Master's thesis

Cell Image Transformation Using Deep Learning

Jesper Jönsson and Emmy Sjöstrand

May 29, 2018

Supervisors:
Adam Morell, CellaVision AB
Kent Stråhlen, CellaVision AB
Niels Christian Overgaard, Lunds Tekniska Högskola

LUND
UNIVERSITY

Abstract

This thesis was written at CellaVision who sells digital microscope systems, mainly used for blood analysis. Blood tests are an important part of modern health care and today digital microscopes are widely used to replace conventional microscopy. It is important that the digital images of blood cells are of high quality and that they look as they would in a traditional microscope. CellaVision has digital microscope systems with different optics, which means images from different systems do not look the same.

In this thesis we investigate the possibility of transforming images between systems using neural networks. The main focus is on generative adversarial networks, also known as GANs, but we also experiment with a simple CNN and a network with a perceptual loss based on the VGG-16 network. Our results include two variations of GANs, a conditional GAN and a cyclic GAN. An advantage of the cyclic GAN is that it can be used in an unsupervised setting. It does however require a lot more memory compared to the conditional GAN. We present results from four different network setups. With these methods we have attained very good results that are better than previous tries at CellaVision. The networks are however too slow to be implemented in the actual systems today.

Acknowledgements

We would like to thank our supervisor Niels Christian Overgaard at the Department of Mathematics for constructive feedback, especially during the writing process. We would also like to thank our supervisors Kent Stråhlen and Adam Morell at CellaVision who have helped us with everyday matters as well as giving us valuable comments on this report. A special thanks to Martin Almers at CellaVision who has served us with a constant flow of ideas and provided us with tools and data needed to complete this project. Finally we would like to thank all the personnel at CellaVision for making us feel welcome and as part of the group from day one.

Contents

1 Introduction

1.1 Motivation

Blood tests are an important part of modern health care and are widely used to screen for different diseases and confirm diagnoses. A peripheral blood smear is blood that has been smeared onto a microscope slide. The blood sample is stained in order to make the blood cells more visible in a microscope. Examining blood smears is part of the procedure when investigating different hematological diseases. The traditional way to do this is to use manual microscopy, which is time consuming and labour intensive. CellaVision sells analyzers and supporting software that replaces conventional microscopy to improve the efficiency and quality of blood analysis. Their customers consist of large and mid-size laboratories and hospitals.

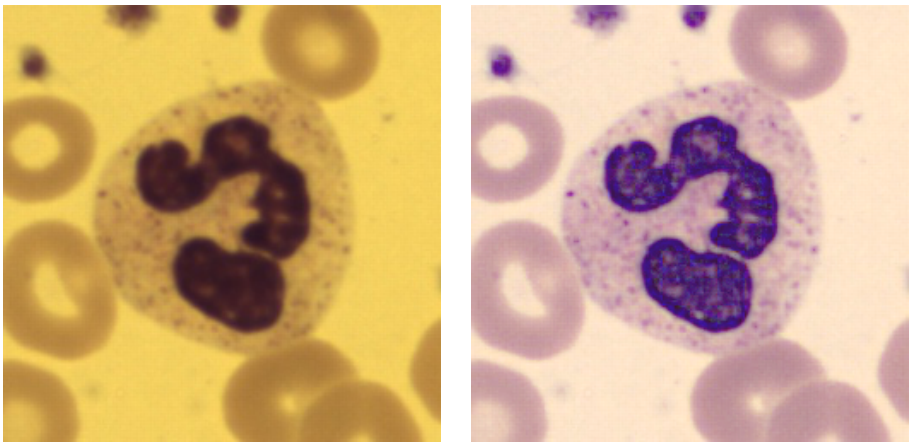
CellaVision's best selling system is an integrated system called DI-60 and it is the result of an OEM partnership with Sysmex. CellaVision has two other systems called DM9600 and DM1200. Both DM9600 and DM1200 are part of the third generation of systems developed by CellaVision. The loading capacity of DM9600 is 96 microscope slides and the loading capacity of DM1200 is 12 microscope slides. A popular product from the previous generation is the DM96 system which is the predecessor of the DM9600 system. The DM96 is no longer for sale. Figure 1 shows the DM9600, the DM1200 and the DM96. CellaVision's main analysis is a differential count of white blood cells. There are many types of white blood cells. A differential count finds a number of white blood cells and gives the proportions of each cell type. If a differential count deviates from the normal distribution that could be an indication of e.g. an inflammation or a bacterial infection. CellaVision also counts immature cells that are not present in a healthy person's blood. Immature cells detected in the blood could be an indicator of more serious conditions such as leukemia and lymphoma. The result of the analysis, which includes images of white blood cells, is presented to the user on a screen. Images from a system that have not been processed in any way are called raw images. When the images are presented to the user they should look like traditional microscope images. The process that makes the images look more similar to how they would look in traditional microscope is called normalization. See Figure 2 for a comparison between a raw image and a normalized image. The normalization algorithm segments the background and sets it to a constant predefined color. The dynamic of the image is also changed to make it look pleasing, for example by enhancing the contrast in the image, especially in the nucleus of the cell.

One major difference between the third generation systems and the second generation systems is the optics and the illumination. The second generation systems use a halogen lamp and the third generation use an LED. Because of the different optics and illumination, raw images taken with systems from different generations do not look the same. Many customers reacted to this change since they were used to the previous generation. Figure 3 shows a raw image from the third generation and a raw image from the second generation. Since all the normalized images should look the same it is obvious that images from different systems cannot be normalized the same way. Ideally one would like to use the same normalization

process for all images. This can be done if it is possible to somehow transform raw images from one generation to look like they came from the other. Otherwise there needs to be different normalizations for systems from different generations.

(a) CellaVision DM9600. (b) CellaVision DM1200. (c) CellaVision DM96.

Figure 1: Digital microscopy systems from CellaVision.

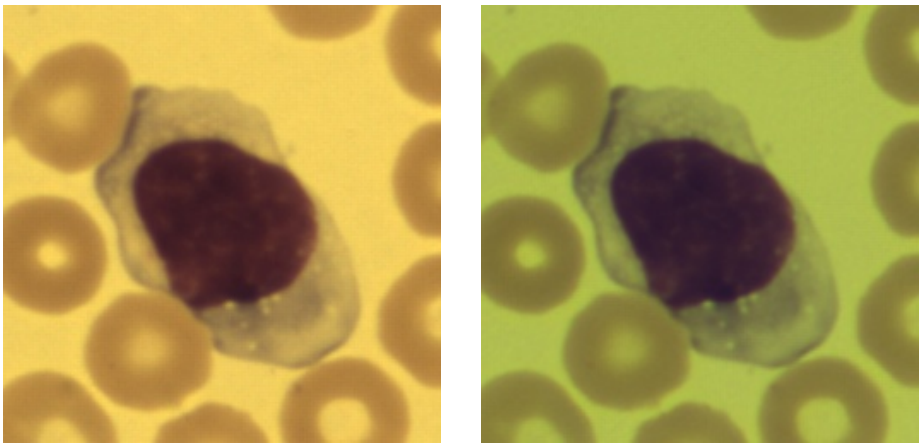


(a) Raw image. (b) Normalized image.

Figure 2: Comparison of a raw image and a normalized image.

CellaVision's third generation systems implement two options for normalizing images, a transformation to the second generation combined with a second generation normalization, as well as a direct normalization of the raw third generation images. It is up to the customer to choose their preferred method. The problem with the direct normalization is that customers that have been using earlier generations feel that the images look very different compared to before. The other option is the transformation from third generation images to second generation images. Unfortunately this transformation is not very simple and a global transformation cannot solve the problem. Today an algorithm called Hedlund-Morell (HM) normalization is used to transform the third generation images to second generation

images and normalize them. The main idea of HM normalization is to segment the image and apply different transformations to different parts. This works fairly well, but the result is not entirely satisfactory. This is where deep learning comes in. The hope is that some kind of neural network that is complex enough will be able to capture the need for different transformations of different cell components without having to do any segmentation. In this thesis a transformation of images from the third generation to the second generation will be discussed. The DM96 system has been used to collect images from generation two and the DM1200 system has been used to collect images from generation three.



(a) Second generation. (b) Third generation.

Figure 3: Raw images taken with systems with different optics and illumination.

The classification of white blood cells that was mentioned earlier is done with artificial neural networks. Those networks were originally trained using images from DM96 and the networks had to be retrained in order to work for DM1200 images. That is something one wants to avoid since training of networks requires data labeled by experts and since the training is time consuming. If the transformation from DM1200 images to DM96 images is good enough the classification networks might work for the transformed DM1200 images as well. This is something that could be useful when developing systems in the future.

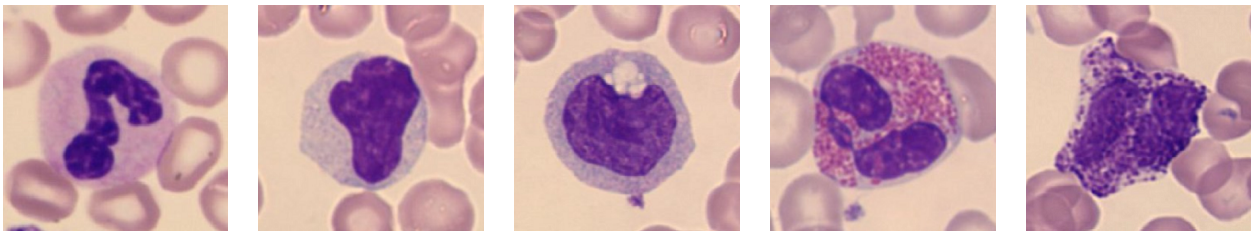
1.2 Aim

The aim of this thesis is to develop a pre-processing step in order to make raw images from different systems look the same: a transformation from the domain X (DM1200 images) to the domain Y (DM96 images). This pre-processing step will be a neural network. The main focus is to transform the DM1200 images to be visually indistinguishable from the DM96 images, but if possible we also want the classification networks to classify the transformed images correctly.

2 Background

2.1 Blood Cells

CellaVision's main analysis is the differential blood count as mentioned in Section 1.1. Blood consists of 60 % plasma and 40 % is made up of three types of blood cells, white blood cells (WBCs), red blood cells (RBCs) and platelets (PLTs) [6]. This thesis will focus on transforming images of white blood cells. White blood cells can be divided into five main categories, neutrophils, lymphocytes, monocytes, eosinophils and basophils, which play different roles in the immune system. Approximately $50 - 80$ % of all WBCs are neutrophils [31] while basophils only consitute < 1 % of the total amount of WBCs [30]. Figure 4 shows examples of white blood cells. Eosinophils have a color in the granules that does not appear as much in the other cell types and has proved to be one of the more challenging colors to get right. The granules are the small particles present in the cytoplasm around the nucleus.



(a) Neutrophil (b) Lymphocyte (c) Monocyte (d) Eosinophil (e) Basophil

Figure 4: Five different white blood cells [3].

2.2 Hedlund-Morell Normalization

The pixels of an image of a white blood cell can be divided into four areas, background, red blood cells, nucleus and cytoplasm. In the HM normalization each class has a separate pre-determined transformation. Furthermore, every pixel is given a probability of belonging to the different classes. For every pixel all transformations are carried out and the result is weighted against the probability of the pixel belonging to that class. This has the effect that even if the classification is not completely correct the transitions in the image are smooth.

Images from the target system as well as original images are needed in order to find the four transformations. All images are segmented into the four classes (background, red blood cells, nucleus and cytoplasm). This is done by first finding the white blood cell, i.e. the nucleus and the cytoplasm. This cell can be cut out from the image and then the background and the red blood cells are easily separated by some thresholding method. Segmenting the white blood cell itself into nucleus and cytoplasm is harder, but there exist good methods to do this. Examples of such segmentation methods are described in [18]. When the segmentation is done a three dimensional Gaussian approximation in the RGB space is calculated for each class. This is done for target images and the original images separately, so there are two Gaussian approximations for each class. Then an affine transformation from every class in

the original images to the corresponding class in the target images is estimated. These are the transformations used when normalizing the images.

2.3 Artificial Neural Networks

One can think of an artificial neural network as a function of two variables, a matrix of weights $\mathbf{w}$ and a matrix of input $\mathbf{x}$. When using a neural network for image transformations the input $\mathbf{x}$ will be an image. A loss function $\mathcal{L}$ is defined to capture how good the output from the network is. To improve the performance of the network some optimization algorithm is used to update the weights of the network based on the loss function.

Neural networks are, as the name suggests, loosely inspired by the structure of the biological neural networks in our brains. The simplest version of a neural net is the perceptron which is a model of a biological neuron. It was introduced by Frank Rosenblatt in 1958 [34]. The definition of the perceptron is as follows

$$f(x) = \begin{cases} 1 & \text{if} \quad w^T x + b > 0 \\ 0 & \text{otherwise,} \end{cases}$$

where w is the weights, x is the input and b is a bias term. Figure 5 shows an illustration of the perceptron. Since it is a linear classifier it can only learn to separate sets that are in fact linearly separable, i.e. the data can be separated by a hyperplane. A famous example that is not linearly separable is the exclusive-or (XOR) function. To produce an XOR function one can create a multi-layer perceptron by adding more layers and introduce non-linear activation functions between the layers. This way the classifier can learn much more complex patterns.

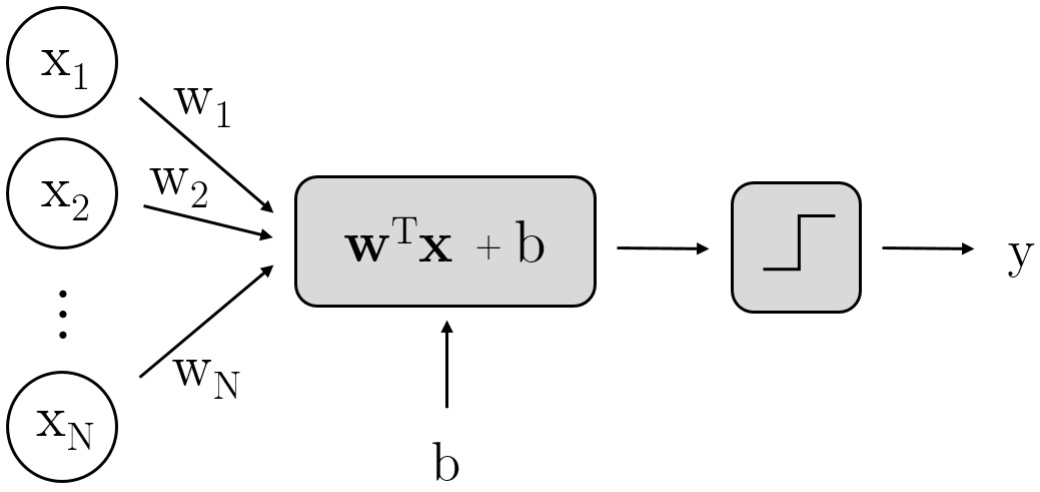


Figure 5: Illustration of the perceptron. The input $\mathbf{x}$ is multiplied by the weights $\mathbf{w}$ and then the bias is added before it is sent through a step function (see (6)) to compute the end result y.

2.3.1 Loss Functions

The training of neural networks is a data driven process and the driving force is the loss function. Therefore it is important to have a good loss function in order for the network to learn the right things. The network can be seen as a function f. If x is the input, y is the target and w is the weights of the network, then the minimization problem can be formulated as

$$\min_{w} \mathcal{L}\big(y, f(x, w)\big), \tag{1}$$

where $\mathcal{L}$ is some loss function.

Different loss functions are used for different learning tasks. The cross entropy loss function is often a very good choice for binary classification tasks [28]. If $\hat{y}_i$ is the output from some network and d_i is its label (either 0 or 1) defining what class the sample belongs to, the cross entropy error function is defined as

$$\mathcal{L} = -\frac{1}{N}\sum_{i}^{N}\Big(d_i \log(\hat{y}_i) + (1 - d_i)\log(1 - \hat{y}_i)\Big). \tag{2}$$

By studying (2) we can immediately verify how the choice of error function makes sense. If a sample has target $d_i = 1$, the second term cancels. The network will then minimize its loss by outputting a value as close to 1 as possible. Similarly, if a sample has target $d_i = 0$ the first term cancels and the network will minimize its loss by outputting a value as close to 0 as possible. Therefore the network is always encouraged to match its target value.

When transforming images it might be useful to introduce an l^1- or l^2-loss between the transformed image and the target image (if a target image exists). This loss would just be the norm of the pixel-wise difference between the transformed image $\hat{y}$ and the target image y,

$$\mathcal{L}_{l^1} := \|\hat{y} - y\|_1. \tag{3}$$

2.3.2 Optimization Methods

A traditional way of optimizing a function is to use gradient descent. This method updates the weights of the network by studying the gradient of the loss function for the entire batch of training data and moving in the direction of the steepest descent, see (4). The hyper-parameter α is the learning rate which controls how much the parameters in the network are updated each iteration. It is possible to have a constant learning rate but it can also decrease over time as the network gets better.

$$\mathbf{w} \longleftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L} \tag{4}$$

This type of optimization algorithm where the whole training set is used is called a deterministic gradient method. If one has a lot of data this probably is not the best method since it requires a lot of memory. It is also not very efficient, for many datasets it can take hours to do just one iteration [12]. To get around this problem one can use a so called mini-batch

of data instead. A mini-batch consists of m randomly drawn samples, x_i, from the training set. The gradient used for the weight update is the average of all gradients in the mini-batch, (5) shows the update rule. This is called Stochastic Gradient Descent (SGD) and is the most commonly used optimization method in machine learning today [9].

$$\begin{aligned} \hat{g} &= \frac{1}{m} \sum_{i}^{m} \nabla_w \mathcal{L}(f(x_i, w)) \\ w &\longleftarrow w - \alpha \hat{g} \end{aligned} \tag{5}$$

The neural networks in this thesis have been trained using the optimization method Adam [21]. Adam is an efficient method for stochastic optimization that requires little memory and often gives good results for most problems without too much tuning of the hyperparameters. Adam is a good choice of optimizer when one has a lot of data or large networks. It has been shown empirically that Adam works better compared to other popular optimization methods when training convolutional neural networks [21]. Adam combines the advantages from AdaGrad [7], which is able to handle sparse gradients well, and RMSProp [42] that deals with non-stationary objectives in a good way [21].

The gradient of the loss function with respect to w is usually calculated with *backpropagation*. First a forward pass is performed. This simply means that the input is sent through the network. Next the loss is calculated and then backpropagated through the network to update the weights. The name backpropagation originates from the fact that the loss is propagated backwards through the network [32].

2.3.3 Activation Functions

Non-linear activation functions are used in all kinds of neural networks, not just different types of perceptrons. A simple activation function is just a hard threshold, a step function (6). However the gradient of this is zero almost everywhere which is not good for the learning [11]. To avoid this problem one can use softer thresholds like the sigmoid function (7), which is useful when the network should output a probability since it squashes $\mathbb{R}$ into $(0, 1)$. The most commonly used activation function today is the ReLU (Rectified Linear Unit) (8). The ReLU looks a lot like the linear function but is actually non-linear, and in fact any function can be approximated by combining ReLUs [35]. The main advantage using ReLU is that the output from the activation function becomes sparse if the input is normalized (mean 0 and unit variance) since approximately 50 % of the input will be less than zero. Sparse output means it is a lighter and faster network [35]. Unfortunately this can cause problems too. Since everything less than zero will not get activated it can cause the gradient to vanish. Then there will be no weight update during gradient descent and the network can get stuck. This is often called the dying ReLU problem [20]. Today there exist several variations of the ReLU that try to get around this problem, one of the most well-known is the Leaky ReLU (9). The idea is to make the gradient non-zero so that the network can recover during training. This is done by replacing the flat part for $x < 0$ with a small slope, αx [20].

Table 1: Examples of activation functions.

$$\text{Step function:} \quad f(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

$$\text{Sigmoid:} \quad f(x) = \frac{1}{1+e^{-x}} \tag{7}$$

$$\text{ReLU:} \quad f(x) = \begin{cases} x & \text{if} \quad x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$\text{Leaky ReLU:} \quad f(x) = \begin{cases} x & \text{if} \quad x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{9}$$

2.3.4 Convolutions and Transposed Convolutions

Today there are a lot of different types of neural networks being used in a wide array of applications. Convolutional neural networks (CNNs) are networks where the main operation is a convolution. This kind of network has proven to be very successful when dealing with images, for example in image classification [19]. The convolution operation consists of a kernel sliding over the input image performing matrix multiplications at every location. The stride controls how the kernel slides, if the stride is 1 then the kernel moves one pixel at a time, if the stride is 2 then it moves 2 pixels etc. The result of the convolution is an image containing the features defined by the kernel. Figure 6 shows an example of a kernel that detects edges in an image. In a neural network the kernel is not predefined, instead the networks learns the parameters during training.

-1	-1	-1
-1	8	-1
-1	-1	-1

(a) Original image (b) Convolution kernel (c) Resulting image

Figure 6: Example of edge detection using convolution [39].

Image generation using neural networks often involves upsampling. Many interpolation methods are fixed and do not have any learnable parameters. Sometimes it might be desirable for the network to learn an optimal upsampling procedure. Then the transposed convolution is a good choice. Transposed convolutions are sometimes called deconvolutions, which might be confusing since it is not actually the mathematical definition of a deconvolution.

Transposed convolutions are especially useful in neural networks with an encoder-decoder setup (see Section 3.6) since upsampling and convolution is combined. For more details see [36].

2.3.5 Generalization

An important term in machine learning is generalization. Generalization is a term used to measure the performance of a trained network on data which has not been part of the training [32]. Data sets are often split into three subsets, a *training set*, a *test set* and a *validation set*. The training set is the largest one and it contains the data used for training. The test set is a data set used to measure how well the model performs on previously unseen data. The test set needs to be completely unbiased since it is used to measure the final performance. The validation set is somewhere in between the other two sets. It is not used to train a model, but can still be used for evaluation during the training process. For example it could be used to tune hyperparameters. The fact that it is used to evaluate performance during the training means it is no longer unbiased even though the model has not actually been trained on the validation set.

Many machine learning algorithms, including neural networks, have a tendency to overfit to training data. An example of the concept is shown in Figure 7. The left model is too flexible and will fit data too well, even noise. The right model on the other hand does not fit as well to the training data but much better to the test data. This model has better generalization performance.

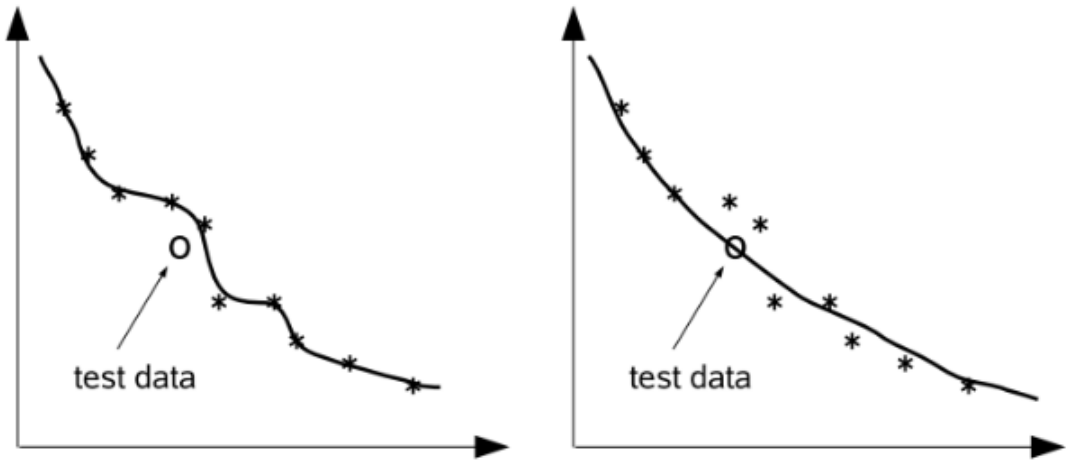


Figure 7: An example of overfitting. The left model fits better to the training data but does not generalize as well. The right model on the other hand captures the characteristics of the training data and fits well to previously unseen data [32].

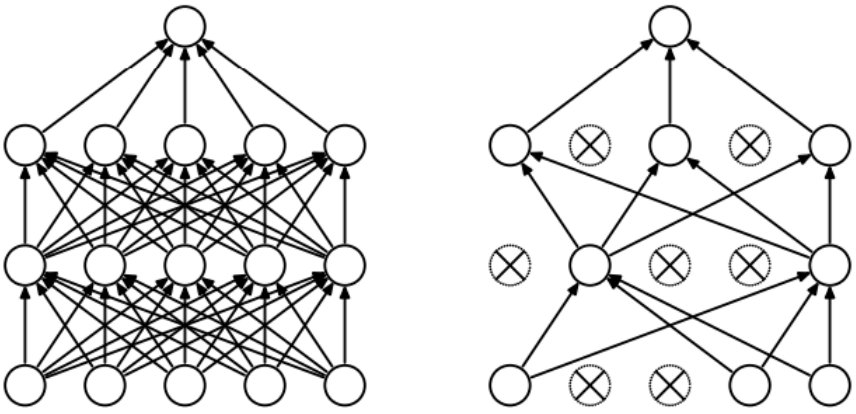
Our aim is of course to do well on test data rather than training data. Therefore ways of improving the generalization performance are needed and there are a lot of them to choose

from. Methods designed to improve generalization are usually called *regularization methods*. There are less subtle ones such as early stopping, meaning one stops the training when the generalization performance deteriorates. A more sophisticated method is dropout, further explained in the following section.

2.3.6 Dropout

A simple idea that many times seems to improve the generalization performance is combining the output of multiple networks into one single prediction. For example one can train a number of different networks and then take the average of all of them to generate a single prediction. These machines are sometimes called committees [2]. However, large networks are slow to train and very source demanding. One can show that the best performance of committees is obtained when the models are different from each other [32]. To achieve this the networks need to be trained on different sets of data or use different architectures. Finding optimal hyperparameters for each network and the resources needed to train all of the networks make it difficult to use committees [41].

Dropout is a simple idea that tries to mitigate the problems which arise when using committees. It builds upon the idea of training multiple networks with different architectures. Only one network is used but during training random nodes and their connections are dropped. For each batch different nodes are dropped. Consider Figure 8; the left graph shows a regular neural network with two hidden layers and in the right graph dropout is applied. Some nodes have been randomly chosen and temporarily dropped. No information will be sent or processed in these nodes. When training on the next batch of images new nodes will be randomly chosen and dropped. Dropout serves as a method of removing co-adaptation between different neurons, forcing each node to learn independently of the other nodes in the layer. Dropout greatly improves the generalization performance [40].



(a) Standard Neural Net (b) After applying dropout.

Figure 8: Illustration of dropout. The left graph illustrates a normal neural network with two hidden layers. In the right graph dropout is applied to each layer. Some nodes and their connections, the crossed ones, have been temporarily dropped when training the network on this batch [41].

2.3.7 Batch Normalization

A common problem in deep neural networks is either vanishing or exploding gradients. One way to deal with this problem is to normalize the input of the network [13]. Another is to make normalization part of the training. This was first introduced in 2015 by Sergey Ioffe and Christian Szegedy [15]. The method is called batch normalization and has shown great performance, not only dealing with the problem of the gradients but also as a regularization method [15].

It has been known for a long time that some pre-processing of the input to the network can speed up convergence substantially. One alternative is to whiten the input, i.e. make sure the input to the network has zero mean, unit variance and is decorrelated [22]. Batch normalization generalizes this concept. It changes the mean and variance of the input to any activation function in the network, but it does not remove correlation between input.

When the input to a neural network, or any other learning system, changes it is said to experience a covariate shift [37]. Ioffe and Szegedy define internal covariate shift as the change in the distribution of the network activations due to the change in network parameters during training [15]. Consider a layer in the network. During training the parameters continuously change as a result of the backpropagation algorithm. For every new batch of training data the input to this layer will change its distribution slightly. This is what is called an internal covariate shift. This makes the training of the network difficult since each time the distribution is changed, the layers of the network need to adapt to the new distribution [15]. Batch normalization tries to reduce the internal covariate shift. It does so by fixing the mean and variance of the inputs to each activation function.

Consider a mini-batch training session and a layer with d-dimensional input where batch normalization should be applied. Let $\mathbf{x} = (x_1, \ldots, x_d)$ be the input vector. The normalization is applied to each feature. The input is normalized as

$$\hat{x}_k = \frac{x_k - \mathbb{E}[x_k]}{\sqrt{\mathrm{Var}[x_k]}},$$

where the expectation and variation are computed over the training data set. This takes care of the normalization of the features. It does however have a major drawback, it limits what the network is able to learn. For example if this normalization is applied before a sigmoid activation the inputs are constrained to be in the almost linear part of the function. This is why Ioffe and Szegedy make sure that the batch normalization can also represent the identity transformation. This way the network can recover the original signal if that is the optimal thing to do. To be able to do this the new learnable parameters γ and β are introduced, which will be part of the training of the network. The transformation will now be

$$y_k = \gamma_k \hat{x}_k + \beta_k.$$

By letting $\beta_k = \mathbb{E}[x_k]$ and $\gamma_k = \sqrt{\mathrm{Var}[x_k]}$ the original signal will be recovered. Batch normalization has many good qualities. As a result of the reduced internal covariate shift

the learning rate can be increased to speed up training. Another advantage is that one does not need to be as careful with the initialization of parameters [15].

2.3.8 Perceptual Losses

Losses like the l^1-loss, which are based on the difference between the output image of the network and the target image, have a huge drawback since they do not capture perceptual differences between the two images [17]. Imagine a target image that has a black and white zebra pattern. Every other column is black and the rest is white. If a network generates an image that has a black and white zebra pattern that is shifted one column, then the generated image will receive the highest possible loss, even though it has actually captured the stylistic features of the target image.

One way to define a *perceptual loss function* which captures the essentials of the image is to make use of a pre-trained classification network. Losses of this kind have been used in multiple applications. For example [43] use a perceptual loss to de-rain images, meaning they try to remove visual signs of rain or snow in an image. In [17] Johnson et al. use this kind of loss to combine the content of one image with the style of another. One example of this would be to combine the content of a photograph with the style of a painting. In [24] a perceptual loss is used to train networks which take an image as input and outputs the same image but with higher resolution. It is a type of loss that has been proven to give good results when the goal is to create high quality images. There are a couple of different variants of this loss. One thing these losses all have in common is that they use the output of some intermediate layer of a pre-trained classification network. One such network is the VGG network [38]. The VGG network won the 2014 edition of the ImageNet challenge [14] and is pre-trained on the ImageNet dataset which consists of 14 million images split into 1000 classes.

Figure 9 shows a simple illustration of how this could work. The generated image $\hat{y}$ and the target image y are sent through the VGG network and a loss based on the output of some intermediate layer ϕ_j of the network is defined. The core idea of a perceptual loss is to represent the differences between the generated image and the target image with high level features. The network is then encouraged to generate images with similar feature representation as the target image [43]. These kind of losses is believed to capture stylistic similarities between images.

The perceptual loss can be defined in many ways. If ϕ_j is the output from the j:th layer of VGG and $(\hat{y}, y)$ is a generated image and a target image, then a common way to define the perceptual loss function is to simply take some norm of the difference of the two outputs,

$$\mathcal{L} = \|\phi_j(\hat{y}) - \phi_j(y)\|. \tag{10}$$

In [17] Johnson et al. define one loss called the feature reconstruction loss of the same structure as (10) and one loss called the style reconstruction loss. Once again, let ϕ_j be the output of the j:th layer of the pre-trained VGG network. If ϕ_j is a three dimensional matrix with size $c_j \times h_j \times w_j$ then ϕ_j can be reconstructed as a $c_j \times h_j w_j$ matrix called ψ_j. A matrix

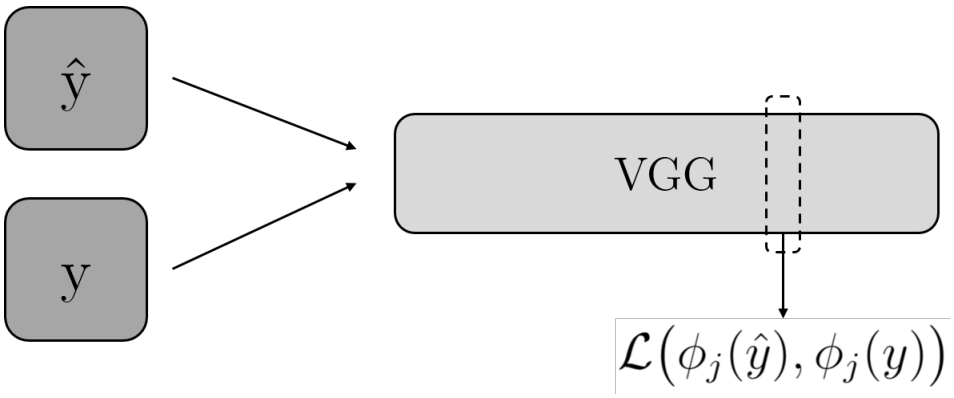


Figure 9: Illustration of the idea of having a perceptual loss function based on a pre-trained network, in this case VGG.

G_j^ϕ is defined as $G_j^\phi = \psi_j \psi_j^T/(c_j h_j w_j)$, which is a $c_j \times c_j$ dimensional matrix (sometimes referred to as the Gram matrix). The feature reconstruction loss is then defined as

$$\mathcal{L}_{feat}(\hat{y}, y) = \|G_j^\phi(\hat{y}) - G_j^\phi(y)\|_F^2, \tag{11}$$

where $\|A\|_F = (tr(A^T A))^{\frac{1}{2}}$ is the Frobenius norm of the matrix A. This loss captures stylistic features of the image but does not preserve its spatial structure [17].

2.4 Generative Adversarial Networks (GANs)

In the previous section a loss function based on a pre-trained classification network was defined. When Goodfellow et al. presented their article "Generative Adversarial Nets" [8] in 2014 they took this idea one step further. Two networks are defined, one network called the *generator* and one network called the *discriminator*. The discriminator will, just as the VGG-16 network did before, serve as a loss function for the generator but it will now be part of the training, in the sense of getting its weights updated. The name originates from the fact that the networks will compete against each other during training.

In order to understand the idea of a GAN better we will start with a simple example before getting to the technical details. This is a common example that appear in many places, for example [8], but this is our interpretation. Imagine a situation where a thief tries to manufacture money and the police tries to determine whether this money is fake or not. If the police detects the fake money, then the thief will have to do better next time to be able to fool the police. On the other hand, if the police thinks the money is real, then the thief has done a good job. In a GAN setting the thief is the generator. The thief tries to generate fake money that looks like real money. The generator's (thief's) goal is to fool the police, which is the discriminator in a GAN. The discriminator (police) is trained to distinguish between real and fake money, it gives a probability of a sample being real. The generator (thief) sees this probability and is able to improve. As the generator (thief) becomes better at generating fake money the discriminator (police) needs to improve as well. This training process forces both the generator and the discriminator to become better and better, competing against

each other.

A generator is a *generative model* and a discriminator is a *discriminative model*. A generative model takes a training set of samples from some distribution p_{data} (the real money in the example above) and tries to learn an estimate of that distribution, called p_{model} (the fake money) [10]. Sometimes the generative model learns the distribution explicitly, but sometimes it will only be able to sample from the learned distribution. The discriminative model learns to represent some posterior probability $p(y|x)$ (probability of a sample being real money) [2]. The generative model takes noise as input and generates a sample. The reason for giving the generator noise as input is that we get a higher variation in what the generator is able to produce. Without noise the generator could only produce a deterministic result. The discriminator takes a sample as input, either the output from the generative model or a sample from p_{data}, and outputs a probability of the sample coming from p_{data}. In Figure 10 an illustration of the framework is shown. From now on the generator is called G and the discriminator is called D.

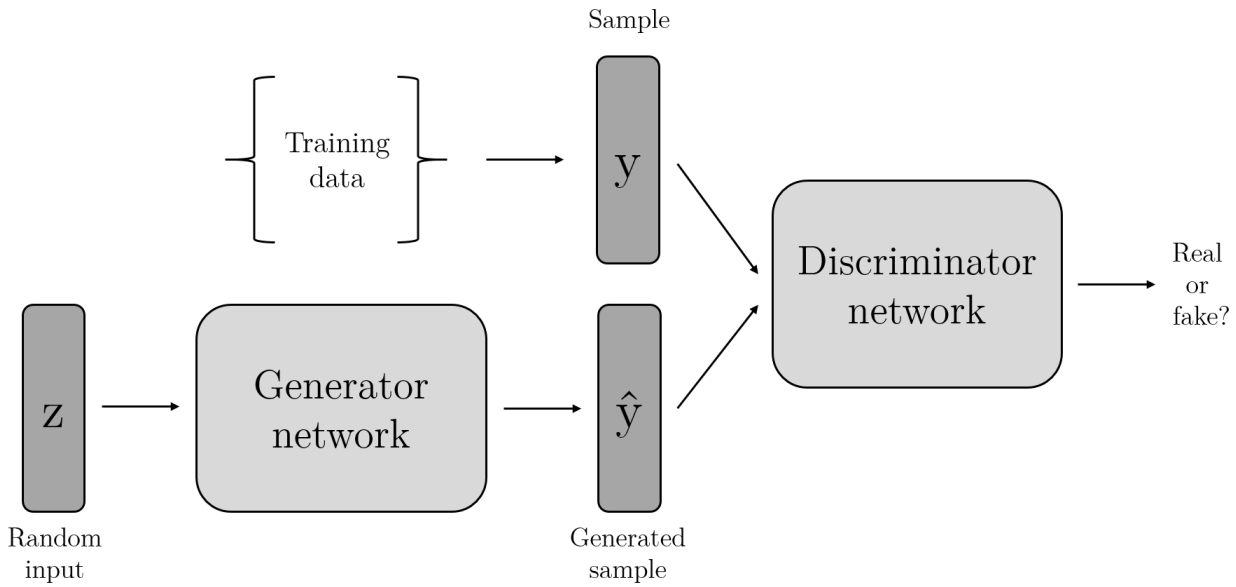


Figure 10: Illustration of the GAN framework. It consists of two networks, the generator and the discriminator. The input to the generator is a sample from some noise prior. The output of the generator is passed along to the discriminator. The discriminator is trained to separate fake samples from real samples and the generator is trained to fool the discriminator.

For GANs the loss functions of the networks will not only depend on its own parameters. The discriminator will try to minimize $\mathcal{L}_{disc}(\theta_G, \theta_D)$, but can only do so by manipulating its own weights, θ_D. Similarly, the generator will try to minimize $\mathcal{L}_{gen}(\theta_G, \theta_D)$, but can only do so by manipulating its own weights, θ_G. This means training the networks will be more like a game than a traditional optimization problem. The solution will be a Nash equilib-

rium, (θ_G, θ_D) which is a local minimum of $\mathcal{L}_{disc}(\theta_G, \theta_D)$ given θ_D and a local minimum of $\mathcal{L}_{gen}(\theta_G, \theta_D)$ given θ_G [10].

The discriminator is designed and trained to solve a binary classification problem. Samples y coming from the real distribution gets the label one and samples $\hat{y} = G(z)$ coming from the fake distribution gets the label zero. Consider having N such samples from each class, then by using the cross entropy error function, see (2), the discriminator loss can be constructed as

$$\mathcal{L}_{disc} = -\frac{1}{2N}\sum_{i}^{N} \log\big(D(y_i)\big) - \frac{1}{2N}\sum_{i}^{N} \log\big(1 - D(\hat{y}_i)\big). \tag{12}$$

By similar arguments as before (see (2) in Section 2.3.1) we realize this formulation encourages the discriminator to output a probability close to 1 when samples are coming from the real distribution and a probability close to 0 when samples are outputs of the generator.

It is also possible to formulate the loss function based on a more statistical approach, see (13), which is the formulation usually presented in GAN papers.

$$\mathcal{L}_{disc} = -\frac{1}{2}\mathbb{E}_{y\sim p_{data}}\big[\log D(y)\big] - \frac{1}{2}\mathbb{E}_{z\sim p_z}\Big[\log\Big(1 - D\big(G(z)\big)\Big)\Big] \tag{13}$$

When thinking of the generator's loss function the easiest formulation would be

$$\mathcal{L}_{gen} = -\mathcal{L}_{disc}. \tag{14}$$

This means the generator is trained to maximize the discriminator's loss function, i.e to make the discriminator's job as difficult as possible. This has the nice property of being a zero-sum game, $\mathcal{L}_{gen} + \mathcal{L}_{disc} = 0$. By defining a value function as $V(\theta_g, \theta_d) = -2\mathcal{L}_{disc}$ the entire problem can be formulated as the minimax game shown below [10].

$$\min_{\theta_G}\max_{\theta_D} V(\theta_g, \theta_d) = \min_{\theta_G}\max_{\theta_D}\left(\mathbb{E}_{y\sim p_{data}}\big[\log D(y)\big] + \mathbb{E}_{z\sim p_z}\Big[\log\Big(1 - D\big(G(z)\big)\Big)\Big]\right) \tag{15}$$

Training a classifier by minimizing the cross entropy error function is highly effective. This is because the cost function always provides a strong gradient when the network classifies the samples to the wrong class. On the other hand when the network does provide the correct output the loss function will saturate. This is positive when training the discriminator, but the generator will be completely without gradient if the discriminator becomes too good. Therefore, if the generator falls behind in the training process it will not be able to catch up again. The loss function defined in (14) has some nice theoretical properties but is usually not the one used when training the generator. The cross entropy error function will still be used but instead of flipping the sign of the discriminator loss when defining the generator loss, the target variables will be flipped [10]. The fake samples get target one and the real ones get target zero.

Let x be a vector with $2N$ elements, N samples from the real distribution and N samples

from the fake distribution, i.e. let $x = [y_1, \ldots, y_N, \hat{y}_1, \ldots, \hat{y}_N]$. Let $\hat{d}$ be the corresponding target vector with flipped targets, i.e. $\hat{d} = [0, \ldots, 0, 1, \ldots, 1]$. The cross entropy loss function becomes

$$\mathcal{L} = -\frac{1}{2N} \sum_{i}^{2N} \hat{d}_i \log\big(D(x_i)\big) + (1 - \hat{d}_i) \log\big(1 - D(x_i)\big). \tag{16}$$

The first term cancels when $i = 1, \ldots, N$, i.e. when the real samples are presented to the loss function and the second term cancels when $i = N + 1, \ldots, 2N$, i.e. when the generated samples are presented. The loss can be rewritten as

$$\mathcal{L} = -\frac{1}{2N} \sum_{i}^{N} \log\big(D(\hat{y}_i)\big) - \frac{1}{2N} \sum_{i}^{N} \log\big(1 - D(y_i)\big). \tag{17}$$

Since the second term does not depend on $\hat{y} = G(z)$ it can be omitted. The loss function that will be used for the generator is

$$\mathcal{L}_{gen} = -\frac{1}{N} \sum_{i}^{N} \log\big(D(\hat{y}_i)\big) \tag{18}$$

With this formulation both networks have strong gradients when falling behind in the game [10]. By choosing the generator's loss function according to (14) we encouraged the generator to make the life of the discriminator as difficult as possible, this implicitly means the generator should generate samples that could be from the real distribution. By choosing the loss function in accordance with (18) (flipping the labels) we explicitly train the generator to fool the discriminator. In pseudo code the training algorithm looks as follows.

Learning algorithm GAN [8]

---

for number of training iterations **do**

- Sample minibatch of m noise sample $z_1, \ldots, z_m$ from noise prior $p_z(z)$.
- Sample minibatch of m examples from $y_1, \ldots, y_m$ from data generating distribution $p_{data}(y)$.
- Update the discriminator by moving in the negative direction of the gradient

$$\nabla_{\theta_d} \mathcal{L}_{disc} = -\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \log D(y_i) + \log\Big(1 - D\big(G(z_i)\big)\Big).$$

- Sample minibatch of m noise sample $z_1, \ldots, z_m$ from noise prior $p_z(z)$.
- Update the generator by moving in the negative direction of the gradient

$$\nabla_{\theta_g} \mathcal{L}_{gen} = -\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\Big(D\big(G(z_i)\big)\Big).$$

end for

2.5 Conditional GANs (cGANs)

When using a GAN one does not have a lot of control over what is being generated, and that is why Mirza et al. introduced the conditional GAN (cGAN) [27] a few months after the original paper. A cGAN is very similar to the normal GAN, but the input to both the generator and the discriminator is conditioned on some additional information, for example a class label. In Figure 11 a GAN is trained on the MNIST dataset. The MNIST dataset consists of 70000 labeled grayscale images of handwritten digits and is commonly used as a benchmark dataset. It is available to everyone online [23]. The network has learned to generate what looks like handwritten digits, but there is no way to control what number the network generates. By using a cGAN it is possible to condition on what number the network should generate. Figure 12 shows the result of training a cGAN on the MNIST database and Figure 13 illustrates the cGAN framework.

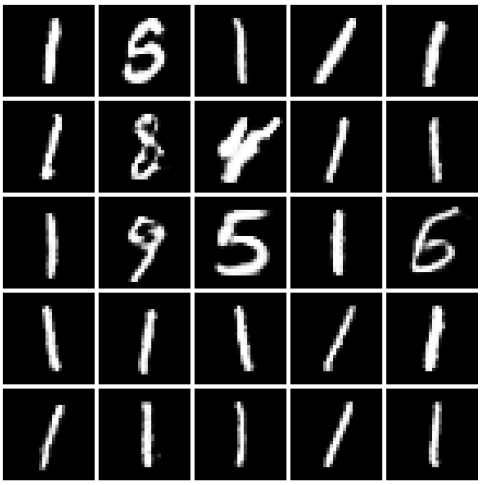


Figure 11: MNIST images generated using a GAN.

The training data is slightly different compared to before since every sample of the data is now paired with a label, x, describing what number it illustrates. The label is called a *conditioning label*. Input to the generator is some noise, just as before, and a conditioning label. The output of the generator is sent with the conditioning label to the discriminator, which outputs the probability of the sample being drawn from the real distribution. This simple change in the input to the networks makes it possible to control what number is being generated. If the generator generates the number one but the conditioning label was zero the discriminator will immediately classify it as fake, assuming the discriminator is good enough to realize the label and the image should be the same number. This way the generator is forced to generate the number the conditioning label is coding for. The minimax game can be formulated in almost the same way as for the GAN case (15), but now the conditional information x is added as well, see below.

$$\min_{\theta_G}\max_{\theta_D}\mathbb{E}_{\substack{y\sim p_y(y)\\x\sim p_x(x)}}[\log D(y|x)]+\mathbb{E}_{\substack{z\sim p_z(z)\\x\sim p_x(x)}}\Big[\log\Big(1-D\big(G(z|x)|x\big)\Big)\Big] \tag{19}$$

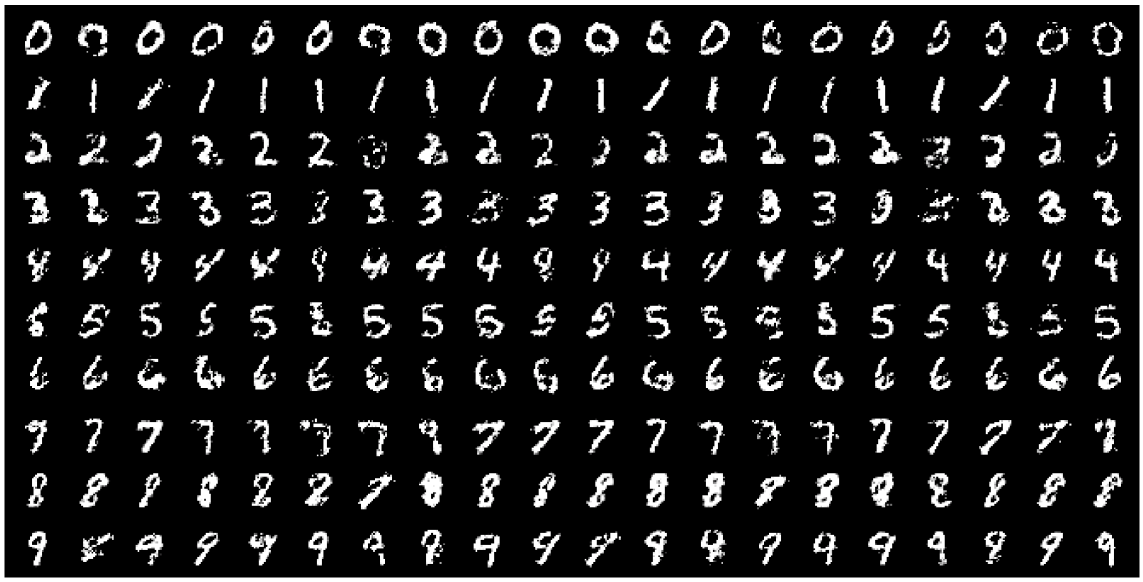


Figure 12: MNIST images generated using a conditional GAN. Every row is conditioned on a class label [27].

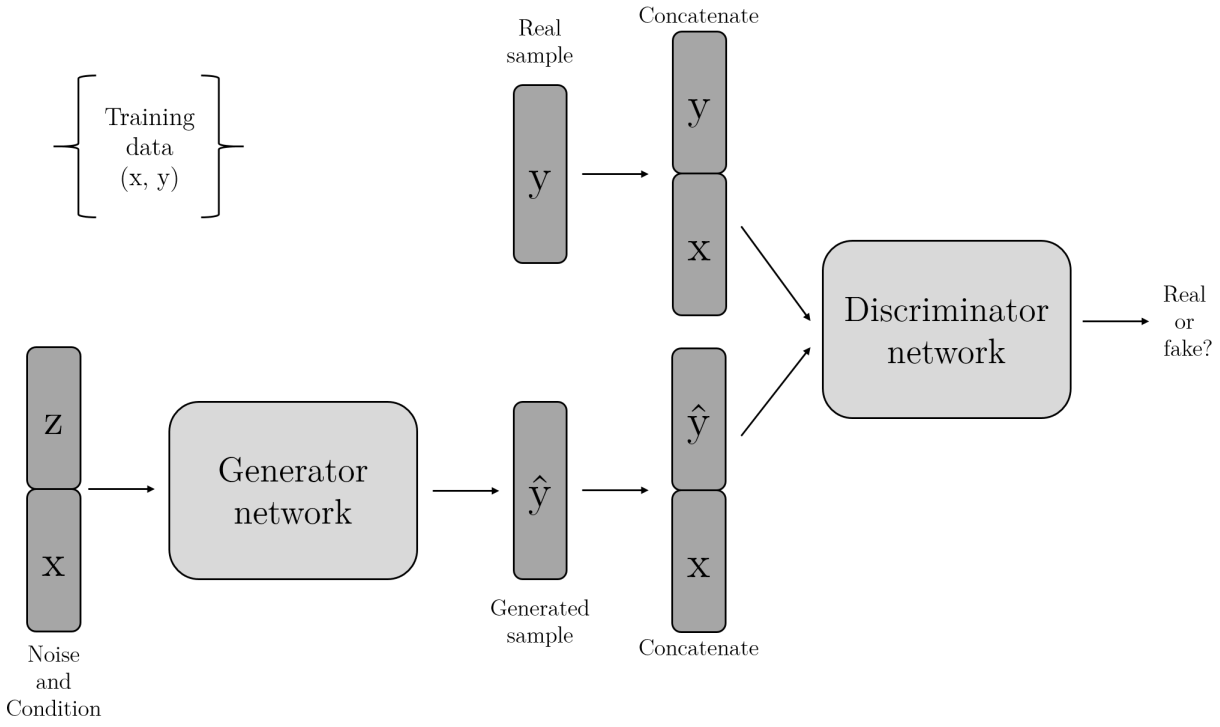


Figure 13: Illustration of the cGAN framework.

2.5.1 A Simple Example

To gain a better understanding of cGANs, show some of the typical challenges of GANs and to illustrate how the training is performed a simple example will be constructed. The goal is to approximate a one-dimensional zero mean Gaussian with unit variance using a cGAN. To simplify the training of the networks no noise will be sent as input to the generator, only the conditioning label. This is not how one usually would do it but it will suffice at this point to illustrate the points we are trying to make.

The architectures of the generator and discriminator networks are not relevant to the example and has been omitted. Figure 14 shows the probability density function which the generator is trying to approximate, $p_{data}(x)$, what the generator has predicted, $G(x)$, and the output from the discriminator when given fake data as input, $D(x, G(x))$. Since this is before any training has been performed the output from the generator and the discriminator is just nonsense. Throughout this example the discriminator plot will be the probability of the generator output being a sample from the true distribution.

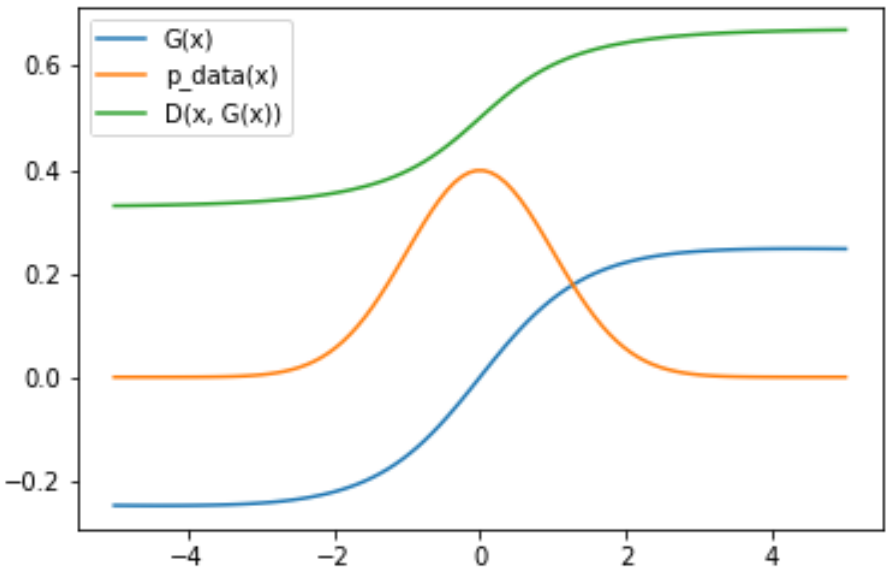


Figure 14: Output from the generator $G(x)$ and the discriminator $D(x, G(x))$ before any training has been performed, along with the probability density function, $p_{data}(x)$, which the generator is trying to approximate.

When training the discriminator a batch $x_1, \ldots, x_N$ is sampled uniformly along the x-axis and then the batch is sent to the generator. Let $\hat{y}_i = G(x_i),\ i = 1, \ldots, N$. This gives N pairs of "fake" data $(x_i, \hat{y}_i)$. Next, a batch from the true distribution is sampled which gives new pairs of data, $(x_i, p_{data}(x_i)),\ i = 1, \ldots, N$. The two sets of data points, the fake one and the real one, are presented to the discriminator along with their labels. The real ones get a

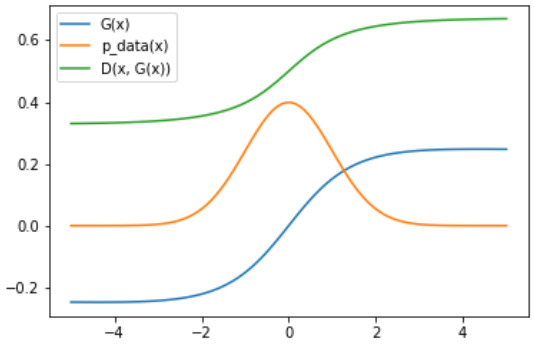
label equal to 1 and the fake ones get a label equal to 0. In accordance with (12) and (19) the discriminator loss will be

$$\mathcal{L}_{disc} = -\frac{1}{2N}\sum_{i}^{N} \log D\big(p_{data}(x_i)|x_i\big) - \frac{1}{2N}\sum_{i}^{N} \log\big(1 - D(\hat{y}_i|x_i)\big).$$

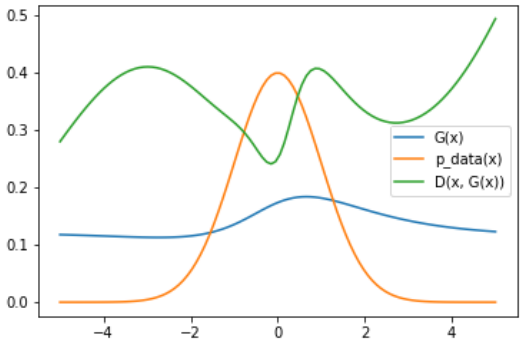
When training the generator a batch $x_1, x_2, \ldots, x_N$ is once again sampled uniformly along the x-axis. These points are sent through the generator to get $\hat{y}_i = G(x_i),\ i = 1, \ldots, N$. The data is paired $(x_i, \hat{y}_i)$ and sent to the discriminator, this time with a label equal to 1. The generator should be encouraged to get as good as it possibly can at fooling the discriminator. This way the generator will receive a lot of loss if the discriminator classifies the fake samples as fake, driving the generator to perform better and better as the training progresses.

In Figure 15 some results during training are shown. After 100 iterations nothing much has happened. After 200 iterations the generator fits a little better to the true distribution. By studying the discriminator a little closer one can see how the probability of $(0, G(0))$ being drawn from the true distribution is a little lower. This is good. As seen in the plot the generator's graph is far away from the true probability density function at this point. The discriminator really cannot say anything about the generator's graph in the tails of the graph, it only gives a probability of 0.5 meaning it does not know if it is a sample from the real distribution or not. After 10000 iterations the discriminator and the generator have become much better. By studying the discriminator it is seen that whenever the generator's graph is close to the true probability density function the discriminator rewards the generator with a high probability of the sample being from the true distribution. When the generator's graph deviates from the Gaussian the discriminator immediately gives the generated sample a low probability, punishing the generator with a higher loss. After 92000 iterations the generator fits well to the true distribution. The discriminator cannot separate the two distributions anymore and just randomly guesses which distribution the samples are coming from.

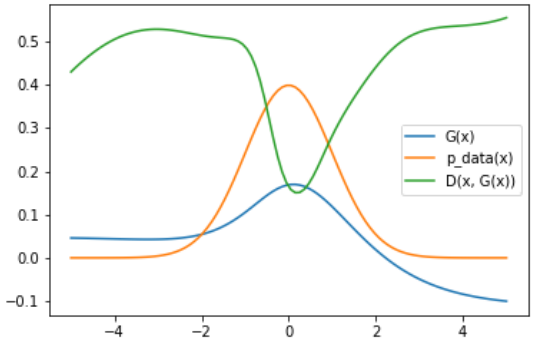
One of the drawbacks of GANs is that you cannot tell by just studying the loss function how well the training progresses. In Figure 16 it is shown how the generator's loss function varies during training. By solely studying the loss function we realize there is no way to decide if the results are good or not. The loss oscillates heavily throughout the training. This is to be expected. The loss is the result of the output of the discriminator which constantly changes depending on how well the training of itself and of the generator progresses. With this simple example it becomes clear that some other metric in which we can measure success is needed. It is simply not enough to study the loss coming from the discriminator. Another consequence of this is how the generator keeps receiving a high loss signal even though it is seen visually that it performs very well. As a result the generator will keep on backpropagating a high loss, changing its parameters. In Figure 17 the result after 120000 iterations is shown. The generator is now substantially worse. The discriminator on the other hand is fully aware of what may be samples from the true distribution or not. As a result the generator manages to recover, but this still illustrates an apparent drawback of having a loss function which only depends on a network that cannot separate the two distributions.



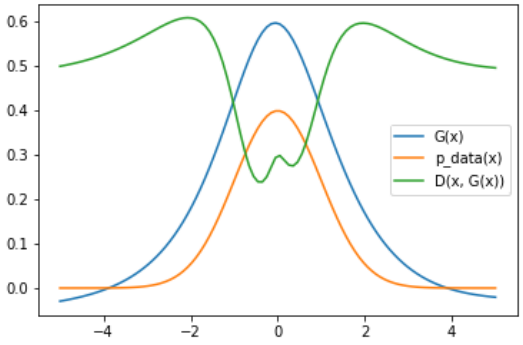
(a) After 0 iterations.



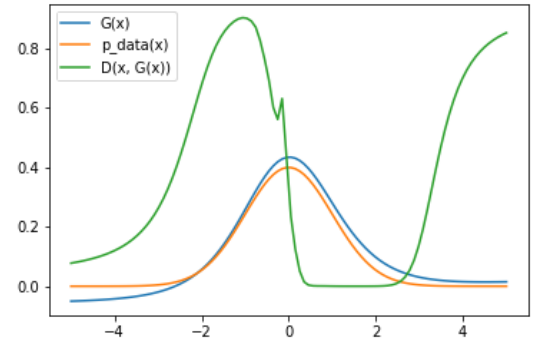
(b) After 100 iterations.



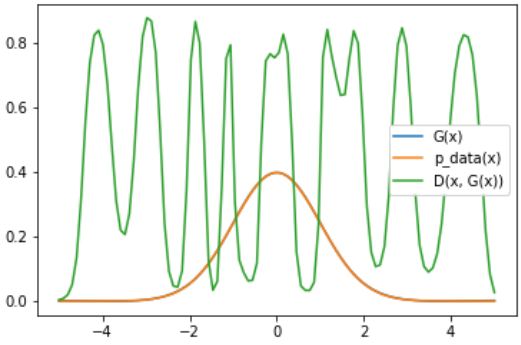
(c) After 200 iterations.



(d) After 400 iterations.



(e) After 10000 iterations.



(e) After 92000 iterations.

Figure 15: Illustration of how the training of the generator and discriminator progresses.

Another well recognized problem of the GAN is the balance between the discriminator and the generator. They need to improve at the same pace. The discriminator needs to be good enough to give the generator a reasonable loss signal, it needs to lead the generator towards the true distribution. If the discriminator is too good compared to the generator,

all of the samples from the generator will be classified as false. In this case the generator will be fumbling in the dark having no clue what it should aim for, resulting in nothing but nonsense.

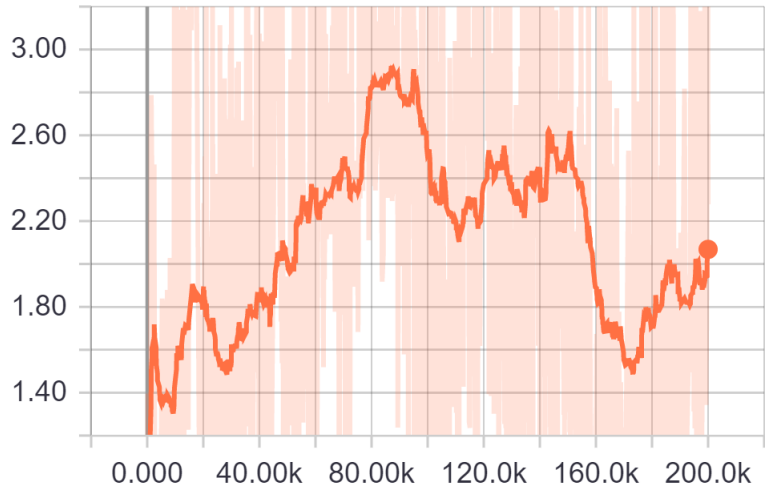


Figure 16: Plot of the generator's loss as a function of iterations.

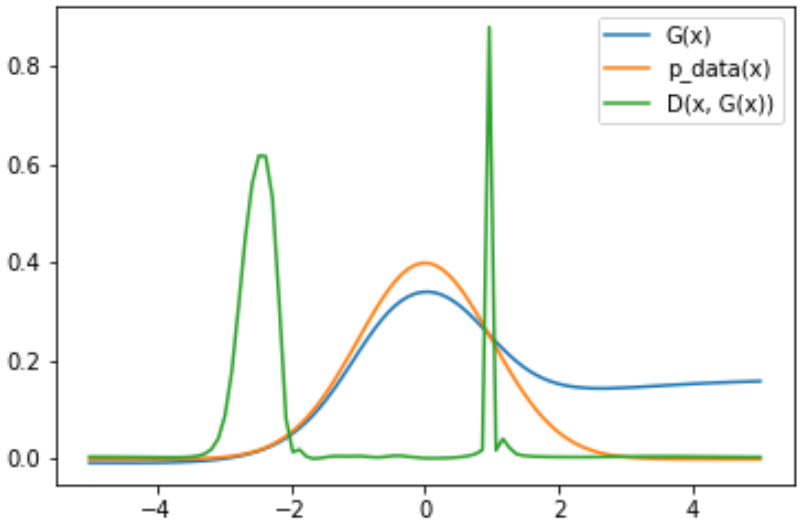
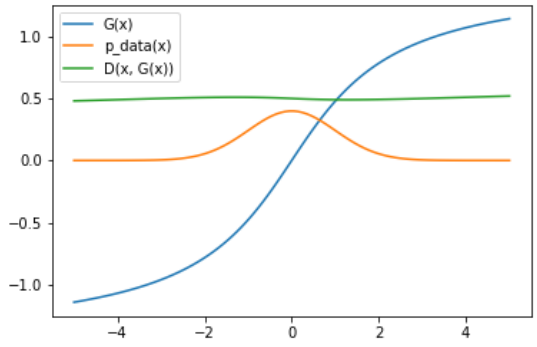


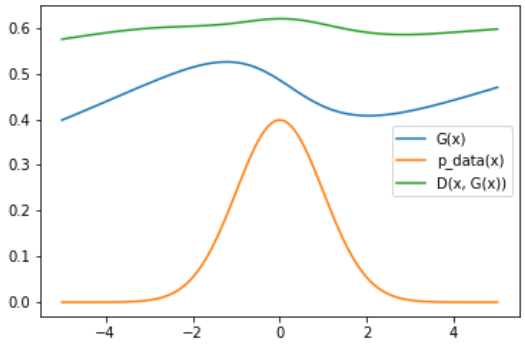
Figure 17: Result after 120000 iterations. The generator has become much worse as a result of the constant high loss from the discriminator.

Figure 18 shows the same example as before but this time the discriminator only sees half as many samples as it did before, resulting in slower training of the network. The discriminator never learns to separate the fake from real samples. This way it keeps sending a high loss to the generator even though it has no idea what it is doing. The generator updates its weights accordingly but since it is not getting qualified information from the discriminator it does

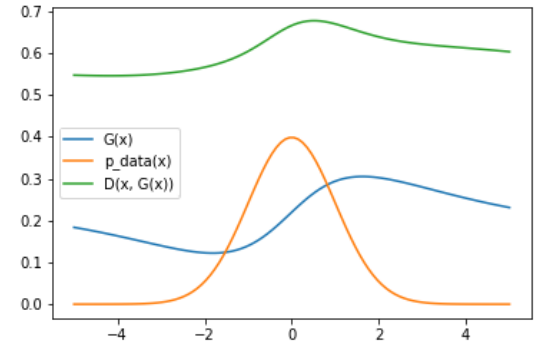
not converge towards the true distribution. As seen in Figure 18 the results are nowhere near as good as the previous results. At the end the discriminator just gives a constant value no matter what the input is.



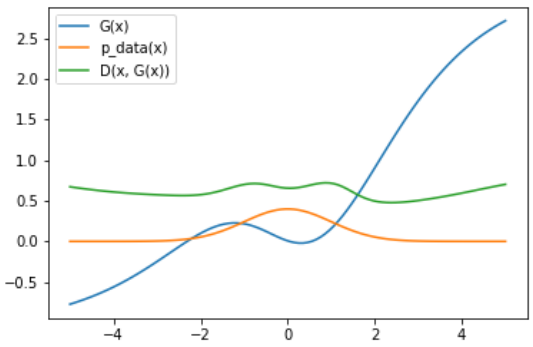
(a) After 0 iterations.



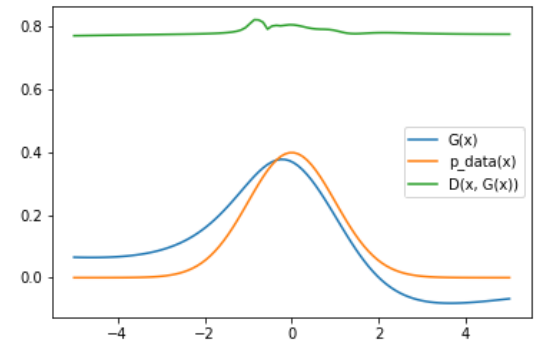
(b) After 500 iterations.



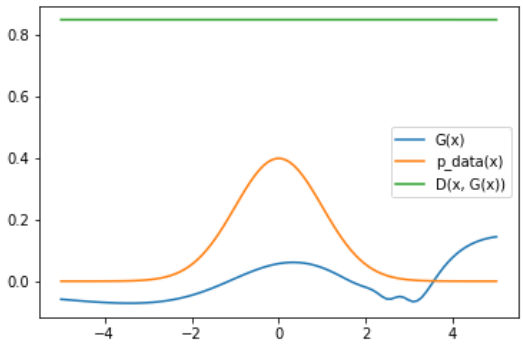
(c) After 1000 iterations.



(d) After 2500 iterations.



(e) After 24000 iterations.



(f) After 120000 iterations.

Figure 18: Illustration of how the training of the generator and discriminator progresses.

2.6 Pix2Pix

In the paper by Isola et al. [16] from 2016 it is shown that a slightly modified version of the cGAN can be used for image-to-image translation with state-of-the-art results. Their network is called Pix2Pix. Figure 19 shows two examples from their article. The left part of the image shows construction of aerial photographs with maps as input. The right part shows the reversed problem, from aerial photos maps are being generated. The minimax game is almost the same as in the original cGAN, but it now also includes a term describing an l^1-loss, see (20). This term only includes the generator and will therefore be of no help when training the discriminator. The fact that paired data is needed is a limitation of this approach. Sometimes paired data is not available or is hard to collect.

$$\min_{\theta_G}\max_{\theta_D}\mathbb{E}_{\substack{x\sim p_x(x)\\y\sim p_y(y)}}\big[\log D(y|x)\big]+\mathbb{E}_{\substack{z\sim p_z(z)\\x\sim p_x(x)}}\Big[\log\Big(1-D\big(G(z|x)|x\big)\Big)\Big]+\lambda\mathbb{E}_{\substack{x\sim p_x(x)\\y\sim p_y(y)\\z\sim p_z(z)}}\big[\|y-G(z|x)\|_1\big] \tag{20}$$

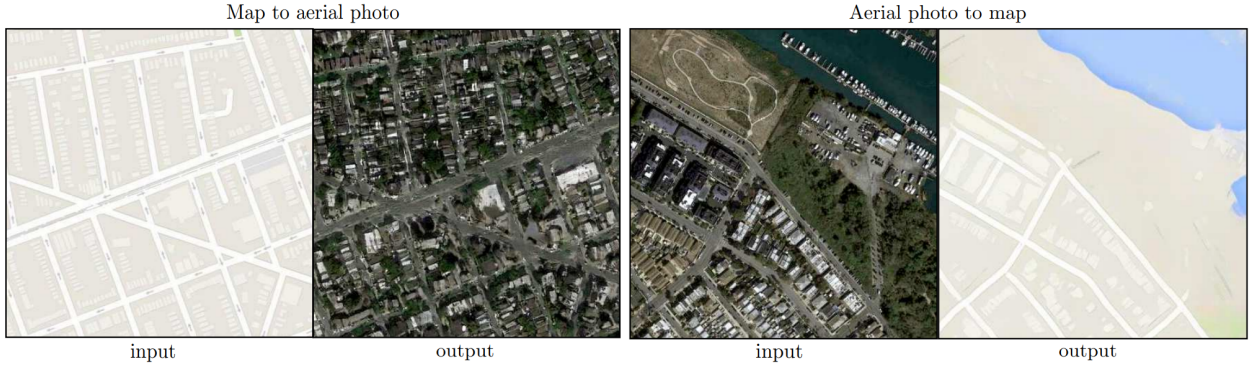


Figure 19: Results from training Pix2Pix on paired images of maps and aerial photos. In the left part of the figure the network is trained to generate an aerial photo given a map as input. In the right part the problem is reversed, the network is trained to generate a map given an aerial photo as input.

2.7 CycleGAN

When paired data is not available another approach is needed. In 2017 Zhu et al. proposed the cycleGAN [45] which is a network that performs image translations between two domains X and Y without the use of paired data. The cycleGAN has two generators and two discriminators. The first generator G transforms an image from domain X to Y and the second generator F transforms an image from domain Y to X. The two discriminators D_X and D_Y are trained on the different domains, D_X is trained to recognize images from domain X and D_Y is trained to recognize images from domain Y. Figure 20 shows an overview of the network structure. This setup makes it possible to explore the idea of a so called cycle loss. Given an image $x \in X$ the first generator, G, transforms it to $\hat{y}$. Then $\hat{y}$ can be given as input to the second generator, F, which transforms it to $\hat{x}$. Now it is possible to compare x and $\hat{x}$, and this is referred to as the forward cycle loss. Reversing the order of

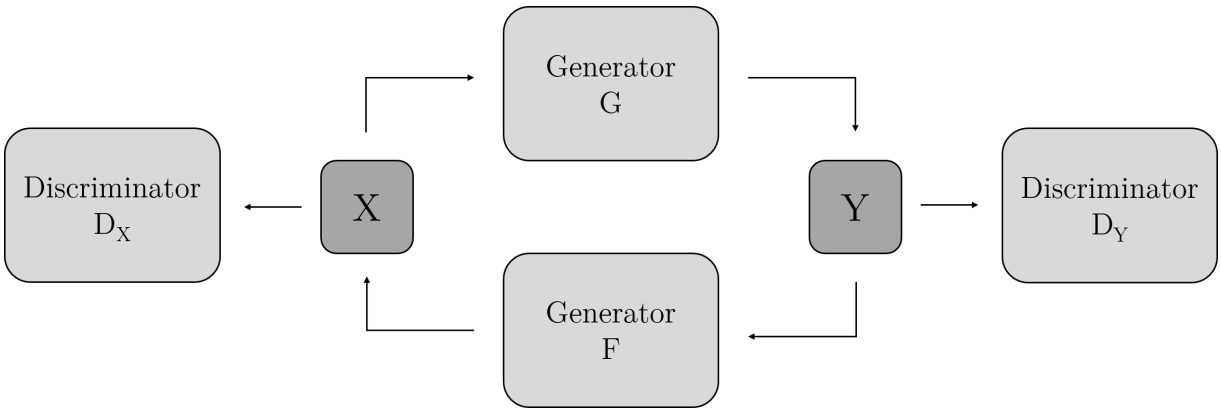


Figure 20: Overview of the cycleGAN framework.

the generators and using $y \in Y$ as input gives another cycle loss, the backward cycle loss. The forward cycle is shown in Figure 21 and the cycle loss is defined as the l^1-norm of the difference between the input and the output from the whole cycle, see (21). In [45] Zhu et al. also experiment with an identity loss, see (22). This loss encourages the generator F to do an identity mapping given input from domain X and similarly encourages the generator G to an identity mapping if G is given input from domain Y. The authors found this loss to help the generators preserve color composition between the input and the output images. Without this loss the generators often changed the tint of the images when there was no need to do so.

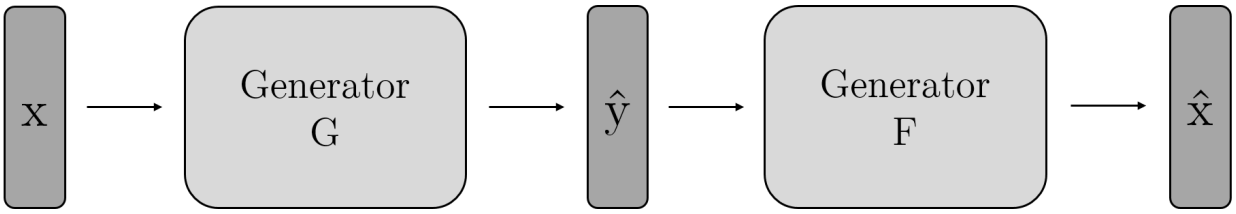


Figure 21: Image describing the idea of a forward cycle loss. A sample from domain X is sent through both generators and is then evaluated how much in l^1-norm it differs from the input.

$$\mathcal{L}_{cycle} = \|y - G(F(y))\|_1 + \|x - F(G(x))\|_1 \tag{21}$$

$$\mathcal{L}_{identity} = \|y - G(y)\|_1 + \|x - F(x)\|_1 \tag{22}$$

Another concurrent work is the DualGAN by Yi et al. [44] which was released a few days after cycleGAN. The basic idea and network structures are very similar, but there are some differences. Both use the discriminator from Pix2Pix [16], but the cycleGAN uses the generator from the paper by Johnson et al. [17] while DualGAN has kept the generator from Pix2Pix as well. Many generative models have some randomness in them, DualGAN included, but cycleGAN is more of a deterministic style-transfer and makes no use of noise or dropout [26].

3 Methodology

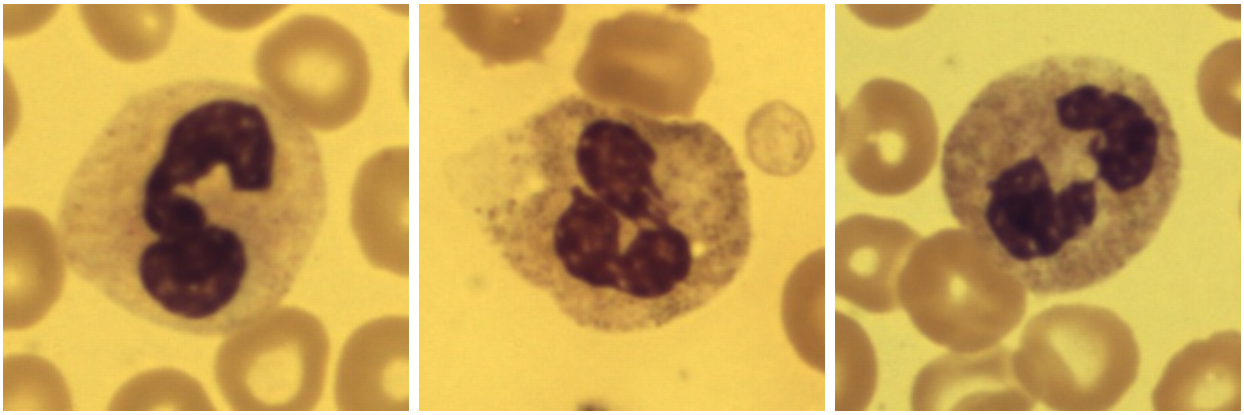
3.1 Cell Images

The raw cell images used for training our networks were collected at CellaVision with their systems DM96 and DM1200. CellaVision provided us with anonymized blood samples to use. The systems output BMP images of white blood cells with 100x magnification, which in this case means that a pixel is equal to $0.1\mu m$. The size of those images are 640×480 pixels. Before using the images for training they are cropped to contain one white blood cell in the middle. The cropped images have the size 256×256 pixels. Because of restrictions from our networks that originate from the up- and downsampling procedures, the size of the images that are put through the networks need to be a multiple of 128. For details about of our network structures see Section 3.6.

3.1.1 Pre-Processing

When the images are used in a conditional setting or in an l^1-loss they need to be paired. CellaVision's systems are able to find the same cells on a slide on different systems, but the images of the cells are not automatically connected or tagged in any way. Therefore we needed to pair them ourselves and this pairing was done using template matching. The nucleus was segmented based on a manual threshold and was then used as the template. Now the images were paired but not aligned exactly. When using an l^1-loss the pairs should match as closely as possible pixel-wise in order to get a good result. If the images in a pair is slightly rotated or scaled compared to each other it is impossible to get an exact matching between the images using template matching. After the pairing is done one can extract SIFT points [25] from both images. SIFT points are feature points in an image which are invariant to scaling and rotation. We found the corresponding feature points in both images and then approximated an affine transformation using a RANSAC algorithm. This gave us images which were more or less matched pixel to pixel.

Blood samples can be colored using three different stainings, May-Grünwald-Giemsa (MGG), Wright-Giemsa (WG) and Wright (W). The stainings look slightly different, see examples in Figure 22. Intuitively one would want an equal distribution of stainings in the data, but there is actually more variation within a staining than between them, so the proportions of the stainings are not very important. The different cell types do however look very different. When collecting blood cells from random slides you get more or less the same distribution of cell types that an average person has in their blood, which means there are a lot less eosinophils compared to neutrophils. Because of this it is reasonable to believe the network would get a lot better at doing the transformation with neutrophils. In order to avoid that and to get a more equal distribution of cells, data was augmented by rotating some of the images 90, 180 and 270 degrees. The rotated images look exactly like real data since there is no specific way a cell is positioned on a slide. Our data consists of 9810 pairs with approximately the same number of cell types. The data was split into a training set (80 %), a validation set (10 %) and a test set (10 %).



(a) May-Grünwald-Giemsa (b) Wright-Giemsa (c) Wright

Figure 22: Cell images showing neutrophils taken with the same DM96 system where the samples have been stained using different stainings.

3.2 Simple Network with an l^1-loss (L1Net)

When performing an image transformation with a neural network we need a transformation network and a loss function. We also need some data to train it. We have paired data (x, y), where x is the image we want to transform and y is the target image. We call our transformation network a generator. As mentioned in Section 2.3.4 convolutional neural networks are very good at handling images, so we will use a CNN as our generator. The generator is based on the U-Net [33], an encoder-decoder structure with skip weights in order to share information between layers. This is very useful in image transformation problems since the underlying structure of the input image and output image often is the same. For more details about the generator architecture see Section 3.6.1. The setup of what we will call the L1Net is shown in Figure 23. The reason for the name L1Net is that the network will only be trained with an l^1-loss. This loss is defined as the l^1-norm of the difference between the transformed image $\hat{y}$ and the original target image y, see (23). There are clear disadvantages with this loss. The paired data needs to be exactly paired, pixel to pixel, for this approach to give satisfactory results. It is well-known that using a norm between images often causes blurrier results. The training of the L1Net is very straightforward, you simply do a forward pass, calculate the loss (23) and then update the weights by backpropagation.

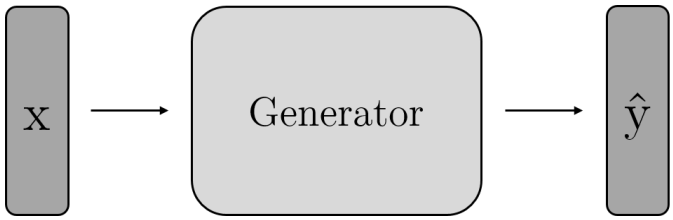


Figure 23: A simple CNN to transform an image.

$$\mathcal{L} = \|\hat{y} - y\|_1 \tag{23}$$

3.3 Conditional GAN with an l^1-loss (L1GAN)

To improve the L1Net we add a discriminator to create a conditional GAN with an l^1-loss. The hope is that a discriminator will help capture more high frequency data compared to only using a loss based on a norm which captures mostly low frequencies [16], see Section 3.6.2 for more details regarding the discriminator. We call this network L1GAN and it is based on the ideas from Pix2Pix, it differs only in the design of the architecture of the generator network. The L1GAN was however first trained using the Pix2Pix generator, but based on the results we got we later changed it to our own generator.

3.4 Conditional CycleGAN (ccGAN)

We have also tried a variant of the cycleGAN. We call this network ccGAN, which is short for conditional cycleGAN. Since we do have paired data, (x_p, y_p), we wanted to make use of that as well, and therefore we introduced another loss that we will call paired loss. This is once again an l^1-loss between transformed images and target images, see (24). The complete loss function (25) for one generator in ccGAN consists of a loss coming from the discriminator (18), the cycle loss (21), the identity loss (22) and the paired loss (24), where the λ_i's are the weights of the loss functions.

$$\mathcal{L}_{paired}(x_p, y_p) = \|y_p - G(x_p)\|_1 + \|x_p - F(y_p)\|_1 \tag{24}$$

$$\mathcal{L}_{tot} = \lambda_1 \mathcal{L}_{gen} + \lambda_2 \mathcal{L}_{cycle} + \lambda_3 \mathcal{L}_{identity} + \lambda_4 \mathcal{L}_{paired} \tag{25}$$

3.5 Perceptual Cycle Network (pcNET)

By combining the idea of a cycle from the cycleGAN and the feature reconstruction loss based on the VGG network, see (11), we define a new framework called perceptual cycle network (pcNET). This framework will consist of three networks, two generators and one pre-trained VGG-16 network, which is a variant of the VGG network. Worth noting is that there are no discriminator networks. The VGG-16 network is only part of the training as an evaluation network (its the cornerstone of the perceptual loss) but it is not part of the training in the sense of getting its weights updated. In this implementation ϕ_j (an intermediate layer in VGG-16) is extracted after the fifth max-pooling layer, see Table 1 in [38] for a full overview of the VGG-16 network. As in the ccGAN the two generators are trained using the cycle losses, the identity losses and the paired loss defined in (21), (22) and (24) respectively. In Figure 24 the setup of the framework is shown.

To make everything a little bit clearer the loss function which generator F is trained against is defined in (26). The samples (x, y) are randomly chosen images from the distributions X and Y and (x_p, y_p) is randomly chosen paired data from the distributions X and Y. We have the cycle loss

$$\mathcal{L}_{cycle}(x, y) = \|y - G\big(F(y)\big)\|_1 + \|x - F\big(G(x)\big)\|_1,$$

the identity loss

$$\mathcal{L}_{identity}(x) = \|x - F(x)\|_1,$$

the paired loss

$$\mathcal{L}_{paired}(x_p, y_p) = \|x_p - F(y_p)\|_1,$$

and the feature reconstruction loss

$$\mathcal{L}_{feat}(x_p, y_p) = \|G_j^\phi\big(F(y_p)\big) - G_j^\phi(x_p)\|_F^2.$$

The weighted sum which F is trained to minimize is

$$\mathcal{L}_{tot} = \lambda_1\mathcal{L}_{cycle} + \lambda_2\mathcal{L}_{identity} + \lambda_3\mathcal{L}_{paired} + \lambda_4\mathcal{L}_{feat}, \tag{26}$$

where the λ_i's are the loss weights. The loss for generator G is constructed in a similar way.

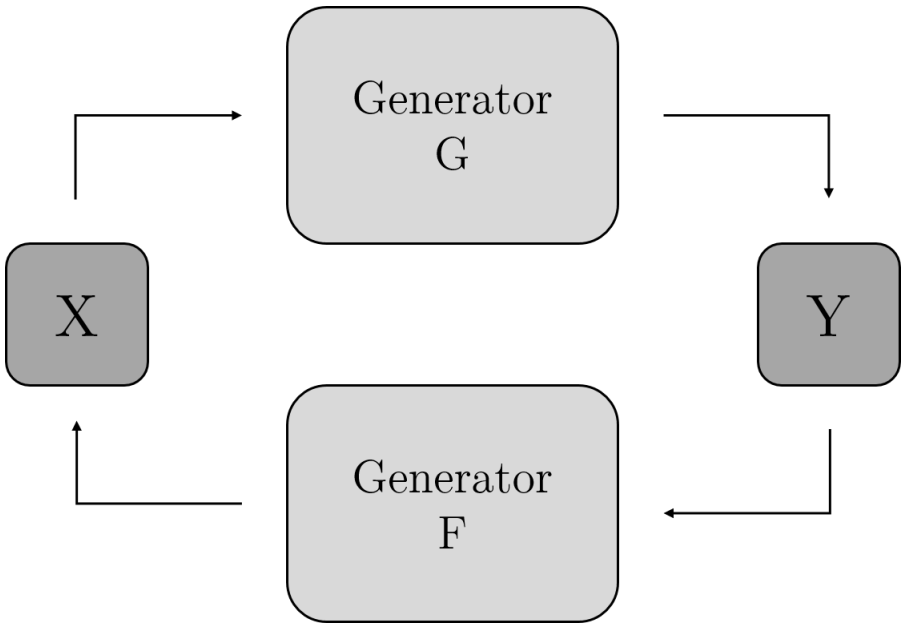


Figure 24: Cyclic network with two generators and no discriminators.

3.6 Network Architectures

All our operations are convolutions which means in theory they can be applied to an image of any size. The images are however up- and downsampled by a factor two, and it is repeated seven times which means the image size should be a multiple of $2^7 = 128$.

3.6.1 Generator

The generator used for all different setups is inspired by the generator in the Pix2Pix paper by Isola et al. [16]. It is based on the U-Net [33], which means it has a lot of skip connections between layers, see Figure 25. Sharing information between layers like this is very common in image transformation tasks where the target image might share a lot of structure with the original image. The generator has 20043919 parameters which is approximately twice the size

of the discriminator. It is built up of encoding blocks and decoding blocks. An encoding block consists of a convolution followed by a batch normalization and a Leaky ReLU activation function. The size of the convolution kernel is 5×5 and the stride is 2. A decoding block starts with upsampling, then a transposed convolution, a batch normalization followed by a ReLU activation function. The difference between our generator and the Pix2Pix generator is the decoding blocks. The Pix2Pix generator use a transposed convolution with stride 2 in order to do upsampling and convolution in one step. We have separated these operations. First an upsampling process is performed followed by a transposed convolution with stride 1. The reason for this is that artifacts like checkerboard patterns can appear in transposed convolutions when the kernel size and stride do not match [29]. In our L1GAN we first used the original Pix2Pix generator, and as seen later in the results we did get a checkerboard pattern.

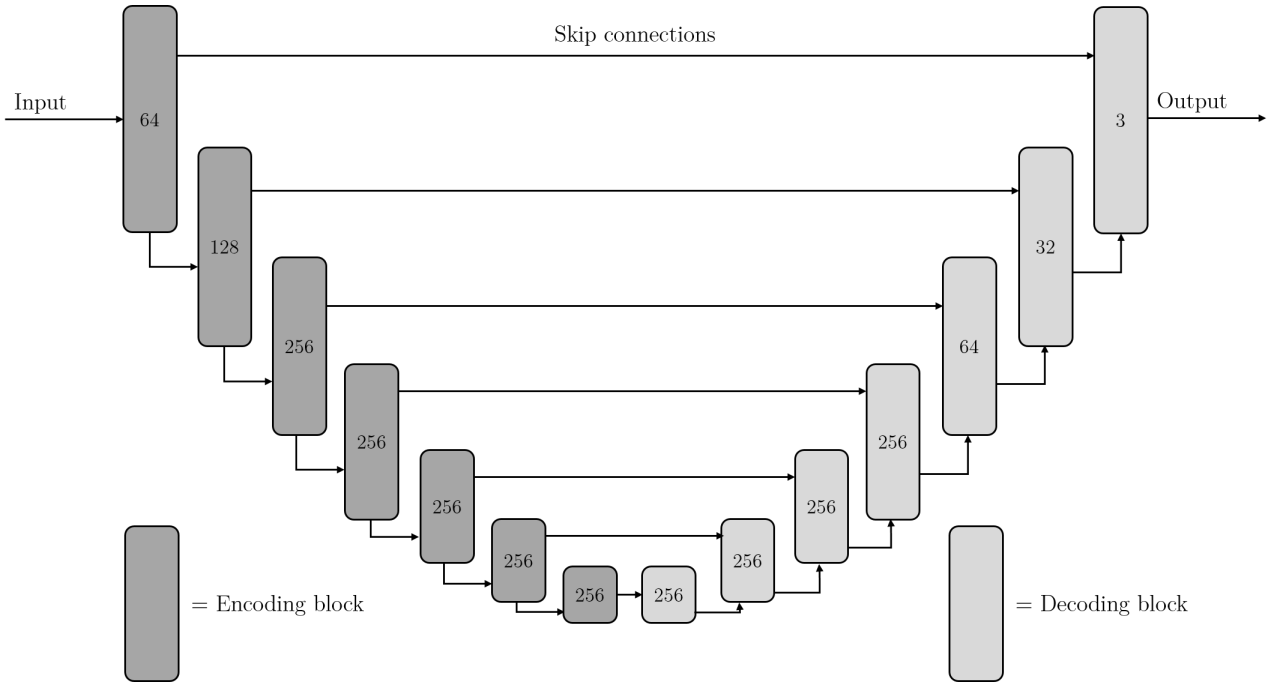


Figure 25: Illustration of the generator architecture. The number in each block is the number of filters in the convolution layer.

3.6.2 Discriminator

In our suggested frameworks the generator is never trained solely against a discriminator. The generator's loss is usually made up of a combination of losses based on norms between two images and a loss given by a discriminator. As previously mentioned losses based on norms are good at capturing low frequency information in images, but might struggle with style and texture. The idea of using a discriminator is to encourage the generator to generate high frequency information in the images. Therefore the discriminator should only penalize lack of high frequency data. This was the idea behind the "PatchGAN" used in [16]. In the article they propose a discriminator with a receptive field of 70×70 pixels. This implies

pixels further away from each other than 70 pixels (in maximum norm) are assumed to be independent from each other. These small patches should capture the local style statistics of the image. The discriminator gives the probability of each patch being real or fake. The output of the discriminator is the mean of the result of the evaluation of all patches. Another advantage of a patch discriminator is that it has less parameters and therefore needs less computing resources compared to a full discriminator that takes the whole image into account [16].

Our patch discriminator has 11047809 parameters and consists of 5 blocks that start with a convolutional layer, then a batch normalization followed by a Leaky ReLU activation. The convolutions have kernel size 4×4 and the stride is 2 for the first 3 blocks and 1 for the last 2 blocks. There is no batch normalization for the first and last block. The final activation function before the mean layer is a sigmoid instead of a Leaky ReLU. Figure 26 shows a simple illustration of our discriminator.

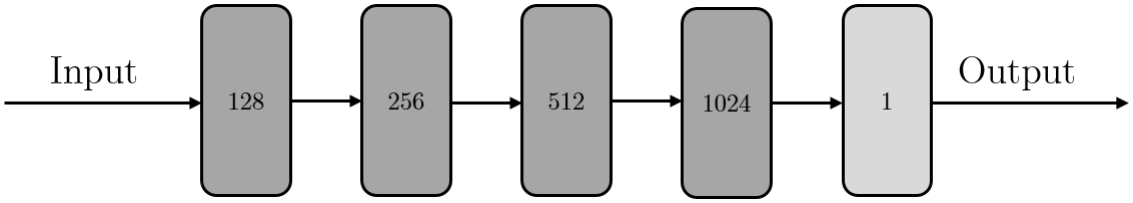


Figure 26: Illustration of the discriminator architecture. The first blocks are regular convolution - batch normalization - Leaky ReLU blocks. The last block has a sigmoid activation function and a mean layer. The number in each block is the number of filters in the convolution layer.

4 Results

First of all it is worth mentioning that most of the images shown in this section are very similar and therefore they should be viewed on a computer screen since a lot of details are lost when printed on paper. For details regarding the training and the tools used see Appendix A. Figure 27 shows an overview of our results. The first column shows the original image from the DM1200 system. The four middle columns are the transformations: the L1Net, the L1GAN, the ccGAN and the pcNET. Finally the original DM96 image is shown in the last column. As mentioned in the introduction of this thesis, the most challenging task for CellaVision has been to transform the eosinophils. Figure 28 shows our transformations of eosinophils. All the images in Figure 27 and Figure 28 have been normalized and results are shown in Figure 29 and 30. The order of the images is the same as before, but the first column of original DM1200 is removed and replaced with images that have been normalized using HM normalization, cf. Section 2.2.

As mentioned in the introduction of this thesis we also wanted to study the classification of the transformed images and see how that compares to the classification of the original images. The most important aspect is not whether the classification is good or not, we just

want the transformed image to be classified the same way as the original image. We used our test set to do the classification. 981 image pairs consisting of a transformed image and its corresponding original DM96 image were sent through the classification network. The pcNet gave the best result, 88.6 % of the transformed images were classified the same way as the original images. Next was the L1Net with 87.7 %, then the ccGAN with 85.2 % and finally the L1GAN with 82.9 %. We also performed the same experiment with the original DM1200 images. It turned out 84.4 % of the images were classified the same way as the original DM96 images. A confusion matrix for the pcNet can be found in Appendix A.

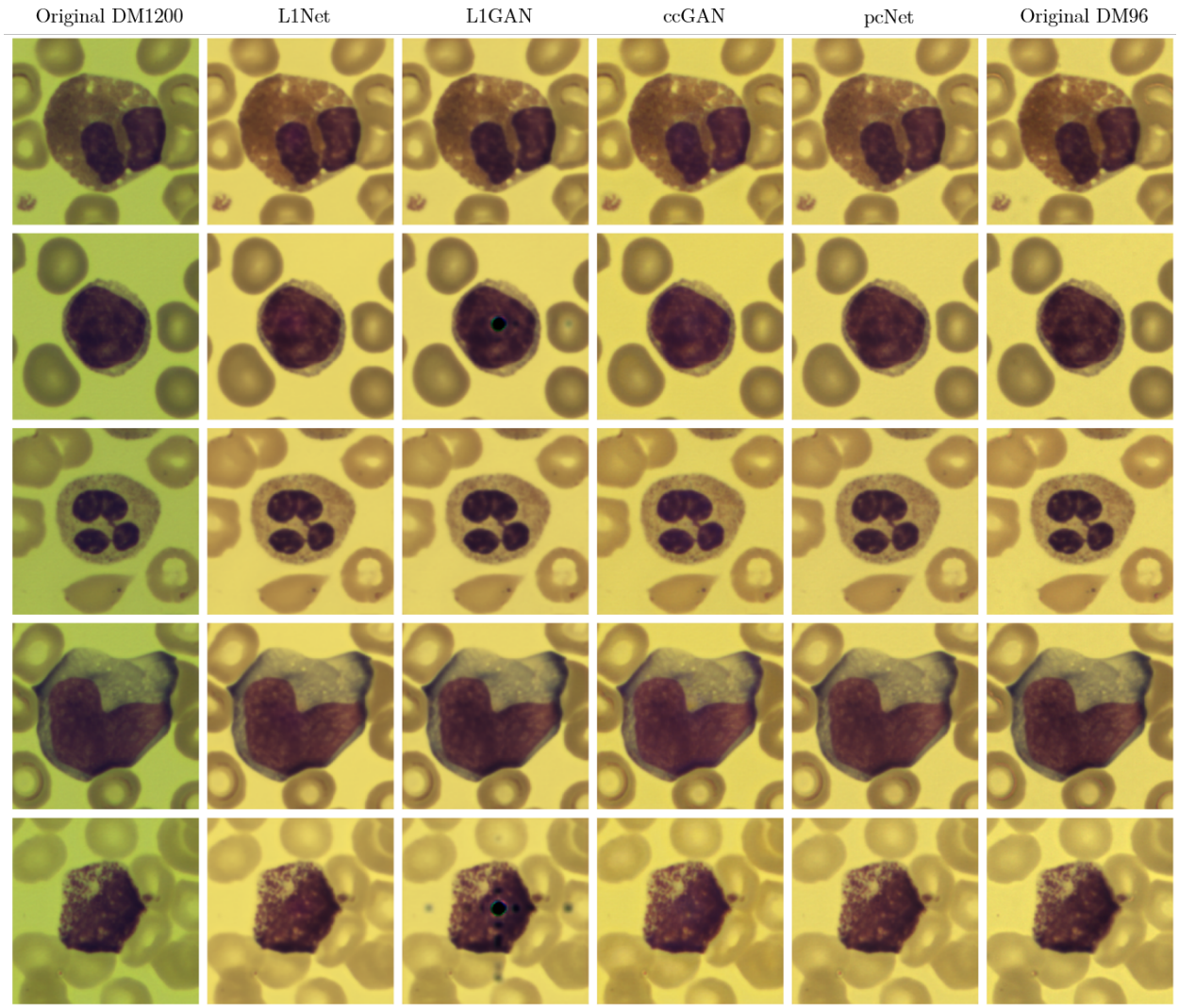


Figure 27: Comparison of all networks with different raw cell images.

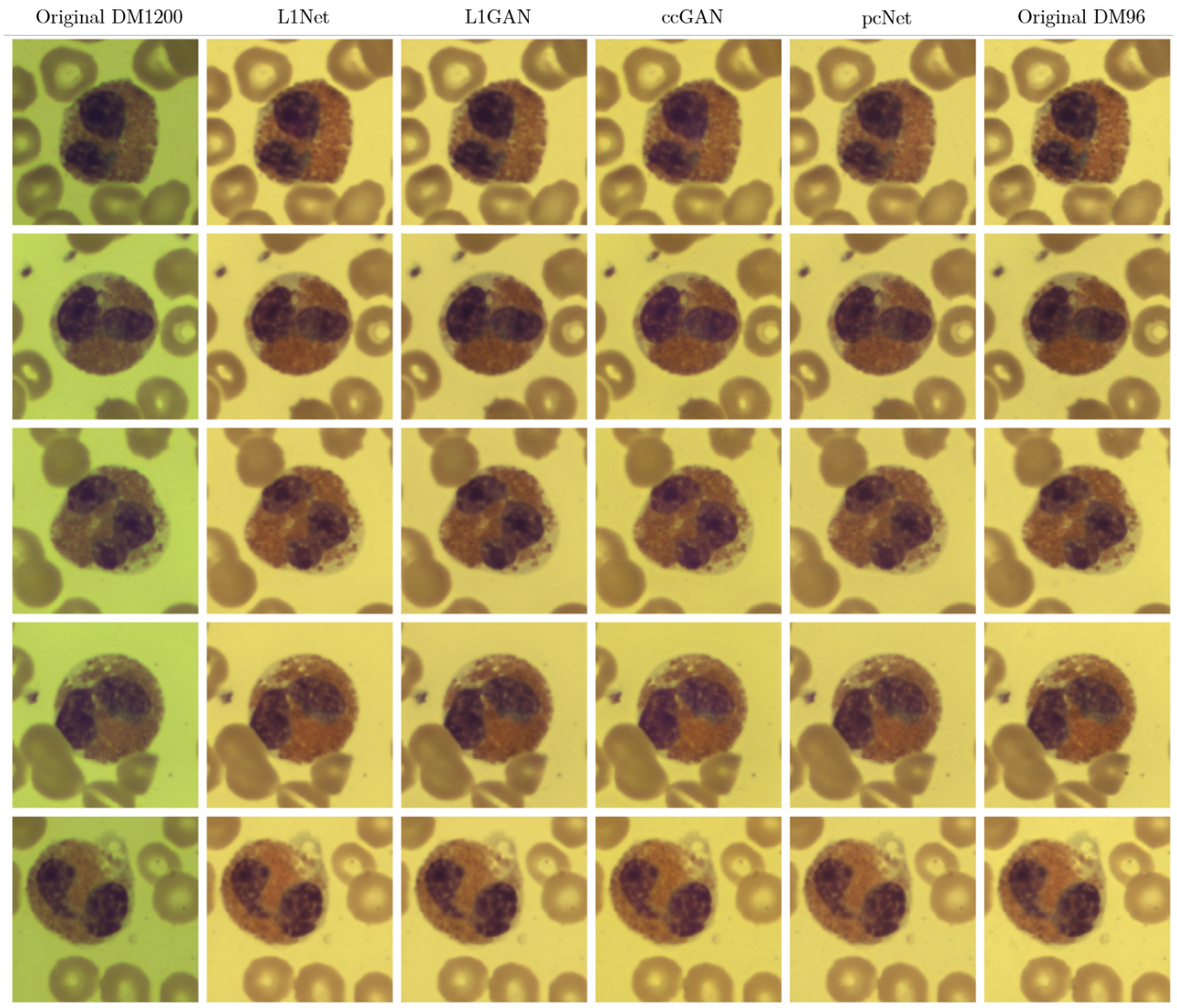


Figure 28: Comparison of all networks with different raw cell images of eosinophils.

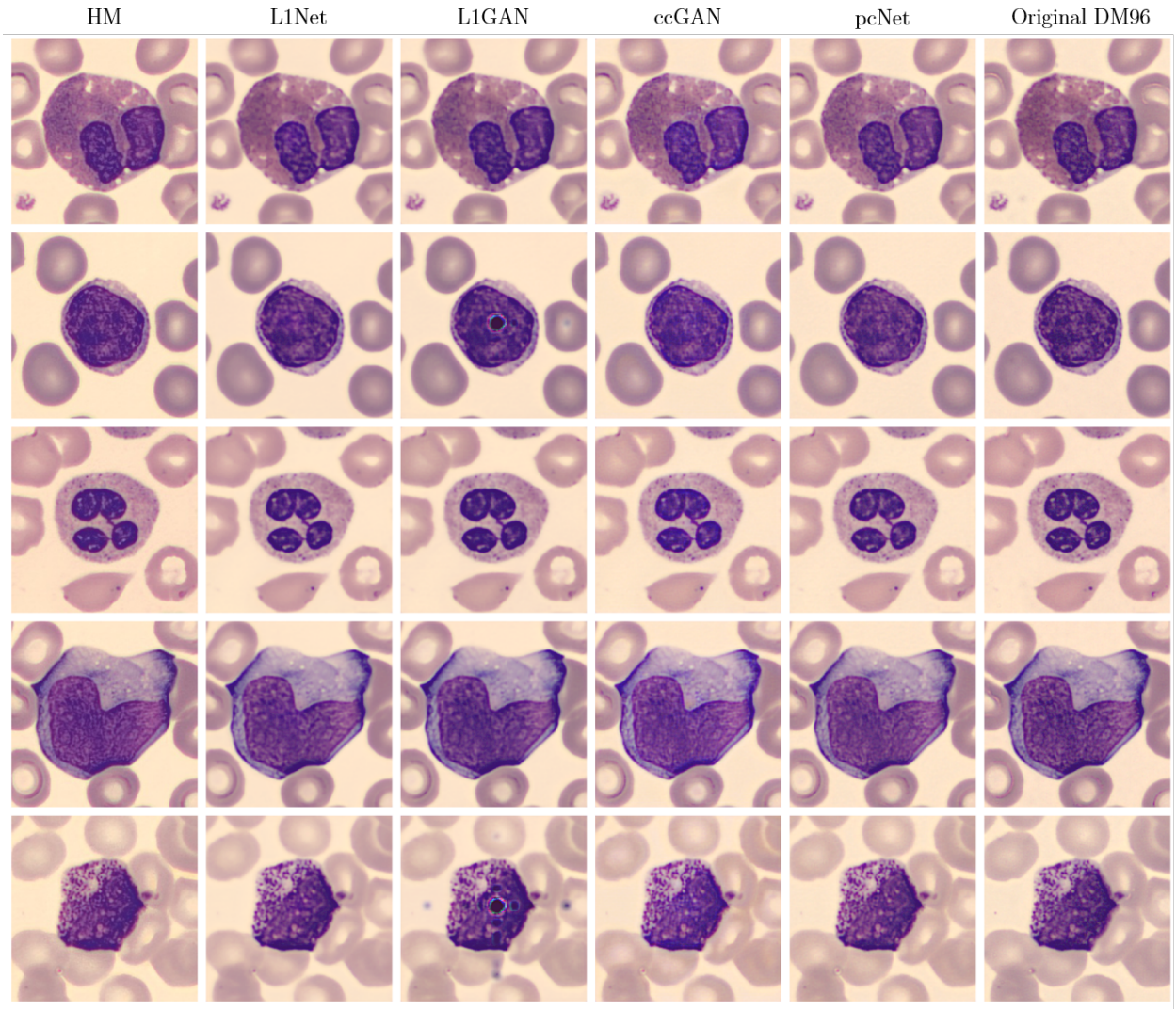


Figure 29: Comparison of all networks with different normalized cell images.

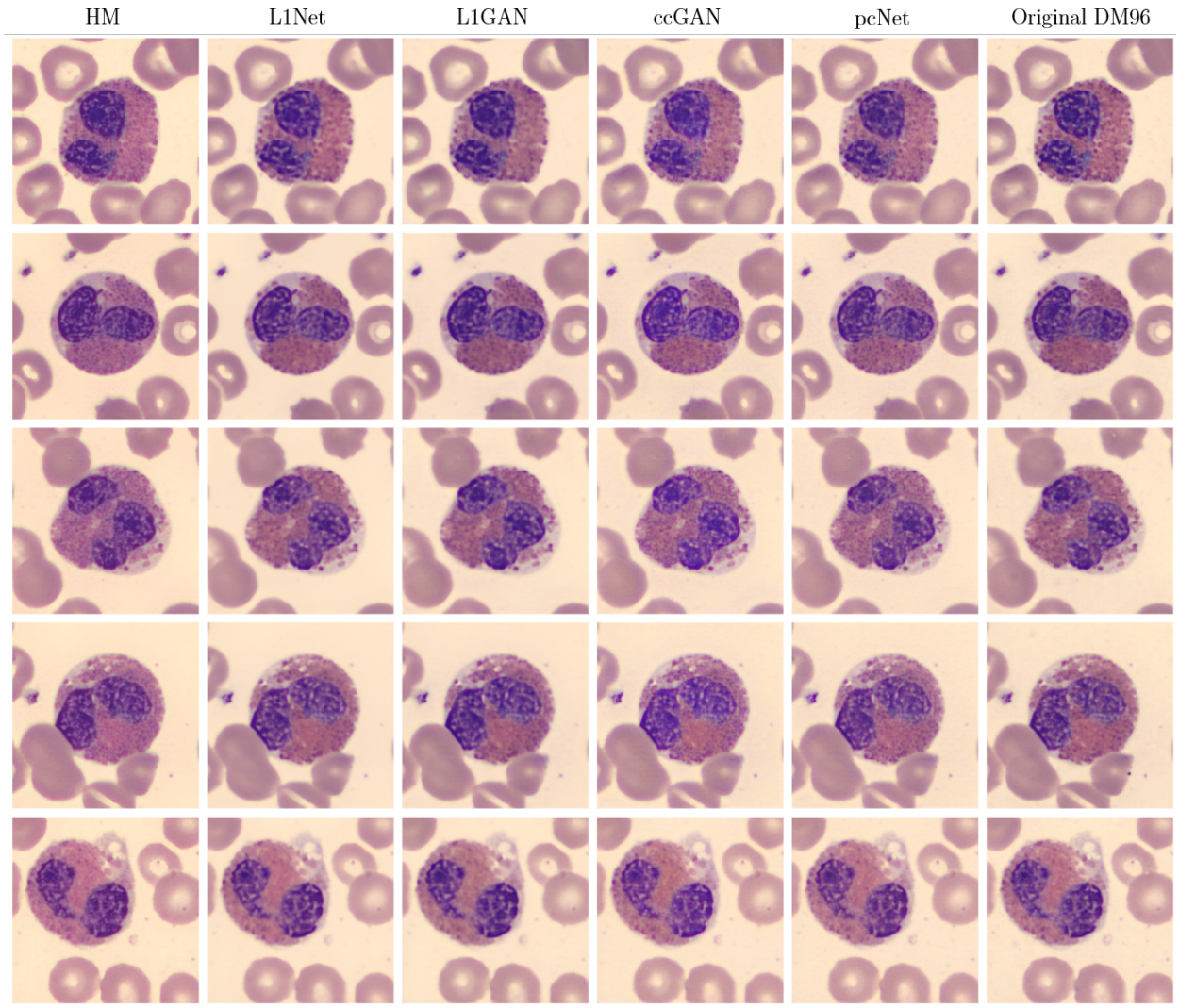
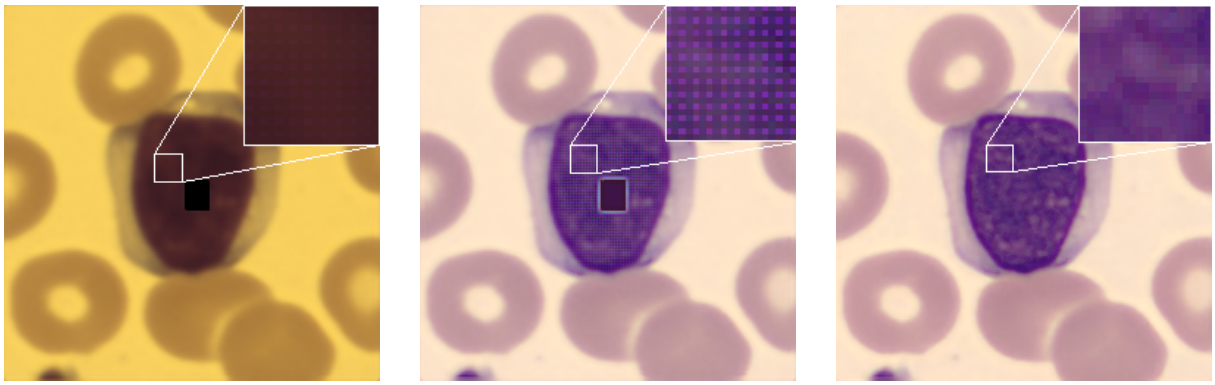


Figure 30: Comparison of all networks with different normalized cell images of eosinophils.

The L1GAN suffers from some occasional black spots in the image when it is trained using the Pix2Pix generator, see Figure 31. It also shows signs of other artifacts such as a checkerboard pattern. This can be difficult to see in the raw image, but in the normalized image (b) it is more clear. Figure 31 (c) shows the transformation with the L1GAN after changing the generator so that it has our own generator. Now the checkerboard pattern is completely gone, but it still produces spots in the images sometimes.



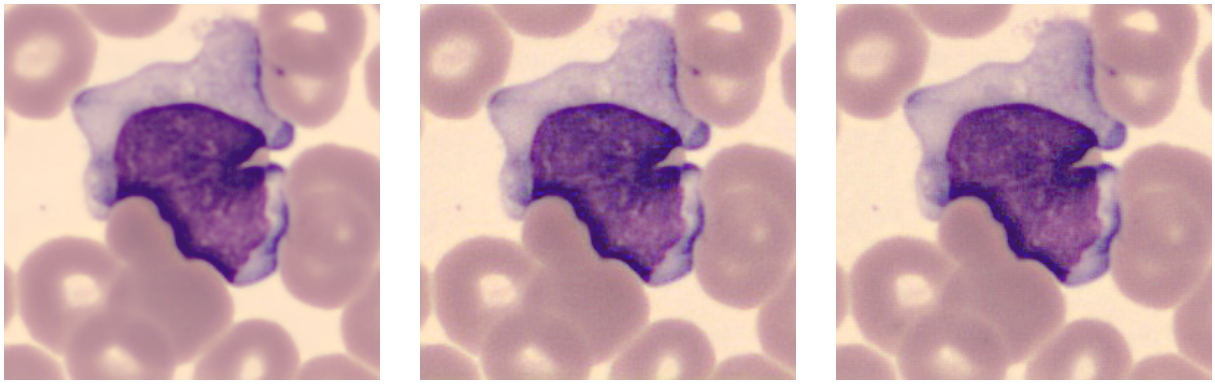
(a) Transformed image from DM1200.

(b) Normalized image with Pix2Pix generator.

(c) Normalized image with our generator.

Figure 31: Example of a black spot that appears in the transformation with the L1GAN is shown in (a) and (b). The checkerboard pattern in the image is visible as well, especially in (b). In (c) our own generator is used and the checkerboard pattern is not present.

Figure 32 shows a comparison of the L1Net, ccGAN and original images. This is to illustrate the fact that training a network with only an l^1-loss gives a blurrier result. At last we have two images of full cycles from the ccGAN, a DM1200 cycle and a DM96 cycle in Figure 33 and 34 respectively.



(a) L1Net

(b) ccGAN

(c) Original DM96

Figure 32: Comparison of transformed DM1200 images from two networks and the original DM96 image. All images have been normalized.

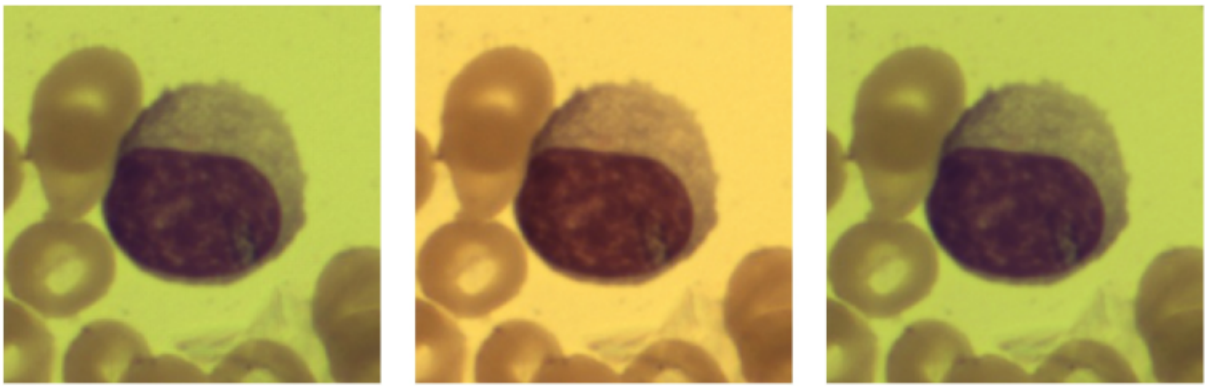


Figure 33: A full cycle starting with a DM1200 image. The DM1200 image is sent through the generator G and then the output from generator G is sent through generator F, cf. Figure 21. As can be seen, the input image is more or less recovered after a full cycle rendering an output which is completely matched pixel to pixel with the input.

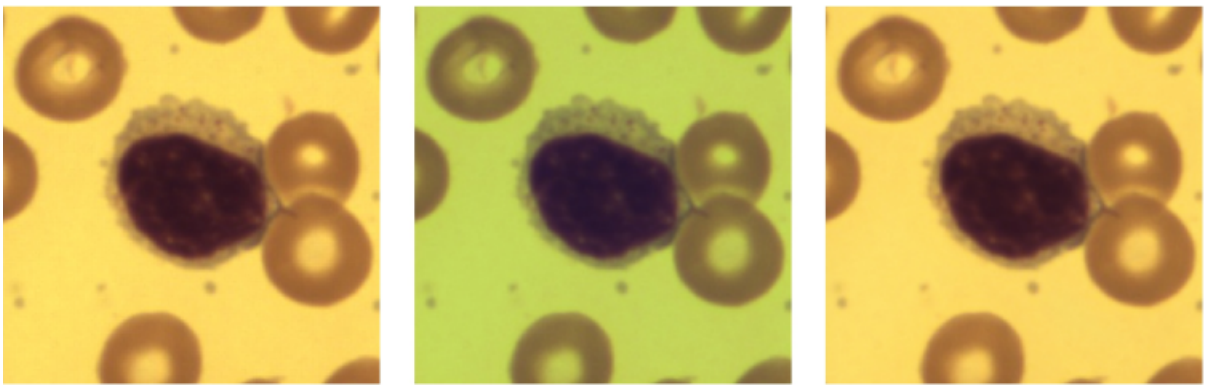


Figure 34: Full cycle starting with a DM96 image. The DM96 image is sent through the generator F and then the output from generator F is sent through generator G, cf. Figure 21. As can be seen, the input image is more or less recovered after a full cycle rendering an output which is completely matched pixel to pixel with the input.

5 Discussion

5.1 General Conclusions

Overall the results are satisfactory. For the best network, pcNet, it is very hard for a person to notice any differences between the original image and the transformed image. It is even harder to decide if an image is transformed or real when shown just one image. However, it is difficult to come up with some metric which captures important characteristics of the generated images and the target images. Popular metrics for comparing images like SSIM (structural similarity) and PSNR (peak signal-to-noise ratio) give very similar results for the images and do not correlate with how good a person thinks the images are. Because of this evaluation difficulty we have mostly evaluated our results by letting a person look

at them. The goal is that the user should not be able to distinguish between images and we think that we have achieved that. We also think that we have achieved better results than the HM normalization that is used today. As can be seen in Figure 29 and 30 the HM normalized images have good structure in the nucleus but the colors are a bit off, especially for the eosinophils in Figure 30. The HM normalization is however a lot faster than all of our networks which is a significant advantage when using it in an actual system.

5.2 L1Net

The colors of the images transformed with the L1Net are very good. They are a little blurry, see Figure 32 for a comparison with the ccGAN and the original image. The image from the L1Net looks like a smoothed version of the image from the ccGAN. Overall it contains less high frequency data compared to the original images and compared to the other transformations. The L1Net is however good when it comes to low frequency data like the colors in the image. An advantage of the L1Net is that it is simple which means it does not need as much memory as for example the ccGAN. This allows us to use a larger batch size than for the other networks. The L1Net is also fast to train since it only consists of one network.

5.3 L1GAN

Column three of Figure 27 shows examples of transformations with the L1GAN. Overall the results are good, but if one looks closer at the nucleus it is a bit blurry and some detail is lost compared to the original image. This is probably because of the l^1-loss function, it is known that using a norm of the difference between images as a loss function causes a blurrier result, but the overall color scheme benefits from this kind of loss. In the L1GAN the l^1-loss is the only loss function except for the discriminator loss which means it has a large impact. Our hope was that the discriminator loss would help sharpen the images by giving a low probability for a blurry image. This did not work out the way we wanted it to and we do not know why. In general it was hard to train the discriminators due to the balancing problem described in Section 2.5.1. The L1GAN is a simple structure and it is faster to train than for example the ccGAN.

As seen in Figure 31 the L1GAN has a problem with occasional black spots. It also has a slight checkerboard pattern which shows up especially in the normalized images. This is a known problem and it is caused by the overlap that can occur in transposed convolutions when the kernel size and stride do not match. This is something we did not realize would be a problem at first. It can be avoided by separating the transposed convolution into an upsampling step and then do the convolution after that. That is what we did in our generator and then the checkerboard pattern disappeared. The result is shown in Figure 31 (c). One can see clearly that there is no checkerboard pattern present in the right image. We do however still have problems with spots in the images. We do not know why the spots appear, but they seem to get smaller the longer we train the networks so maybe the L1GAN needs more training than the other networks. These artifacts are serious and the L1GAN cannot be used until the problem with the spots has been resolved. Another reason why

the other networks are better than the L1GAN could be the different cycle losses, which immediately punish the generator if it generates something that is structurally wrong.

5.4 Conditional CycleGAN (ccGAN)

The ccGAN shows promising results as can be seen in the fourth column in Figure 27. The overall structure and detail is good especially in the nucleus, but as one can see the colors are a bit off. Compared to the original images the colors of the ccGAN results are too saturated. The cycle losses work well, as can be seen in Figure 33 and 34 the input image is almost exactly reproduced after the entire cycle. As mentioned the detail in the ccGAN images is good, and we believe that is because the majority of the losses are calculated based on images that are an exact pixel-to-pixel match. When the images are normalized the colors are slightly better, see Figure 29, but they are still too saturated. Overall the normalized images of the ccGAN are better than the raw images.

A disadvantage of the ccGAN approach compared to the other frameworks is the large number of networks. Four different networks are being trained but only one is used. Because of memory limitations the large number of networks forces us to use a smaller batch size which might slow down the training and make it more sensitive to bad training samples. If a training batch does not capture the characteristics of the training data, the networks might be updated in a suboptimal way.

5.5 Perceptual Cycle Network (pcNET)

Instead of using a discriminator we also tried a perceptual loss function as an attempt to improve the resolution of the transformed images. The result is shown in Figure 27, column five. As one can see the details are very good, there are not many visual differences in the structure of the cell. The colors are also very good and in our opinion this is the best network. It combines the advantages of other frameworks presented in this thesis. The paired l^1-loss used in for example L1Net helps capture low frequency data like the colors. The cycle losses from the ccGAN and a perceptual loss make sure the high frequency data, such as details and structure, is preserved.

5.6 Classification

Another thing that is a measure of quality is how well one of CellaVision's classification networks performs on the transformed images. Since the goal is to make the transformed DM1200 images look the same as the original DM96 images we do not necessarily care whether the images are classified correctly, but instead that they are classified as the same cell type. We did not know if this would be a good way to measure similarity between the images, but the results of the classification agree with our visual evaluation of the networks. The pcNet was the best network according to the classification results, but all of the networks performed quite well.

5.7 Future Work and Improvements

In this thesis the networks are very large which means the transformation takes a while, approximately 300 ms on a CPU (Intel Core i7-7700 at 3.60 GHz) and 10-20 ms on a GPU (NVidia GeForce GTX 1080 Ti). The actual systems the transformation would be run on do not have a dedicated GPU and in many cases the CPU is worse than the one we did our tests on. To be able to use the networks in those systems the transformation needs to be a lot faster, less than 100 ms on an average CPU. In order to achieve this one option is to try to scale down the networks. Implementing the networks in a low level language could also improve the efficiency.

Another thing that would be interesting to investigate is how the intensity of the images affect the transformation. It is known that images taken with different systems can have different intensity, and this is something we have not taken into consideration while training our networks. All our images were collected from the same systems at the same time, so they have more or less the exact same intensity. The easiest way to investigate this would be to augment data by randomly changing the intensity of the images slightly while training the network. The more thorough approach is to actually change the settings in the system while taking the images to get real images with different intensities.

5.8 Ethical Considerations

As always in medical applications that use patient data the integrity of the patient is very important. The blood samples used in this thesis were anonymized so we never knew whose blood it was.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*, pages 653, 43. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] CellaVision. Leukocytes in peripheral blood: http://www.cellavision.com/en/cellavision-cellatlas/leukocytes, 2018.

[4] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient Primitives for Deep Learning. abs/1410.0759, 2014.

[5] François Chollet et al. Keras. https://github.com/keras-team/keras, 2015.

[6] L. Dean. Blood groups and red cell antigens: Chapter 1, blood and the cells it contains. *Bethesda (MD): National Center for Biotechnology Information (US)*, 2005.

[7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courvile, and Y. Bengio. Generative Adversarial Networks. *NIPS*, pages 2672–2680, 2014.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning - chapter 8.1.3.* MIT Press, 2016. http://www.deeplearningbook.org.

[10] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. abs/1701.00160, 2017.

[11] Roger Grosse. Lecture 5: Multilayer perceptrons (csc321). *University of Toronto*, 2018.

[12] Roger Grosse. Lecture 8: Optimization (csc321). *University of Toronto*, 2018.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. abs/1512.03385, 2015.

[14] ImageNet. ImageNet Object Localization Challenge. 2018.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. abs/1502.03167, 2015.

[16] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. *ArXiv e-prints*, November 2016. ArXiv:1611.07004.

[17] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. abs/1603.08155, 2016.

[18] Adam Karlsson. Area-Based Active Contours with Applications in Medical Microscopy, 2005. Licentiate Thesis.

[19] Andrej Karpathy. Convolutional neural networks (cnns / convnets). *Convolutional Neural Networks for Visual Recognition*.

[20] Andrej Karpathy. Neural networks. *Convolutional Neural Networks for Visual Recognition*.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. abs/1412.6980, 2014.

[22] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag.

[23] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[24] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint*, 2016.

[25] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[26] Ruotian Luo. Reading notes: Triplets from different parents, cyclegan, discogan, dualgan. 2017.

[27] M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. *arXiv:1411.1784 [cs.LG]*, 2014.

[28] Michael A. Nielsen. Neural Networks and Deep Learning: Chapter 3, the cross-entropy cost function, 2015.

[29] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[30] Kara Rogers/The Editors of Encyclopaedia Britannica. Basophil. *Encyclopædia Britannica*, 2016.

[31] Kara Rogers/The Editors of Encyclopaedia Britannica. Neutrophil. *Encyclopædia Britannica*, 2016.

[32] Mattias Ohlsson. Lecture notes from course "Introduction to Artificial Neural Networks and Deep Learning" (FYTN14), Lund University. 2017.

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. abs/1505.04597, 2015.

[34] Frank Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65:6, 1958.

[35] Avinash Sharma. Understanding activation functions in neural networks. *The Theory Of Everything*, 2017. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[36] Naoki Shibuya. Up-sampling with Transposed Convolution. 2017.

[37] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227 – 244, 2000.

[38] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. abs/1409.1556, 2014.

[39] Utkarsh Sinha. Image convolution examples. *AI Shack*, 2017.

[40] Johannes Skog. Re-identification with recurrent neural networks, 2017. Master's Thesis.

[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[42] T. Tieleman and G. Hinton. COURSERA: Neural Networks for Machine Learning. technical report. 2012.

[43] C. Wang, C. Xu, C. Wang, and D. Tao. Perceptual Adversarial Networks for Image-to-Image Transformation. abs/1706.09138, 2017.

[44] Zili Yi, Hao Zhang, Ping Tan, and Minglun Gong. Dual-GAN: Unsupervised Dual Learning for Image-to-Image Translation. abs/1704.02510, 2017.

[45] J.-Y. Zhu, T. Park, P Isola, and A. A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. abs/1703.10593, 2017.

Appendix A

Tools Used

All code was written in Python 3.6.4. The two main frameworks used were TensorFlow [1] (version 1.4.0) which is an open source library for numerical computations and Keras [5] (version 2.1.3), a high-level neural networks API. The GPU version of TensorFlow was used together with the CUDA Toolkit (version 8.0) and cuDNN (CUDA Deep Neural Network library, version 5.1) [4] from Nvidia. Keras has built-in GPU support, it automatically runs on the GPU if TensorFlow does which is very convenient. All the training was performed on an NVidia GeForce GTX 1080 Ti GPU.

Training Details

The optimization method Adam was used in all training with learning rate $\alpha = 0.0002$. The L1Net only has one loss with weight 10 and the batch size was 8. For the L1GAN the loss weights were $\lambda_{disc} = 1$ and $\lambda_{paired} = 10$ and it was trained with batch size 4. The ccGAN had $\lambda_{disc} = 1$ and the rest of the loss weights were all 10. The batch size for the ccGAN was 4. The pcNet had the loss weight 10 for all losses and was trained with batch size of 2. All the networks have been trained for 50000 iterations.

Confusion Matrix

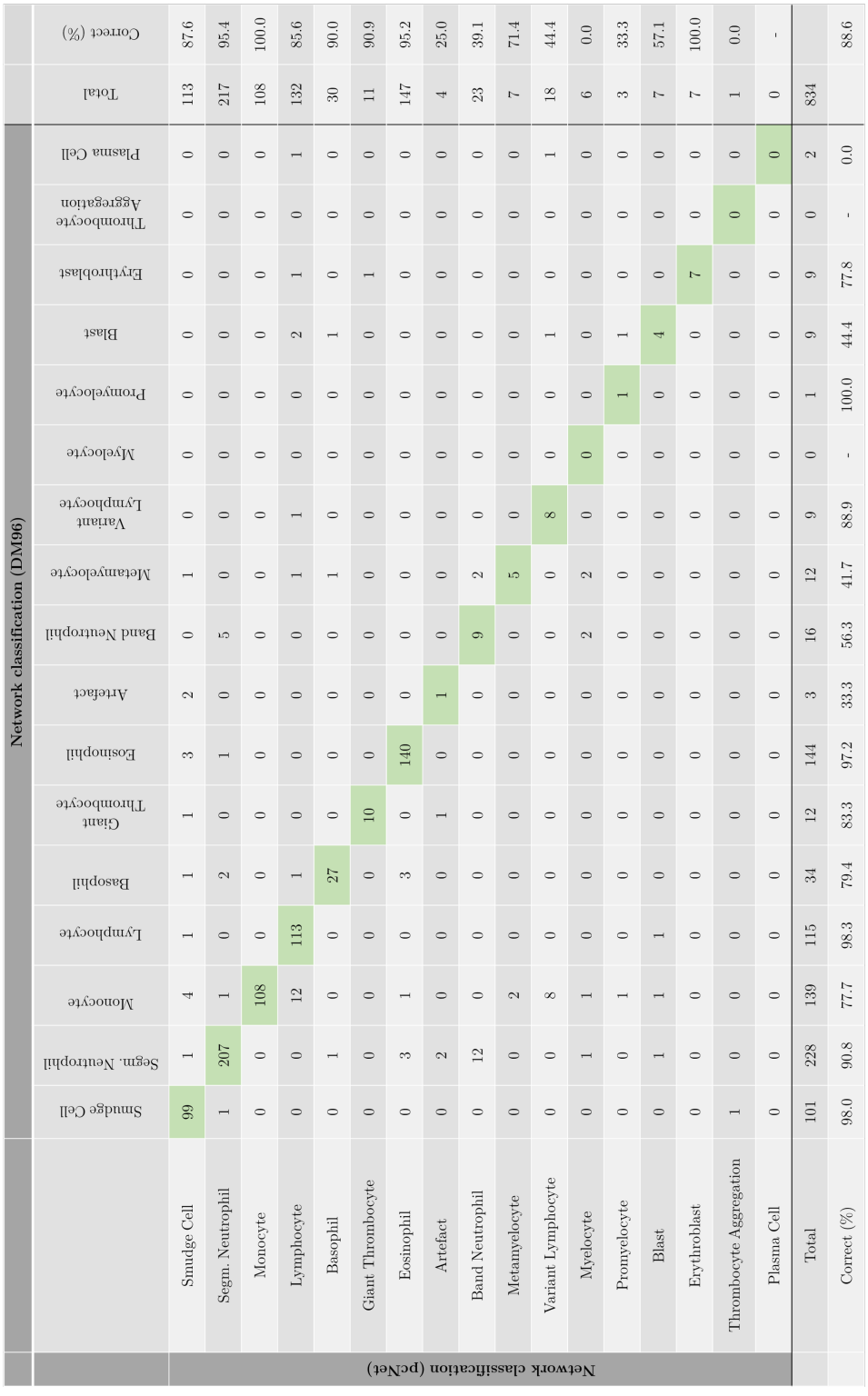


Figure 35: Confusion matrix for the classification of images transformed with pcNet. Some images have been classified as "Unidentified" or "Undefined". These images have not been included here which is why the total amount of images is not 981.