. They were then asked to assess which cell images were synthetic. The medical expert was able to classify 96% of the real images as real, but only 32% of the synthetic images were correctly classified. In turn, the experienced CellaVision employee was able to correctly classify 44% of the real images and 24% of the synthetic.

Acknowledgements

We would like to thank our LTH supervisor Anders Heyden for supporting us and answering our questions. Anders has contributed with constructive feedback and guidance in the steps needed to do a master thesis.

Furthermore, we would like to thank CellaVision for giving us the opportunity to write our thesis and for the support we have been given. A special thank you to our supervisors at CellaVision, Gert-Ola Carlsson and Karolina Lind, for proposing the project. Gert-Ola and Karolina encouraged us through the process and helped us brainstorm new ideas as well solutions to emerging problems.

A thank you to Emily Källström, biomedical scientist at CellaVision, and Kent Stråhlen, product manager at CellaVision. Emily and Kent helped us evaluate our synthetic images. We would also like to thank Martin Almers and Håkan Wieslander at CellaVision for sharing their knowledge in deep learning, particularly generative models.

Lastly we would like to thank the team, Applications PB, for making us feel welcome and for many nice thursday fikas.

Contents

1 Introduction

1.1 Motivation

CellaVision's systems find and pre-classify white blood cells. The user examines and, if needed, adjusts the classification of each cell. Knowing the proportion of cells belonging to each class is important to be able to diagnose patients correctly and evaluate the performance of medical treatments. Serious diseases such as leukaemia can in some cases only be diagnosed by examining the distribution of cells.

Some types of white blood cells are less common than others, however, the pre-classification is still expected to suggest the correct class with few exceptions. The imbalanced distribution has consequences. One example is a larger workload when it comes to data gathering and evaluation by experts to obtain the desired precision of the pre-classification.

1.2 Aim of the Thesis

The aim of the thesis is to examine the possibility of using denoising diffusion to generate synthetic images of white blood cells with abnormalities.

We aim to answer the following questions:

1. Is it possible to implement a Denoising Diffusion Implicit Model [1] that can generate images of white blood cells with abnormalities such that an expert in the field cannot differentiate them from real images?
2. Can the performance of a classifier be improved by training it on data augmented with generated images?
3. Is it possible to generate images using a diffusion model that outperforms images generated with a Generative Adversarial Network (GAN)?

2 Related Work

2.1 Blood Cell Data Augmentation using Deep Learning Methods

In a previous master thesis done at CellaVision, Klang and Carlberg [2] constructed and tested Generative Adversarial Networks (GANs) in order to generate images of white blood cells. The goal of the master thesis was to generate images with GANs and evaluate the performance as well as investigate the possibility of improving the classification of white blood cells by adding the synthetic images to the training data.

To test the quality of the generated images, a medical expert was asked to evaluate if the images in a set of real and synthetic images seemed real or not. The results show that the expert classified 46% of the synthetic images as real.

The synthetic images were lastly added to a training set which was classified with a Xception-classifier. The weighted F1-score for the classifier that was trained without synthetic images was 0.946 and when adding the synthetic images, the classifier had a F1-score of 0.953.

One problem that arose when the authors created the model was a gridlike texture visible in the background of the generated images. According to the authors, this could be caused by the choice of kernel size and the number of convolutional filters.

2.2 Brain Imaging Generation with Latent Diffusion Models

In a report from 2022, Pinaya et al. [3] presented the results of their attempts to generate synthetic images of the brain using latent diffusion models. To train their model, they used the UK Biobank dataset which consists of 31740 images of the brain as well as information about the person. As a baseline they also created and trained a GAN.

The authors concluded that the diffusion model was able to generate high quality images. Comparing it to the GAN, the authors found the diffusion model easier to train since it was more stable and converged more easily.

The end result was 100 000 generated images that the authors have made available for the public to use.

3 Background

3.1 Blood Cells

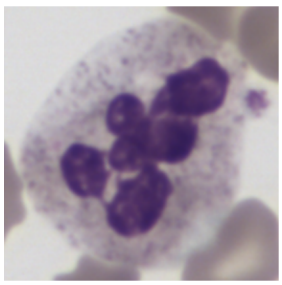
Blood consists of plasma, red blood cells, platelets and white blood cells (WBCs) [4]. WBCs consist of a nucleus and cytoplasm. White blood cells can be divided into 5 types: monocytes, lymphocytes, basophils, eosinophils and neutrophils [5]. Neutrophils are the most prevalent type of WBC in the blood and fight off bacteria and fungi. In this thesis, the focus is on neutrophils.

Abnormalities in Neutrophils

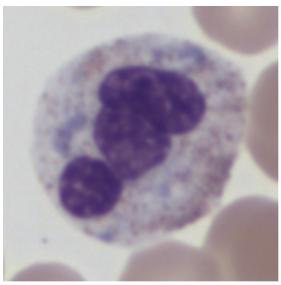
Neutrophils can have abnormal features, which can indicate that the patient has health issues. Three of these are hypersegmentation, Döhle bodies and hypergranulation.

Normally, neutrophils have 3 or 4 nuclei [6]. One considers the neutrophils to be hypersegmented when there are many neutrophils with 5 nuclei or any neutrophils with 6 or more nuclei, as visible in Figure 1a.

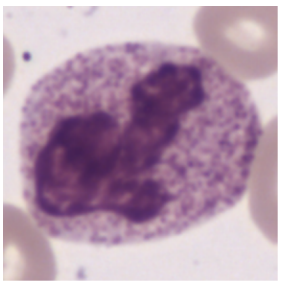
Sometimes oval, blue-gray structures can appear in the cytoplasm [7]. These are called Döhle bodies and an example of a cell with this abnormality is visible in Figure 1b. The Döhle bodies often occur together with hypergranulation. A cell is considered hypergranular when there are a larger number of granules in the cytoplasm than normal [8]. Usually, hypergranulation also entails that the granules are dark purple. A hypergranular neutrophil is visible in Figure 1c.



(a) Hypersegmented neutrophil. The neutrophil has 6 nuclei.



(b) Neutrophil with Döhle bodies.



(c) Hypergranular neutrophil.

Figure 1: Examples of images of cells with different abnormalities.

3.2 Machine Learning

Machine learning aims to solve tasks that are difficult to solve with a fixed program [9]. The algorithms are designed to learn from data and experience.

Examples of such tasks are classification and synthesis. Besides training the algorithm on a training dataset, it is common to validate the model on a validation dataset using some metric and finally evaluate the performance on a test dataset.

When performing classification, the task is to specify which category the input belongs to. When the task is synthesis, the machine learning algorithm is meant to generate new data. The generated data should be similar to the dataset used for training the algorithm.

3.3 Deep Learning

Deep learning is a method of machine learning which is built on artificial neural networks (ANNs) [9]. ANNs are, as the name implies, networks inspired by the function of the neurons in the human brain. The network consists of connected nodes, which can be likened to the neurons in the brain.

The simplest example of a neural network is the perceptron, which can be seen in Figure 2. The network begins by multiplying all inputs $x_1, ..., x_K$ with their weights $w_1, ..., w_K$ and adds them together with a bias b, which results in a weighted sum. The weighted sum is then run through an activation function φ which produces the output y.

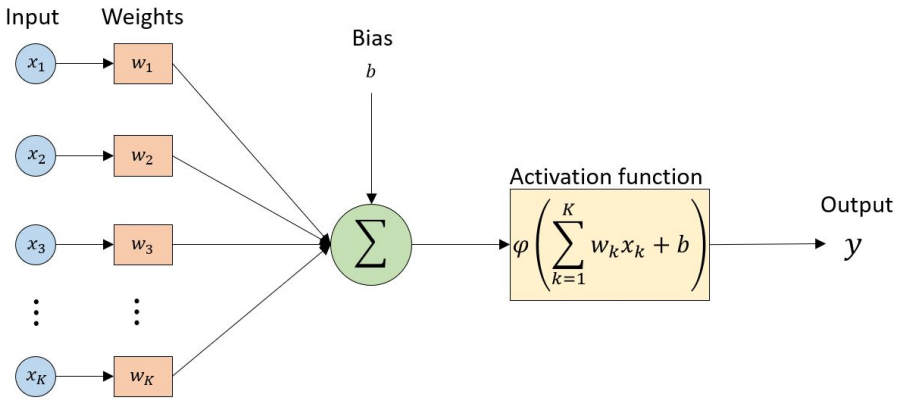


Figure 2: Illustration of a simple perceptron.

One of the most common types of ANN is the feedforward network, also called Multilayer Perceptron (MLP). The MLP consist of three types of layers: input layers, hiddens layer and output layers, where the nodes are connected to each other. All nodes have an input, bias, weight and output associated with them. A simple example of a MLP with one hidden layer can be seen in Figure 3. The name feedforward network comes from the fact that no feedback is included in the network. Information is only passed from the input, through the network and lastly to the output. The purpose of the MLP is to find an output function

$y(x)$ for inputs x that results in the outputs being equal to targets d. Since the output of the MLP depends on the biases and weights for the nodes in the network, the task is solved by finding the weights w that fit the purpose as well as possible.

The output of a node is calculated as

$$y = \varphi\left(b + \sum_{k=1}^{K} w_k x_k\right).$$

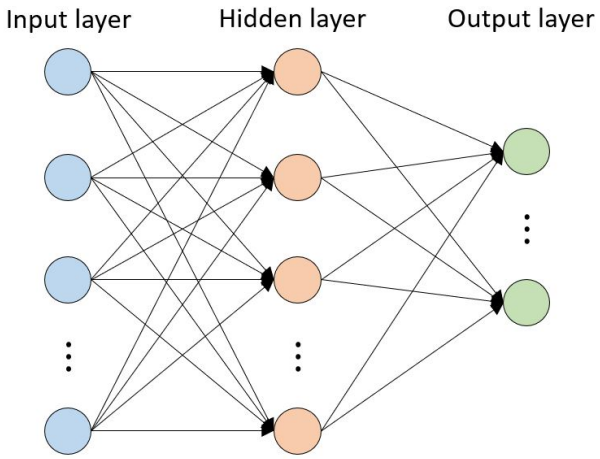


Figure 3: Illustration of multilayer perceptron with one hidden layer.

3.4 Network Types

Convolutional Neural Network (CNN)

The convolutional neural network is a type of ANN that is often used when working with images [11]. CNNs typically have three different types of layers: convolutional layers, pooling layers and fully-connected layers.

Convolutional layers consists of a convolutional operation between the input to the layer and a convolutional kernel [9]. In short, the kernel slides over the input data whilst performing elementwise multiplication. In the two-dimensional case, such as when the input is an image, the discrete convolution operation is defined as

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n),$$

where I is the input image and K is the two dimensional kernel.

The pooling layer performs downsampling of the input [11]. Two common types are average pooling layers and maxpooling layers. Average pooling layers output the average input value within a rectangular neighbourhood. If the input is an image, an average pooling layer will result in a downsampled image which is somewhat smooth. Maxpooling layers output the maximum input value within a rectangular neighbourhood. If the input is an image, a maxpooling layer will result in a downsampled image with higher contrast. Furthermore, the fully connected layers are traditional layers of an ANN. All neurons in a fully connected layer have connections to all neurons in an adjacent layer.

Recurrent network

As opposed to the feedforward networks, Recurrent Neural Networks (RNNs) also have feedback included in the network. Because of this, RNNs are suited for data that is sequential [9]. In a recurrent neural network there is a feedback loop that feeds the output of the current layer as input to the layer in the next time step. An illustration of a simple recurrent neural network can be seen in Figure 4.

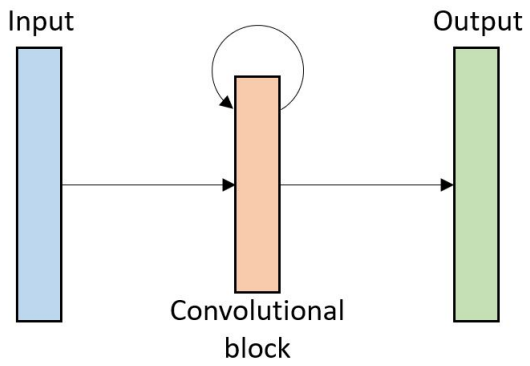


Figure 4: Illustration of a simple recurrent network with a convolutional block.

Residual network

To reduce the risk of vanishing gradients when performing stochastic gradient descent (Section 3.7) and the risk of bad generalization with deep networks, one can use residual networks [10]. Instead of fitting the weights in a layer directly on the input, the layer will fit a residual mapping which will then be added to the input to produce the output. An illustration of a simple residual network can be seen in Figure 5.

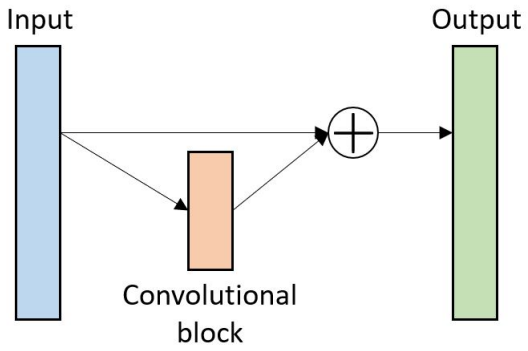


Figure 5: Illustration of a convolutional block with a residual connection.

3.5 Activation Functions

The Rectified Linear Unit (ReLU) is a common activation function in feedforward networks [9]. Its functional form is given in Equation 1. It is a piecewise linear function, and hence many of the properties that make linear models easy to optimize with gradient-based methods are preserved. Linear models also generalize well. A visual representation of the ReLU is visible in Figure 6a.

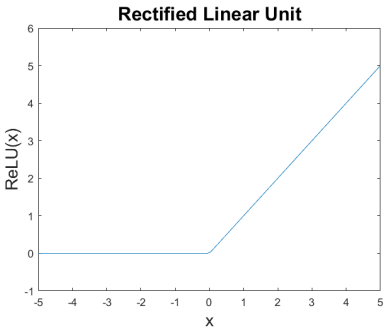
$$\text{ReLU} = \max\{0, x\} \tag{1}$$

To avoid the activation function or its derivative taking on zero-values in the negative region, LeakyReLU (LReLU) is an option [12]. Its functional form is visible in Equation 2. A visual representation of the LReLU with $\alpha = 0.3$ is visible in Figure 6b.

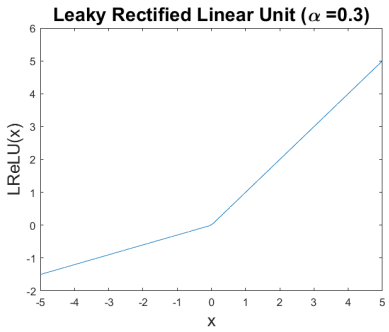
$$\text{LReLU} = \max\{x, \alpha x\} \tag{2}$$

Another option is to use the hyperbolic tangent function, *tanh*, as activation function. Its functional form is visible in Equation 3. A visual representation of the hyperbolic tangent function is visible in Figure 6c.

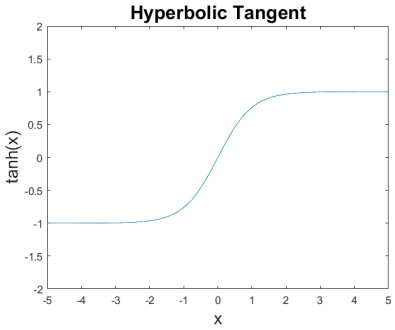
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$



(a) The rectified linear unit.



(b) The leaky rectified linear unit.



(c) The hyperbolic tangent.

Figure 6: A variety of activation functions. Mind the difference in y-axis limits when comparing the different activation functions.

Lastly, the softmax activation function is popular when performing multiclass classification [13]. It transforms the raw output of a network into probabilities of belonging to each class. Its functional form is visible in Equation 4. Here, the vector $\mathbf{z}$ is the raw output and $softmax(z_i)$ is the predicted probability that the output belongs to class i.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{4}$$

3.6 Loss Functions

The goal of most machine learning models is to optimize a function f, called the objective function. When the goal is to minimize the function it can also go under the names loss function and cost function [9]. There are several loss functions to choose from and they have varying performance for different problems. One often used loss function is the binary cross entropy loss, which is defined

as

$$\text{Binary Cross Entropy Loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1-y_i)\log(1-p(y_i)),$$

where y_i is the target label [14].

Sparse categorical cross entropy can be used when there are two or more classes. Here the truth labels are integer encoded, for example: [0], [1] and [2] for a problem with three classes. The loss function is defined as [15]

$$\text{Sparse Categorical Cross Entropy} = -\sum_{i=1}^{n} y_i \log(p_i),$$

where n is how many classes the problem have.

Another commonly used loss function is the mean absolute error, often shortened MAE. MAE is defined as the mean of the absolute difference between the prediction and the true value [16],

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}.$$

3.7 Training and Optimization

In order to optimize a loss function, a minimum of the function needs to be found [9]. In machine learning, the function that should be optimized can be very complex and it can be difficult to find the precise minima. Gradient descent is a method which is able to give a good approximation of the minima and uses the partial derivatives

$$\frac{\partial}{\partial x_i} f(x)$$

of the inputs $\mathbf{x}$, which together form the gradient. In the beginning, one starts at a random point and evaluate the directional derivative, $\mathbf{u}$, at this point. Since the goal is to find the minima, the idea is to find which direction the derivative decreases the fastest in, by determining

$$\min_{\mathbf{u}, \mathbf{u}^T\mathbf{u}=1} \mathbf{u}^T \nabla_{\mathbf{x}} f(\mathbf{x})$$

and take a step in that direction. The new step proposed by gradient descent is determined by

$$\mathbf{x}' = x - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x}). \tag{5}$$

This procedure is continued until the partial derivatives in all directions are zero or close to zero, meaning we are close to a minimum.

The ϵ in Equation 5 is a scalar called the learning rate, which represents the length of the steps taken when determining a new point. A large learning rate means longer steps taken. This might mean that the minimum is found in fewer steps, but it might also result in the minimum being missed. This may be solved by lowering the learning rate, but choosing a too low learning rate can result in slow convergence. The learning rate is therefore a hyperparameter that has to be tuned carefully for each task.

The same procedure can be used when the objective is to maximize the function. This is then called gradient ascent. Instead of taking a step in the direction where the derivative is the lowest, one should take a step in the direction of the highest derivative.

To be able to compute the gradient in order for the network to learn with gradient descent, information from the loss needs to be passed back through the network. This can be done with the back-propagation algorithm, which passes the information backward through the network and computes the gradients.

When the training set gets larger, performing gradient descent will become more computationally challenging and demand longer time. In order to speed up the training, one can use stochastic gradient descent (SGD). Instead of forming the gradient for all partial derivatives for the input, SGD uses the partial derivatives for a smaller portion of the input called a minibatch. Using stochastic gradient descent as opposed to ordinary gradient descent removes the influence of the size of the dataset, since SGD only depends on the size of the minibatch.

One method using stochastic gradient-based optimization is called Adam. It has been shown to have multiple advantages, e.g., that is is straightforward to implement and well-suited for large problems [17]. In order to improve the regularization of the Adam optimizer, the authors of the paper "Decoupled Weight Decay Regularization", Loshchilov and Hutter [18], presented a modified version of Adam called AdamW. In the modified optimizer, when and how weight decay is applied, have changed. The effect of this is that AdamW yields a better generalization performance, meaning it performs better on unseen data.

3.8 Batch Normalization

One of the complications that arises when training the network is the fact that the distribution of the inputs to the layers changes throughout the training. As a way to solve this problem, Ioffe and Szegedy [19], proposed using batch normalization. As the name would suggest, the method consist of normalizing the layer inputs and to do this for every batch. The effect of batch normalization will be faster training and it has the advantage that it is simple to include in the model architecture.

3.9 Generative Adversarial Network (GAN)

The Generative Adversarial Network is a deep learning technique that was introduced by Ian J. Goodfellow et al. [20] in 2014. The idea of GAN is to have two neural networks, a generator and a discriminator, that are pitted against each other. During the training of the model, the generator has the task to generate new data with the purpose of tricking the discriminator that the data is from the training dataset, and the discriminator has the task of separating the generated data from the real data. The output of the discriminator will then help the generator to generate images with a larger probability of seeming real. An illustration of the GAN can be seen in Figure 7.

The training of the model works by first extracting m samples

$$\mathbf{x} = \{\mathbf{x^{(1)}}, ..., \mathbf{x^{(m)}}\}$$

from the real dataset, p_{data}, and then as many samples

$$\mathbf{z} = \{\mathbf{z^{(1)}}, ..., \mathbf{z^{(m)}}\}$$

from the latent space, p_z.

Generated samples are then produced by passing the latent vector $\mathbf{z}$ to the generator. The real images $\mathbf{x}$ and the generated images $G(\mathbf{z})$ are then sent through the discriminator and the discriminator returns an output, $D(\mathbf{x})$, which can be interpreted as the probability that the real data, $\mathbf{x}$, is real, meaning it comes from the training data instead of the generator distribution (p_g). If the generator was completely unable to fool the generator, the output would be $D(\mathbf{x}) = \mathbf{1}$ and if the discriminator was unable to tell the generated images from the real ones, the output instead would be $D(\mathbf{x}) = \frac{\mathbf{1}}{\mathbf{2}}$.

The discriminator wants to maximize the probability of assigning both real and generated images their correct label, which can be described as

$$\max \log D(\mathbf{x}) + \log(1 - D(G(\mathbf{z}))).$$

With the output of the discriminator, the discriminator is updated using the help of gradient ascent according to

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)})))]$$

In turn, the generator wants to minimize the probability that the discriminator classes the generated samples as fake, i.e.,

$$\min \log(1 - D(G(\mathbf{z}))).$$

The generator is updated using the discriminator output and gradient descent according to

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} [\log(1 - D(G(\mathbf{z}^{(i)})))].$$

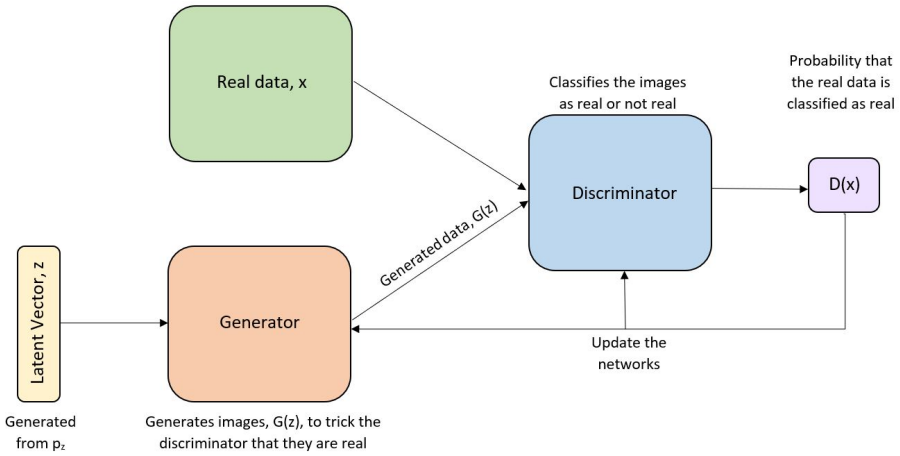


Figure 7: Illustrated structure of the GAN.

3.10 Transfer Learning

Transfer learning is a technique where one can use the knowledge gained by solving a problem and apply it to another problem [21]. This can be a suitable method when the dataset is not very large. Having a larger but similar dataset, one can then train a model on the larger dataset and then save the model. The model can then be used as a base model to be trained further on the smaller dataset.

To learn from the smaller dataset, one can use different approaches. One option is to fine-tune the full base model, i.e., train all layers of the base model on the smaller dataset with a low learning rate. One can also freeze the weights of some of the layers of the base model and train the unfrozen layers on the smaller dataset. A third option is to freeze the full base model and add additional layers to the network. These additional layers will then be trained on the smaller dataset.

FreezeD

When applying transfer learning to a GAN, a simple baseline called FreezeD has been shown to outperform other techniques [22]. In short, it consists of training a GAN on a large dataset until desired results are achieved. Next, the lower layers of the discriminator are frozen and the rest of the discriminator weights as well as all generator weights are kept trainable. These are then trained on the smaller dataset.

3.11 Positional Embeddings

Positional encoding, or positional embedding, is a way of maintaining knowledge of the order of objects in a sequence [23]. Each entity in a sequence is assigned a low-dimensional continuous vector, which is unique for all positions. Mapping each index to a vector results in the output of the embedding layer being a matrix. Each row of the matrix represents the object with the row number as position, summed with its positional information.

More specifically, one can use sinusiodal embeddings. When the input sequence is of length L, the positional encoding of the k:th element can be calculated as follows:

$$P(k, 2i) = \sin\left(\frac{k}{n^{\frac{2i}{d}}}\right)$$

and

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{\frac{2i}{d}}}\right).$$

This will result in an encoding matrix where the row number corresponds to the position in the sequence. Moreover, d is the output embedding dimension and n is a user-defined scalar, originally set to 10000. Lastly, $0 \leq i < \frac{d}{2}$ is used to map column indices of the encoding matrix to sine and cosine functions.

3.12 Diffusion Models

Diffusion models are a family of generative models. First, they progressively destruct data by injecting noise. Next, they learn to reverse this process for sample generation. The intuition of diffusion models is depicted in Figure 8.

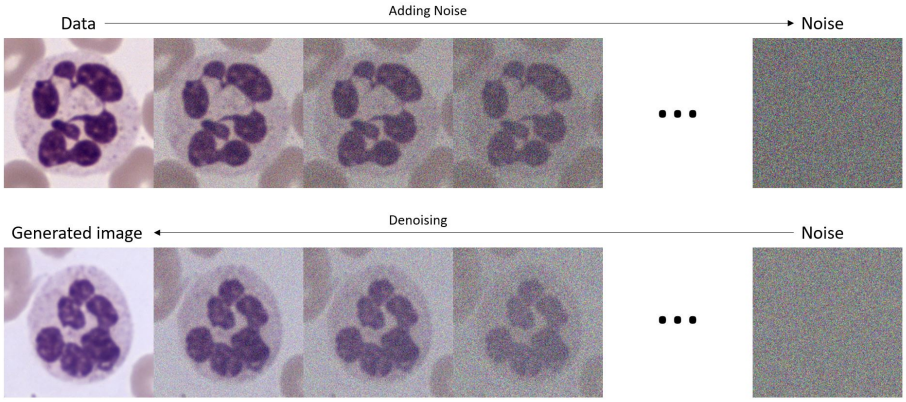


Figure 8: The intuition of diffusion models.

Mainly there are two different types of diffusion models: Denoising Diffusion Probabilistic Model (DDPM) and Denoising Diffusion Implicit Model (DDIM).

The key difference between the two models lies in the generative process (from noise to images). In DDPM this is done with Markov chains, which is a time consuming process since the denoising is done step by step. This is a problem that the developers of the DDIM wanted to solve by making the process non-Markovian [1]. A common network architecture for diffusion models is the U-net. According to Oktay et al., adding attention gates to the U-net could improve performance [24].

U-net

A U-net is a U-shaped network architecture with two parts [25]. The first part, the encoder, consists of convolutional layers and maxpooling layers. This contracting network is then followed by the second part, the decoder. The decoder also has convolutional layers but the pooling operators are replaced by upsampling operators. In the decoder, the resolution is increased. Moreover, high resolution features from the encoder layers are combined with the upsampled output using skip connections. Using the information from the skip connections as well as the information from the upsampling operators, one can achieve a better performance and a more precise output. Simplified, the layers with the highest numbers of feature channels handles details and the layers with fewer feature channels gives context and overview. A visual representation of an example of a U-net can be seen in Figure 9.

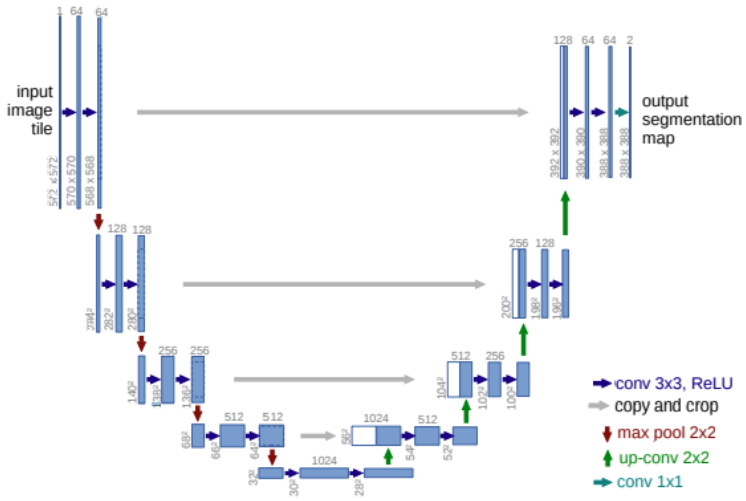


Figure 9: Example of U-net architecture used in a segmentation task [25].

R2U-Net

As a way to combine the positive effect of U-Nets, RNN and residual networks, Alom et al. [26] presented a combined network called R2U-Net in order to

increase performance for segmentation tasks. Alom et al. motivates the new model by stating that adding residual and recurrent connections in the U-Net have several advantages, for example better feature representation and simplified training of deeper networks. The results show an increased performance for the segmentation task compared to using other existing networks, including U-Net.

Attention

In general, the attention mechanism consists of mapping a query and a set of key-value pairs to an output [27]. The output of the attention layer is a weighted sum of the values. The different weights are determined by a compatibility function taking the query and key as input. Recently, the attention layers have become popular to use with a U-net architecture. In that case, the query comes from the previous decoder block and key-value pairs comes from the skip connections.

A way of using attention in U-nets was proposed by Oktay et al. in Attention U-Net: Learning Where to Look for the Pancreas [24]. Their attention mechanism, which they call an attention gate, can be seen in Figure 10. It is meant to weight different parts of an image and assign larger weights to more relevant parts. The attention weights are updated during training using back-propagation.

The attention gates are placed at the skip connections, to lessen the impact of the insufficient feature representation from the initial layers in the encoder. The attention gate takes two inputs, the query or gating signal, g, and x which is the skip connections. The gating signal g comes from a lower layer in the network and hence has better feature information whilst the skip connections have better spatial information. The inputs are then run through convolutional layers to have the same dimensions and are then added. In the addition, the aligned weights get larger and the unaligned weights get relatively smaller. The sum is then sent through a ReLU activation function as well as a sigmoid activation function, to normalize. The attention weights have been found and only need to be upsampled to have the same dimensions as x. The output of the attention gate is found by multiplying the attention weights with x elementwise.

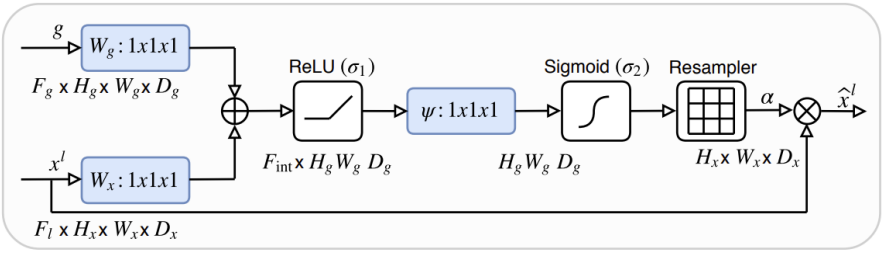


Figure 10: Illustration of the attention gate, introduced in Attention U-Net: Learning Where to Look for the Pancreas [24].

Denoising Diffusion Probabilistic Model (DDPM)

A sequence of random variables is a Markov chain if, at any time point, the future states depend on history only through the present state [28]. This is called the Markov property.

Denoising Diffusion Probabilistic Models use two Markov chains; a forward chain that destructs data by adding noise according to a chosen noise schedule and a reverse chain which does the opposite [29]. The forward chain is designed to transform the data distribution into a simple prior distribution. The reverse chain reverses the forward process by learning transition kernels. These transition kernels are parameterized by deep neural networks.

The forward process starts with a data distribution $x_0 \sim q(x_0)$, i.e, training images. From this, it generates a sequence of random variables $x_1, ..., x_T$, using the transition kernel $q(x_t|x_{t-1})$. This transition kernel has been designed to transform the data distribution into a simple prior distribution, e.g., Gaussian. The most common choice is

$$q(x_t|x_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}x_{t-1}, \beta_t \mathbf{I}), \tag{6}$$

where $\beta = \{\beta_1, ..., \beta_T\}$ represents the variances of the forward process and is given by the chosen noise schedule.

Furthermore, one can factorize the joint distribution using the chain rule of probability as well as the Markov property into

$$q(x_1, ..., x_T|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}). \tag{7}$$

Using Equation 6 and Equation 7 with $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=0}^{t} \alpha_s$ results in

$$q(x_t|x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}x_{t-1}, (1-\bar{\alpha}_t)\mathbf{I}).$$

This means that a sample x_t can be generated using the initial distribution x_0 and a sample of a Gaussian vector $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, as

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon. \tag{8}$$

The reverse process Markov chain has a prior distribution which is a Gaussian, $p(x_T) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. It is also parameterized by a learnable transition kernel,

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t)).$$

Here θ is the model parameters, $\mu_\theta(x_t, t)$ is the mean and $\Sigma_\theta(x_t, t)$ is the variance.

To generate new data, one samples a prior vector (an unstructured noise vector) from the simple prior distribution and then gradually removes noise from it by running the learnable Markov chain in the reverse time direction. In other words, generate a data sample by first sampling a random noise vector $x_T \sim p(x_T)$. Next, sample repeatedly from the transition kernel $x_{t-1} \sim p_\theta(x_{t-1}|x_t)$ until the selected number of diffusion steps T is done.

To be able to generate high quality images, the reverse process needs to match a time reversal of the forward process. To achieve this, the parameter θ needs to be adjusted until the joint distribution of the reverse Markov chain approximates the forward process joint distribution.

This is done by minimizing the variational upper bound on the negative log likelihood [30], which is defined as

$$L = \mathbb{E}_q \bigg[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \bigg]. \tag{9}$$

To train the model, L is minimized using stochastic gradient descent.

Furthermore, Ho et al. [30], shows that L in Equation 9 can be rewritten using Kullenback-Leibler divergences. The effect of this is a reduced variance of the variational bound.

Denoising Diffusion Implicit Model (DDIM)

As alternative to the time-consuming DDPM, Song et al. [1] presented the Denoising Diffusion Implicit Model in a report from 2022. In order to reduce the number of steps in the generative process, DDIM make use of non-Markovian methods. To remove the Markov property in the forward process, a new family of inference distributions is defined as

$$q_\sigma(x_{1:T}|x_0) = q_\sigma(x_T|x_0) \prod_{t=2}^{T} q_\sigma(x_{t-1}|x_t, x_0),$$

indexed by $\sigma \in \mathbb{R}^T_{\geq 0}$ and the factors are defined as

$$q_\sigma(x_T|x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_T}x_0, (1-\bar{\alpha}_T)I),$$
$$q_\sigma(x_{t-1}|x_t, x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_{t-1}}x_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2} \cdot \frac{x_t - \sqrt{\bar{\alpha}_t}x_0}{\sqrt{1-\bar{\alpha}_t}}, \sigma_t^2 I). \quad (10)$$

Comparing this to the inference distribution for DDPM (Equation 7), there is now an added dependence on x_0. From the new inference distribution, the forward process can be derived with the use of Bayes' rule

$$q_\sigma(x_t|x_{t-1}, x_0) = \frac{q_\sigma(x_{t-1}|x_t, x_0)q_\sigma(x_t|x_0)}{q_\sigma(x_{t-1}|x_0)}.$$

As opposed to the forward process for the probabilistic model (Equation 6), x_t can depend on both x_{t-1} and x_0, making it non-Markovian.

In the generative process $p_\theta(x_{0:T})$ of the DDIM, the first step is to make a prediction of x_0, given x_t. This can be done by a rewrite of Equation 8, which results in

$$f_\theta^{(t)}(x_t) := \frac{x_t - \sqrt{1-\bar{\alpha}_t} \cdot \epsilon_\theta^{(t)}(x_t)}{\sqrt{\bar{\alpha}_t}}. \quad (11)$$

Here ϵ_θ is a set of functions, $\{\epsilon_\theta^{(t)}\}$, with parameters θ.

The prediction of x_0, $f_\theta^{(t)}(x_t)$, will then be used to define the generative process as

$$p_\theta^{(t)}(x_{t-1}|x_t) = \begin{cases} \mathcal{N}(f_\theta^{(t)}(x_1), \sigma_1^2 I) & t = 1, \\ q_\sigma(x_{t-1}|x_t, f_\theta^{(t)}(x_t)) & \text{otherwise} \end{cases} \quad (12)$$

where q_σ is defined in Equation 10 and $f_\theta^{(t)}$ in Equation 11.

In the report, Song et al. present the observation that the objective function L (Equation 9) does not depend on the inference function $q(x_{1:T}|x_0)$ and only on the marginals $q(x_t|x_0)$, meaning that the objective function is not dependent on σ or whether the forward process is Markov or non-Markov. Therefore, the objective for the implicit model will be equal to the one for the probabilistic and same for any choice of σ.

From the generative process 12, a sample x_{t-1} can be generated as follows

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\left(\frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta^{(t)}(x_t)}{\sqrt{\bar{\alpha}_t}}\right) + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2} \cdot \epsilon_\theta^{(t)}(x_t) + \sigma_t \epsilon_t.$$

The choice of different σ will result in different generative processes. One of the special cases is when

$$\sigma_t = \sqrt{(1-\bar{\alpha}_{t-1})/(1-\bar{\alpha}_t)}\sqrt{1-\bar{\alpha}_t/\bar{\alpha}_{t-1}},$$

since the generative process then becomes Markovian. This means that the generative process will be equivalent to that of a DDPM. Another case is when $\sigma_t = 0$ for all t. The forward process then becomes deterministic and the model becomes an implicit probabilistic model. It is this case that Song et al. [1] call a Denoising Diffusion Implicit Model (DDIM).

The motive for generalizing the forward process as a family of inference functions is to be able to speed up the generative process. A way to do this is to skip steps in the generative process, which is visualized in Figure 11. This works since the objective function is not dependant on a specific forward process. When generating, a subset of the latent variables are chosen: $\{x_{\tau_1}, ..., x_{\tau_S}\}$, where τ is an increasing sequence of length S, $t \in [1, T]$. Because of the decrease in sampling steps, this will result in faster sampling than the DDPM.

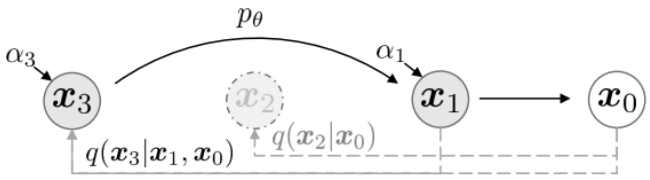


Figure 11: Visualization of skipping steps in the generative process [1]. Here, $\tau = [1, 3]$ whilst $t = [1, 2, 3]$.

Noise Schedules

The addition of noise to the training data is done according to a chosen noise schedule. The noise scheduling is crucial for the performance of the model and different schedules are preferable for different tasks.

In the report "*On the Importance of Noise Scheduling for Diffusion Models*" [31], Ting Chen shows that the best choice of noise schedule is dependant on the resolution of the data. A noise schedule that works well for a higher resolution may not be a good fit for lower resolution data. Chen motivates this by stating that the task of denoising a higher resolution image is easier since the redundancy of information in data increases.

In the report Chen presents the results of experiments with two different types of noise schedules with different hyperparameters, for three different resolutions. The first schedule is a cosine noise schedule,

$$\text{cosine schedule} = \frac{\cos(end \cdot \frac{\pi}{2})^{2\tau} - \cos((t \cdot (end - start) + start) \cdot \frac{\pi}{2})^{2\tau}}{\cos(start \cdot \frac{\pi}{2})^{2\tau} - \cos((t \cdot (end - start) + start) \cdot \frac{\pi}{2})^{2\tau}},$$

with parameters *start* (start value), *end* (end value) and τ. Examples of cosine noise schedules are shown in Figure 12. The plots show the signal rate, which is

how much of the original data is left (1 meaning no added noise and 0 meaning the image is all noise).

The second schedule that was tested was a sigmoid noise schedule,

$$\text{sigmoid schedule} = \frac{\frac{1}{1-e^{end/\tau}} - \frac{1}{1-e^{(t(end-start)+start)/\tau}}}{\frac{1}{1-e^{start/\tau}} - \frac{1}{1-e^{(t(end-start)+start)/\tau}}},$$

which has the same parameters as the cosine noise schedule. Examples of sigmoid noise schedules can be seen in Figure 13. In the report by Chen, the result show that a sigmoid noise schedule produces a better result than the cosine noise schedule for all three resolutions.

Another schedule that is often used, is the linear schedule where the noise variance increase equally for each time step. This is the schedule used in the earliest diffusion models and is one of the simplest schedules to implement.

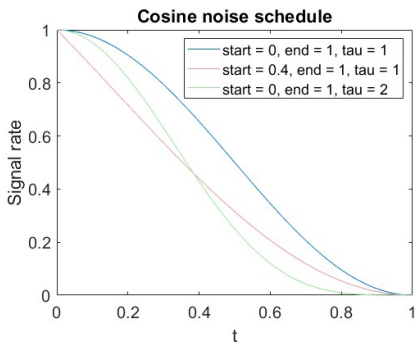


Figure 12: Examples of cosine noise schedules.

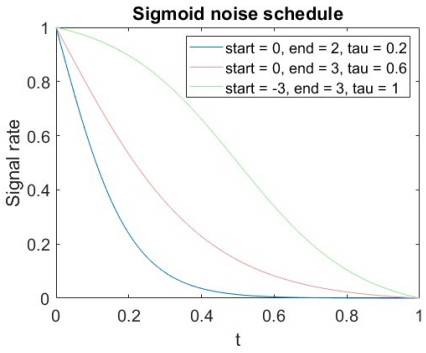


Figure 13: Examples of sigmoid noise schedules.

3.13 Metrics

To compare the results of a classification problem, there are different standard metrics that can be used [32]. They have different characteristics and the metric choice depends on the classification problem.

One can use a confusion matrix, as shown in Table 1. It displays the number of true positives, the number of true negatives, the number of false positives and the number of false negatives.

Table 1: Confusion matrix example.

		Predicted	
		Positive	Negative
Actual	Positive	True positives (TP)	False Negatives (FN)
	Negative	False Positives (FP)	True Negatives (TN)

Accuracy is a widely used metric. It is calculated using the total number of predictions made and how many of these predictions that are correct. In mathematical representation,

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{TP + TN}{TP + FP + TN + FN}.$$

If one wants to penalize the number of false negatives, recall is a suitable choice. In mathematical representation,

$$\text{Recall} = \text{Sensitivity} = \frac{TP}{TP + FN}.$$

One can also penalize false positives using specificity,

$$\text{Specificity} = \frac{TN}{TN + FP}.$$

It is notoriously hard to measure the quality of synthetic images. However, a few suitable metrics have been introduced recently. Fréchet Inception Distance (FID) was introduced by Heusel et al. in 2018 [33]. It is calculated as

$$\text{FID} = ||\mu_X - \mu_Y||^2 - \text{Tr}(\Sigma_X + \Sigma_Y - 2(\Sigma_X \Sigma_Y)^{1/2}).$$

Here μ_X is the mean of the activations from the last pooling layer prior to the output layer of an Inception model, when running the real images through the model. In turn, μ_Y is the same but for the generated images. Moreover, Σ_X and Σ_Y are the covariances of the activations from the last pooling layer prior to the output layer. A lower FID is to prefer since it means that the feature distance between the real and the generated images is lower.

4 Methods

4.1 Data

The data that was used to train our generative models consists of images of white blood cells, more specifically neutrophils with and without abnormalities. The datasets with abnormalities are divided into three categories: No abnormality, slight abnormality and severe abnormality.

The base models were trained using the data specified in Table 2. This data consists of images depicting neutrophils with and without abnormalities. Fine-tuning of the base models was done with images that depict cells with severe abnormality. This data is specified in Table 3. For the classification nets, the data with slight and severe abnormality was merged, which created a dataset with two categories: without abnormality and with abnormality.

Table 2: Data for training base models.

Cell Type	Number of cell images
Neutrophil	49793

Table 3: Data for training specialized generative models. The images depict cells with severe abnormality according to medical experts.

Abnormality	Number of cell images
Hypersegmentation	988
Döhle bodies	2040
Hypergranularity	10200

In Table 4 the division of the data used when training the classifier is shown. The first number in the brackets specifies how many images that have been classified as normal and the second shows how many that have been classified as abnormal. For the validation data, the set was randomly oversampled to match the number of images of cells that have been classified as normal.

Table 4: Division of generated and real data for training and evaluating the classifier.

Abnormality	Dataset	Number of real cell images	Number of generated images
Hypersegmentation	Training	[20000, 9459]	[0, 9459]
	Validation	[4983, 267]	-
	Test	[11330, 441]	-
Döhle bodies	Training	[20000, 8976]	[0, 8976]
	Validation	[4786, 464]	-
	Test	[11112, 660]	-
Hypergranularity	Training	[20000, 9823]	[0,9823]
	Validation	[4191, 1059]	-
	Test	[11028, 733]	-

Examples of real cell images can be seen in Figures 14-16. Figure 14 depicts nine images where the cells have been classified as severely hypersegmented and Figure 15 shows nine images of cells that have been classified to contain Döhle bodies to a severe degree. Lastly, Figure 16 contains nine images of cells that have been classified as severely hypergranular.

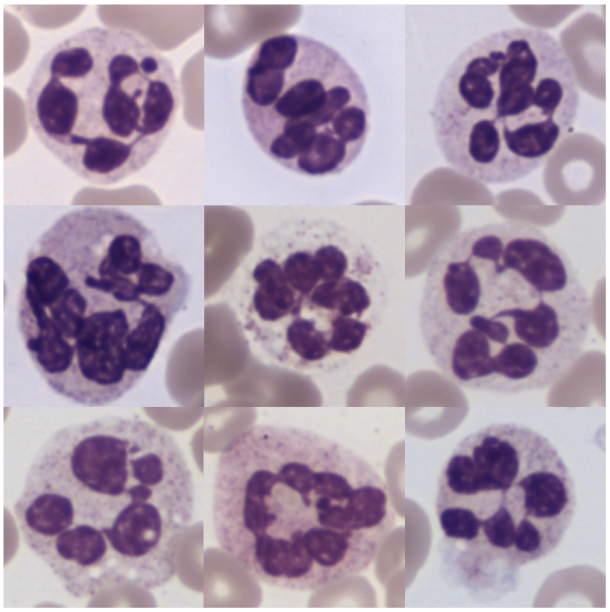


Figure 14: Examples of real cell images where the cells have been classified as severely hypersegmented.

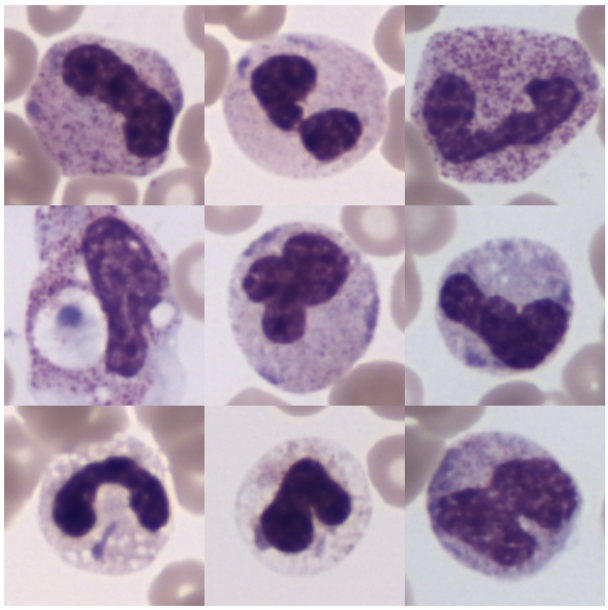


Figure 15: Examples of real cell images with Döhle bodies, classified to be severely abnormal.

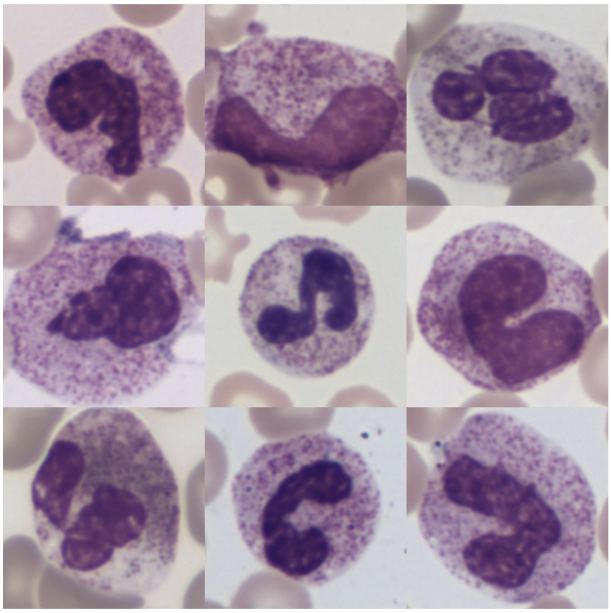


Figure 16: Examples of real cell images where the cells have been classified as severely hypergranular.

4.2 GAN

The first step in creating the GAN was to train a base model on the training data presented in Table 2. Every tenth epoch, the model weights were saved and images were generated. The images were then visually assessed, and the model where the images were deemed the best was used as the base model. The base model was then fine-tuned for images of cells with the abnormalities: hypersegmentation, hypergranularity and Döhle bodies.

Network Architecture

Klang and Carlberg [2] have previously created a Vanilla-GAN, with the goal of generating images of white blood cells of several different classes. The generator and discriminator in their Vanilla-GAN was CNNs with batch normalization included and Leaky ReLU as activation function. The architecture of the GAN for our work with generating images of neutrophilis with abnormities, was for the most part the same as Klang and Carlbergs original Vanilla-GAN. One of the changes made was exchanging the transpose convolutions in the generator with an upsampling layer followed by a convolution layer with stride (1,1). Another small change made, was changing the kernel size to be divisible by the stride.

The fine-tuning of the model was done according to the FreezeD procedure. This meant loading the weights of the base model and freezing the lower layer of the discriminator. All other weights of the discriminator as well as the generator were set to trainable. The number of layers that were frozen was tuned in order to optimize the performance.

Hyperparameters

There were several hyperparameters that were tuned to create the model. Among these were the batch size, kernel size, number of epochs, stride and noise dimension. For both the generator and the discriminator, the learning rate was set as well as the depth (number of filters in the first layer). Lastly the discriminator also had an update ratio, meaning how often the discriminator is updated relative the generator.

4.3 Denoising Diffusion Model

The choice was made to use a denoising diffusion implicit model, i.e., a diffusion model with non-Markovian methods.

Network Architecture

The diffusion model architecture was initially a standard U-net with maxpooling as upsampling operator. The number of filters in the convolutional layers doubled with each encoder block and was divided by two for each decoder block. The inputs were noisy images and the noise variance. Before concatenation, the

noisy images were run through an convolutional layer and the variances were run through an sinusoidal embedding layer. The output was of the same dimensions as the noisy input image.

To improve the performance, several changes were made to the U-net. Instead of the basic convolutional blocks of the standard U-net, a R2U-net was used. By each skip connection, attention gates, as shown in Figure 10, were implemented. The diffusion schedule was finally chosen to be a sigmoid schedule.

Firstly, a base model was trained using the data specified in Table 2 whilst tuning parameters until the performance was satisfactory. Next, three models were fine-tuned, one for each abnormality. This was done on the data specified in Table 3.

Hyperparameters

Different hyperparameters were tested as the aim was to find parameter values that gave optimal performance. The optimal values were found by visually assessing the different generated images. The number of features in the first encoder block, or the last decoder block, was varied to find an optimal number. The number of encoder blocks or the number of decoder blocks will be referred to as the network depth. This was also tuned, as well as the learning rate and batch size. The model was trained using the AdamW optimizer and therefore the value of the weight decay was a parameter in need of tuning.

In the noise schedule, there were also a few parameters in need of tuning such as start value, the end value and τ. The minimum signal rate and maximum signal rate were also set. The signal rate corresponds to $\sqrt{\bar{\alpha}}$. The minimum signal rate and maximum signal rate are the lower and upper bounds for this value, respectively. The number of diffusion steps, i.e., how many iterations it takes to go from noise to generated image, was tuned.

Lastly, there are two parameters in need of tuning in the embedding layer: the output embedding dimension and the maximum embedding frequency. The maximum embedding frequency decides how much of a noise variance change is needed to impact the result.

4.4 Testing the Generated Images

Testing our Images with an Expert

To test the generated images, a set of 100 images was given to a medical expert. 50 of the images were generated and 50 images were real. The medical expert was given the task to assess which images were real and which were generated. The expert was not aware of the number of generated images included. The dataset was shuffled so the real and generated images were ordered randomly.

This test was done with images generated with the diffusion model. The result was compiled and different metrics were calculated.

The same test was then repeated but performed by a CellaVision employee with lots of experience looking at cell images.

Testing our Images with a Classification Net

The classification net used to test the images was chosen to be MobileNetV2, introduced by Sandler et al. in 2018 [34], with a width multiplier $\alpha = 0.75$. The weights had been pre-trained on ImageNet, a dataset introduced by Deng et al. in 2009 [35]. To this, a global average pooling layer and a dense layer with a softmax activation were added.

Firstly, the net was trained using only real cell images as specified in Table 4. Secondly, the dataset was expanded with the generated images as specified in Table 4 and the net was trained. This was done using the Adam optimizer and sparse categorical cross entropy loss.

This test was repeated for images generated with the diffusion model for the three different abnormalities; hypersegmented neutrophils, hypergranular neutrophils and neutrophils with Döhle bodies.

Fréchet Inception Distance

The FID was calculated for the hypergranular cell images generated with both the GAN and the DDIM. This abnormality was chosen since a sample size of at least 10000 real images is recommended to calculated the FID. Embeddings for the real and generated images were computed using the InceptionV3 network, with weights pre-trained on ImageNet, before calculating the feature distance.

5 Results

5.1 GAN

Table 5 shows the hyperparameter settings that were used to train the GAN base model. In Table 6, the hyperparameters for fine-tuning the GAN are summarized. One exception being the learning rate for the hypergranular model, which was set to $5 \cdot 10^{-7}$. When fine-tuning the model for all three abnormalities, the two last convolutional layers in the discriminator were frozen, meaning the weights were not updated. In the generator, all batch normalization layers were frozen as well.

Table 5: Hyperparameter settings for training of the GAN base model.

Hyperparameter	Value
Learning Rate (Generator)	$1 \cdot 10^{-5}$
Learning Rate (Discriminator)	$1 \cdot 10^{-5}$
Batch Size	20
Depth (Generator)	1024
Depth (Discriminator)	64
Kernel Size	6
Epochs	150
Noise Dimension	100
Stride	2
Discriminator Update Ratio	1

Table 6: Hyperparameter settings for fine-tuning of the GAN base model.

Hyperparameter	Value
Learning Rate (Generator)	$3 \cdot 10^{-6}$
Learning Rate (Discriminator)	$3 \cdot 10^{-6}$
Batch Size	16
Depth (Generator)	1024
Depth (Discriminator)	64
Kernel Size	6
Noise Dimension	100
Discriminator Update Ratio	2

The generator and discriminator loss for the training of the base model are shown in Figure 17. Here we can see that after around 20 epochs, the loss increases for the generator and decreases for the discriminator. The images that were generated at this epoch were not very good, instead they were visually the best at epoch 50. The base model that was chosen was therefore the generator and discriminator weights that were saved at 50 epochs.

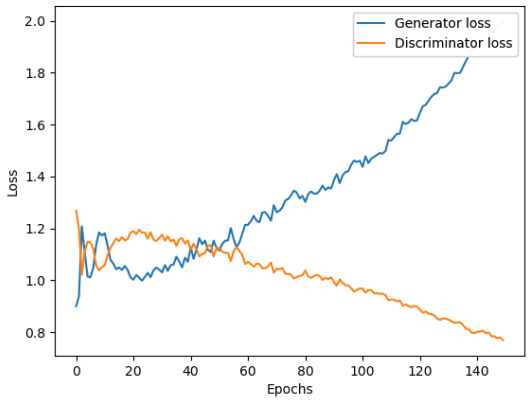


Figure 17: The loss plot of the GAN base model.

Examples of Generated Images

In Figure 18, nine examples of generated cell images are shown. The GAN has in this case been trained on images that have been classified as hypersegmented and was trained for 50 epochs. Figure 19 shows nine examples of cell images

with Döhle bodies generated with the GAN. The base model was fine-tuned on the cell images with Döhle bodies for 40 epochs. Lastly Figure 20 shows nine examples of cells generated by the GAN trained with images of hypergranular cells. The model was trained for 40 epochs.

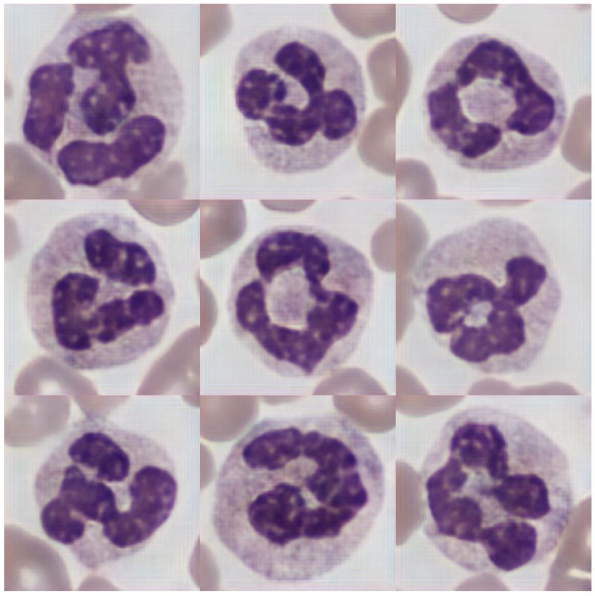


Figure 18: Example of cell images generated by a GAN. The model was trained on cell images that have been classified as hypersegmented.

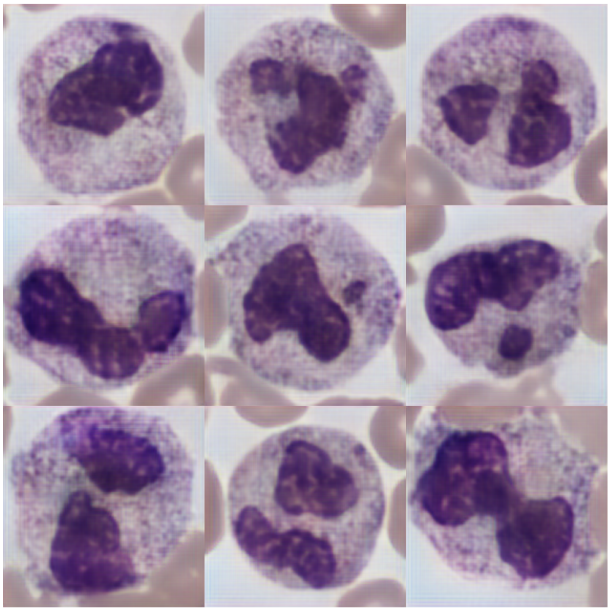


Figure 19: Example of cell images generated by a GAN. The model was trained on cell images that have been classified to have Döhle bodies.

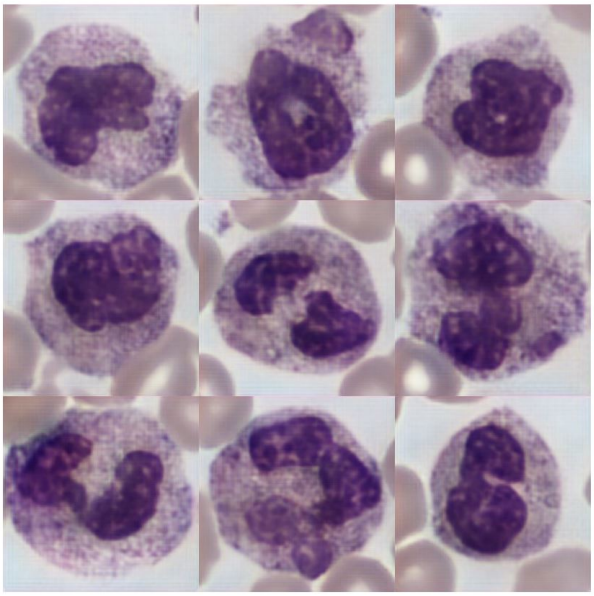


Figure 20: Example of cell images generated by a GAN. The model was trained on cell images that have been classified as hypergranular.

5.2 Denoising Diffusion Models

In Table 7, the hyperparameter settings for the DDIM base model training are presented.

Table 7: Hyperparameter settings for training of the DDIM base model.

Hyperparameter	Value
Learning Rate	$3 \cdot 10^{-4}$
Batch Size	20
Network Depth	5
Number of Features	16
Weight Decay	$1 \cdot 10^{-4}$
Epochs	60
Diffusion Steps	20
Embedding Dimensions	32
Embedding max Frequency	1000
Start Value (noise schedule)	0
End Value (noise schedule)	3
τ (noise schedule)	0.6
Min Signal Rate	0.01
Max Signal Rate	0.95

Hyperparameter settings for fine-tuning the base model on cell images with abnormalities can be seen in Table 8, apart from the number of epochs which varied with abnormality. The fine-tuning using images of hypersegmented cells was trained for 64 epochs, the fine-tuning using images with Döhle bodies was trained for 40 epochs and the fine-tuning using images of hypergranular cells was trained for 74 epochs.

Table 8: Hyperparameter settings for fine-tuning the base model on cell images with abnormalities.

Hyperparameter	Value
Learning Rate	$3 \cdot 10^{-4}$
Batch Size	20
Network Depth	5
Number of Features	16
Weight Decay	$1 \cdot 10^{-4}$
Diffusion Steps	20
Embedding Dimensions	32
Embedding max Frequency	1000
Start Value (noise schedule)	0
End Value (noise schedule)	3
τ (noise schedule)	0.6
Min Signal Rate	0.008
Max Signal Rate	0.95

Examples of Generated Images

Figure 21 shows nine examples of images that have been generated with a model trained on images of hypersegmented cells. In Figure 22, nine examples of generated images is shown. In this case, the generative model has been trained on images of cells with Döhle bodies. Figure 23 shows nine examples generated images where the model has been trained on images of hypergranulated cells.

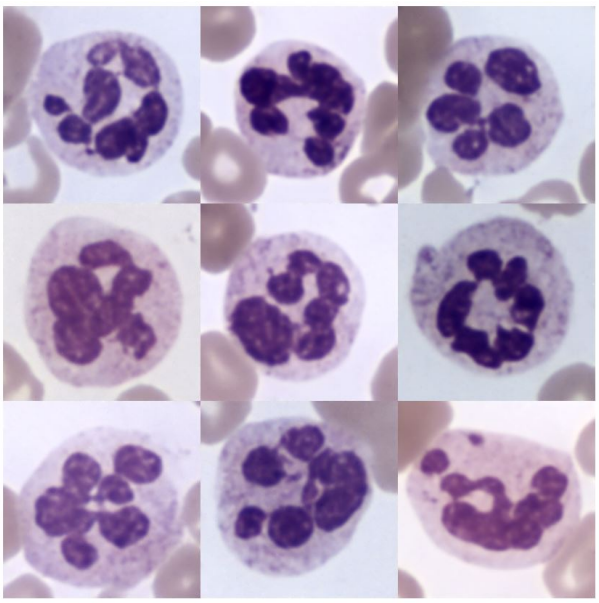


Figure 21: Examples of generated cell images. The generative model has been trained on images of cells that are classified as hypersegmented.

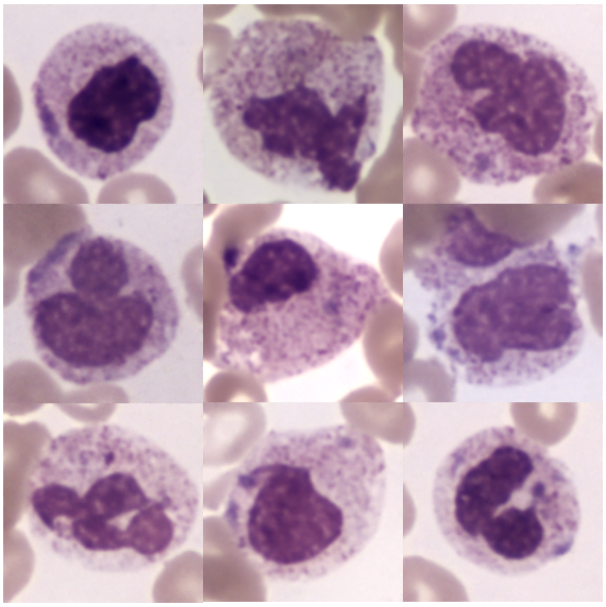


Figure 22: Examples of generated cell images. The generative model has been trained on images of cells with Döhle bodies.

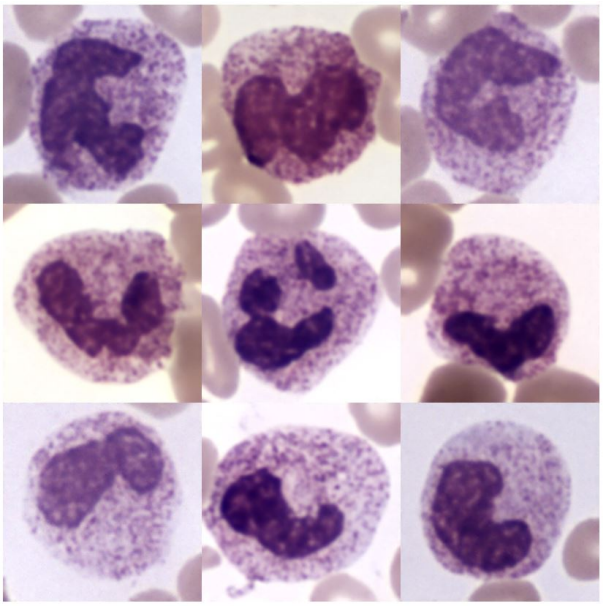
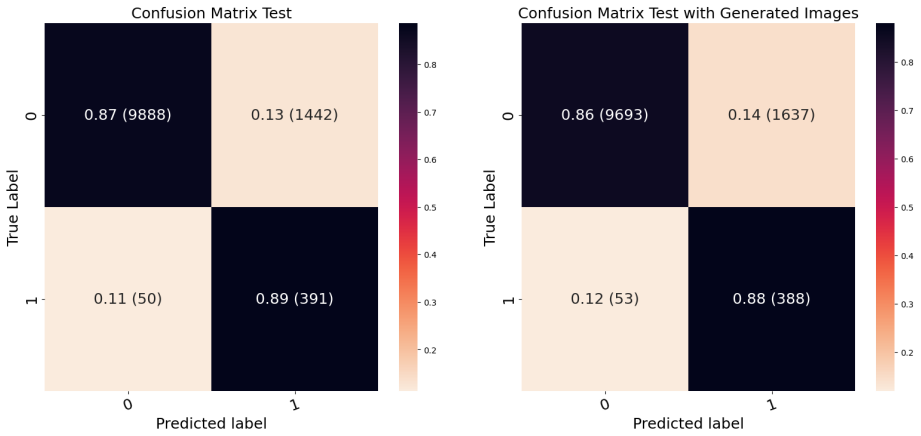


Figure 23: Examples of generated cell images. The generative model has been trained on images of cells that are classified as hypergranulated.

Classifier Net Test Results

The classifiers for all three abnormalities were trained using sparse categorical cross-entropy loss. The model weights from the epoch with the lowest validation loss were used. Label 0 represents the cells that have been classified by an expert to not have the abnormality that the network was trained to classify and label 1 represents the ones that have been classified as abnormal.

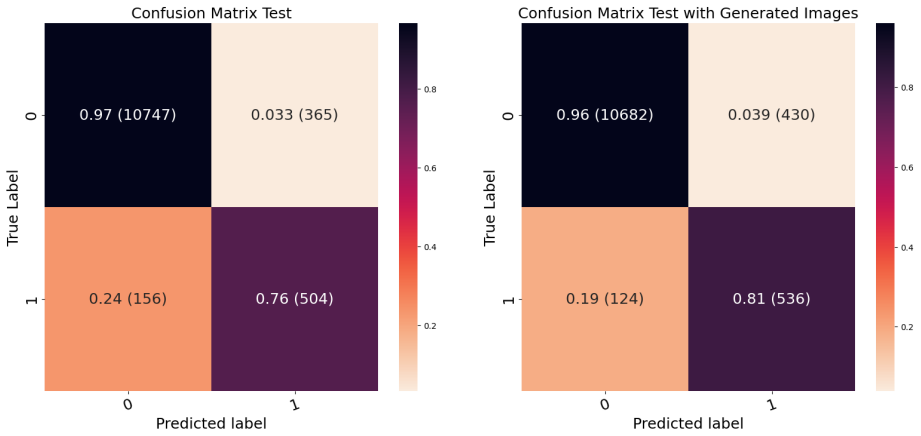
In Figure 24, the result of running the classifier net for hypersegmentation on the test set is shown. The net was trained for 150 epochs and with learning rate $5 \cdot 10^{-6}$. Figure 24a shows the classifier result when no generated images had been added to the training data and Figure 24b shows the result for when generated images have been added to the training set according to Table 4. The numbers in parenthesis in the boxes show how many of the cell images with the specified true label that have the correct predicted label. The first number in each box is which percentage of the total number of cells with the true label that have the correct predicted label.



(a) No generated cell images have been added to the training data.

(b) Generated cell images have been added to the training data.

Figure 24: The results of the classification of hypersegmented cells (True Label = 1) and cells that are not hypersegmented (True Label = 0).

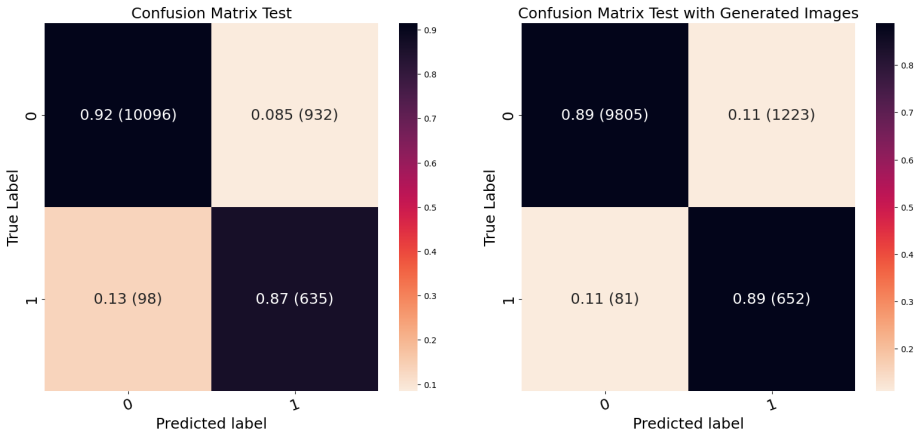


(a) No generated cell images have been added to the training data.

(b) Generated cell images have been added to the training data.

Figure 25: The results of the classification of cells with Döhle bodies (True Label = 1) and cells without Döhle bodies (True Label = 0) for the test set.

In Figure 25 the result of running the classifier net for Döhle bodies on the test set is shown. Figure 25a shows the classifier result when no generated images had been added to the training data and Figure 25b shows the result for when generated images have been added to the training set according to Table 4. Both models were trained for 150 epochs with a learning rate of $2 \cdot 10^{-6}$.



(a) No generated cell images have been added to the training data.

(b) Generated cell images have been added to the training data.

Figure 26: The results of the classification of hypergranulated cells (True Label = 1) and cells that are not hypergranulated (True Label = 0) for the test set.

In Figure 26 the result of running the classifier net for hypergranulation on the test set is shown. Figure 26a shows the classifier result when no generated images had been added to the training data and Figure 26b shows the result for when generated images have been added to the training set according to Table 4. Both models were trained for 150 epochs with a learning rate of $5 \cdot 10^{-6}$.

Medical Expert Test Results

The results of the medical experts test can be seen in Table 9. In total, 64 out of the 100 cell images were correctly classified. The results separated by abnormalities are visible in Table 10, Table 11 and Table 12.

Table 9: Confusion matrix for the medical expert's classification of real images and images generated with the diffusion model.

		Predicted		
		Real	Synthetic	Total
Actual	Real	48	2	50
	Synthetic	34	16	50
	Total	82	18	100

Table 10: Confusion matrix for the medical expert's classification of real images with WBCs classified as hypersegmented and images generated with the diffusion model fine-tuned on images with WBCs classified as hypersegmented.

		Predicted		
		Real	Synthetic	Total
Actual	Real	19	1	20
	Synthetic	20	0	20
	Total	39	1	40

Table 11: Confusion matrix for the medical expert's classification of real images of WBCs with Döhle bodies and images generated with the diffusion model fine-tuned on images with WBCs with Döhle bodies.

		Predicted		
		Real	Synthetic	Total
Actual	Real	14	1	15
	Synthetic	10	5	15
	Total	24	6	30

Table 12: Confusion matrix for the medical expert's classification of real images with WBCs classified as hypergranular and images generated with the diffusion model fine-tuned on images with WBCs classified as hypergranular.

		Predicted		
		Real	Synthetic	Total
Actual	Real	15	0	15
	Synthetic	4	11	15
	Total	19	11	30

Table 13: Confusion matrix for the experienced CellaVision employee's classification of real images and images generated with the diffusion model.

		Predicted		
		Real	Synthetic	Total
Actual	Real	22	28	50
	Synthetic	38	12	50
	Total	60	40	100

The results of the test done by the experienced CellaVision employee can be seen in Table 13. In this case, 34 out of the 100 cell images were correctly classified. The metrics calculated on the results can be seen in Table 14.

Table 14: Metrics of the classification tests.

Test type	Metric	Value
Medical expert: hypersegmented	Accuracy	0.48
	Recall	0.95
	Specificity	0
Medical expert: Döhle bodies	Accuracy	0.63
	Recall	0.93
	Specificity	0.33
Medical expert: hypergranulated	Accuracy	0.87
	Recall	1
	Specificity	0.73
Medical expert: all abnormalities	Accuracy	0.64
	Recall	0.96
	Specificity	0.32
CellaVision employee: all abnormalities	Accuracy	0.34
	Recall	0.44
	Specificity	0.24

Fréchet Inception Distance

The FID calculated using 10000 real images of hypergranular cells and 10000 images generated with the GAN trained on images of hypergranular cells was computed to be 386.5. The FID calculated using 10000 real images of hypergranular cells and 10000 images generated with the DDIM trained on images of hypergranular cells was computed to be 353.0.

6 Discussion

One of the problems that Klang and Carlberg [2] reported with their model was that it generated images with checkerboard artefacts. This is a problem that has proven to be quite common and according to Odena et al. [36], this can be improved by first making sure the kernel size is divisible by the stride. The kernel size in Klang and Carlberg's model was set to $5 \cdot 5$ with a stride of 2, so the kernel size was instead set to $6 \cdot 6$ with the stride kept as 2. Another solution that Odena et al. presented was changing the transposed convolutional layers in the generator to an upsampling layer with nearest neighbour interpolation followed by a convolutional layer. When applying these changes to the GAN, there was a difference for some epochs, but the artefacts are still noticeable. There are more changes that could be done to possibly improve the GAN, such as using different loss functions. However, optimizing the GAN was not the main focus of this thesis and therefore we did not spend time on this.

The images generated by the GAN were not tested as the ones generated by the diffusion model, by evaluation by a medical expert and by training the classifier net. This was a decision based on a visual evaluation of the generated images. As seen in Section 5, one can distinguish the images generated by the GAN from the images generated by the DDIM. Firstly, the overall image quality is better in the DDIM generated images. They are less blurry and do not show any gridlike texture in the background, which the GAN generated images do. Secondly, the images generated with the GAN all look very similar, especially in color. The images generated with the diffusion model are more diverse. Finally, the abnormalities are more distinct in the DDIM generated images. Furthermore, the FID calculated on images generated with the DDIM is lower than the FID calculated on images generated with the GAN. This indicates that the DDIM images are to prefer. Since the main goal with the GAN was to generate images and compare these with the images generated with a diffusion model we elected not to proceed with further testing of the images generated by the GAN.

A classifier for each abnormality was trained to see if the DDIM generated images could improve the results. There was a slight decrease in the performance when training the classifier net for images of hypersegmented neutrophils and including generated images in the training dataset, compared to when the dataset only contained real images. The accuracy of the normal cells sunk from 0.87 to 0.86 and the accuracy of the abnormal cells sunk from 0.89 to 0.88. When training the classifier net using images with Döhle bodies, there was an increase in accuracy for the abnormal cell images when including generated images. The accuracy for cells with Döhle bodies was 0.76 when training without generated images and 0.81 when adding the generated images. However, the accuracy of the normal cell images sunk from 0.97 to 0.96 when adding generated images to the training dataset. Adding generated images to the training set and training a classifier net for hypergranular cells resulted in a slight increase in accuracy

for the abnormal cells, from 0.87 to 0.89. Just as for the hypersegmentation and Döhle bodies classification nets, there was a slight decrease in accuracy for normal cells, which was 0.92 before adding generated images and 0.89 afterwards.

The results of the classifier test are ambiguous. There was no obvious improvement when adding generated images to the training set, but also not a clear decrease in performance. For two of the nets, Döhle bodies and hypergranular, there was an increase in accuracy for abnormal cell images when adding generated images, but combined with the decrease in accuracy for normal cells it is uncertain whether adding images improves the overall performance of the classifier. A potential cause to why the generated images did not improve the classification, even though they look quite similar to the training dataset, is that the abnormalities in the generated images are not as prominent as for the real images. Another reason might be the difference in quality between the generated images and the real images, for example some of the generated images appear to be foggy and to have low contrast. The number of generated images to be added was chosen to be the number of real abnormal images in the training dataset. This is something that could be tweaked to reach a better performance. One could also try other types of networks and to tune the hyperparameters. Using a balanced validation set instead of oversampling may be helpful when it comes to choosing the best model. Lastly, one could attempt to generate normal cell images as as well and include these in the classifier training set. This could reduce the risk of the classifier learning to differentiate between real and generated images, instead of normal and abnormal.

Parts of the dataset used to train the classifier had already been used to train the generative models. It is custom to separate the dataset used to train a generative model from other datasets. Since there was a shortage of cell images with the type of abnormalities we were aiming to generate, an exception was made. None of the images in the test set had been used to train the generative model and hence the comparison between different results is still considered valid.

The classification networks were also trained, validated and tested for images which had been normalized to have the same background color. The results of the classifiers showed little difference between the classifier trained without generated images and the classifier trained with generated images. There were also small differences between the test results of the classifiers trained with and without normalized images. For this reason, these results were omitted from the report.

Apart from adding images to train the classifier, a medical expert and an experienced CellaVision employee performed the classification test. The CellaVision employee could not distinguish between the real images and synthetic images, as the accuracy and recall are less than 0.5 and the specificity is less than 0.25. The medical expert could not distinguish between real and fake images of hypersegmented neutrophils. The images with other abnormalities were easier to

separate real from synthetic, especially hypergranulated. Looking at the results for all abnormalities, we can see that the expert could classify the true positives, the real images, well, as the recall is 0.96. Examining the specificity shows that the medical professional only could correctly classify approximately a third of the generated images. Hence, the images generated by the DDIM can be considered credible.

As in most new technologies, the ethical perspective of using a new method is not crystal clear. Synthetic images can be used in a harmful way, for example to deceive and spread fake news. However, there are situations when the benefits outweigh. In the current case, the synthetic images can possibly improve the pre-classification and thereby save time and resources for the users.

6.1 Tested Methods

Below follows a brief description of some changes made to our model which were found not to work well for our problem.

In their report from 2022, Karras et al. [37] presented some changes they had included in their diffusion model which they found to improve the performance. Among these were stochastic sampling and second order correction. The motivation Karras et al. give of adding stochastic sampling is that the model is better at correcting errors that occur in previous sampling steps. The authors also note some of the possible negative effects of using stochastic sampling, for example oversaturation and loss of detail in the image. These effects can be somewhat resolved by tweaking the parameters. When using stochastic sampling in our network we found it to make the training slower without visual improvement of the image generation. The same effects occurred when adding second order correction, which the authors found to improve their model when using it to their sampling step. They found the method to give a good balance between the truncation error and the number of steps needed to produce an image. Since the two methods did not improve our performance they were not included in the final model. Working with the model further it would be interesting to revisit the methods and try them again, but with different combinations of hyperparameters and noise schedules in order to see if they could improve the performance.

In the beginning of the development of the DDIM, an image of a smaller size was found to be easier to generate. The training of the model was started with $256 \cdot 256$ images and it was found that our model was not able to generate quality images of this size. The results did not look like a white blood cell, but more a blurry version of the red blood cells in the background. This led us to think that the model may be more successful generating images without that much background included. The training data was therefore cut out to $192 \cdot 192$ images, which meant there was less background included and more focus on the white blood cell. The results show that the concept of smaller images was successful.

Therefore, one idea was to downsample the training set and generate smaller images. Next, the plan was to enhance the resolution with a super resolution network. However, this was only tested briefly as we found that we were able to generate an image of size $192 \cdot 192$ with good quality.

When training the model, several different types of noise schedules were tested. The schedules tested were a linear schedule, a cosine schedule, a quadratic schedule and lastly the sigmoid schedule. Here it was found that a quadratic and cosine schedule worked well for the problem at hand, but a sigmoid schedule produced the best results. The network architecture and hyperparameters of the diffusion model were also explored. Different types of network, e.g., with or without residual and recurrent connections and with different placements of attention gates, were evaluated. The learning rate was varied and learning rate schedules were evaluated. However, the optimal learning rate was found to be constant. Besides the optimal optimizer, which was found to be AdamW, two other optimizers were tested: Adam and SGD. The network architecture found to be optimal is described in Section 4.3.

6.2 Further Work

The medical expert test was performed simply by letting the professional assess whether the cell images were real or generated. One possible improvement would be to ask the medical expert to determine whether generated images contain cells with abnormalities and to what degree. As we trained the classifier, we assumed that all images generated by a model trained on images of cells with abnormalities also depicted abnormal cells.

When we fine-tuned our generative models, we decided to only train on images of cells with a grade 2, or severe, abnormality. The reason was that it would be easier to notice when the generative model had been fine-tuned enough. Cells with a grade 1 abnormality have a discrete abnormality, sometimes hardly visible to a non-expert. When training the classifier all abnormal cells were merged into one class with both cells with a slight abnormality (grade 1) and cells with a severe abnormality (grade 2). Since the generative models were fine-tuned only on cells classified to be grade 2, it would be interesting to attempt to generate images of grade 1 as well. Next, one could train a classifier with output categories: normal, grade 1 and grade 2, and add generated images to the two latter.

When comparing the images that were generated with the diffusion model to the real images, there was a noticeable difference in contrast between them, where the contrast was clearer for the real images. In order the lessen the difference, the contrast for the generated images was sharpened. This was done according to

$$\text{image} = \text{image} \cdot (\text{contrast}/127 + 1) - \text{contrast},$$

where contrast was chosen to be 20. The positive effects of this were more vi-

brant coloring as well as clearer outlines for the cells and nucleus. This made the majority of the generated cell images more similar to the real images, but it also resulted in some generated cell images getting more vibrant than the real images. This is mostly clear when comparing the background between the real and generated images. Examples can be seen in Figure 22 and 23 where the center image has a very white background, something that is not seen in real images. This is something that possibly could be improved further by trying different values for the contrasts as well as finding a good balance with brightness and other sorts of filters.

Denoising diffusion models are relatively new, so improvements and new versions are presented almost daily. New methods were presented during the writing of this thesis and used for our model. An example of this is the noise schedule presented by Chen [31]. Improvements that could improve the performance of our diffusion model might therefore still be in development. Working with the model further it could be a good idea to try out new methods.

6.3 Conclusions

In conclusion, it is possible to implement a Denoising Diffusion Implicit Model that can generate images of WBCs with abnormalities such that a medical expert cannot distinguish them from real cell images. The medical expert only classified 32% of the generated images as synthetic.

The classifier net trained to classify cells with Döhle bodies and the classifier net trained to classify hypergranular cells had an increase for the accuracy of the abnormal cell images when adding generated images. However, the accuracy decreased for the classifier net trained to classify hypersegmented cells when adding generated images. Because of the ambiguous results, we are unable to draw the conclusion that inclusion of generated images in the training dataset can improve the classification. One way to possibly improve the performance of the classification with added generated images is to devote more time and evolve the diffusion model to obtain higher quality images of cells with more distinct abnormalities.

Furthermore, visually comparing the generated images presented in Section 5, one can draw the conclusion that images generated with the DDIM outperforms images generated with the GAN. The images generated with the DDIM have higher quality as well as greater variety.

References

[1] Song J., Meng C., Ermon S. *Denoising Diffusion Implicit Models*. arXiv (2022).

[2] Klang O., Carlberg M. *Blood Cell Data Augmentation using Deep Learning Methods*. Lund University (2020).

[3] Pinaya W.H.L., Tudosiu P-D., Dafflon J, Da Costa P.F., Fernandez V, Nachev P., Ourselin S., Cardoso M.J. *Brain Imaging Generation with Latent Diffusion Models*. arXiv (2022).

[4] Dean L. Blood Groups and Red Cell Antigens. National Center for Biotechnology Information (US), Bethesda, MD, (2005).

[5] Gotwals J., Karlin R., Gersten T. *What Are White Blood Cells?*. `https://www.urmc.rochester.edu/encyclopedia/content.aspx?ContentID=35&ContentTypeID=160` (read 2023-04-18).

[6] *CellWiki: Hypersegmentation*. `https://www.cellwiki.net/en/aberrations/\neutrophils-hypersegmentation` (read 2023-04-18).

[7] *CellWiki: Döhle bodies*. `https://www.cellwiki.net/en/aberrations/neutro-\phils-dohle-bodies` (read 2023-04-18).

[8] *CellWiki: Hypergranulation*. `https://www.cellwiki.net/en/aberrations/\neutrophils-hypergranulation` (read 2023-04-18).

[9] Goodfellow I., Bengio Y., Courville A. *Deep Learning*. MIT Press. (2016).

[10] He K., Zhang X., Ren S., Sun J. *Deep Residual Learning for Image Recognition*. arXiv (2015).

[11] O'Shea K., Nash R. *An Introduction to Convolutional Neural Networks*. arXiv (2015).

[12] Eger S., Youssef P., Gurevych I. *Is it Time to Swish? Comparing Deep Learning Activation Functions Across NLP tasks*. arXiv (2019).

[13] Priya B. *Softmax Activation Function: Everything You Need to Know*. `https://www.pinecone.io/learn/softmax-activation/` (read 2023-04-17).

[14] Godoy D. *Understanding binary cross-entropy / log loss: a visual explanation*. `https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23` (read 2023-04-12).

[15] Koech K.E. *Cross-Entropy Loss Function*. `https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e` (read 2023-04-17).

[16] Parmar R. *Common Loss functions in machine learning*. https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23 (read 2023-04-12).

[17] Kingma D.P., Ba J.L. *Adam: A Method for Stochastic Optimization*. arXiv (2017).

[18] Loshchilov I., Hutter F. *Decoupled Weight Decay Regularization*. arXiv (2019).

[19] Ioffe S, Szegedy C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv (2015).

[20] Goodfellow I.J., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y. *Generative Adversarial Networks*. arXiv (2014).

[21] Radhakrishnan P. *What is Transfer Learning?*. https://towardsdatascience.com/what-is-transfer-learning-8b1a0fa42b4 (read 2023-03-31).

[22] Mo S., Cho M., Shin J. *Freeze the Discriminator: a Simple Baseline for Fine-Tuning GANs*. arXiv (2020).

[23] Saeed M. *A Gentle Introduction to Positional Encoding in Transformer Models, Part 1*. https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/ (read 2023-03-31).

[24] Oktay O., Schlemper J., Le Folgoc L., Lee M., Heinrich M., Misawa K., Mori K., McDonagh S., Hammerla N.Y., Kainz B., Glocker B., Rueckert D. *Attention U-Net: Learning Where to Look for the Pancreas*, arXiv (2018).

[25] Ronneberger O., Fischer P., Brox T. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv (2015).

[26] Alom M.Z., Hasan M., Yakopcic C., Taha T.M., Asari V.K. *Recurrent Residual Convolutional Neural Network based on U-Net (R2U-Net) for Medical Image Segmentation*. arXiv (2018).

[27] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., Polosukhin I. *Attention Is All You Need*, arXiv (2017).

[28] R. Serfozo. *Basics of Applied Stochastic Processes, Probability and its Applications*. Springer-Verlag Berlin Heidelberg (2009).

[29] Yang L., Zhang Z., Song Y., Hong S., Xu R., Zhao Y., Shao Y, Zhang W., Cui B., Yang M. *Diffusion Models: A Comprehensive Survey of Methods and Applications*. arXiv (2022).

[30] Ho J., Jain A., Abbeel P. *Denoising Diffusion Probabilistic Models*. arXiv (2020).

[31] Chen T. *On the Importance of Noise Scheduling for Diffusion Models*, arXiv (2023).

[32] Raj R. *Evaluation Metrics to Evaluate Classification Models*. `https://www.enjoyalgorithms.com/blog/evaluation-metrics-classification-models` (2019). (read 2023-03-31).

[33] Heusel M., Ramsauer H., Unterthiner T., Nessler B. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium* arXiv (2018)

[34] Sandler M., Howard A., Zhu M., Zhmoginov A., Chen L. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. arXiv (2018).

[35] Deng J., Dong W., Socher R., Li L., Li K., Fei-Fei L. *ImageNet: A large-scale hierarchical image database* (2009).

[36] Odena A., Dumoulin V., Olah C. *Deconvolution and Checkerboard Artifacts*. `https://distill.pub/2016/deconv-checkerboard/` (2016) (read 2023-04-20).

[37] Karras T., Aittala M., Aila T., Laine S. *Elucidating the Design Space of Diffusion-Based Generative Models*. arXiv (2022).