Lunds Tekniska Högskola

Master's Thesis

Virtual Staining of Blood Cells using Point Light Source Illumination and Deep Learning

Joel Wulff

January 17, 2022

Supervisors:
Jesper Jönsson
Martin Almers
Alexandros Sopasakis[LU]

LUND UNIVERSITY

LTH

FACULTY OF ENGINEERING

Abstract

Blood tests are an important part of modern medicine, and are essentially always stained using chemical colorization methods before analysis by computational or manual methods. The staining process allows different parts of blood cells to be discerned that would be unnoticeable in unstained blood. However, this process is complicated to perform correctly and often takes much time. It often requires a trained professional to perform, and the method may damage or alter cells during the staining process. There is also no globally defined standard for staining, which means different labs use different methods and professionals are trained using different stainings.

In this thesis, I investigate the possibility of avoiding the chemical staining process by digitally transforming an image of an unstained blood cell to its stained version using deep neural networks and point light source illumination. This problem is thought to be ill-posed, as there is much more information in an image of a stained cell than an unstained one. To counteract this and provide additional information to the neural networks a programmable LED-array is employed as the lighting device in a digital microscope, and each image of an unstained blood cell is accompanied by several extra images taken while lit by one LED at a time. This creates an input with considerably more information than if only traditional lighting had been used, and is key to the performance of the presented models. Virtual staining presents several advantages over chemical staining, as it 1) avoids potentially damaging cells through an invasive method, 2) saves time and resources, 3) allows concurrent staining of different cells using different virtual stains. A complete dataset of 29999 white blood cells was designed and collected during the process of this thesis. As far as I know, this is the first time a setup of this type is used for the purpose of virtual staining.

The neural networks investigated consist of architectures inspired by ESRGAN, using Residual-in-Residual blocks in combination with different loss functions ranging from pixel-wise losses to perceptual losses. Furthermore, the use of a generative adversarial network, or GAN, is also investigated to evaluate whether it improves performance. The results include models with four different loss functions, with one of them being a GAN. Variations of these models are also tested with different model depths and dataset size. The presented methods show promising results, and provide a strong proof-of-concept for the proposed solution to virtual staining. The results are however too unreliable and the networks are too slow to implement in a system in their current state.

Acknowledgements

First and foremost, I wish to thank Jesper Jönsson and Martin Almers for giving me the opportunity to work on such an interesting problem. Their enthusiastic support and guidance has been incredibly valuable during my work on this thesis, and I am grateful for it. Them and my other colleagues have really made me feel like a part of the team, and our weekly meetings and discussions have been a great source of inspiration.

I would like to extend thanks to my supervisor at the department of Mathematics at LTH, Alexandros Sopasakis, for the encouraging meetings and discussions and his helpful comments throughout the project.

I would also like to thank Elina Bergskans and Åse Sykfont Snygg for their guidance and expertise in understanding chemical staining, and for their help in staining blood smears for the investigated dataset.

Special thanks to Sara Arnetorp, for comments on writing style and language.

And finally, I would like to thank my friends and family who have always been there for me during my education and helped me get to this point.

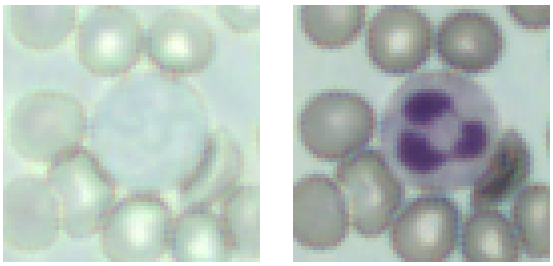
Contents

Chapter 1

Introduction

1.1 Background and motivation

In the past years modern medicine has taken several steps toward digitalizing and automating previously manual tasks. The area of heamatology, blood analysis, is no exception. Blood analysis has long been an important tool leveraged to screen for many different diseases and confirm diagnoses. One important procedure in heamatological analysis is the examination of blood smears. A peripheral blood smear is created by smearing a sample of blood onto a microscope slide. Traditionally, the sample is then chemically stained in order to make the cells more visible before being analyzed manually by experts using optical microscopes. Before staining most cells appear mostly translucent. In Fig. 1.1 an example of a white blood cell (WBC) can be seen before and after chemical staining. The chemical staining process is time-consuming, invasive, and complicated, requiring strict protocols for each staining solution used. No international standard method exists, and different laboratories use different staining solutions and protocols when creating their blood smears. Furthermore, the machines and solutions used for this process are expensive, and require specifically trained professionals to guarantee the quality of the samples.



(a) Unstained (b) Stained

Figure 1.1: A white blood cell before and after chemical staining.

As the traditional workflow of blood analysis is both labor intensive and expensive for hospitals to perform, many are exploring methods of optimizing it. One increasingly popular solution has been the adoption of digital microscopes, complete with supporting software capable of performing analysis providing increased information to the analysts. In recent years, further work has been done investigating whether one can use specialized digital microscopes to virtually stain the blood smears. With *virtual staining*, it is meant that a digital microscope is first allowed to image an unstained blood sample. This image is then transformed through the use of some computer software into its stained form, artificially mimicking the chemical staining process.

When working in heamotology or cytology most types of stains are variations of so called *Romanowsky stains*. The term Romanowsky stain is a generic description of a family of staining methods focusing on combining the acidic/basic dye pairs Azure B (or polychromed methylene blue) and eosin. Objects

stained by Azure B become blue, while objects stained by eosin become red. A simple way to conceptualize how the dyes bind to the staining material is to view it as an ion-exchange process. The negatively charged colored anions of eosin Y exchange with mobile inorganic anions to neutralize fixed positive ions on the sample, and ends up staining it. Likewise, the colored cations of Azure B exchange with the mobile inorganic cations to neutralize fixed negative ions on the sample, attaching to it. The process is schematically shown in Fig. 1.2. In this way, the two stains attach to different biological material and a polychromatic staining can be created. However, dyes are not only ions, and so there are several more phenomena at play deciding the final colorization. An example of this is the possibility of dye-dye complexes appearing, which is a key characteristic of Romanowsky stains: in nuclei of blood cells an azure B-eosin or azur B-eosin-biological substrate is formed coloring the nuclei a distinct purple color, as seen in Fig 1.1. This is sometimes referred to as the "Romanowsky-Giemsa effect". Commonly used variants of Romanowsky stains include the Wright, Giemsa, and May-Grünwald stains. [1]. In Fig. 1.3 two different combinations of them are presented to illustrate their visual difference.

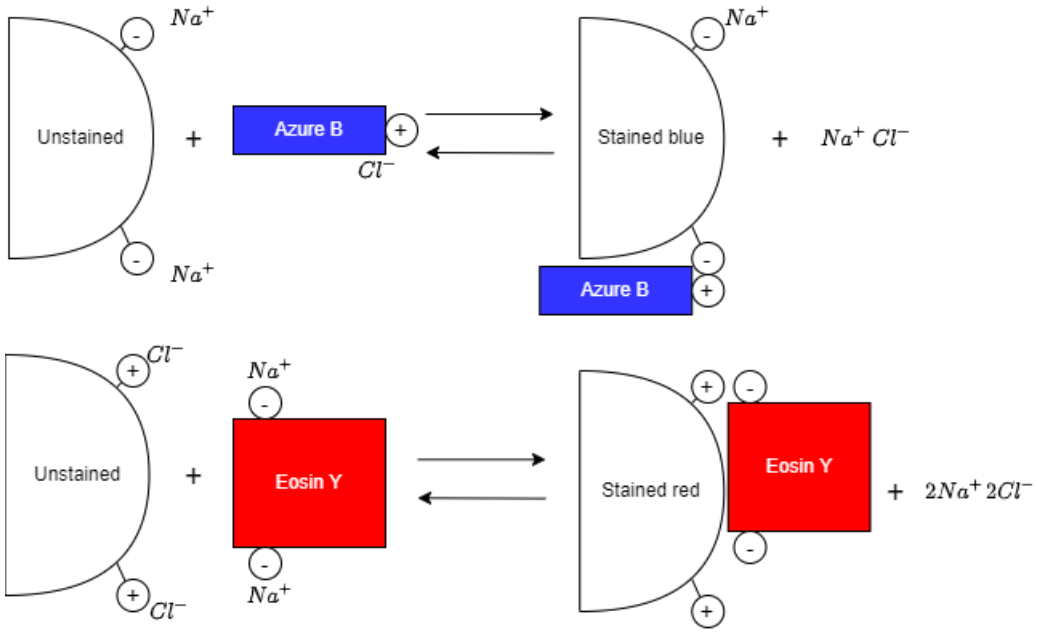
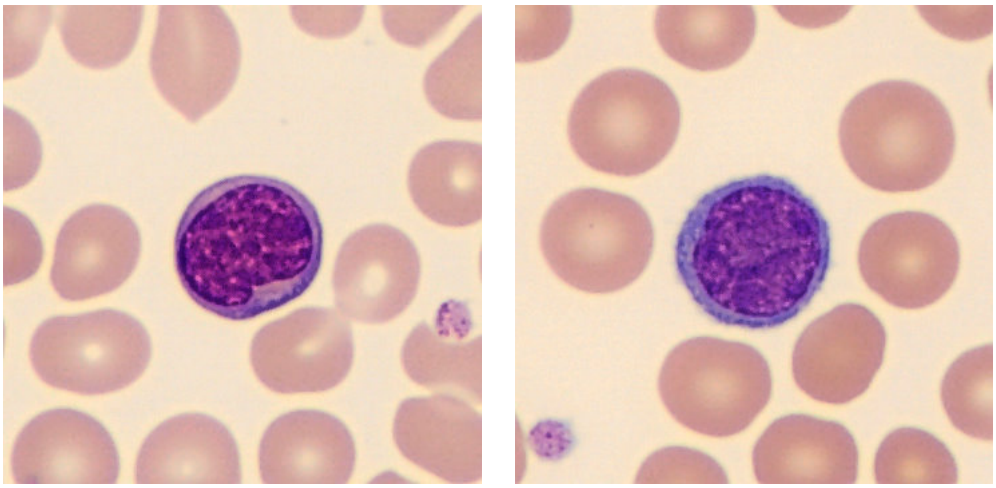


Figure 1.2: Illustration of acid and basic dyeing as an ion exchange process. The actual inorganic mobile counterions present when staining depend on staining solution, but are shown here as Na^+ and Cl^-.

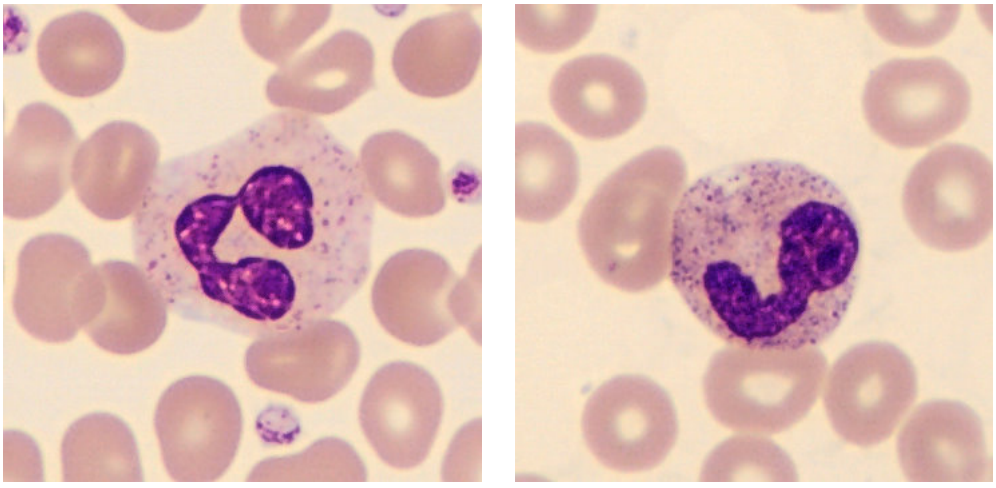
The popularity of Romanowsky stains is attributed to the relatively simple method of achieving polychromy in stained cells: minimally, this requires just one dye bath using the two dyes. This creates red, blue, and purple stained structures allowing for morphological anlaysis to be performed. However, laboratories still struggle to consistently get satisfactory results due to the many sensitive steps involved, such as fixation of the sample, pH of involved solutions, impurity of commercially sold dyes, specific staining times, and more. Often the process involves toxic substances requiring the use of fume cupboards, further complicating application.



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

Figure 1.3: Two images of lymphocytes stained with either a May-Grünwald Giemsa (MGG) or Wright-Giemsa (WG) stain. Note the more purple/reddish hue of the MGG and the more blue tones of WG.

Since the staining process is also a physically altering process, it means that even the morphological information of the starting cell can be altered during the staining process: cells may shrink as water is drawn out of them and morphological features can become distorted. In some cases cells may even be damaged or burst. Different types of staining artifacts can appear, that could obstruct or obfuscate the actual cells to be inspected. The staining process is also not reversible; once a sample has been stained with a specific stain, it cannot be unstained. This means an error in staining can essentially ruin a sample for good. The large variety of staining procedures also mean that different analysts are used to working with differently colored samples, and may not be able to recognize the same diagnosing traits in samples stained with another staining type. Even more troublesome is the large variation of stain colors produced even when using the same protocol and stain type due to differences in the commercially supplied dyes or simply other factors of usage; how long the dye containers have been opened, what temperature they have been stored at, how oxidized they have become, how the actual sample has been handled, et. cetera. Some examples of how different cells can appear even when stained using the same method and protocol are shown in Fig. 1.4.



(a) MGG stain from site 1 (b) MGG stain from site 2

Figure 1.4: Two images of neutrophils stained with May-Grünwald Giemsa (MGG) stain at two different testing sites. Note the difference in purple color of the nucleus.

Virtual staining offers several advantages over chemical staining, as it 1) avoids potentially damaging cells through an invasive method, 2) saves time and resources, taking only seconds to a few minutes

as opposed to tens of minutes to hours 3) allows concurrent staining of different cells using different virtual stains, and 4) is a reversible process. The speed at which a virtual staining process can be performed is limited only by the computational power available, and not by a physical process as the chemical staining, which reduces the problem of speeding up staining to one of purchasing stronger GPUs. A successful implementation of virtual staining means laboratories could completely skip the chemical staining process, and save huge amounts of time and money. Furthermore, virtual staining could also allow for more digital microscope software developed directly for unstained blood (e.g. classification), which has previously not been explored much. One reason for this is that it is very hard to verify whether a model is correct or not, as experts cannot analyze unstained blood cells, but if one could virtually stain the cells an expert would be able to verify classifications and hold the software accountable.

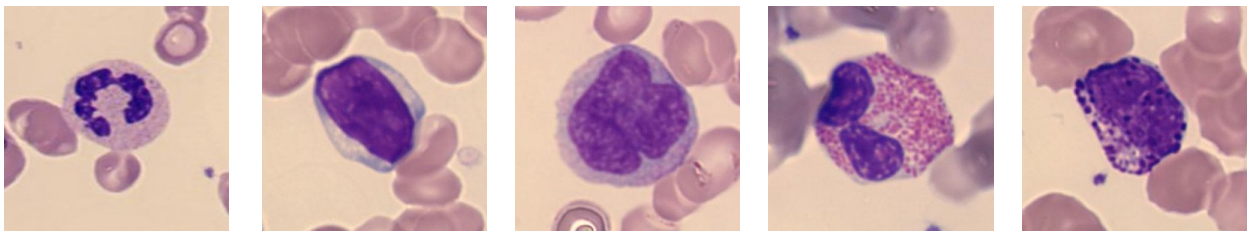
In this thesis, a novel method of data collection, processing, and transformation with the purpose of virtually staining blood cells is proposed. It leverages data-driven deep learning algorithms alongside an innovative hardware solution providing the necessary data to perform the virtual staining. The thesis will primarily focus on virtual staining of white blood cells, as they are more detailed and varied in appearance than other commonly occuring cells in blood. Furthermore, they are of significant importance in heamatological analysis and are vital in diagnosing patients.

1.1.1 Blood cells

Blood is composed of about 60% plasma and 40% blood cells. The blood cells are then divided into three different categories; white blood cells (WBCs), red blood cells (RBCs) and platelets. RBCs and platelets generally show far less variation both when it comes to shape and colour after staining than WBCs, and therefore this thesis will focus mostly on virtually staining WBCs. WBCs are usually divided into five distinct categories, each with their own role in the immune system. The categories are listed below, along with the approximate percentage of total WBCs [2]:

1. Neutrophils, 50-75%;
2. Lymphocytes, 20-40%;
3. Monocytes, 3-11%;
4. Eosinophils, <5%;
5. Basophils, <1%.

Fig. 1.5 shows typical WBCs of the different categories. The rarest of the WBCs, basophils and eosinphils, are also more distinct and differently colored than the other types: Eosinphils have large amounts of reddish granules (the small particles present in the cytoplasm around the nucleus), while basophils have large amounts of dark, almost black granules.



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

Figure 1.5: Examples of different types of WBCs [3].

1.2 Problem formulation

In this thesis the problem to be solved is defined loosely as:

- Find a transform function capable of altering images of unstained blood cells into images of stained blood cells, as if they had been chemically stained.

Essentially, the transformation model needs to approximate the chemical process of staining cells. This means it needs to identify different areas of the unstained image such as cytoplasm, WBC nucleus, RBC or background and color them accordingly. The problem can be setup mathematically as follows. Assume images of unstained cells exist as part of some distribution, $x_{unstained} \in \mathbb{P}_{unstained}$. The same is assumed of stained cells, where $x_{stained} \in \mathbb{P}_{stained}$. The sought after transform function can then be described mathematically through the following equation:

$$\mathcal{T}(x) = \hat{x}, \quad \text{where} \quad x \sim \mathbb{P}_{unstained},\ \hat{x} \sim \mathbb{P}_{stained},\ \mathcal{T} \sim C^0 \tag{1.1}$$

The function $\mathcal{T}(x)$ can be thought of as modeling the physical process of the chemical staining. Because of this, the problem becomes more difficult than the common colorization problem: Instead of using a one channel luminance image as input and generating plausible color channels for the output, the starting point is a three-channel color image of an unstained blood sample. As described in Sec. 1.1, the staining can even lead to structural differences between the unstained and stained samples, further differentiating the task from standard colorization. Trying to find the transform function $\mathcal{T}(x)$ is an ill-posed problem. To further illustrate why this is the case one can view Fig. 1.6. Here, the two distributions $\mathbb{P}_{unstained}$ and $\mathbb{P}_{stained}$ are represented as two separate sets. The transform function is then tasked using only the information from $\mathbb{P}_{unstained}$ to construct the cell pair in the larger set $\mathbb{P}_{stained}$. However, the cell in $\mathbb{P}_{stained}$ contains more information than the one in $\mathbb{P}_{unstained}$; information has been added in the form of physical, staining particles binding to the cell structures. As the chemical staining procedure itself is not entirely deterministic, and the same cell can look different after staining depending on many factors, several possible solutions exist that could be acceptable.

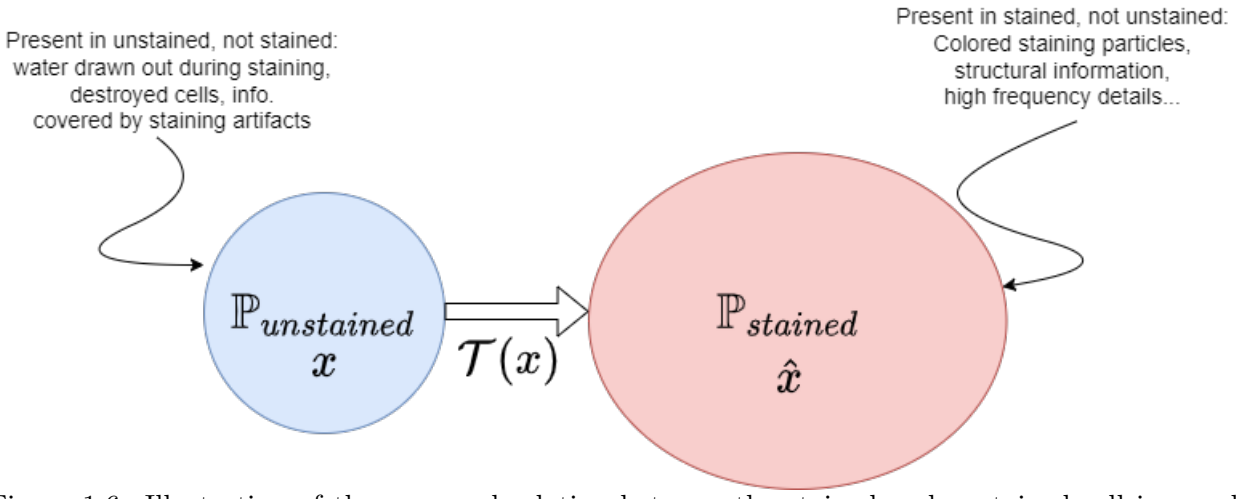


Figure 1.6: Illustration of the proposed relation between the stained and unstained cell image distributions. Using only the information present in $\mathbb{P}_{unstained}$, the transform function is tasked with reconstructing the appropriate cell in $\mathbb{P}_{stained}$.

The identification of areas to color in different ways could be seen as a segmentation problem, but segmentation alone is not enough to complete the task: the different areas of a WBC cannot be colored in one single hue, and should instead vary as in true stained images. The information present in an unstained brightfield image may not be enough to solve this problem, and so additional information needs to be gathered.

Previous papers regarding virtual staining of biological material have used different methods to gather this extra information, e.g using Quantitative Phase Information and lensfree holography [4] or Deep ultraviolet microscopy [5]. These techniques produce impressive results, but require less common and

expensive instrumentation. In this thesis, another solution is proposed that can be performed with a normal brightfield microscope with only a change of lighting.

1.2.1 Proposed solution

The solution proposed in this thesis is divided into two parts: the first part concerns providing increased information to allow more accurate virtual staining, and the second part concerns how to approximate the transform function $\mathcal{T}(x)$.

3D information through Point Light Source (PLS) illumination

The motivation behind the proposed solution is related to the need for more structural information about the sample to allow for accurate staining. If there is not enough information present in the input to deterministically stain the cells, then the transform function will be forced to approximate in its predictions. This may be fine if only done to a small extent, but if the approximations are too general then it may start misstaining cells to such a degree that it could lead to misclassification by analysts. A programmable LED-array is used for illumination of the sample instead of the standard halogen microscope lighting, allowing the sample to be illuminated from specific angles and directions as in [6]. In Fig. 1.7 a traditional digital microscope setup is shown, with a single light source. The light source used is decided by the specifications of the objective, more specifically its Numerical Aperture (NA). NA is directly related to the resolution of the image, and is defined as

$$NA = nsin(\theta), \tag{1.2}$$

where n (= 1.00 for air) is the refractive index of the medium in which the lens is working, and θ is the maximum half-angle of the cone of light that enters the lens.

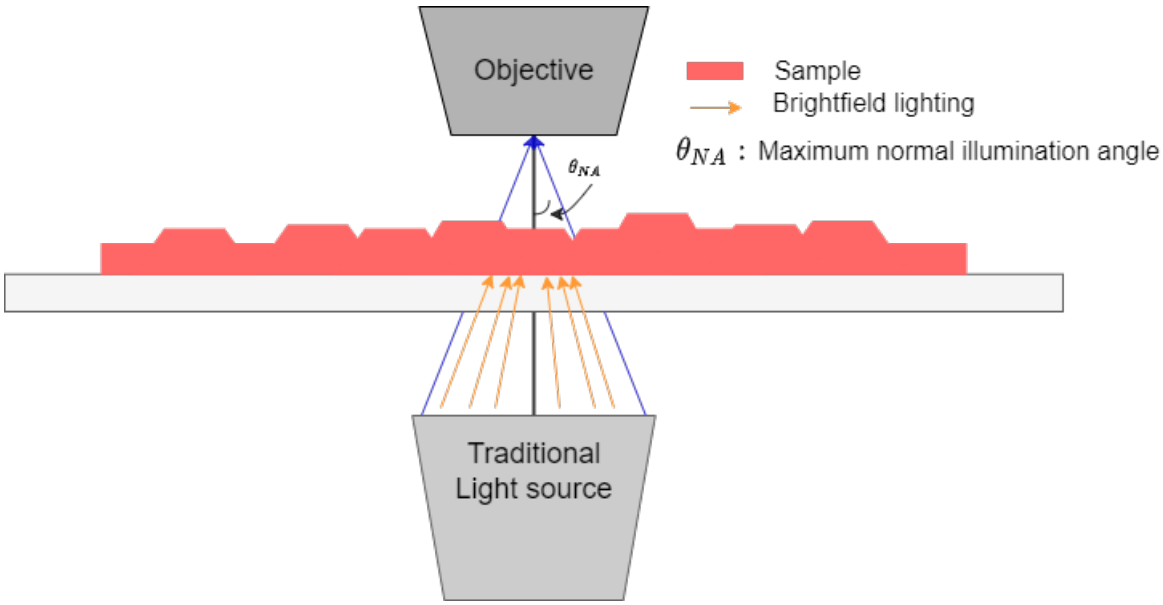


Figure 1.7: Illustration of traditional microscope lighting. The light source fills the entire inner light cone defined by the objective's NA at once, producing a standard brightfield image.

The traditional light source is then designed to produce light completely filling the cone defined by the maximum half angle θ, from now on referred to as θ_{NA}, as shown in Fig. 1.7. Consider what would happen if one shone a light at an angle higher than θ_{NA}? No light would be able to pass straight from the light source to the objective, so it would mean a much darker picture. However, some light could still be observed; that which has scattered of the sample. Microscopy using this type of lighting is usually referred to as Dark Field (DF) microscopy, and holds the potential to gather more information about the sample.

To allow for control over angle and direction of light the LED-array construction is introduced, illustrated in Fig. 1.8. The traditional single light source has been replaced by many small LEDs, distributed at different angles and directions toward the optical axis. This setup allows for much more flexible microscopy: lighting up all LEDs within the objectives θ_{NA} creates a classic brightfield image, lighting up LEDs outside the θ_{NA} allows for DF-images, and it allows for the use of highly directional light by using only one LED at a time.

The key use of this setup in this thesis is the highly directional light. When this light hits the sample from different angles, it will scatter into the objective lens in different manners depending on incoming direction and angle. Further, the increased maximum angle relative to the optical axis of the light means scattering patterns previously unseen will be observable. Furthermore, allowing each angle to be lit one by one instead of all at once may allow more structural information to be collected; when all are lit at once, different LED:s end up cancelling each other out in the digital image. Imagine lighting a persons face from all directions, removing the shadows: the image will look well lit and the face can be studied easily. However, if one had lit the face from an angle, the shadows cast over different portions of the face could be used to calculate 3D-information about it, like the length of the nose. This comes at the cost of occluding parts of the image in shadow, but to compensate for this one only needs to light the face from another angle, making sure that all portions occluded in the first image end up lit in another. The same thoughts can be applied to the proposed setup.

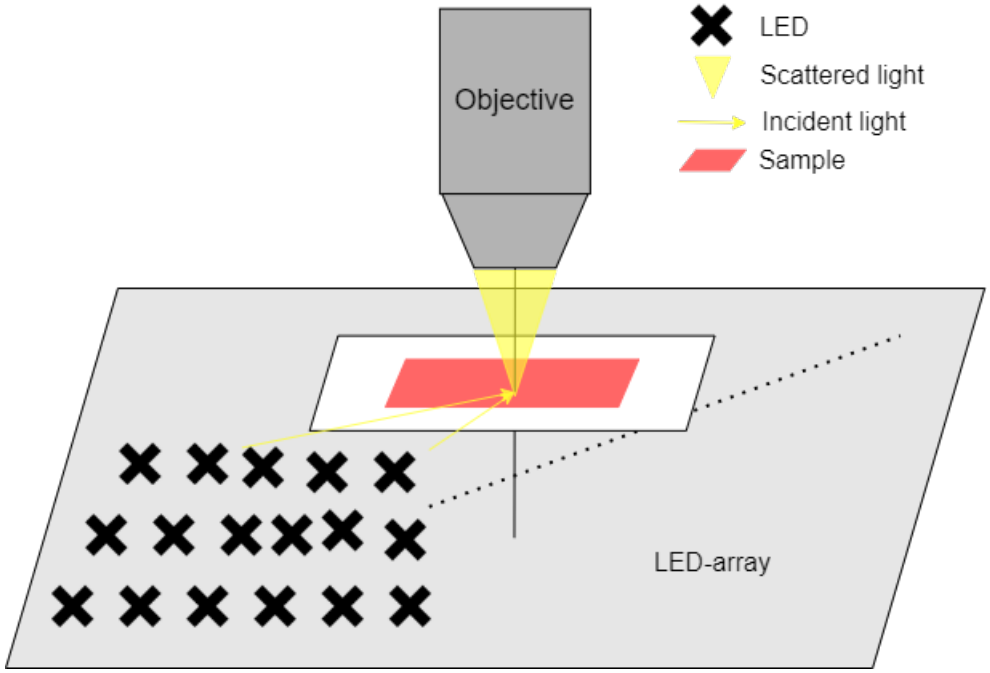


Figure 1.8: Overview of the LED-array setup.

For some examples of images collected with one LED lit at a time at different angles, see Sec. A.3.1 in the Appendix. A cross-sectional view of the setup is shown in Fig. 1.9 where the increase in possible angles to the optical axis is clear.

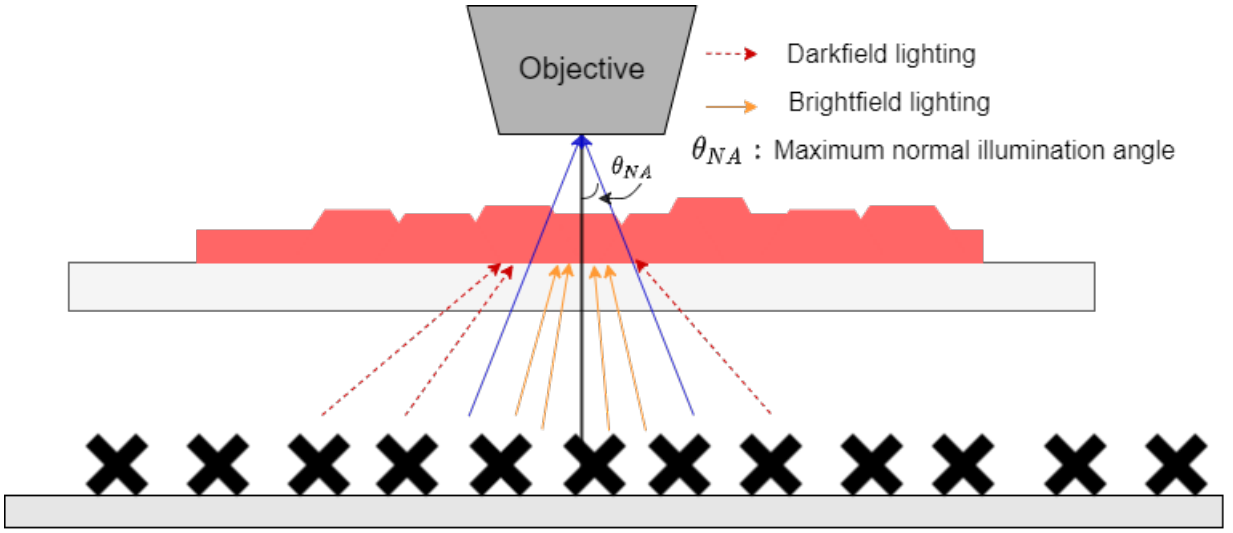


Figure 1.9: Cross-sectional view of the LED-array setup. This setup allows for directional light from many different angles, including above the θ_{NA} of the objective. By illuminating from different individual angles, information about the 3D-structure of the sample is encoded in the diffracting light that reaches the lens. The image is not to scale and the height differences present in an actual blood sample are very small.

Several computational imaging techniques that leverage the increased information from using similar programmable LED-arrays for illumination have been presented in previous works, allowing for algorithms to synthetically increase Numerical Aperture (NA), digitally refocus images, and perform aberration correction [7][6]. These results often require advanced spectral reconstruction algorithms and specific light sources, but provide an additional indication of the increased information that can be gathered using the images from multiple point light sources. In the case of virtual staining, the method can be thought of as attempting to make up for the missing chemical information usually used in chemical staining by providing increased optical information through the PLS images.

Artificial Neural Networks as Transform function

In this thesis, a neural network will be leveraged to extract the extra information from the PLS images and approximate a transform function $\mathcal{T}(x)$. Several different architectures are presented and tested, including feed-forward Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs). Both architectures have shown great performance in image transformation tasks, especially GANs. The proposed network architectures are presented in Sec. 3.2.

Chapter 2

Artificial Neural Networks

2.1 Introduction

Artificial Neural Networks (NNs) are a fundamental part of modern machine learning and have produced state-of-the-art results in a variety of different areas such as image classification, image generation and natural language processing. The application of NNs in the world of medical microscopy has also proven succesful, allowing super-resolution in microscopes [8], diagnosing of patients[9] and enhancing blurred images [10]. As the name implies, NNs are inspired by the way biological neural networks in the human brain process information. They are comprised of many simple, interconnected computational units called *neurons*, capable of passing information between each other through a network of connections, mirroring the interconnected pathways of the neurons in our brain. Neurons in NNs are often called nodes. The simplest possible neural network is comprised of one single node, which models the behaviour of a biological neuron. Such a network is sometimes referred to as a perceptron, and was first introduced in 1958 by Frank Rosenblatt [11]. It was presented as a probabilistic model of computation and memory storage in the brain, and its variations are still a basic building block of modern NNs. The classic perceptron is pictured in Fig. 2.1 and is defined by the following equation:

$$y = f(\sum_{i}^{n}(w_i x_i) + b), \tag{2.1}$$

It takes an n-dimensional vector $\mathbf{x}$ as input, computes a weighted sum according to the weights $w_1, w_2 ... w_n$, adding a bias term b and feeds the result through an activation function $f(x)$ which gives the final scalar output y. In the classic perceptron the activation function is a step-function, which produces a y-value of either 1 or 0 depending on whether the weighted sum reaches a set threshold or not.

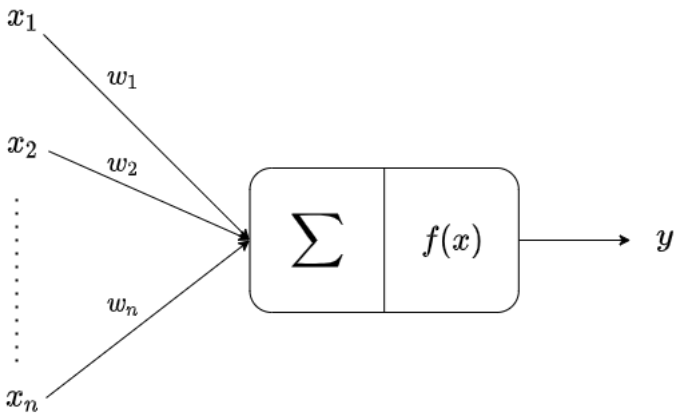


Figure 2.1: Illustration of the perceptron. A neuron takes an n-dimensional vector as input and computes a one-dimensional output y through summation and an activation function $f(x)$.

The perceptron model constitutes a linear classifier, and as such it can only solve linearly separable tasks, where the data can be separated by a hyperplane. To allow for more complex tasks to be solved, connections between several of these nodes needs to be leveraged. This can be achieved by adding layers to create a multi-layer perceptron (MLP), and introducing non-linear activation functions between the layers. An example of an MLP network is pictured in Fig. 2.2. It has an input layer of size 2 (meaning it contains two nodes), n *hidden* layers of size 4, and an output layer of size 2. NNs with one or more hidden layers are considered deep neural networks (DNNs), and constitute the basis for Deep Learning (DL), the practice of using DNNs for machine learning.

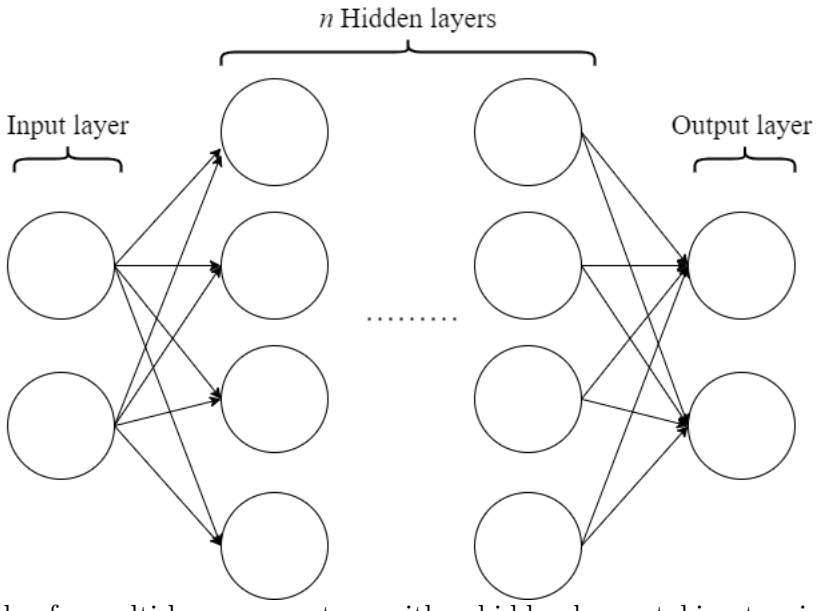


Figure 2.2: Example of a multi-layer perceptron with n hidden layers taking two input variables and computing two output variables.

The training of a NN consists of learning the appropriate network weights in order to solve a given task. In practice this is achieved through minimization of a *loss function*, described in Sec. 2.2. During this process, the gradients of the weights of the NN are calculated through a method called *backpropagation* [12] and then used to guide the optimization of the NN (discussed further in Sec. 2.3). In order for backpropagation to work efficiently, the choice of activation function becomes important.

2.1.1 Activation functions

Some common activation functions are presented in Tab. 2.1. The simplest activation function is the step-function, a function which is either 0 or 1 depending on some pre-defined threshold. This was the activation function suggested by Rosenblatt for the perceptron, and was inspired by the all or nothing activation of a biological neuron. However, the gradient of this function is almost always zero which can be problematic during training. This problem can be partially negated by using functions with a softer threshold such as the logistic (also called sigmoid) function or the hyperbolic tangent function (tanh). Both of these functions are squashing functions, accepting any input from $\mathbb{R}$ and outputting a number in a limited range ((0,1) for the logistic function and (-1,1) for hyperbolic tangent function). This feature can sometimes be beneficial. As an example the logistic function is often used as the final activation of a NN in the case of binary classification, as the output can then be interpreted as the probability that the input is one of two classes. However, the squashing also means that the functions return small gradients for values larger than ≈ 10. In deeper NNs this gives rise to the well-known problem of *vanishing gradients*. In short, the vanishing gradient problem arises because of the way backpropagation finds the gradient of the NN. The derivatives of the network are found layer by layer starting from the final one. Using the chain rule the derivatives of each layer are then multiplied down the network to compute the derivatives of the inital layer. Imagine then that you have n layers with logistic/tanh activation functions. There is a risk that n very small derivatives are multiplied together, leading to an exponentially decreasing gradient during propagation to the initial layers. The small gradient will cause the weights of initial layers to be updated very slowly, which can lead to slow convergence and poor performance.

In order to facilitate training of deeper networks the ReLU function was introduced. It is a piecewise linear function, preserving many beneficial aspects of linear functions while actually being non-linear (allowing for complex modeling). The function is either zero, if the input is less than zero, or identity if the function is larger than zero. This means that it avoids the vanishing gradient problem by remaining linear and unbounded for positive inputs. It is also much faster to calculate than the logistic or tanh activations, as it does not require the exponential function to be computed. The simplicity of the function does however have a downside. Assigning all negative inputs to zero can give rise to the dying ReLU problem [13]. This happens when the inputs to many neurons get stuck in the negative range meaning the neurons will output zero, the gradients will fail to flow backwards during backpropagation and the weights will stop updating. Once dead it also becomes unlikely to recover due to the lack of slope in the negative input range.

Different variations of solutions exist to the dying ReLU problem. One such solution is the Leaky ReLU. For positive inputs it is identical to ReLU, but for negative inputs it replaces the flat part with a small slope αx. This means the gradient will always be non-zero, and the network will be able to recover when a normal ReLU would lead to death. A common choice of α is 0.1.

Name	Formula
Step-function	$f(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{otherwise} \end{cases}$
Logistic/sigmoid function	$f(x) = \frac{1}{1+e^{-x}}$
Hyperbolic tangent	$f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
ReLU	$f(x) = \begin{cases} x & \text{if} \quad x > 0 \\ 0 & \text{otherwise} \end{cases}$
Leaky ReLU	$f(x) = \begin{cases} x & \text{if} \quad x > 0 \\ \alpha x & \text{otherwise} \end{cases}$

Table 2.1: Example activation functions.

2.2 Loss functions

The loss function is the key component through which NNs learn: it provides an evaluation of how well your current algorithm models the given data. If the models predictions are poor, the loss function should output a higher value, and if they improve, it should output a lower value. This indicates to the model which weight adjustments increase performance and provides a direction of improvement. The loss function is generally defined as

$$\mathcal{L}(y, f(x, w)), \quad \mathcal{L} : \mathbb{R}^{\mathbb{N}} \longrightarrow \mathbb{R} \tag{2.2}$$

where $f(x, w)$ is the (N-dimensional) output of the NN given the input x and the current weights w, and y is the target output. With a well formulated loss function the training of the NN is transformed into a minimization problem,

$$\min_{w} \mathcal{L}(y, f(x, w)) \tag{2.3}$$

which can then be optimized through some optimization method. The optimal loss function will vary from problem to problem, and choosing a suitable one is crucial in order to guide the NN to actually solve the intended problem. Loss functions can generally be divided into two categories, *Regression losses* or *Classification losses*. The main difference is that in classification, the prediction is limited to assigning a class from a finite set of classes to the sample, while in regression the predicted output is a continuous value (e.g. a pixel color value). As an example, a common choice of loss function for classification problems is the crossentropy loss, or log-loss [14]. For a binary classification problem, the task is defined as accurately assigning one of two classes to the datapoint, e.g. cat or dog. In practice the class labels cat and dog are mapped to the values '0' and '1', and the loss function is defined as

$$\mathcal{L} = -\frac{1}{N}\sum_{i}^{N} \Big(l_i \log(\hat{y}_i) + (1 - l_i)\log(1 - \hat{y}_i) \Big) \tag{2.4}$$

where $\hat{y}_i$ is the output of the NN, l_i is the target label ('0' or '1') and N is the number of samples in the training batch. When the true label is 1 ($l_i = 1$), the second half of the function disappears and the function is minimized when the neural network reaches the desired output,

$$\min_{w} \mathcal{L} = \min_{w}(-\frac{1}{N}\sum_{i}^{N} \Big(\log(\hat{y}_i) \Big)) \longrightarrow \hat{y}_i = 1. \tag{2.5}$$

When the true label is 0 ($l_i = 0$) the first half of the function disappears instead, and the loss is minimized when the neural network outputs $\hat{y}_i = 0$:

$$\min_{w} \mathcal{L} = \min_{w}(-\frac{1}{N}\sum_{i}^{N} \Big(\log(1 - \hat{y}_i) \Big)) \longrightarrow \hat{y}_i = 0. \tag{2.6}$$

In the two equations above it is implied that the NN output $\hat{y}_i$ is dependent on the weights w, $\hat{y}_i(w)$, for a given input x. These equations show that through the cross entropy loss function the network is encouraged to match its target value.

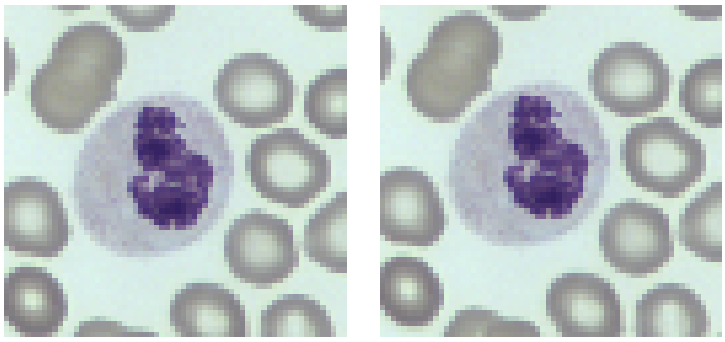
The main task considered in this thesis is one of image transformation towards a known target image. Since we have a ground truth target to aim for, a regression loss can be used to guide the network towards the target. Common regression losses for image transformations are Mean Average Error

(MAE, l_1) and Mean Square Error (MSE, l_2). They ensure a close similarity between the generated images ($\hat{y}$) and the ground truth (y) by minimizing the mean pixel-wise norm between them:

$$MAE : \mathcal{L}_{l_1} := \frac{\|y - \hat{y}\|_1}{N} \tag{2.7}$$

$$MSE : \mathcal{L}_{l_2} := \frac{\|y - \hat{y}\|_2}{N} \tag{2.8}$$

Here, N is the number of pixels times the number of channels. In this thesis only the l_1 loss will be used as it has been shown to produce less blurry results in image generation tasks [15]. While the simplicity of the l_1 and l_2 losses is nice, it also gives rise to problems. As both methods compute their metrics based on pixel values, they do not accurately capture perceptual differences between images [15]. The losses also become very sensitive to misalignment between the ground truth and the input data, as a small shift in pixels may result in large losses. As an example, view the images in Fig. 2.3 below. Two images of cells are shown. They are copies of eachother, except for the fact that the right image is shifted two pixels down. Most human observers would state that the images are very similar, as almost all perceptual aspects of the images are equal. However, computing the $\mathcal{L}_{l_1}$-loss between them results in a loss of 120.6, in a possible range of 0-255, meaning the mean error is almost 50% of the maximum possible despite the close resemblance of the images. To account for differences such as this, a new form of loss needs to be introduced, called the *perceptual loss.*



(a) Original (b) Translated 2 pixels

Figure 2.3: a) Original image of WBC. b) Same image as in a) but translated two pixels down. The two images are perceptually very similar, but the $\mathcal{L}_{l_1}$-loss (≈ 120) between them is almost half of the maximum possible.

2.2.1 Perceptual loss

A *perceptual loss function* is a loss function specifically engineered to target perceptual information not captured in traditional pixel-wise losses. There are many ways to construct a perceptual loss function, and a succesful approach has been to leverage a pre-trained fixed *loss network* ϕ. In [15] Johnson *et al* suggest using an image classification network as the loss network, specifically the VGG network pre-trained on the ImageNet dataset. The key motivation behind these methods is that convolutional neural networks pre-trained for image classification have already learned to encode the perceptual and semantic information to be measured in a loss function. Johnson *et al* present impressive results using different perceptual losses both for style-transfer (transforming the style of an image to that of a target, e.g. from realistic to painting) and super-resolution (low resolution input to high resolution output). Other variants of the loss exist, but the general structure is often the same: a pre-trained network is allowed to predict on the generated and target image, the feature maps of one or several intermediate layers are extracted and a norm is computed between the outputs. If ϕ_j is the chosen layer in the loss network ϕ to extract from, y is the target image and $\hat{y}$ is the generated image a general perceptual loss can be defined as

$$\mathcal{L}_p = \|\phi_j(y) - \phi_j(\hat{y})\|. \tag{2.9}$$

An illustration of this is shown in Fig. 2.4. In [15] Johnson *et al* present a *Feature Reconstruction Loss* using the euclidian distance (l_2-norm) between the outputs of an intermediate layer extracted *after* activation. Further work in the realm of super-resolution have shown that it may be more advantageous to extract the outputs of the intermediate layer *before* the activation instead [16]. The motivation behind the change is that for a deep network, the activated features become very sparse, providing weaker supervision to the model. In [16] Wang *et al* also show that using features before activation caused the reconstructed brightness of images to be more consistent with that of the ground-truth.

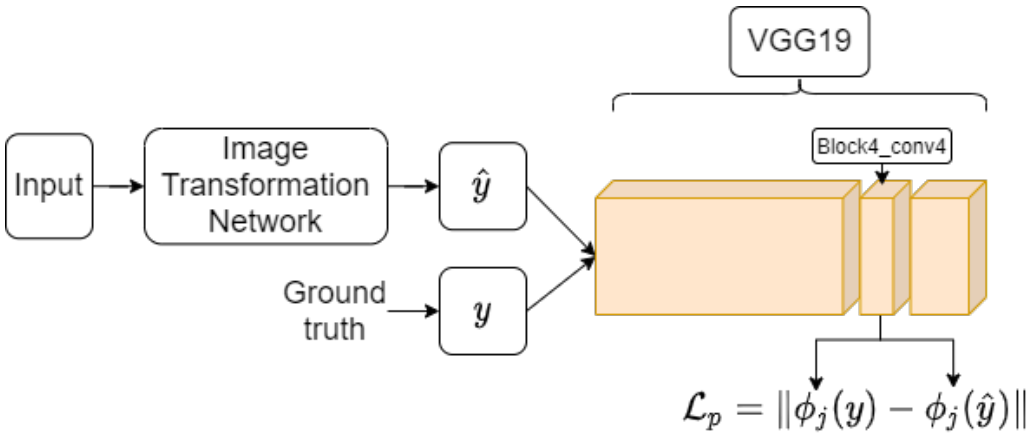


Figure 2.4: An illustration of a perceptual loss using the VGG19-network, extracting the layer Block4_conv4.

2.2.2 Fourier domain loss

While the previous section describes the most common way to define a perceptual loss, other perceptual losses exist as well. An example of that is a Fourier space loss, where the input image and the ground truth are transformed to the frequency domain before being compared. An advanced version of a Fourier loss was recently developed to facilitate more efficient networks in super-resolution by Fuoli *et al* [17]. They state that in order to recreate a high-resolution image from a low-resolution one, one must recover the missing high-frequency information. They further argue that in Fourier domain these missing frequencies can be clearly separated, in contrast to spatial domain. Furthermore, due to the nature of the Fourier transform, the loss is directly calculated on the global frequency components therefore providing global guidance to the model, as opposed to local pixel-based losses in spatial domain.

In this thesis, a simple version of a Fourier loss is investigated to provide an extra loss-signal to the network. It is a less advanced than the one presented in [17], but the arguments for using a Fourier domain loss still apply. The loss is calculated by allowing a NN to generate a sample, in this case a real-valued image, calculate the FFT of both the generated sample and its paired ground truth, and take the absolute value of both. The output real-valued matrices are then compared through an MAE-loss creating the final loss signal,

$$\mathcal{L}_\mathcal{F} = \||\text{FFT}(y)| - |\text{FFT}(\hat{y})|\|. \tag{2.10}$$

The Fourier loss presented in this thesis was designed and implemented before reading the paper by Fuoli *et al*. Their paper [17] was not discovered until the end of the project, and the more advanced supervision and GAN Fourier losses proposed there were not implemented due to time restrictions. They do however provide interesting avenues for further improvements.

2.3 Optimization

Many different optimization methods exist for neural networks with different strengths and drawbacks. They all share a common goal, to minimize a given loss function. Most modern methods stem from the idea of *Gradient Descent* (GD) [18]. In gradient descent an objective function (e.g. a loss function) is minimized by updating the parameters of a model in the opposite direction of the gradient of the objective function w.r.t the parameters (e.g. the weights of a NN). If the loss function is thought of as an *optimization landscape* gradient descent can be understood as walking down the steepest descent until a valley is reached. For classic gradient descent the gradient of the loss function w.r.t the weights is computed for the entire training set before updating according to eq. 2.11.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \Delta_{\mathbf{w}_t} f(\mathbf{w}_t), \tag{2.11}$$

Here, $\mathbf{w}_t$ represent the parameters to be updated at timestep t, α is the learning rate, $\Delta_{\mathbf{w}_t}$ is the gradient operator w.r.t to $\mathbf{w}_t$ and $f(\mathbf{w})_t$ is the objective function to be minimized. The hyperparameter α decides how large the update should be, and can be both constant or adaptive during optimization.

Computing the gradient for the entire dataset before updating can be computationally expensive and redundant for large datasets, as it recomputes gradients for similar examples before each parameter update. To overcome this redundancy and improve the speed of optimization a variation of gradient descent called Stochastic Gradient Descent (SGD) [19] can be used. In it, the update rule is altered to perform a parameter update for each training example x^i and label y^i, see eq. 2.12. The training examples are chosen randomly (i.e., stochastically) from the training set.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \Delta_{\mathbf{w_t}} f(\mathbf{w_t}; x^i, y^i). \tag{2.12}$$

SGD is much faster than vanilla gradient descent, but may have trouble finding the exact minimum. Combining the best parts of GD and SGD one can construct mini-batch gradient descent: it computes the average gradient for a mini-batch of n training examples before performing an update. This reduces the variance of parameter updates while at the same time making use of highly optimized matrix operations, making it very efficient. In neural network training mini-batch gradient descent is often the algorithm of choice, and the term SGD is frequently used whether mini-batches are employed or not. The update rule for mini-batch GD is shown in eq. 2.13.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \frac{1}{n} \sum_{i}^{n} \left(\Delta_{\mathbf{w}_t} f(\mathbf{w}_t; x^i, y^i) \right) \tag{2.13}$$

While the mini-batch SGD algorithm works well for some problems, there are still a few challenges in employing it. For example, it can be difficult to choose a proper learning rate α. If chosen too small the convergence may be slow, and if chosen too large it can cause the loss function to fluctuate around the minimum or start to diverge. One can use learning rate schedules to try to adjust the learning rate during training, but these schedules need to be defined before training and cannot adapt to a dataset's characteristics [20]. The learning rate is also applied equally to all parameter updates, which may not be optimal.

A method that attempts to solve the issues of mini-batch SGD is Adam [21]. The name comes from *Adaptive Moment Estimation*, and refers to two properties of the Adam algorithm: individual adaptive learning rates for parameters and an added *momentum* [22] term. In [21] Kingma and Ba describe the method as combining the advantages of two other extensions of SGD. Like Adagrad [23], it is well suited for problems with sparse gradients, and like RMSProp [24], it performs well on online or non-stationary problems. Adam has been empirically shown to achieve good performance in stochastic optimization while requiring little hyperparameter tuning and being computationally efficient even when compared with the latest methods [25]. In his technical report from 2016 [18], Sebastian Ruder

recommends the Adam optimization method as the go-to method for training NNs. In this thesis, all NNs are trained using the Adam method.

2.4 Generalization, Overfitting and Regularization

When training machine learning models we attempt to fit a model to some dataset, in the hopes that the fitted model will be able to work when shown new data of the same type. In other words, we want our model to be able to *generalize* its solution and accurately interpret previously unseen data. Several factors can affect how well a model generalizes, like the model complexity, the number of training epochs and the size of the dataset. As an example we can consider constructing a polynomial fit of some degree to data sampled from a sine function. To represent the inaccuracies inherent in any real world dataset, some noise is added to the samples. Specifically, for this example, the goal is to accurately reconstruct the underlying pattern of the original sine wave without the noise. Three polynomial fits of degree 1, 3 and 9 estimated by minimizing the mean squared error are shown in Fig. 2.5.

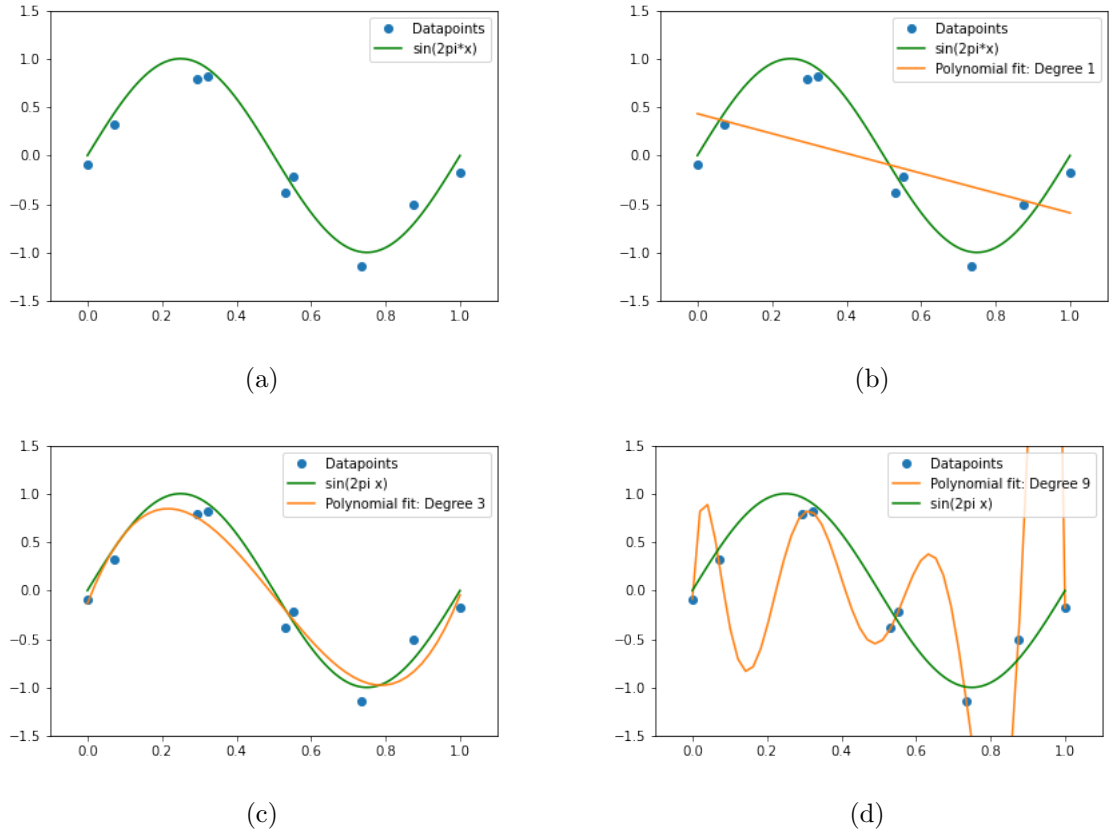
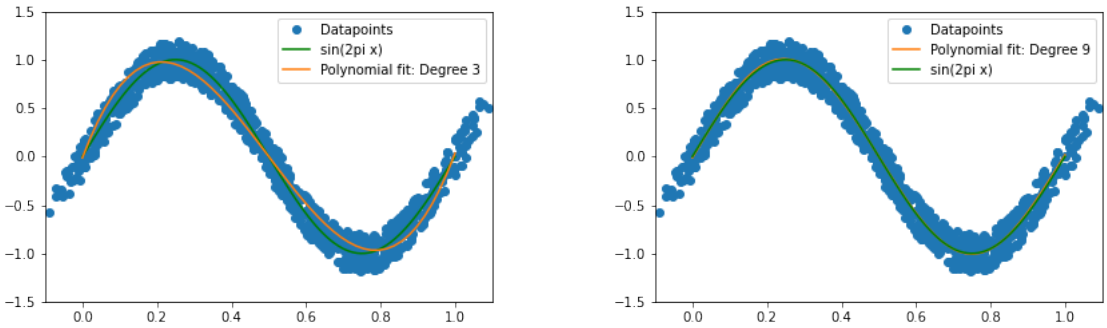


Figure 2.5: Polynomial fits of different degrees to datapoints sampled from a sine function with some added noise. The polynomial fit of degree 9 perfectly models the datapoints, but fails completely in reconstructing the underlying sine function.

The datapoints used as input are shown as blue dots and the true $sin(2\pi x)$ function is shown in green. In Fig. 2.5 b), the polynomial fit of 1 degree becomes a line, which of course fits poorly with the original sine wave. The model simply lacks the required complexity to reconstruct the original data. This problem is often referred to as *underfitting*, and can often be solved by adding complexity to the model. Increasing the degree of the fitting polynomial to 3 gives much better results, as shown in Fig. 2.5 c). But increasing model complexity can also lead to problems. When the fitted polynomial is of degree 9 as shown in Fig. 2.5 d), the model accurately predicts all of the datapoints from the training set, but fails to reconstruct the original function. The model ends up *overfitting* to the training data, and fails to generalize. There exists many techniques to reduce overfitting of a model. One of them

is increasing the number of datapoints: building on the previous example, the polynomial of degree 9 becomes very accurate if we have 1000 datapoints instead of only 9 (see Fig. 2.6).



(a) (b)

Figure 2.6: Same polynomial fit as in Fig. 2.5 using more datapoints. Here, the fit of degree 9 reconstructs the underlying sine function better than that of degree 3.

DNNs constitute complex models by nature of their design: they often have thousands or even millions of learnable parameters. Because of this they are often prone to overfitting, and require a large amount of data to train well. In order to catch an overfitting model one can evaluate its generalization power by splitting the available data into two parts, the training set and the validation set. The validation set is kept separate from the model during training, and is used to evaluate the models performance on unseen data between epochs of training. The training set is then the data actually used to train the model. By monitoring the models performance on both the training and validation sets its generalization capability can be analyzed: if the model generalizes well the gap between the training and validation loss is small, and if the model does not generalize the gap will be large. A sketch illustrating the training and validation losses behavior as a function of the number of training epochs is shown in Fig. 2.7. The fractions of data to use for validation and training sets depend on the problem, but generally the validation set is much smaller than the training set as training on more data is almost always beneficial.

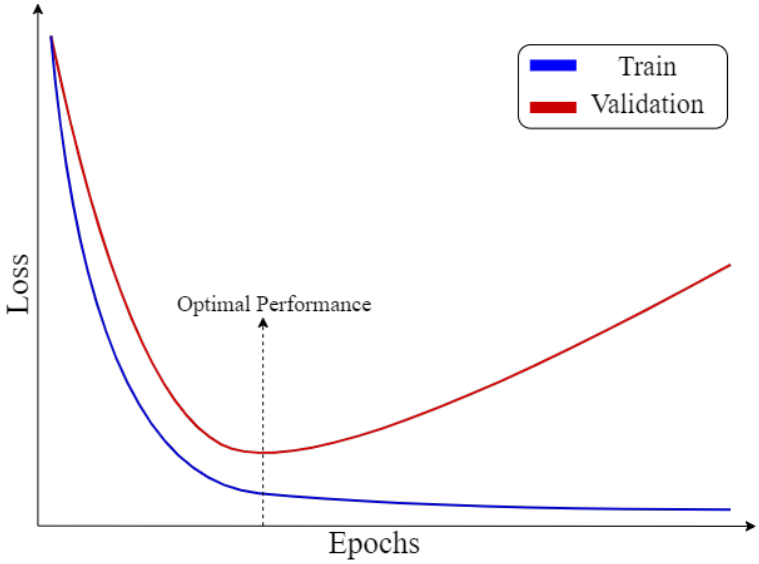


Figure 2.7: Illustration of loss behavior during training.

As training progresses both the training and validation losses typically decrease but at some point they end up diverging: the validation loss starts to increase while the training keeps decreasing. After this point, the model starts to memorize the examples in the training set instead of learning the underlying

pattern, similarly to the example in Fig. 2.5 d) above. This behavior presents a simple idea to prevent overfitting called *early stopping*, where the training of the model ends when the generalization performance deteriorates (for example, when the validation loss starts increasing). Techniques used to prevent overfitting are often referred to as *regularization* methods, of which early stopping is a simple example. Other commonly used methods include dropout [26] and batch normalization (BN) [27].

2.5 Convolutional Neural Networks

One of the most successful types of NN for computer vision tasks are Convolutional Neural Networks (CNNs). The main operation of these networks is the convolution. Given an input matrix A, a convolution is performed as follows: first, one defines a kernel matrix K (sometimes called a filter) of a certain size with certain weights. This matrix will then be slid across the input matrix A, computing a matrix multiplication between the filter and the input for every visited position. The operation can be defined mathematically as

$$\mathcal{F}(i,j) = (A * K)(i,j) = \sum_{m=1}^{M} \sum_{n=1}^{N} A(i,j)K(i-m, j-n) \tag{2.14}$$

where the output $\mathcal{F}$ is referred to as a *feature map*. When the input matrix A represents an image, the convolution can be interpreted as sliding a window over the image, collecting the view through the window and combining it into a representative *feature*. In classical image processing, convolutions are often used for tasks such as edge detection, sharpening, or blurring of images. In these cases the values, or weights, of the filter are hand-engineered before application to achieve a specific purpose. An example of applying an edge detecting kernel is shown in Fig. 2.8. Convolutions are especially effective at analyzing images as they are capable of catching local dependencies between pixels. The dependencies caught by the filter depends on its size; a bigger filter takes more pixels into account for each computed feature, giving it a larger *receptive field*.

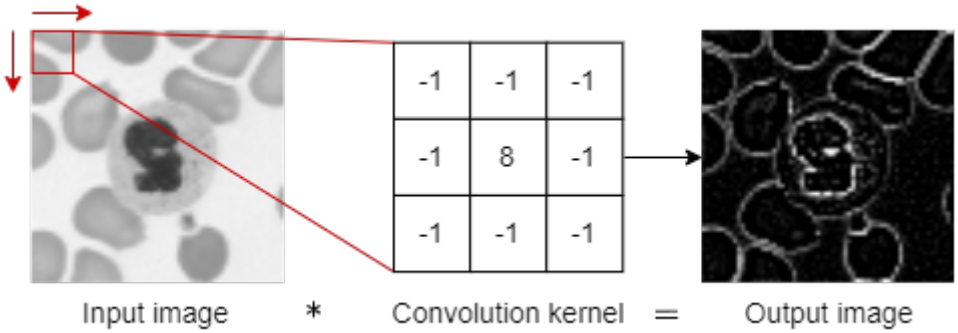


Figure 2.8: Example demonstrating application of an edge detecting kernel.

When using convolutions in NNs, the weights of the kernel are not pre-determined. Instead, they are learned through training with some loss function and backpropagating the gradients to update the weights. In addtion to the filter size and weights there are a few more components that affect a convolutional layer. When initializing the layer, one can define the number of filters to use, creating the depth of the layer. More filters means more feature maps, and potentially more extracted information. The input data can also be *zero-padded* around its boundaries, to preserve information around the edges of the image and to maintain the same spatial dimensions between the feature maps and the original input image. One can also introduce *stride*, which decides the step-size of the sliding kernel over the input: stride = 1 means the kernel is slid one pixel at a time, stride = 2 means the kernel is slid two pixels, etc.. A larger stride is sometimes desired to reduce overlapping, and since fewer features are computed the number of parameters is also reduced.

2.6 Residual Networks

Residual networks (ResNets) were first introduced in [28]. They aimed to resolve the problem of training very deep networks, which was not possible using only normal regularization techniques such as BN. They proposed a new type of basic building block for deeper networks, called a residual block. A sketch of a residual block is shown in Fig. 2.9.

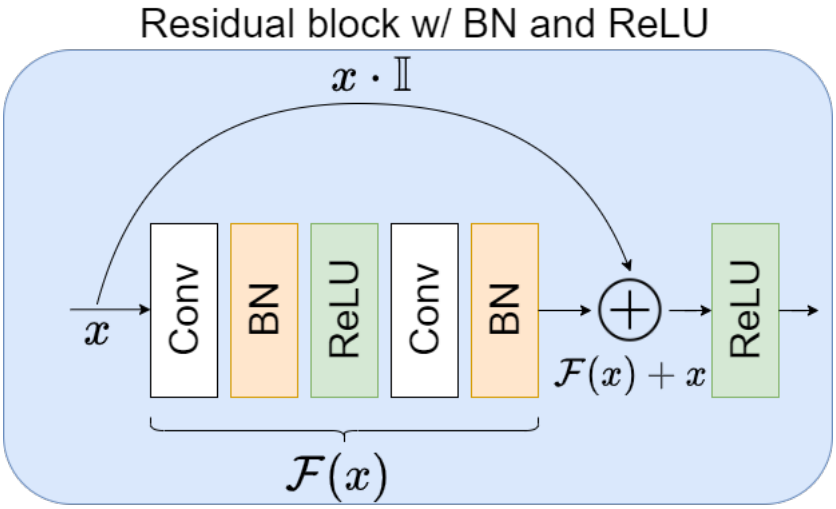


Figure 2.9: Possible example of a residual block. Note that the layers in $\mathcal{F}(x)$ could be different, the structure shown here is the one used in SRGAN [29].

Each residual layer is given an input x which is then passed through some layers $\mathcal{F}$ (often convolution/BN/ReLU). Normally, the output $\mathcal{F}(x)$ would be what is sent forward to the next part of the network, but in a residual network the original input x is added back to $\mathcal{F}(x)$ through an identity skip connection, giving the total output $\mathcal{F}(x) + x$. Since x is given the model only learns the *residual* $\mathcal{F}(x)$, creating what is called *residual learning*.

2.7 Dense networks

Dense networks were a continuation of the residual network style of increased short connections between layers. In the introducing paper [30], they define a Dense Convolutional Network (DenseNet), consisting of dense blocks. The blocks connect each layer to every other layer in the block in a feed-forward fashion. A schematic view of a dense block is shown in Fig. 2.10.

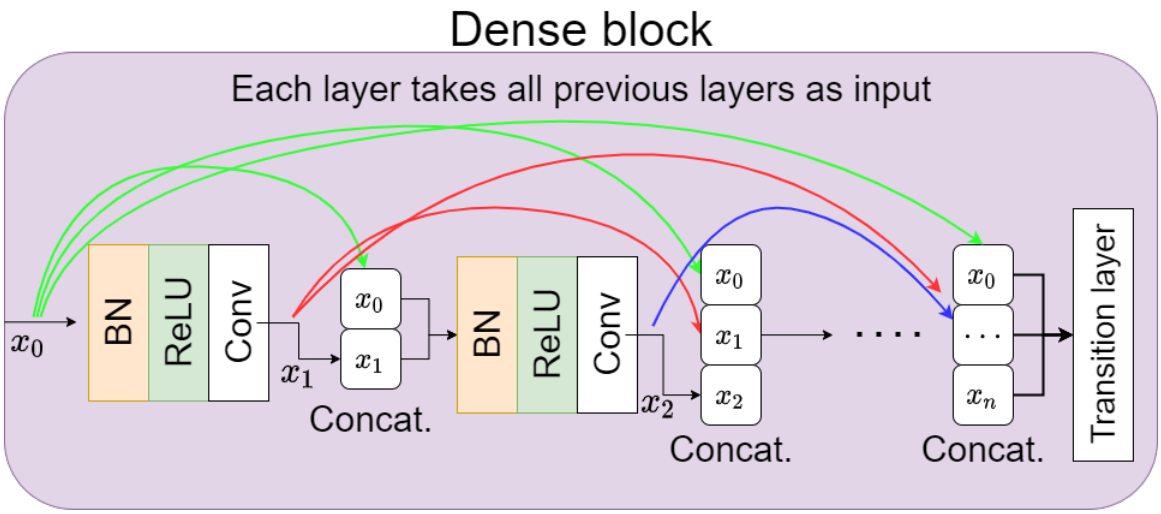


Figure 2.10: An example of a dense block. The BN/ReLU/Conv structure is commonly used but could be replaced by other convolutionally based layers. It is however important that the feature map size remains fixed within the dense blocks (meaning no 2-strided convolutions or downsampling).

As in the residual block, the input to the first layer is passed over to other layers through identity skip connections. However, in a dense block each layer is connected to every other layer that is ahead of them, instead of just the next one. This means that each layer will obtain additional inputs from all of the preceding layers and passes on its own feature-maps to all following layers. In contrast to ResNet, the skip connected features are not combined through summation, but are instead concatenated.

The authors show many compelling advantages of DenseNets, including: alleviation of the vanishing gradient problem, strenghtening feature propagation, encouraging feature reuse, and substantially reducing the number of learnable parameters.

2.8 U-net

The U-net is an encoder-decoder architecture originally designed for biomedical image segmentation tasks [31]. It has proven to be a very successful architecture for general image transformation tasks, such as colorization [32], virtual staining [4] and more general image-to-image transformations [33]. The U-net is a convolutional neural network, starting with an encoder path. This encoder path downsamples and translates the input (often an image) to features. This is then followed by a decoder path, that takes the encoded features and upsamples them to more high-level information. Encoder-decoder architectures are often used for image transformation, as it becomes simple to input an image, extract features during encoding, and then leverage those features to produce a new image.

Another important aspect of the U-net is that it employs *skip connections* between the encoding and decoding blocks of equal size. This allows some information to skip directly from encoder to decoder, which could be beneficial if the output shares structure or features with the input as is common in image transformation.

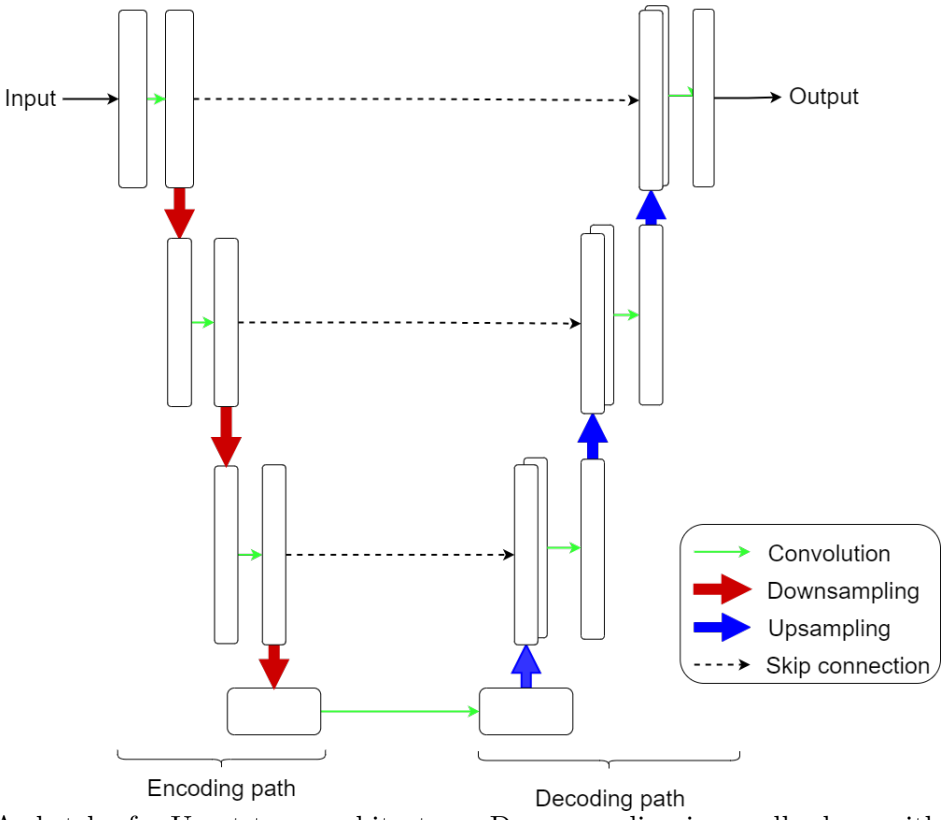


Figure 2.11: A sketch of a U-net type architecture. Downsampling is usually done with max-pooling layers or strided convolutions, and upsamling is done either through strided transposed convolutions or image resizing followed by unstrided transposed convolution.

2.9 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are an algorithmic machine learning architecture designed to increase the performance of deep generative models. First introduced by Ian Goodfellow *et al* in 2014 [34], it consists of two separate NNs: a generator G and a discriminator D. The networks are constructed with diametrically opposed objectives, and compete against each other during training as adversaries.

The generator is a *generative* model and the discriminator is a *discriminative* model. Given a training set of samples from an unknown data-generating distribution p_{data}, the generator attempts to learn an estimate of that distribution, called p_{model} [35]. It can then use its estimated distribution to *generate* new data instances that (if trained well) could have originated from the original p_{data} distribution. The discriminator is then tasked with distinguishing whether a given sample is from the true data distribution or the model distribution, and a loss is computed based on its decision. The two models iteratively take turns generating/judging samples, and as training progresses the generator creates better and better samples. At the same time, the discriminator becomes more adept at distinguishing between the model and data distributions. A common analogy for the two networks is that the generator is like a counterfeiter, trying to produce fake money. The discriminator then takes on the role of the police, trying to detect the counterfeit currency. As the counterfeiters increase the quality of their fakes, the police are forced to learn more advanced methods of detection in order to catch them. The counterfeiter can only succeed by fooling the police, making them fail their task, while the police can only succeed by detecting the counterfeit money, making the counterfeiters fail their task. The two networks end up playing a game, where they can only increase their own performance by reducing their opponents. This game could, in theory, continue until the generator perfectly models the real data distribution, and then the fake currency is indistinguishable from the real currency in the analogy.

A more formal definition of the GAN process is given in [34]. Goodfellow *et al* define a prior on

input noise variables $p_z(\mathbf{z})$ and represent a mapping to the data space as $G(\mathbf{z}, \theta_g) = G(\mathbf{z}) = \hat{y}$. Here, G is a NN with weights θ_g and $\hat{y}$ is a newly generated data sample (or the fake money in the analogy). The input noise is drawn from some random distribution (e.g. $p_z(\cdot)$ = Gaussian noise function). An accompanying NN is defined $D(x, \theta_d)$, which outputs a single scalar. $D(x)$ represents the probability that x came from the p_{data} distribution rather than the p_{model} distribution (i.e., the police attempting to discern whether a banknote is real or not). A loss is calculated based on $D(x)$ and is then backpropagated to update the weights of G and D. A sketch of the GAN framework is shown in Fig. 2.12.

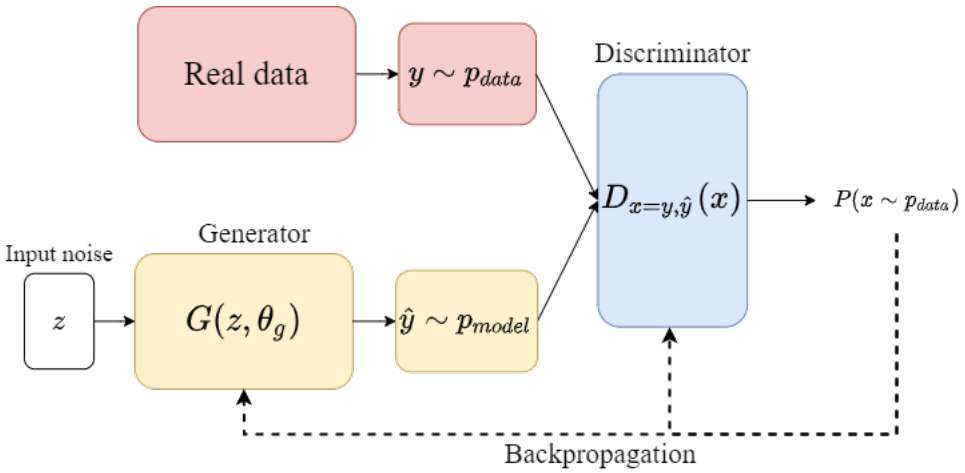


Figure 2.12: Schematic illustration of the GAN training framework.

2.9.1 Original implementation: Defining the game

In order to pit the two networks against each other the game they will play needs to be defined. The first player to consider is the discriminator. The discriminator is tasked with distinguishing real from fake; in other words it is a binary classifier. As such, its loss is defined as

$$\mathcal{L}_D = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}}[\log D(y)] - \frac{1}{2}\mathbb{E}_{G(z)\sim p_{model}}[\log(1 - D(G(z)))], \tag{2.15}$$

which is just the standard cross-entropy loss traditionally used for binary classifiers with a sigmoid output (see eq. 2.4 in Sec. 2.2). There is one difference, which lies in the training data given to the classifier: it is given two minibatches of data, one containing real data (with labels '1') and one containing generated data (with labels '0'). By minimization of this loss D will learn to assign the correct label to both real and fake data.

To create the second player, the adversary to D, a generator loss needs to be constructed. The simplest version of the game is created when we let the sum of all players loss be equal to zero. This is achieved by setting

$$\mathcal{L}_G = -\mathcal{L}_D. \tag{2.16}$$

This type of game is referred to as a *zero-sum game*, or a *minimax* game. Because $\mathcal{L}_G$ is completely dependent on $\mathcal{L}_D$ in this formulation, the entire game can be summarized with a single value function $V(G, D)$ based on the discriminator loss:

$$V(G, D) = -\mathcal{L}_D(G, D) \tag{2.17}$$

Playing the minimax game then becomes equivalent to minimizing an outer loop and maximizing an inner loop over the value function:

$$\theta_g^* = \arg\min_G \max_D V(G, D), \tag{2.18}$$

with θ_g^* being the optimal generator weights. This formulation of the game is mainly used for theoretical analysis of the GAN framework, and is often not optimal for practical use in training. In [34] Goodfellow *et al* use it to show that playing the game resembles minimizing the Jensen-Shannon divergence between p_{data} and p_{model}, providing theoretical validity to the model idea. They also show that given enough capacity and training time the game should converge to its equilibrium. However, as the proof is based on the assumption of a model with infinite capacity it does not translate well to practical implementations. One reason for this, which is stated already in the original GAN paper [34], is vanishing gradients in early training. Here, G has not yet learned to create convincing samples, so D can reject the generated samples with high confidence as they are clearly different from the training data. This could cause the term $\log(1 - (D(G(z))))$ in $V(G, D)$ to saturate (and be equal to zero) causing the gradients of the generator to completely vanish and halt the training of the model [36]. Goodfellow *et al* propose to solve this problem by assigning a different generator loss: instead of flipping the sign in front of the D-loss the target labels are flipped,

$$\mathcal{L}_G = -\frac{1}{2}\mathbb{E}_{x\sim p_{model}}[\log D(G(z)))] - \frac{1}{2}\mathbb{E}_{x\sim p_{data}}[\log(1 - D(y))]. \tag{2.19}$$

Since the second term is not affected by the generator weights it can be omitted during training, giving the generator loss

$$\mathcal{L}_G = -\mathbb{E}_{x\sim p_{model}}[\log D(G(z)))]. \tag{2.20}$$

This loss provides much stronger gradients early in learning while still sharing the same fixed point dynamics of G and D as the original formulation. The loss function is dubbed *non-saturating*, and is the loss used in practice when working with standard GANs. The action of flipping the labels can be thought of as explicitly instructing the generator to fool the discriminator; In the original game, the generator minimizes the log-probability of the D being correct, while in the non-saturating version, it maximizes the log-probability of D being mistaken [35]. For convenience, this version of GAN will be called the standard GAN (SGAN) during the rest of the thesis.

While this formulation of GANs was a groundbreaking step forward in the world of generative modeling, it was not without its faults. In practice GANs are difficult to train, require large amounts of data and are not guaranteed to converge under gradient descent-based optimization [37]. Despite its flaws, it is capable of modeling complex real-world distributions and has become one of the leading model frameworks for state-of-art generative models today.

GANs for image transformation

The original GAN structure as defined above was constructed to generate new, previously unseen data. An example could be to train a network to generate new images of "fake" WBC:s that look photorealistic, perhaps to create a synthetic dataset. However, that is not what is being attempted in this thesis, as I want to perform an image transformation, from unstained to stained. The simplest way to modify the structure in Fig. 2.12 for this new task is to replace the random noise input with whatever deterministic input you wish to transform (e.g. an image of an unstained blood cell).

2.9.2 Relativistic GAN

In the years following its inception many methods have tried to improve on the original GAN structure. Some have focused on altering the loss functions used (e.g. LSGAN [38]), while others have tried altering the discriminator to no longer strictly be a classifier (e.g. WGAN [39]). A successful class of GAN variants are the ones based on Integral Probability Metrics (IPM), which have been able to

improve the stability and reduce the convergence issues common to the original formulation. In 2018, Jolicoeur-Martineau published a paper stating that one key property present in the IPM variants but missing from the standard GAN (SGAN) formulation could be a strong contributing factor to their success: a *relativistic* discriminator [40].

In [40] the missing property of the SGAN is stated: "... the probability of real data being real ($D(x_r)$) should decrease as the probability of fake data being real ($D(x_f)$) increase." Here, x_r represents a real sample and x_f a fake one. Jolicoeur-Martineau further presents three arguments for why this property is needed: the prior knowledge, divergence minimization, and gradient arguments.

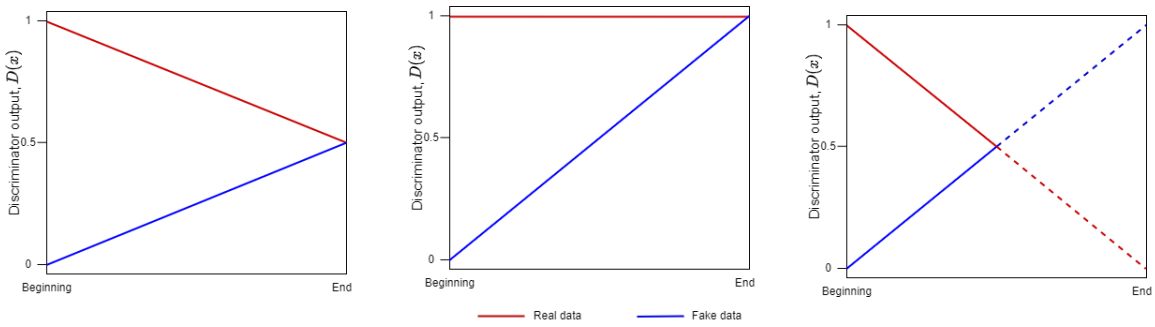
Prior knowledge

The prior knowledge argument is based on the fact that there exists *a priori* knowledge about the distribution of a training mini-batch that is not taken into account in SGAN: half of the samples are fake, and half are real. With this knowledge in mind, a discriminator that perceives all images of a batch as equally real should output a probability of .50 for each sample. However, that is not the case when using the SGAN.

Consider a discriminator trained to a point where it is able to correctly classify most real samples as real and most fake samples as fake. Then, a generator is trained to a point where it manages to fool the discriminator to think fake samples are actually real. Since the generator will not directly influence the discriminators output on real samples, the discriminator will end up classifying almost all samples as real ($D(x) \approx 1$ for all x). This is because in SGAN (or other non-IPM-based GANs), it is implicitly assumed that the discriminator does not know that half of the samples are fake. Designing a new D that actually takes this available *a priori* knowledge into account should lead to more reasonable predictions.

Divergence minimization

The divergence minimization argument can be summarized as the fact that the minimization of the Jensen-Shannon divergence does not actually correspond to the minimization of the saturating loss in SGAN. The discrepancy is illustrated in Fig. 2.13 below: in a), the expected behavior of minimization of JSD is shown with $D(x_r)$ decreasing as $D(x_f)$ increases. In b) however, we see the actual behavior of the SGAN, where the output from $D(x_r)$ is not minimized. To bring SGAN closer to the originally intended divergence minimization, training the generator should also lead to decreasing $D(x_r)$ as in c) [40]. For a more thorough derivation, see [40].



(a) Divergence minimization (b) Actual generator training (c) Ideal generator training

Figure 2.13: Illustrations depicting the expected discriminator output of real and fake data during a) minimization of the Jensen-Shannon divergence, b) Actual generator training minimizing the saturating loss function, and c) ideal training of the generator to minimize its loss function [40].

Gradient argument

In [40] the gradient of SGAN is shown to be

$$\nabla_{\theta_D}\mathcal{L}_D^{SGAN} = -\mathbb{E}_{x_r\sim p_{data}}[(1-D(x_r))\nabla_{\theta_D}C(x_r)] + \mathbb{E}_{x_f\sim p_{model}}[D(x_f)\nabla_{\theta_D}C(x_f)] \tag{2.21}$$

$$\nabla_{\theta_G}\mathcal{L}_G^{SGAN} = -\mathbb{E}_{z\sim p_z}[(1-D(G(z)))\nabla_w C(G(z))\nabla_x C(G(z))J_\theta G(z)], \tag{2.22}$$

where θ_G and θ_D are the weights of G and D, $C(x)$ is the D output before its activation function (for SGAN, $D(x) = \text{sigmoid}(C(x))$), and J is the Jacobian. The function C is often referred to as the *Critic*. When the discriminator is optimal, the term $1 - D(x_r) \longrightarrow 0$. This means that the gradient in (2.21) will mostly come from the fake images x_f, indicating that the discriminator stops learning from real images and learns only from generated images. In this case fake samples will no longer become more realistic, and training can get stuck. However, if the term $D(x_r)$ always decreases when $D(x_f)$ increases, real data will always be incorporated in the gradient of $\mathcal{L}_D^{SGAN}$, ensuring that D does not lose sight of what makes a sample realistic.

Relativistic standard GAN

With the motivation clear the relativistic standard GAN can be defined. As before let $C(x)$ be the output of D before activation. Then the SGAN discriminator is defined as

$$D(x) = \text{sigmoid}(C(x)). \tag{2.23}$$

To make this discriminator relativistic one can sample real/fake data pairs $\hat{x} = (x_r, x_f)$ and use the pairs to compute a relative distance between the critic output of the two

$$D(\hat{x}) = \text{sigmoid}(C(x_r) - C(x_f)). \tag{2.24}$$

While the original discriminator output the probability that a given sample was from the real distribution, this relative discriminator instead outputs the probability that the given real data is **more** realistic than a randomly sampled fake data. Similarly, the reversed version $D_{rev}(\hat{x}) = \text{sigmoid}(C(x_f) - C(x_r))$ can be understood as outputting the probability that a given fake sample looks more realistic than a random real sample. With this discriminator the loss functions for the SGAN in (2.19) and (2.15) can be rewritten into

$$\mathcal{L}_D^{RSGAN} = -\mathbb{E}_{\hat{x}=(x_r,x_f)\sim(p_{data},p_{model})}[\frac{1}{2}\log((D(\hat{x})) - \frac{1}{2}\log(1 - D_{rev}(\hat{x}))], \tag{2.25}$$

$$\mathcal{L}_G^{RSGAN} = -\mathbb{E}_{\hat{x}=(x_r,x_f)\sim(p_{data},p_{model})}[\frac{1}{2}\log(D_{rev}(\hat{x}) - \frac{1}{2}\log(1 - D(\hat{x}))]. \tag{2.26}$$

These equations can be further simplified thanks to a property of the sigmoid activation function. The term $D_{rev}(\hat{x}))$ becomes the complement to the term $D(\hat{x})$, and as such one can rewrite the term $\log(1 - D_{rev}(\hat{x}))$:

$$1 - D_{rev}(\hat{x}) = 1 - \text{sigmoid}(C(x_f) - C(x_r)) = \text{sigmoid}(C(x_r) - C(x_f)) = D(\hat{x}) \tag{2.27}$$

Using this property (2.25) and (2.26) simplify into the following forms,

$$\mathcal{L}_D^{RSGAN} = -\mathbb{E}_{(x_r,x_f)\sim(p_{data},p_{model})}[\log(\text{sigmoid}(C(x_r) - C(x_f))] \tag{2.28}$$

$$\mathcal{L}_G^{RSGAN} = -\mathbb{E}_{(x_r,x_f)\sim(p_{data},p_{model})}[\log(\text{sigmoid}(C(x_f) - C(x_r))] \tag{2.29}$$

and the Relativistic Standard GAN (RSGAN) is complete. In [40] this relativistic view of GANs is generalized to show that almost any GAN can have a relativistic discriminator, but as this thesis will only use SGAN formulations the generalized version is not included here.

Relativistic paired SGAN

If one has access to paired data, i.e. data where the input x is paired with a ground truth y (like an unstained and stained WBC), then one can form a Relativistic Paired SGAN (RpSGAN). This is just an RSGAN but instead of randomly sampling data for comparison as in eq. (2.24) one compares the generated/real image to its paired partner. If a minibatch consists of N pairs of images x_r^i and x_f^i, the loss for that batch is calculated as follows:

$$\mathcal{L}_D^{RpSGAN} = -\sum_i^N \mathbb{E}_{(x_r^i, x_f^i)\sim(p_{data}, p_{model})}[\log(\text{sigmoid}(C(x_r^i) - C(x_f^i))] \tag{2.30}$$

$$\mathcal{L}_G^{RpSGAN} = -\sum_i^N \mathbb{E}_{(x_r^i, x_f^i)\sim(p_{data}, p_{model})}[\log(\text{sigmoid}(C(x_f^i) - C(x_r^i))] \tag{2.31}$$

This paired formulation will be used for a model in this thesis.

2.9.3 Spectral Normalization

In [41] Miyato *et al* introduce a novel weight normalization method meant to increase training stability of discriminator networks used in a GAN setting. The method is named *Spectral Normalization* (SN), and functions by restricting the Lipschitz constant of the discriminator. This is done through direct constraining of the spectral norm of each layer $l : \mathbf{h}_{in} \longrightarrow \mathbf{h}_{out}$ in the discriminator. In [41], it is shown that for a linear layer $l(\mathbf{h}) = W\mathbf{h}$, the Lipschitz norm is given by

$$\|l\|_{\text{Lip}} = \sup_{\mathbf{h}} \sigma(\nabla l(\mathbf{h})) = \sup_{\mathbf{h}} \sigma(W) = \sigma(W). \tag{2.32}$$

Here, $\sigma(W)$ is the spectral norm of the matrix W, which is equivalent to the largest singular value of W. Furthermore, it is shown that given an activation function with Lipschitz norm equal to 1 (true for ReLU and Leaky ReLU), an upper bound can be observed on the Lipschitz norm of the full discriminator f:

$$\|f\|_{\text{Lip}} \leq \prod_{l=1}^{L+1} \sigma(W^l) \tag{2.33}$$

where W^l represents the individual weights W of each layer l. The proposed spectral normalization then normalizes the spectral norm of the weight matrix W so that $\sigma(W) = 1$:

$$\bar{W}_{SN}(W) := W/\sigma(W). \tag{2.34}$$

If each W^l is normalized using (2.34) then it is clear from inequality (2.33) and the fact that $\sigma(\bar{W}_{SN}(W)) = 1$ that $\|f\|_{\text{Lip}}$ is bounded from above by 1. For a more detailed derivation and motivation, see [41].

In essence, this method provides a data dependent regularization technique that is simple to implement and computationally efficient (thanks to the use of a fast approximation of the spectral norm). The addition of spectral normalization greatly increased stability in the original papers findings, and has previously been used in combination with a Relativistic discriminator in [42]. Because of this, weights of the discriminators in this thesis will be spectrally normalized.

2.10 Enhanced Super Resolution GAN

In 2018 Wang *et al* released a paper introducing a new network architecture developed for single image super-resolution titled "Enhanced Super Resolution GAN" (ESRGAN). The paper focused on improving the perceptual quality of super-resolved images, basing their new model on the previous work by Ledig *et al* [29]. They take inspiration from ResNets and DenseNets to create their network, with a novel building block called the Residual-in-Residual-Dense block.

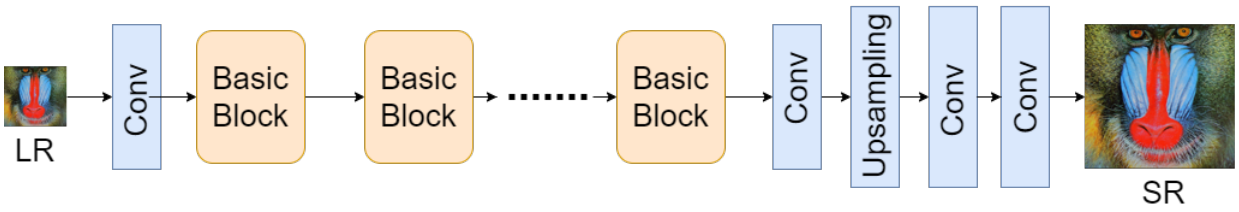


Figure 2.14: Basic architecture of the generator in ESRGAN. This architecture was originally employed in SRResNet [29]. Different basic blocks were tested in [16] and the best performance was achieved when using RRDB blocks.

In Fig. 2.14 the high-level architecture of the ESRGAN generator is shown. Most of the computation of the model is done in the low-resolution feature space through a number of basic blocks, followed by an upsampling step to achieve their super-resolved image. The architecture was originally proposed by Ledig *et al* for the generator in Super Resolution GAN (SRGAN), and in this network the basic blocks were normal Residual blocks. In ESRGAN, better performance is achieved by making two modifications to the structure of the generator: 1) removing all batch normalization layers; 2) replacing the residual block with a new Residual-in-Residual block (RRDB). The RRDB block combines a multi-level residual network with the dense connections of a dense block, see Fig. 2.15.

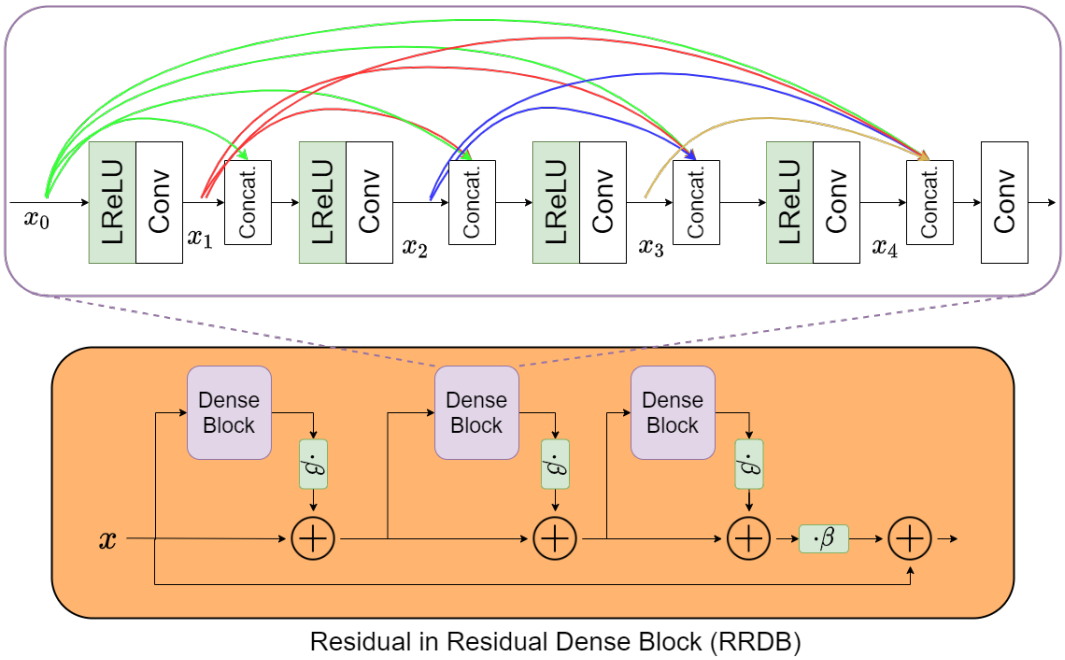


Figure 2.15: The RRDB block as defined in [16]. Note the lack of BN layers in the dense block, the leaky ReLU activations, and the added residual scaling factor β.

The authors motivate the removal of BN layers by referring to other studies that have proven it leads to increased performance as well as reduced computational complexity in peak signal to noise ratio tasks such as SR [43] and deblurring [44]. They also observe empirically that BN layers are more likely to introduce artifacts when using deeper networks and training under a GAN framework. The more complex structure of the RRDB model was motivated by the observation that more layers and

connections usually improve performance, given sufficient training data [43][45][46]. The removal of BN layers does however lead to less regularization of the model, making it more difficult to train a deep model. To facilitate training without the BN layers, the authors employed two techniques: 1) residual scaling, and 2) smaller initialization, as they had empirically found that this makes residual architectures easier to train.

For their GAN setup they used a Relativistic average GAN loss, and found increased performance compared to using an SGAN formulation. They also employed a weighted loss function when training their generator with both perceptual and pixel-wise parts, defined as follows:

$$\mathcal{L}_G^{esr} = \mathcal{L}_p + \lambda \mathcal{L}_G^{RaSGAN} + \eta \mathcal{L}_{l_1}. \tag{2.35}$$

Here $\mathcal{L}_p$ is the perceptual loss defined in eq. (2.9), $\mathcal{L}_G^{RaSGAN}$ is the Relativistic Average generator loss which is defined in [40] and $\mathcal{L}_{l_1}$ is the l_1-norm between the generated and true image defined in (2.7). λ and η are the coefficients used to balance these different loss terms, and become hyperparameters that require finetuning. It was in this paper that the idea of using a perceptual loss before activation, as discussed in Sec. 2.2.1, was introduced.

The design of ESRGAN makes it simple to alter the architecture to perform other image transformation tasks, as one can simply replace the upsampling step with some other operation to receive another output. The structure of the esr-generator and its loss function are used as inspiration for the networks produced in this thesis.

While the transformation attempted in this thesis is not one of super-resolution, many of the sought after qualities are similar: we seek a high perceptual similarity with our paired ground truth and we want high-frequency information to appear in the generated image. The input is also highly correlated with the output, as we have matched before and after images of the same cell, just as with the low-resolution/high-resolution image pairs in ESRGAN.

The ESRGAN formulation also provides a smaller network than a comparative UNET-architecture, which also speaks in its favor.

Chapter 3

Method

3.1 Gathering the dataset

As the methods proposed to solve the problem of colorization in this project are data driven, a good dataset was essential. Since no analysis is currently performed on unstained blood smears, there was no existing dataset available for use. Thus, one had to be collected. The method of collecting data is demonstrated in Fig. 3.1 below and is explained in detail in the following sections. All images were taken using a digital microscope with a 20x/0.4 NA Plan C Olympus optical lens and an LED-array for illumination.

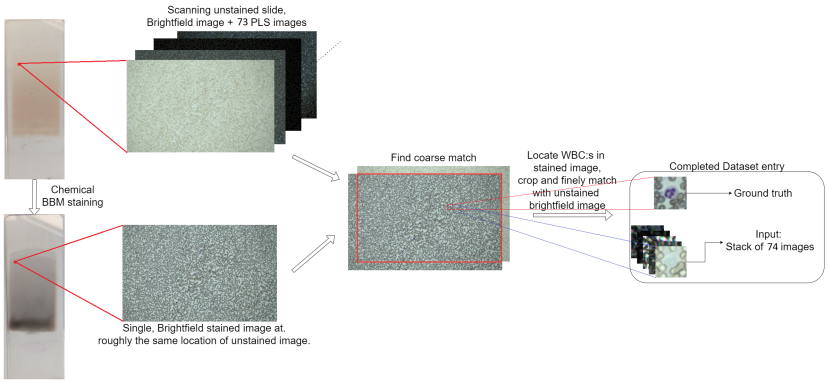


Figure 3.1: Illustration of the method of data gathering. First, the unstained slide is scanned and images of different lighting are collected. The slide is then chemically stained, and rescanned taking a single brightfield image at each position. These stained/unstained images are matched, segmented to find WBCs, and the matching is then finetuned to create the final GT and image stack pairs. A further step collecting a higher quality GT image is explained in Fig. 3.6

3.1.1 Choice of PLS images and calibration

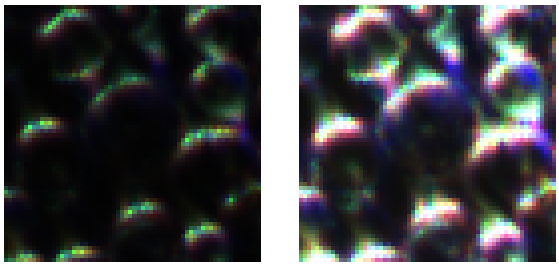
The first step of collecting the dataset was defining which PLS images to collect. A tradeoff was considered between more information and reducing raw data size to accomodate physical storage restrictions (one 4TB harddrive). In the end 43 LED:s were chosen spanning the largest possible angles to the optical axis. The LED:s are divided by three on a bottom plate, close to the optical axis of the microscope, and the rest symmetrically chosen along five arms with 8 LED:s per arm. The 8 LED:s on the arms are of increasing angle to the optical axis, with a maximum angle of 57°. Because of all the different angles of light, the settings for the camera need to be pre-determined for each LED to be collected. This was done manually by observing a sample live in a microscope with the different LED:s on and altering shutter time / camera gain until a good quality was reached. The measurement of good was determined through manual inspection of the image and the histogram distribution of the pixel values in order to choose the image with the most detail present and the least ”maxed out”

pixel values (pixels of value 255 in either R, G or B channels). The reason for this is to increase the dynamic range of the image and therefore the information contained in the image.

This created a problem for some of the higher angles of illumination. Because of the NA of 0.4 of the objective any illumination angles above

$$arcsin(\frac{NA}{n}) = arcsin(\frac{0.4}{1.00}) \approx 23.6^\circ \tag{3.1}$$

will lead to the collected image being in the Darkfield (DF) domain. All of the light present in these images are from light scattered on the subject, and for higher angles less light reaches the camera requiring longer exposure. Increasing the exposure too much will however lead to many "burned" pixels, but also appears to increase the dynamic content inside WBC:s. To compensate for the burned pixels in the high exposure image, for each DF PLS two images are collected: One with a high exposure, increasing the dynamics of the cells, and one with low exposure compensating for the pixels that are burned in the high exposure image. See an example of a high/low exposure DF image in Fig. 3.2 below. The effect of the double exposures is investigated in Sec. 4.1.



(a) Low exposure (b) High exposure

Figure 3.2: a) Low exposure image and b) high exposure image of the same WBC.

Of the chosen LED:s six on each arm produce DF images, meaning a full PLS stack ends up being 73 images. A normal brightfield image is also collected by lighting up all available LED:s with an angle smaller than 23.6°, producing light within the full range of angles that the 0.4 NA objective can accept. This brightfield image is then also added to the PLS stack to be used as input to the model, making the final stack consist of 74 images. For an example of one full stack, see Fig. A.7 in the Appendix.

3.1.2 Scanning full slides

The first step of collection consisted of scanning a portion of an unstained blood smear in the RBC monolayer region. In the monolayer cells are spread out enough that they are rarely on top of each other, and is the region usually used for heamatological analysis such a WBC differential counts. The microscope scans over a rectangle of coordinates specified a priori and saves the coordinates of each visited position. The coordinates were chosen by manual inspection of the blood sample both using the microscope and by simply looking at the sample and measuring using a ruler. A brightfield image is taken with normal lighting, followed by 74 images using different Point Light Sources (PLS) on the LED-array as described in Sec. 1.2.1. After the sample was scanned and imaged, it was taken to a lab for chemical staining using a RAL Stainbox and BBM staining kit. This staining solution is a type of Romanowsky staining, and produces stained cells of similar quality as other commonly used staining techniques such as May-Grünwald Giemsa. While only one particular staining will be used for the entire dataset, changing virtual stain target should be as simple as collecting another dataset using the same method as presented and retraining the networks. The stained sample was then scanned again and one image with brightfield lighting was collected for each position imaged when it was unstained. This was done for two reasons: 1. WBC:s can be located easily in the stained blood sample and then relocated in the unstained sample, while they are difficult to locate in the

unstained sample. 2. A matched stained-unstained pair can be constructed to provide a ground truth when training the neural networks.

3.1.3 Segmenting, matching and cropping full images

After scanning the same slide both unstained and stained the images are roughly matched, and the WBC:s can be clearly seen in the stained samples. To improve the rough matching of the images a large portion of the stained image is cropped, with a margin of 264 pixels to each side.

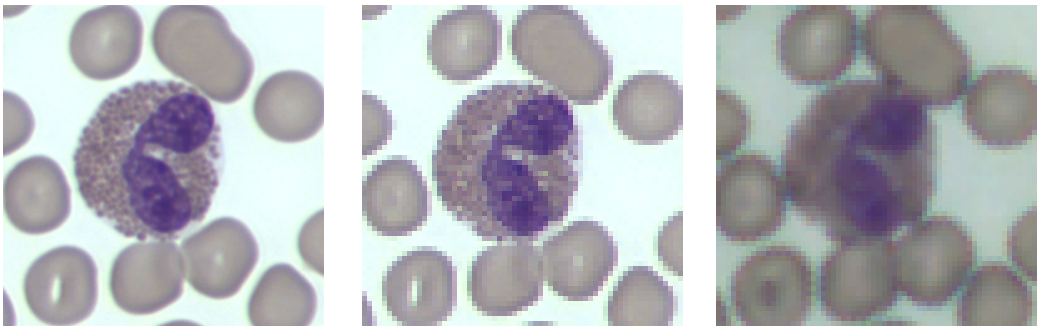
The cropped stained image and the unstained brightfield image are then transformed into binary edge detected images using Canny filtering [47], and subsequently matched by sliding the cropped stained image over the unstained one and calculating the correlation coefficient of the two according to (A.1), and choosing the overlap with the highest correlation. The overlapping portion of the images is then cropped and used for future processing.

After this step the images are well matched, but there can still be errors of a few pixels between the two. The stained image is now processed using a color segmentation script, in which the image is transformed from the RGB colorspace to HSV space. The image is then thresholded in a range of purple colours to segment out the WBC:s. The threshold values were chosen by examining the pixel color distribution of images of stained WBC:s in HSV space and then testing on a few full images. An example of a wbc image and its color distribution are shown in Sec. A.1.1 Fig. A.1.

After this step a binary image is produced with rougly located WBC:s. Sometimes one WBC is split up into two detected objects, and to attempt to reduce any noise in the image and connect closely located objects (often one WBC nuclei) a (morphological) closing operation is applied to the binary image. The remaining connected components of the image are then classified as individual WBC:s, and the coordinates for their centroids are passed to the next step of the algorithm.

To allow for finetuning of ± 5 pixels any detected WBC:s closer than 5 pixels to the edge of the image are removed from the dataset. For each of the remaining WBC:s, an image of the stained WBC is cropped around it's centroid with a previously specified crop width (in the final dataset, 64 pixels). This cropped (64x64) image is then finely matched with the unstained image by sliding it across the same coordinates ± 5 pixels in both height and width. For each position, an SSIM similarity score is calculated and the position with the highest value is chosen as a match. The pixel position of the WBC is converted to the position in micrometers on the slide and saved in a csv file.

After the matching is complete, the PLS images corresponding to the matched unstained image are cropped in the same location as the unstained image, creating the matched input stack and the ground truth. For the extended High Resolution dataset, another collection of the slides is run using a higher power objective (100x/NA 1.25) where the WBC:s are relocated using the saved coordinates in the csv file. The high resolution images are matched against the stained low resolution images and downsampled, creating a ground truth of the same resolution but with much higher dynamic range due to the better physical properties of the objective.



(a) HR (NA=1.25, 320x320) (b) DS HR (NA=1.25, 64x64) (c) LR (NA=0.4, 64x64)

Figure 3.3: a) High resolution image b) downsampled high resolution image c) Low-resolution image. The downsampled high resolution image retains more structural information despite the downsampling procedure and produce a less blurred image. It also has the added effect of removing the dark circles around the red cells, which is an undesired artifact. Using this as the ground truth allows the model to not only stain the cells, but also improve image quality compared to the 20x/0.4 NA objective.

To allow for greater precision the low-resolution image was linearly upsampled and the matching was performed in the high-resolution space. At first the same method used for the 20x stained-unstained pairs was attempted, but it proved ineffective. The images were in some cases too different, and the best correlation match was not actually the best match. Using SSIM for finetuning did not remove the problem either, and was computationally expensive in the HR space. A hypothesis for the difficulty of matching can be seen in Fig. 3.4. Dark edges are present around RBC:s in the images from the 20x objective, which are not present in the 100x images, instead being replaced by an almost white background. This would cause a pixel-by-pixel correlation score to be highly discouraged from a perfect match, as the dark shadow of the 20x is then compared with the white background of the 100x.

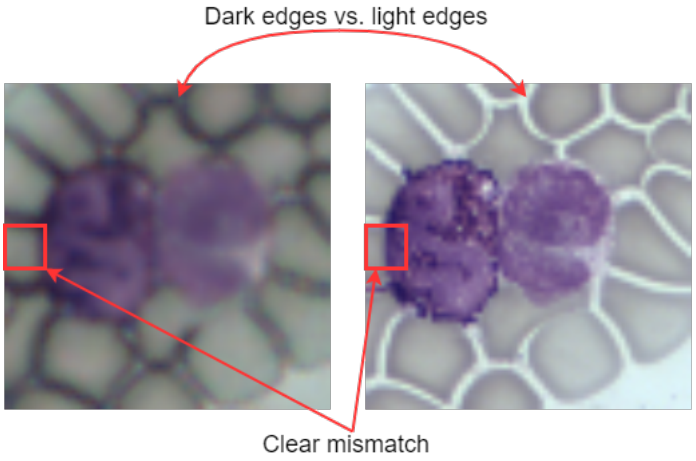
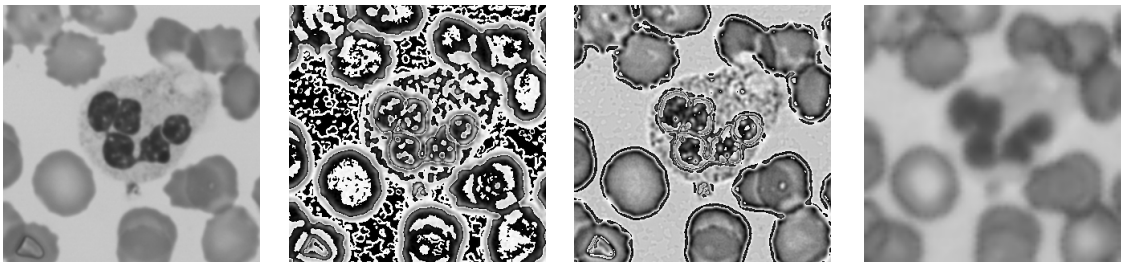


Figure 3.4: Example of a difficult to match sample. Dark edges present in the 20x image but not in the 100x image makes correlation score a poor metric for a good match. The red square shows a mismatch of a few pixels which has remained after a correlation match and SSIM finetuning.

To remedy this image disparity, pre-processing was applied to the HR-image before matching. The high resolution image was filtered by conversion to grayscale and brightness adjustment, followed by application of a blurring Gaussian filter of kernel 9x9 and a Laplacian filter with kernel 7x7 (creating a Laplacian-over-Gaussian (LoG) filter). The output was then subtracted from the grayscale image to create an image with accentuated edges [48]. An example of an LoG-subtracted image is shown in Fig. 3.5 c) next to an upsampled grayscale 20x image.



(a) Gray HR (b) LoG(HR) (c) HR-LoG(HR) (d) Upsampled LR

Figure 3.5: a) Grayscaled HR image originally captured with 100x objective. Note lack of dark shadows. b) LoG-filtered HR image. c)LoG-subtracted high-res image b) Upscampled, grayscaled, low-res image taken with the 20x objective.

Note the darker shadows now present around the RBC:s. The final matching was then performed by sliding the upsampled 20x image over the filtered HR-image and choosing the position of highest correlation. A schematic view of the process is shown in Fig. 3.6.

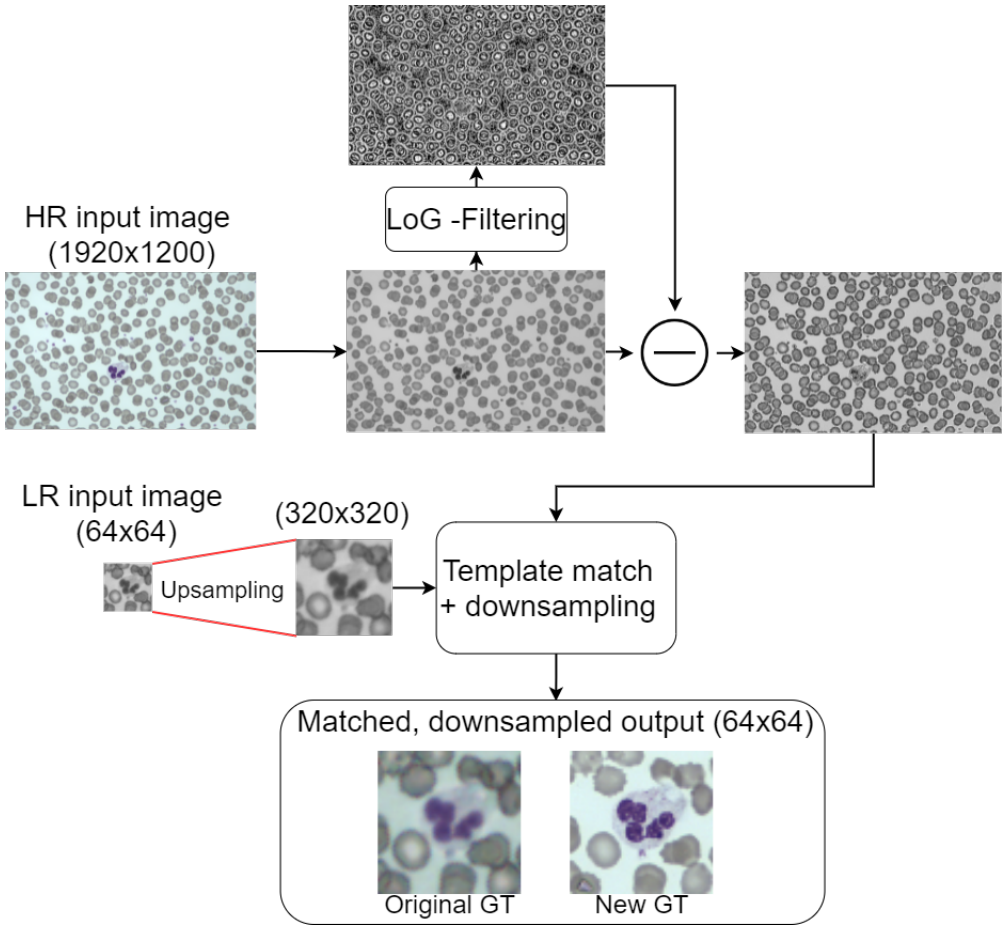


Figure 3.6: Schematic illustration of the matching process to create the improved ground truth from a high-res image.

A final note about the matching is the scaling of the HR image. As it is known a priori that there is a scaling factor of 5 between 20x and 100x, that is the scaling applied when downsampling the HR-image. However, because these are physical instruments and objectives they are not always exact. In general, there seems to be a slight difference left in scale after the downsampling of the HR-image. A better match may be possible to attain by also applying a scaling transform, but this was not done due to time restrictions. The match was deemed good enough for this thesis and the process shown in Fig. 3.6 is the one used to create the HR dataset.

3.1.4 Dataset distribution

Most of the blood smears used for the final dataset came from one healthy individual, with approximately 15% of the slides coming from another unknown patient. With this in mind the assumption is made that the dataset should have a similar distribution of different types of WBCs as those given in Sec. 1.1.1. This means that the dataset will be much more saturated with neutrophils and lymphocytes than basophils and eosinophils, which may affect the model performance on different types of cells.

3.1.5 Summary of datasets

All data collection was performed using the same imaging system and all staining using the same scheme and machine. 44 slides were processed and collected, but due to misstaining or PLS lighting errors the samples from 8 of those slides had to be discarded. From the remaining 36 slides 29999 samples of WBC:s were located and used in the final dataset. For the ablation study, the training and validation ratio was set 9:1 in favor of training, giving a training set of 26999 samples and a validation set of 3000 samples. Using only validation and training sets was deemed appropriate as no hyperparameter optimization was done during the ablation study. For the final models however, a traning/validation/test split was made in ratios 8:1:1. The validation set was used to allow for hyperparameter tuning during training and the test set was used for evaluation after training was concluded. Staining and data collection was performed at a medtech company in Lund by the author of this project.

- **dataset_HSV** - Dataset using HSV segmentation and with 20x images for GT.
- **dataset_HSV_HighRes** - Dataset using HSV segmentation with 100x images used for GT.

3.1.6 Note on Data Augmentation

In machine learning problems it is common to use different types of *data augmentation* to increase the variation in a dataset. For image classification/generation, one often use operations that do not invalidate the task, e.g. flipping or rotating images when performing classification: an upside down dog is still a dog. Employing these types of variations in the data is thought to help guide the model towards the important parts of the image in order to converge to the sought after solution. However, using the common operations such as flipping and rotating are not as trivial in the case of this thesis as one might think. This is due to the use of the PLS images, where specific LEDs at specific angles and *directions* are employed to create the PLS stack. The order of the different images in the stack is also fixed, as they are directly connected to the learned weights of the layers of the model. Consider then what happens when we flip a full stack; All PLS images that used to have directional light from the right, now appear to receive directional light from the left and vice versa. Similar problems appear when trying rotation. This may be possible to solve using a very specific augmentation scheme but due to the complexity required it was not attempted in this thesis. No data augmentation has been used on either of the datasets constructed.

3.2 Network Architectures

In this section the network architectures employed in this thesis are defined and presented.

3.2.1 The Generator

The generators presented in this thesis will be based on the ESRGAN-generator, with the exception of one network, which is a U-net. While U-net is a more common architecture choice for colorization, because of the large input size used in this thesis due to the PLS-stack a U-net generator becomes very large very quickly. This makes the network both difficult and slow to train, and so an alternative was sought. The ESR-based generator becomes much smaller, while still being able to correctly transform

images. A U-net model with the maePF-loss defined later in (3.4) is presented and compared with the respective ESR-based generator in the appendix.

The basic architecture is shown in Fig. 3.7. It is the same as the one shown in Fig. 2.14, except with the upsampling step removed. The input has also changed to be a stack of several images, creating a wider network. The output feature channels are downsampled into one single color image using a convolutional layer creating the output.

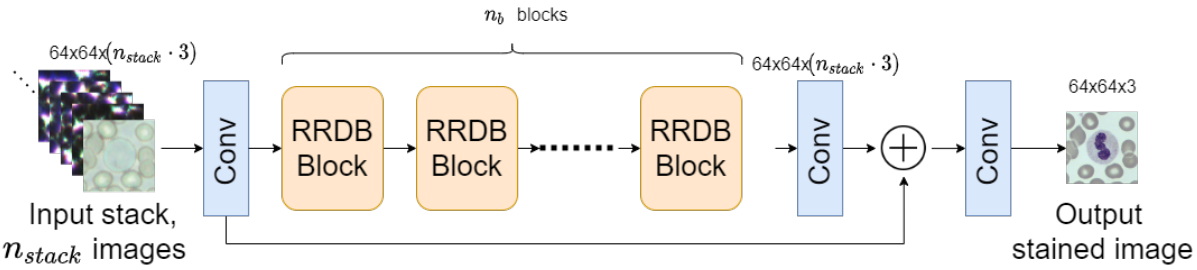


Figure 3.7: Generator architecture used in this thesis. n_b is the number of RRDB blocks used and n_s is the number of images present in the input stack. The RRDB block is defined in Fig. 2.15.

The RRDB blocks are as defined in Fig. 2.15, including the residual scaling parameter. Smaller initialization will also be used during training as was done in the original ESRGAN paper [16].

3.2.2 The Discriminator

The discriminator used for the GAN formulations presented is a classic CNN inspired by VGG-128 [49]. It consists of three initial blocks, in which two convolutions are performed: the first with stride 1 and a kernel size 3x3, and the second with stride 2 and kernel size 4. The number of filters are also doubled between each block. The final block is flattened into a 1D-vector of length 4096, and through two fully-connected layers it is compressed into one single output.

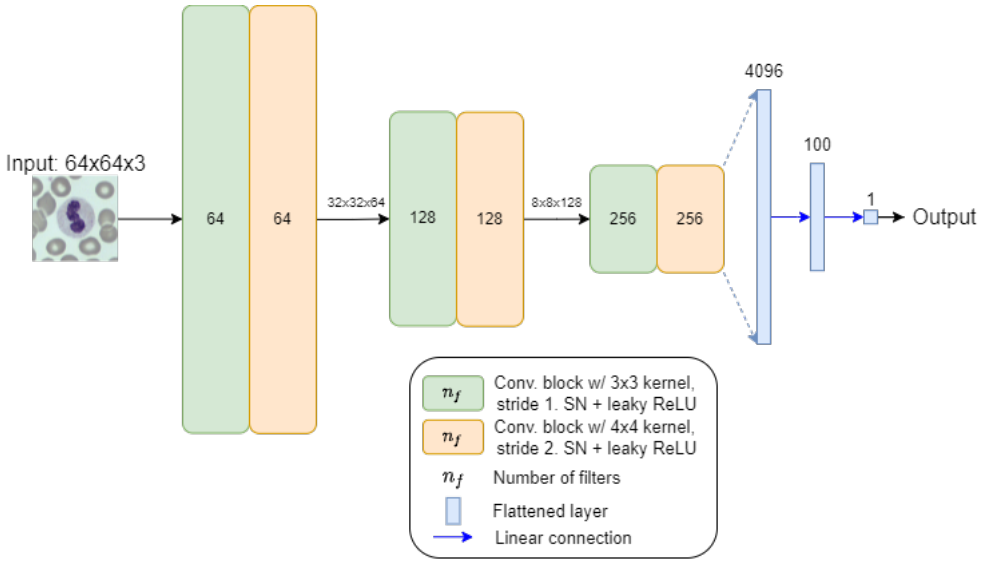


Figure 3.8: The discriminator architecture used in this thesis. SN stands for Spectral Normalization.

The output from the network is the critic output $C(x)$ mentioned in Sec. 2.9.2. The activation function creating the discriminator will be the relativistic paired formulation presented in Sec. 2.9.2.

3.3 Generator networks

Different Generator networks were tested before usage in a GAN setting. This allowed for comparisons between more classical CNNs and the GAN framework to be made, and one can evaluate whether a GAN setup is truly necessary for this problem. The testing also helped in tuning a better generator loss for future models. The generators were trained using different loss functions to evaluate which one would result in the optimal generation of stained images. The model names are connected to the loss function used to train them.

All generator networks share the same general architecture shown in Fig. 3.9, with different loss functions $\mathcal{L}_G$. They attempt to model the transform function $\mathcal{T}(x)$ using only a generator of the design presented in Fig. 3.7 with $n_b = 4$. The data given to the models consists of paired PLS stacks x and stained images y.

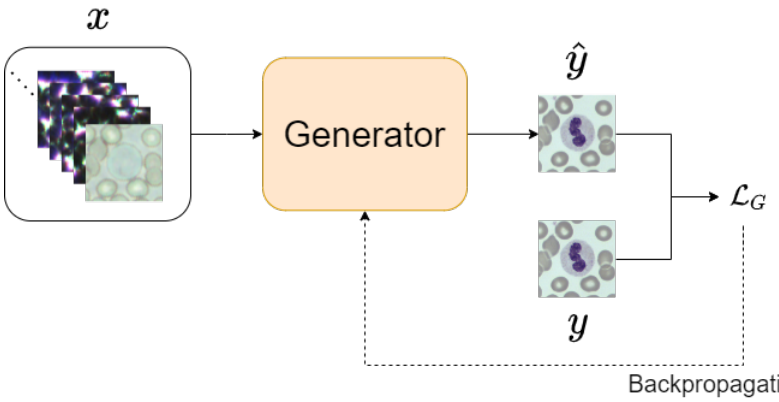


Figure 3.9: Generator network setup. A more traditional feed-forward network.

3.3.1 maeNET

maeNET is the simplest network presented in this thesis. The model is guided only by an MAE loss function between generated images $\hat{y}$ and true images y, as defined in (2.7). The generator loss then becomes,

$$\mathcal{L}_{maeNET} = \mathcal{L}_{l_1}. \tag{3.2}$$

3.3.2 maePNET

maePNET uses the same network architecture as maeNET with a different loss function. A perceptual loss term $\mathcal{L}_p$ as defined in (2.9) is added to the loss function of the network, giving it the following total loss:

$$\mathcal{L}_{maePNET} = \mathcal{L}_{l_1} + \lambda \mathcal{L}_p. \tag{3.3}$$

The coefficient λ decides the weighting between the two loss terms.

3.3.3 maePFNET

maePFNET uses the same network architecture as maePNET with another added loss term. It employs a Fourier loss in addition to the l_1- and perceptual losses, giving it the following loss function:

$$\mathcal{L}_{maePFNET} = \mathcal{L}_{l_1} + \lambda_1 \mathcal{L}_p + \lambda_2 \mathcal{L}_{\mathcal{F}}. \tag{3.4}$$

The coefficients λ_1 and λ_2 decide the weighting between the three loss terms.

3.3.4 PLS Ablation Study

With the networks defined, an ablation study could be performed to determine the relevance of the PLS images. Using the full stack makes the networks large and slow to train, and if the input could be reduced it would make both training and predicting faster. In addition the study will show whether the PLS images truly provide additional information; if the model performs equally well with only the brightfield image as input, then the stack is not needed.

Seven different inputs were designed and trained using the same general model architecture, that of the maePNET based on the generator from ESRGAN. A generator only model was used because it is more straight-forward to train than future GAN-based models and simplifies the procedure of training several networks. Evaluation of the models was based on minimum achieved validation loss, maximum achieved SSIM score (on the validation set) and manual inspection of the output images. The models were trained using the HR GT dataset.

The seven different inputs are defined in Table 3.1 below. All input stacks include one brightfield unstained image.

Name	Images in stack	Max θ_{LED}	Description
Full	74	56°	Input is all collected PLS images, including double exposures of DF.
Full NoHE	44	56°	Same as 'Full' but with the high exposure images removed.
≤36deg	44	36°	Images from LED:s with angles >36° to the optical axis removed. Remaining DF images still have both high and low exposures.
≤36deg NoHE	30	36°	Same as ≤36deg but with high exposure images removed.
≤18deg	14	18°	Images from LED:s with angles >18° to the optical axis removed. No DF images remain.
≤8Deg	4	8	Images from LED:s with angles >18°to the optical axis removed.
Brightfield	1	N/A	Only the brightfield image is used as input

Table 3.1: Names and descriptions for the different input stacks used in the PLS ablation study. Results are presented in Sec. 4.1.

3.4 GAN-networks

3.4.1 maePF-RpGAN-SN

maePF-RpGAN-SN combines many different ideas to create a GAN-type network, with a generator and discriminator. The structure is very similar to that shown in Fig. 2.12, with some key differences; As in ESRGAN, random noise is not sent as input to G. Instead, an input to be transformed is sent, in this case the PLS stack of images (in ESRGAN it was the low resolution image). Schematically this is shown in Fig. 3.10. The generator is based on the maePF model, and the model loads the pretrained weights from the previous maePFNET to initialize training. The main idea for pretraining the generator is simply to provide the discriminator with reasonable generated images early, to make it learn more key distinguishing features between generated and real images and thus provide a better loss signal. The discriminator is described in Fig. 3.8. The weights of each convolutional layer is also spectrally normalized (SN). The adversarial loss itself is constructed in Relativistic paired fashion as in (2.25) and (2.26). The total generator loss is then made up of four components as follows,

$$\mathcal{L}_{maePF-RpGAN-SN} = \mathcal{L}_{l_1} + \lambda_1 \mathcal{L}_p + \lambda_2 \mathcal{L}_{\mathcal{F}} + \lambda_3 \mathcal{L}_G^{RpGAN}. \tag{3.5}$$

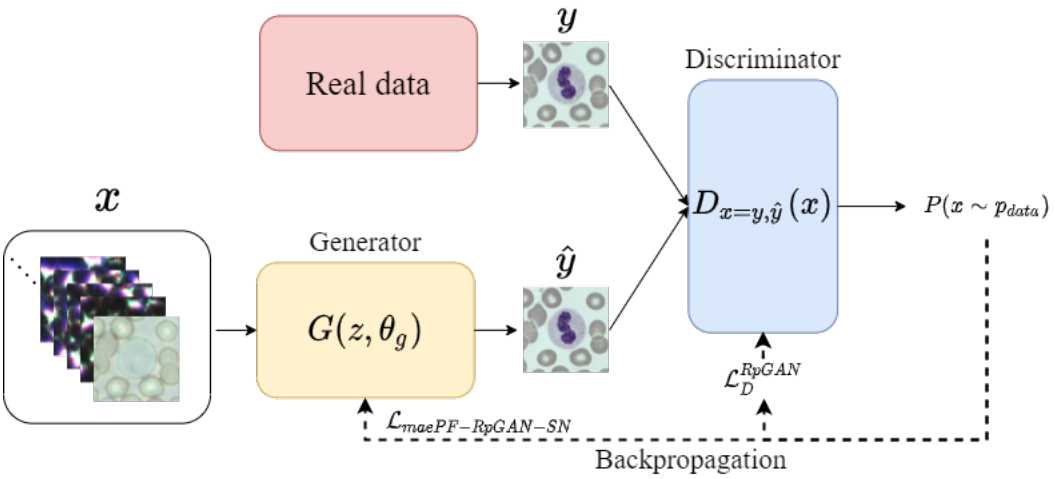


Figure 3.10: Illustration of the maePF-RpGAN-SN model architecture.

3.5 Exploring future improvements

With these model architectures general tests are able to be made to evaluate the efficacy of the different loss functions and the final model performance. However, there are also further avenues of research that could be explored, such as what the effects of using a deeper model or a larger dataset would mean for the general performance. Two small tests were made to explore this: 1) a deeper model with twice the RRDB blocks is trained using both the maePFNET-loss and the maePF-RpGAN-SN-loss and are compared with the shallower models, 2) An maePFNET-model is trained with only half the dataset available.

3.6 Training details

All models described above were optimized using the Adam [21] method. The generator optimizer settings were: learning rate $\alpha = 0.0002$, exponential decay rates $\beta_1 = 0.5$ and $\beta_2 = 0.9$. The discriminator optimizer settings were: learning rate $\alpha = 0.0004$, exponential decay rates $\beta_1 = 0.5$ and $\beta_2 = 0.9$. Using a slightly larger learning rate for the discriminator has been shown to often increase performance [50]. The discriminator and generator were both updated once for each training minibatch, which consisted of 20 stack/GT pairs for the normal models. When using the double depth generator the minibatches contained only 12 stack/GT pairs due to memory constraints. The loss weights for the different networks are presented in Tab. 3.2, where λ_{1-3} are as defined in (3.5).

Model	λ_1	λ_2	λ_3
maeNET	-	-	-
maePNET	0.0125	-	-
maePFNET	0.0125	0.0125	-
maePFNET-RpGAN-SN	0.0125	0.0125	0.005

Table 3.2: Loss weights used when training the presented networks.

All presented models were allowed to train as long as required to reach convergence, by which it is meant that valdidation loss stopped decreasing.

The computer used during the project was equipped with an Nvidia GeForce RTX 2080 Ti GPU and an 11th Generation Intel Core i9-11900K CPU @ 3.50GHz.

Chapter 4

Results

In this chapter results are presented along with important observations. Summarizing discussions are presented in the following chapter.

4.1 PLS Ablation study

The results are presented in two sections: the first detailing the quantitative results in the form of numerical values, while the second focuses on qualitative results in the form of predicted images.

4.1.1 Quantitative results

Validation loss and average SSIM score on the validation set are presented for the different input types in Tab. 4.1.

Name	Nbr model params.	Val. loss	SSIM
Full	13021833	0.06726	0.8652
Full NoHE	6515463	**0.06176**	**0.8660**
≤36deg	6515463	0.06801	0.8617
≤36deg NoHE	4027728	0.06801	0.08630
≤18deg	2050293	0.06935	0.8593
≤8Deg	1015503	0.07196	0.8499
Brightfield	749292	0.08411	0.8065

Table 4.1: Validation loss and average SSIM score on the validation set. The generator weights from the epoch corresponding to the lowest validation loss are loaded and used to generate qualitative results in Figs. 4.1 and 4.2. Best result is written in bold text.

4.1.2 Qualitative results

In Figs. 4.1 and 4.2 the predicted images from the ablation study models are shown, along with the ground truth and the unstained cell for easy comparison.

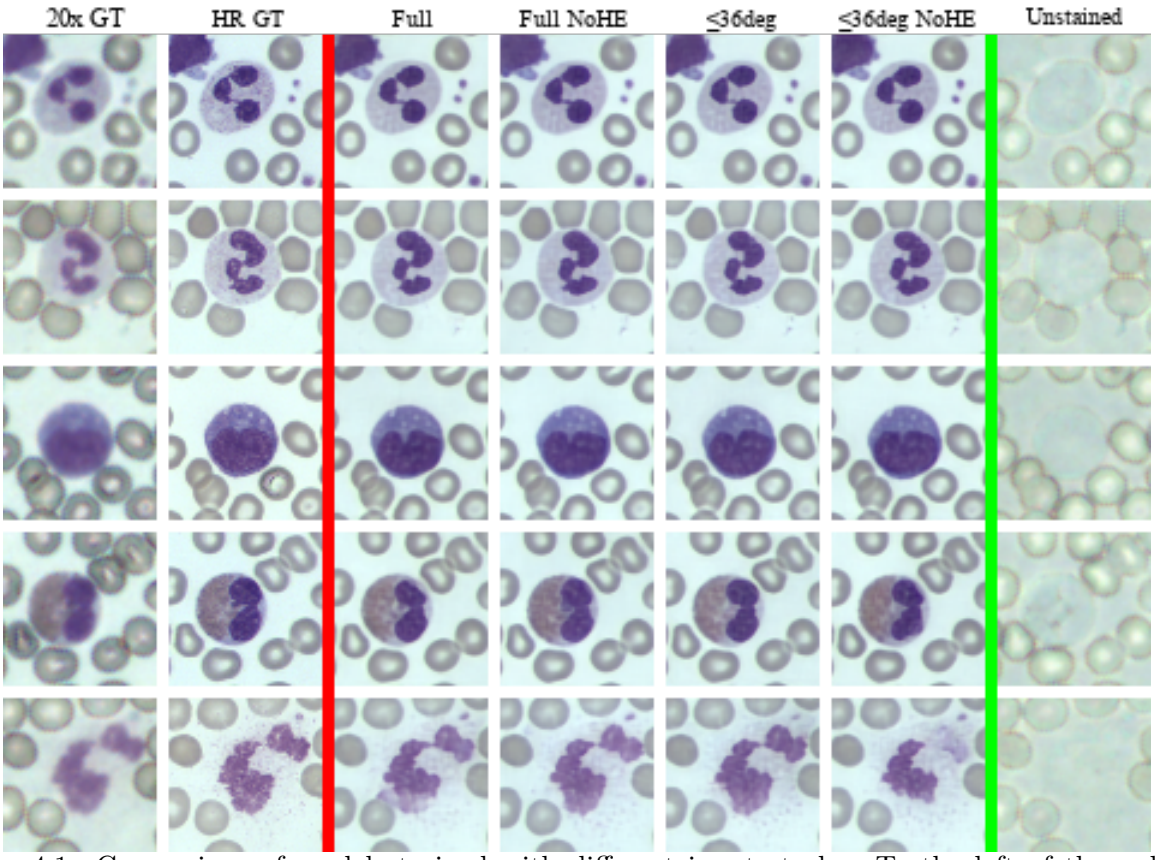


Figure 4.1: Comparison of models trained with different input stacks. To the left of the red line GT images are shown, in the middle the different model predictions are shown, and to the right of the green line the unstained original image is shown for reference. All models were trained with the HR GT images as their target. From top to bottom, the WBC:s are examples of a neutrophil and platelets, a neutrophil, a monocyte, an eosinophil and a smudge cell (a destroyed cell).

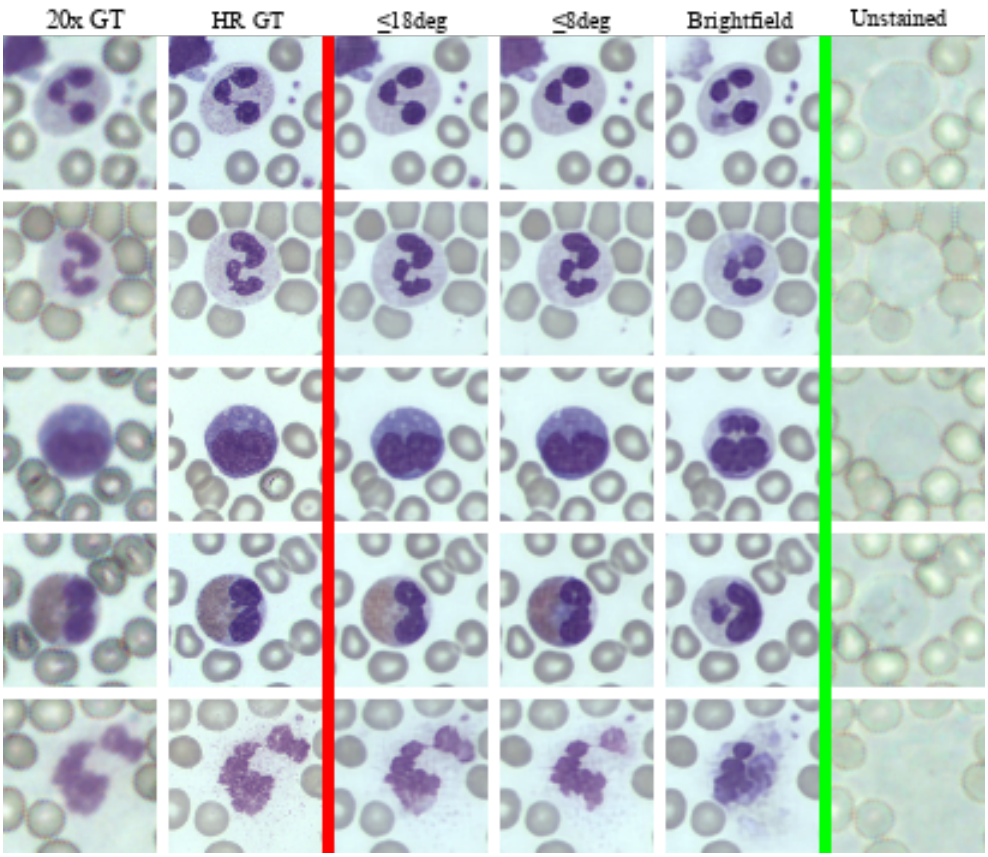


Figure 4.2: Comparison of models trained with different input stacks. To the left of the red line GT images are shown, in the middle the different model predictions are shown, and to the right of the green line the unstained original image is shown for reference. All models were trained with the HR GT images as their target.

4.1.3 Minor discussion

By analysis of both the quantitative and qualitative results it is observed that removing the double exposed DF images from the 'full' setup does not negatively impact performance. In fact, the quantitative results in Tab. 4.1 even shows slightly improved performance for the 'Full noHE' model. This aligns well with the idea that the important factor of the PLS images is the maximum angle of illumination, which is the same for both models. The increased performance could be caused simply by the fact that the generator becomes much smaller when reducing the input, and that the 'Full' model was too complex for a dataset of this size. Removing PLS images from the higher angle LED:s does however seem to negatively impact performance of the models, this is clear both in Tab. 4.1 and Figs. 4.1 and 4.2. As an example, one can view the smudge cell in the fifth row of the figures. For the first three models, 'Full', 'Full noHE' and $\leq$ 36deg, the cell is relatively equal in quality, but for the rest of the models it begins to deteriorate. One can also view the neutrophil, and the small connection between the lower and left lobes of the nucleus. The connection is present for models 'Full'-'$\leq$18deg', but disappears in the rest. From the 'Brightfield' model alone it clearly shows that the added information from the PLS images contribute well to models performance, which supports the choice to collect and include it in the dataset.

The conclusions drawn from this ablation study is that keeping PLS images from higher angle LED:s can lead to greater performance, but keeping double exposures of DF images may not. Since removing the double exposed images also leads to a significantly smaller network (meaning faster training and inference), the input used for all future models in this thesis is that of the 'Full noHE' model.

4.2 Final models

In this section the results of the previously defined models are presented. The presented models are chosen by allowing training to continue until validation loss stops decreasing, and then saving the generator weights from the epoch of training with the lowest total loss.

4.2.1 Quantitative results

The quantitative results presented here show some interesting metrics for the different models. They are calculated by evaluating the model on a previously unseen test set of 3000 samples. Metrics presented in Tab. 4.2 are the average of all predictions on the test set. It is difficult to accurately determine from these alone which model is best performing, as all of the metrics have their downsides and do not correlate perfectly with human perception; this will become increasingly clear when viewing the qualitative results in the next section.

Name	SSIM	MAE-loss	VGG-loss	Fourier loss
maeNET	0.8626	0.0468	n/a	n/a
maePNET	0.8625	0.0469	*0.0213*	n/a
maePFNET	**0.8665**	**0.0463**	**0.0207**	**0.0121**
maePF-RpGAN-SN	*0.8590*	*0.0473*	0.0212	*0.0126*

Table 4.2: Loss and average SSIM score on the test set. The generator weights were loaded from the epoch corresponding to the lowest validation loss and used to generate qualitative results in Figs. 4.1 and 4.2. Best result is written in bold text, worst result is italicized.

As the models may one day feature in a product, it is also interesting to know how fast the predictions can be made. As all of the presented models have the same architecture, their prediction speed should be the same. This was measured by saving the average time per batch during testing using either a GPU or CPU, and dividing the total time by the batch size to get the average prediction time per sample. The inference time was measured to be ∼7.19 ms when using a GPU and ∼139.04 ms when using a CPU. Considering one raw image collected by the camera is of resolution 1900x1200, and each inference can color one 64x64 patch, virtually staining one entire raw image would take approximately ∼4 s on a GPU and ∼77 s using a CPU.

4.2.2 Qualitative results

Predictions from the four previously defined models are shown in Figs. 4.3 and 4.4 below. The predicted cells were manually evaluated to determine which were of best perceptual quality and closest to the ground truth.

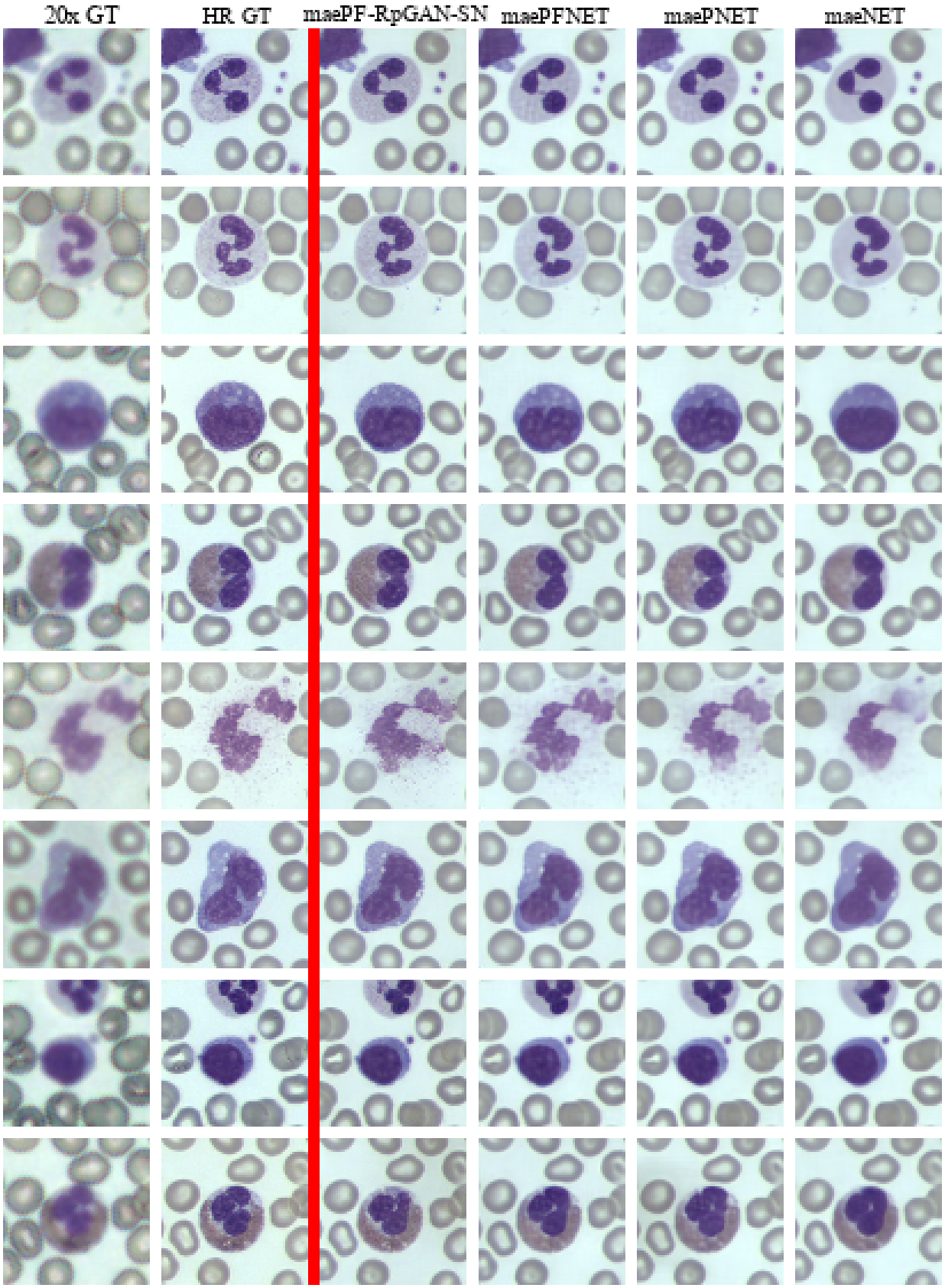


Figure 4.3: Predictions from the four different models constructed. The training target was the HR GT image, but the 20x GT is also included for easy comparison.

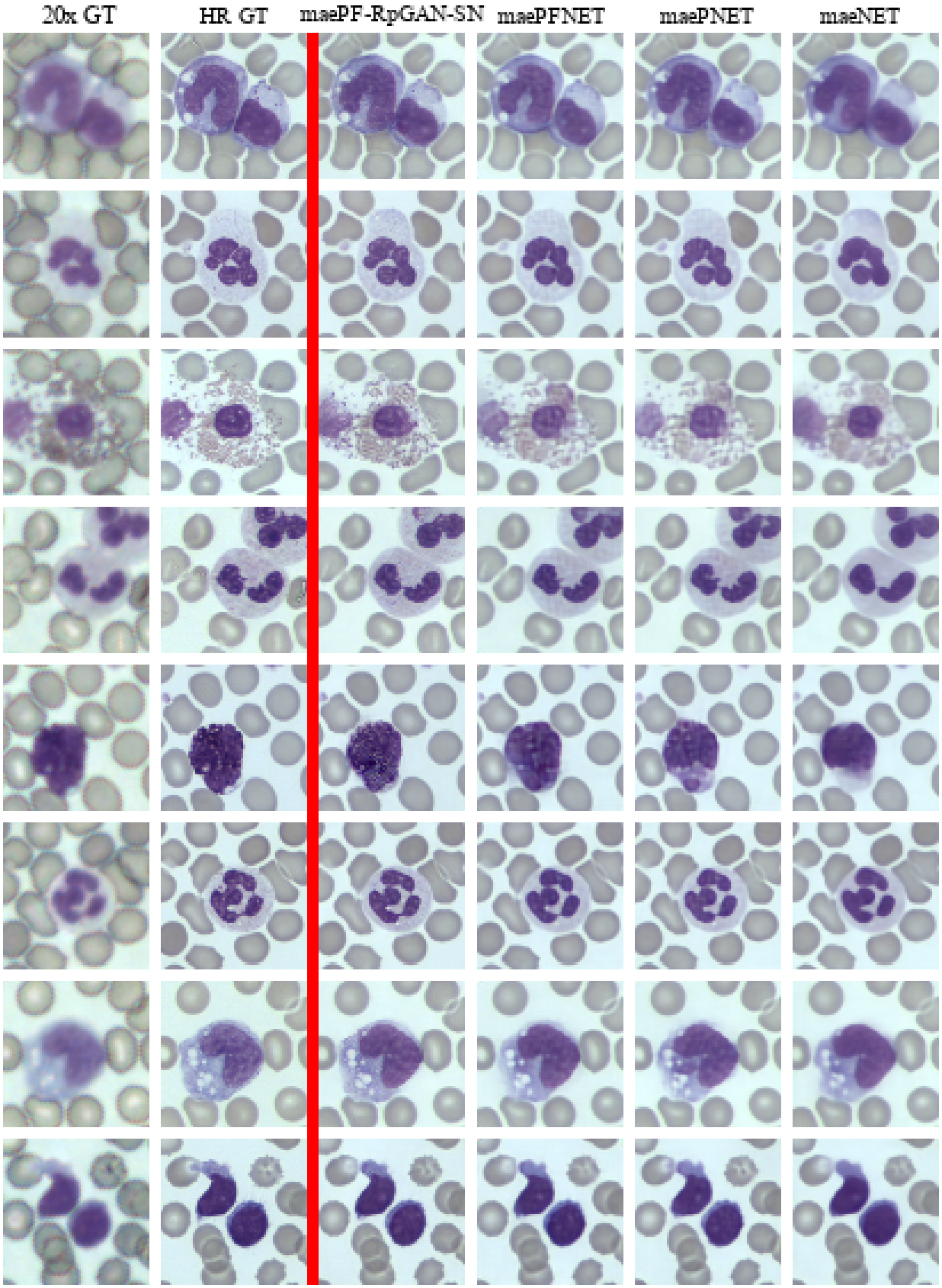
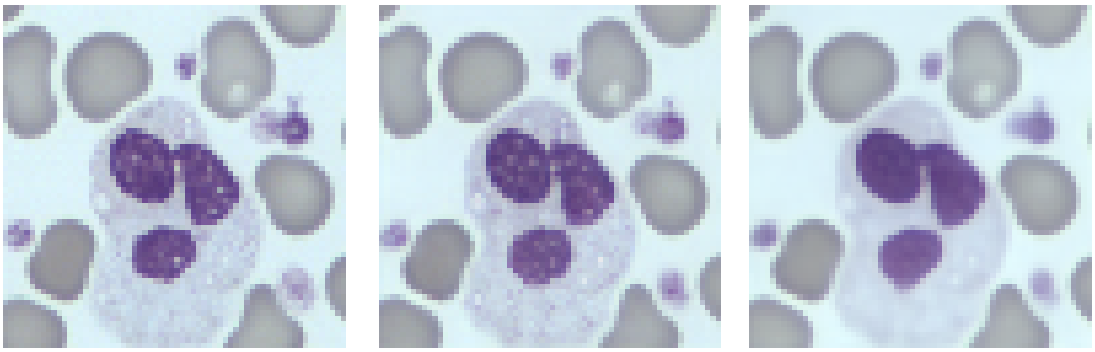


Figure 4.4: Predictions from the four different models constructed. The training target was the HR GT image, but the 20x GT is also included for easy comparison. Two things of note: the model correctly identifies the smudge granules in row 3 as coming from an eosinophil, making them red. In row 5 one can also see a basophil, a rare type of WBC.

The different models have their respective strengths and weaknesses. All of the generator networks

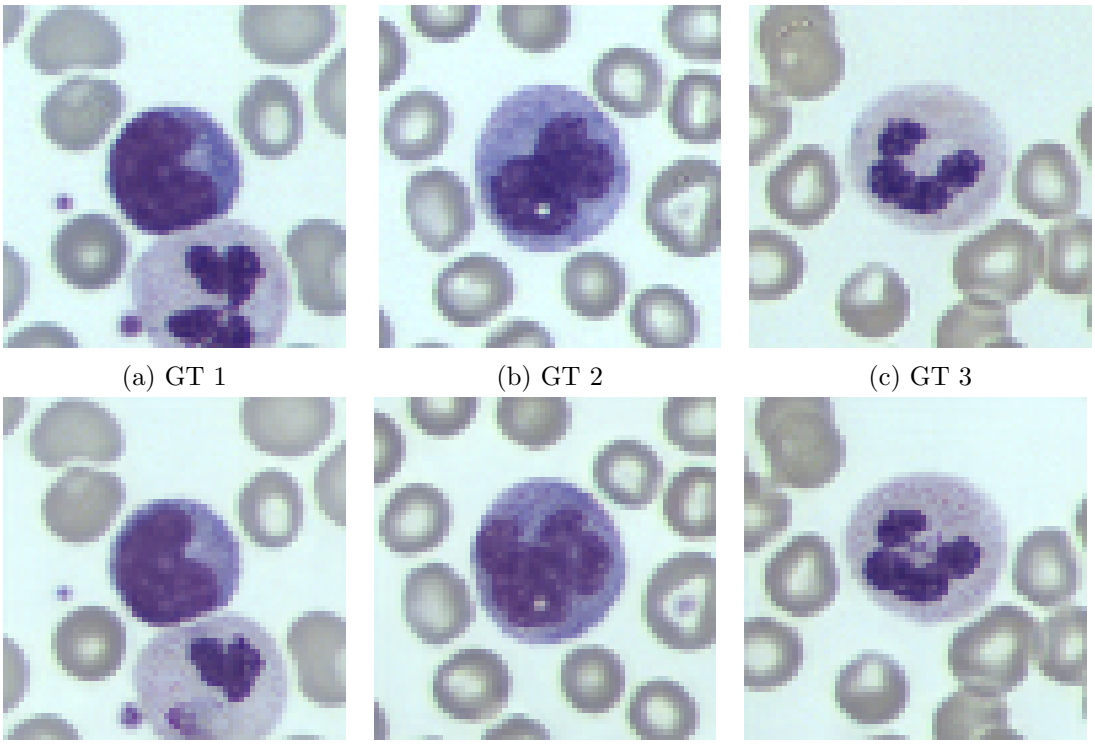
suffer from somewhat blurry results compared to the GT. This is most likely due to the MAE-loss term present in all models, and is especially clear in maeNETs predictions. Fig. 4.5 shows a comparison of maeNET, maePF-RpGAN-SN, and the HR GT, clearly showing the much blurrier result from the MAE-only model.



(a) GT (b) maePF-RpGAN-SN (c) maeNET

Figure 4.5: Comparison between GT and predictions from maePF-RpGAN-SN/maeNET illustrating blurring effect of using only MAE loss.

The addition of a discriminator seems to produce sharper images, as was intended. However, the GAN-model is not without its faults. While the predictions are generally more detailed than the other models, it is unclear whether all those details are truly present in the GT. Additionally, it seems that the GAN model is more likely to remove or add parts of the nucleus at times, which is an alarming property. Some examples of this are shown in Fig. 4.6. It is possible that using the improved HR GT image is too difficult a target to actually construct, which might be forcing the model to hallucinate the details that it cannot ascertain from the input data. This is investigated further by training an additional model with the original low-resolution GT in Sec. 4.2.5.

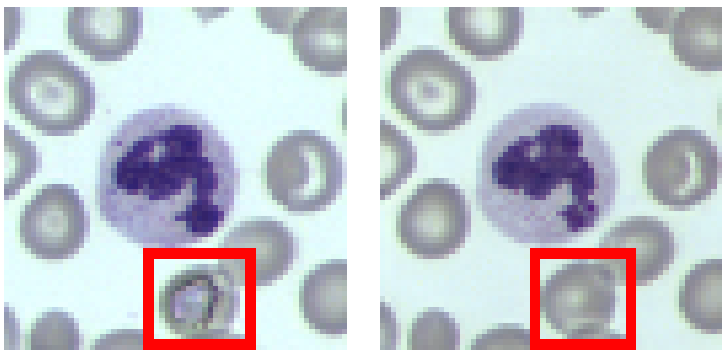


(a) GT 1 (b) GT 2 (c) GT 3

(d) Pred: removed nucleus (e) Pred: Added nucleus (f) Pred: Increased granulation

Figure 4.6: Pairs of GT and predictions made using maePF-RpGAN-SN. Upper row shows the ground truths and lower row the predictions. In d) the predicted example has removed a portion of the nucleus of one cell. In e) the nucleus has been expanded with an additional part not present in the GT. In f), increased granulation is observed in the predicted image compared to the GT, indicating that perhaps the model is sometimes "hallucinating" the finer details.

An interesting feature of virtual staining is also that it can avoid artifacts that originate from physical staining. A clear example of this is that the models avoid creating water artifacts present in some of the GTs, see Fig. 4.7.



(a) GT: Artifact (b) Pred: No artifact

Figure 4.7: a) GT image showing an artifact on the red blood cell marked by the red square. Same artifact not present in predicted image shown in b).

It is important to note that the model is not removing any "real" information when not predicting the artifact. Since the prediction is based on the unstained input, no artifact is present in the input to the model.

4.2.3 Error analysis

This section provides a short analysis on how and on what samples the model fails or performs extra well, to see if some more information can be deduced about the models. In order to find the 'best' and 'worst' samples, the quantitative metrics of total generator loss and SSIM scores were used. Manual inspection was also performed to see if the quantitatively best/worst examples actually line up with human perception of best/worst predictions.

Due to limited time only one model is investigated in this section, although the results may extrapolate to the other models as well. The model used during the error analysis was the maePF-RpGAN-SN.

4.2.4 Best/worst samples

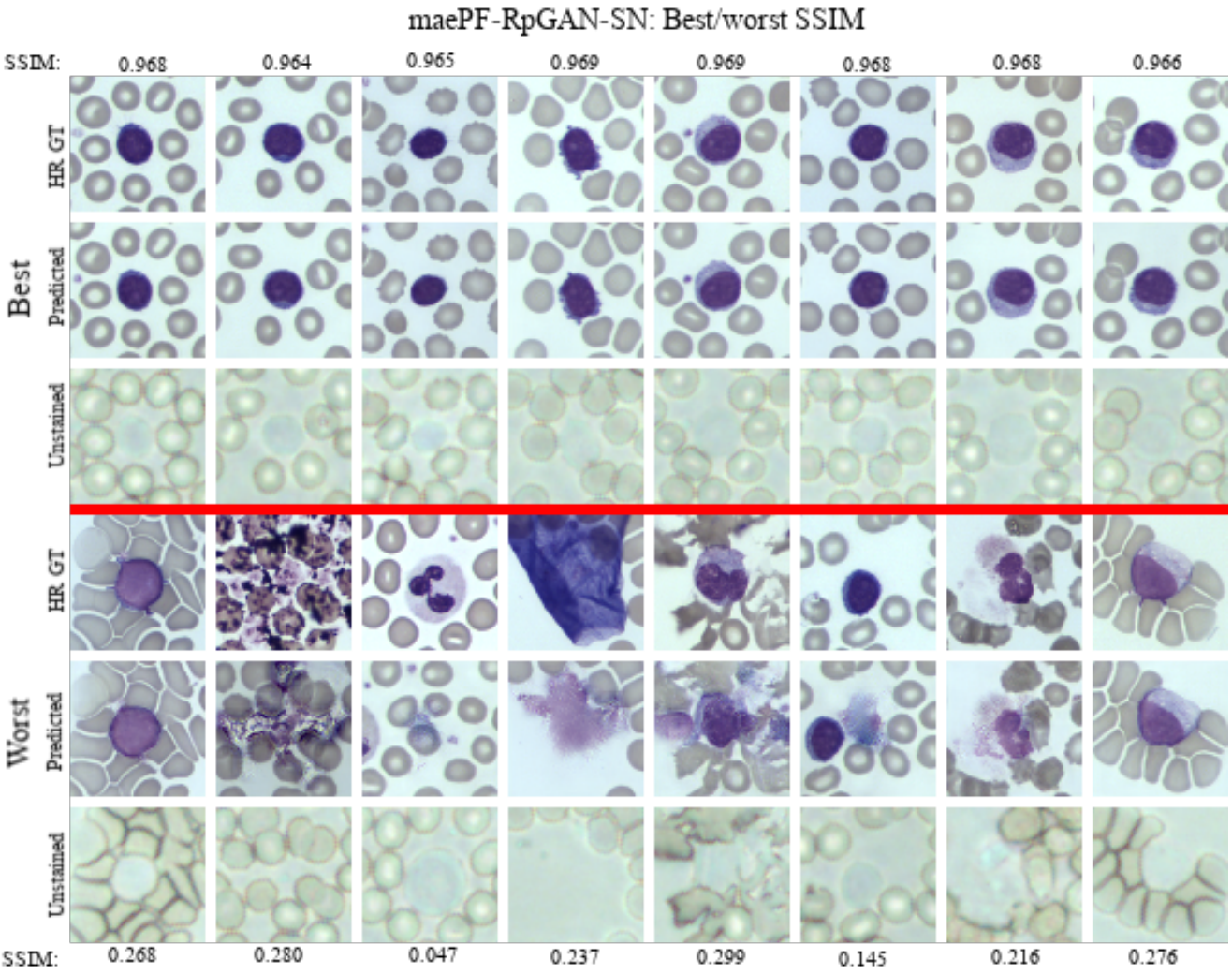


Figure 4.8: Eight examples of the best/worst predictions made by the maePF-RpGAN-SN model. The prediction is evaluated based on SSIM calculated between a grayscaled prediction and HR GT. The unstained brightfield image used as part of the input is also shown for comparison. Above the red line are the best examples, below are the worst.

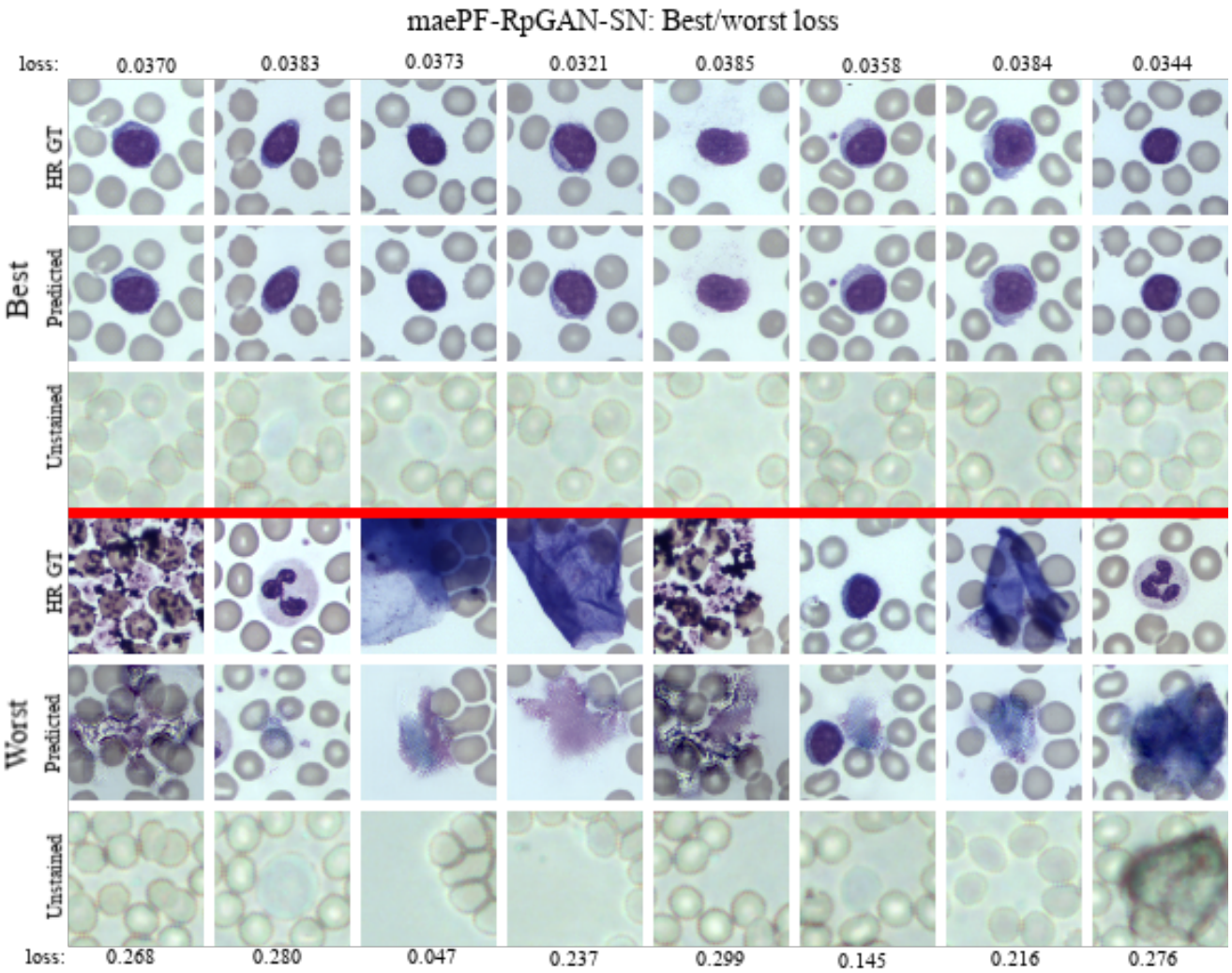


Figure 4.9: Eight examples pf the best/worst predictions made by the maePF-RpGAN-SN model. The prediction is evaluated based on the total generator loss not considering the disriminator part, in other words the loss defined in (3.4). The loss is calculated between the prediction and the HR GT. The unstained brightfield image used as part of the input is also shown for comparison. Above the red line are the best examples, below are the worst.

In both metrics, it seems the quantitative measure of the quality of the prediction is heavily tied to the quality of the matching between unstained and GT. The best predictions in both cases also seem to be of less complex cells with rounder shapes, which could be both because these are easier to match accurately and that they are simpler to reproduce, as they are less detailed. The cells are also of a commonly occurring type in that many of them are lymphocytes, which make up ∼20-40% of WBCs in blood.

Observing the worst results gives some more interesting information. It seems the worst errors occur under one of the following conditions: 1) There exists color artifacts/dirt in the GT that are not present in the input. Examples of this are shown in columns 2 and 4 in Fig. 4.8 and columns 1, 3, 4, 5 and 7 in Fig. 4.9. 2) The cells in the image are damaged, see columns 5 and 6 in Fig. 4.8, 3) There is a matching error between the full input and the ground truth, see columns 1 and 8 in Fig. 4.8 and column 8 in Fig. 4.9. Of these examples most mismatches are only of a few pixels, except for the sample in column 8 in Fig. 4.9 which is a true mismatch. 4) There is a mismatch in the PLS images and the brightfield/GT images, see columns 3 and 6 in Fig. 4.8 and columns 2 and 6 in Fig. 4.9.

The fact that two of the worst predictions in the entire test set according to the SSIM-metric are the very slightly mismatched samples in columns 1 and 8 in Fig. 4.8 shows the problem with using only a pixel-wise metric to judge the quality of the prediction. The predicted images are of a high

perceptual likeness to their respective GTs, especially compared with some of the other examples in the figure, but are still judged very harshly by the metric. One can note that no examples of this type are present in the eight worst predictions according to the generator loss in Fig. 4.9. This is due to the fact that the loss is not only based on a pixel-wise error, but also the global feature matching from the Fourier- and perceptual-losses.

All of the worst examples, except for perhaps the damaged cells in columns 5 and 7 of Fig. 4.8, should not be in the dataset in their current form, as they provide erroneous guidance to the model. The images with color artifacts from the staining process will actively encourage the model to hallucinate objects where there are none, as will the mismatched images. Removing them from the dataset as a whole could lead to further improvement, and the models presented here could be used to locate them by predicting on the full dataset and investigating the results with the highest loss.

The samples fullfilling condition 4) above were a surprise, as it was thought that the PLS images were always matched with the brightfield image: they are gathered mere milliseconds apart with no movement of the objective in between. However, it seems in at least a few examples, somehow a misalignment has happened, potentially due to hardware failure. Since there was no post-processing step that checked whether the brightfield image was matched with the PLS images, and all GTs were matched using the brightfield image only, this was not detected beforehand. Adding a step in the processing of the dataset checking the matching between PLS and brightfield images could potentially catch errors such as this and help in removing the poor samples from the dataset.

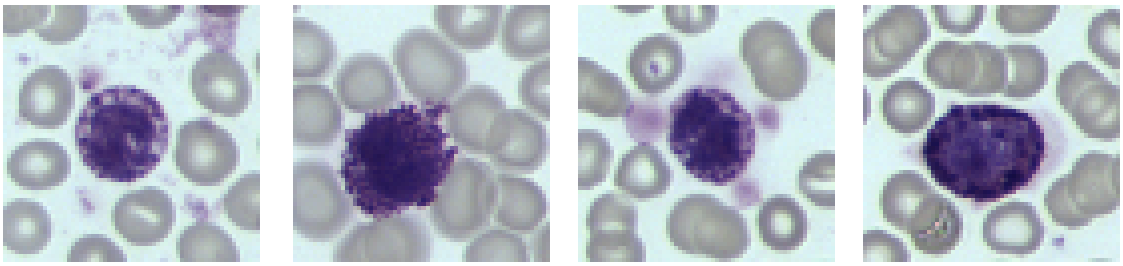
To estimate how much of the dataset consisted of poor samples, the 100 worst loss predictions on the test set were manually examined. The results of the examination are presented in Tab. 4.3 below.

Category	Nbr
Artifact in GT only	6
Dirt in input only	1
True mismatch	1
PLS mismatch	2
Dirt/damaged cells	27
Basophils w/ color artifacts	4
OK data	59
Total examined samples	100

Table 4.3: Classification of the 100 worst loss predictions made by the maePF-RpGAN-SN on the test set. In the OK data class misalignments of just a few pixels were included as it is not considered a fundamental error, requiring only some finetuning.

Given that the the errors caused by an artifact only in the GT, dirt in the input only, a true mismatch or a PLS mismatch generally leads to the highest loss values (as there is a direct difference between GT and input), one could assume that the samples detected among the 100 worst are all of the samples with these particularly serious errors in the entire test set. Given this assumption, that would mean approximately $10/3000 \approx 0.33\%$ of the test set consists of such samples. Extrapolating this number to the full dataset indicates that there should only exist a small amount of such egregious samples. However, efforts should still be made to remove them in future work.

Another interesting discovery made during this examination were the four Basophilic cells that appear to have been misstained, and are surrounded by color artifacts, see Fig. 4.10. It is not known why this has occurred, but it may indicate there is an issue with the staining procedure used that more often misstains basophils. This could be a partial factor in the poor performance when predicting on the basophil in the general results above.



(a) (b) (c) (d)

Figure 4.10: Potentially misstained Basophils. It appears almost as if some cell parts have "leaked" out of the actual cell, creating these purple patches on what would otherwise have been the background.

Summarizing, both the best and the worst samples of the model seem to be highly linked to the quality/diversity of the dataset. With this in mind, future work has much to gain from iterating over the dataset formulation in order to create a stronger and more reliable model.

4.2.5 Using 20x Ground Truth as target

To provide additional results for discussion the two best performing models, maePFNET and maePF-RpGAN-SN, were also trained using the 20x GT as their target. The results of those experiments are presented in this section.

Quantitative results

In Tab. 4.4 quantitative results are shown. One can note the increased SSIM values when compared with the results using the HR GT target. This is due to the increased blur present in the 20x GT images; it naturally leads to less sensitivity to matching errors, and provides an easier prediction.

Name	SSIM	MAE-loss	VGG-loss	Fourier loss
20x GT: maePFNET	**0.8993**	**0.0278**	**0.0194**	0.0076
20x GT: maePF-RpGAN-SN	0.8968	0.0287	0.0195	0.0076

Table 4.4: Average loss and SSIM scores on the test set. The generator weights from the epoch corresponding to the lowest validation loss are loaded and used to generate qualitative results in Figs. 4.1 and 4.2. Best result is written in bold text.

Qualitative results

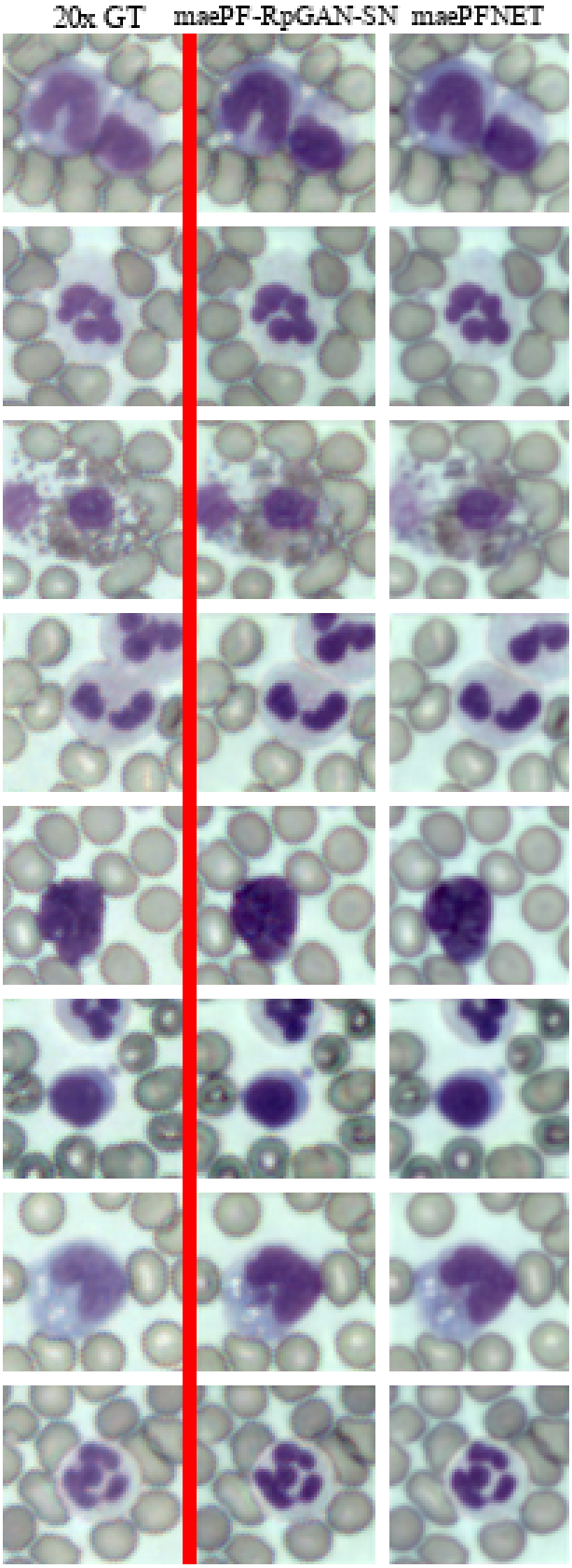


Figure 4.11: Predictions made by maePFNET and maePF-RpGAN-SN networks trained using the 20x stained image as GT. Note the difference in color between GT and predictions, with predictions being more saturated. One can also note the expanded nucleus in row 7 of the maePF-RpGAN-SN prediction, indicating that the GAN formulation still may lead to more unstable solutions even when using the 20x GT.

4.3 Exploring future improvements

In this section results from training models using either half of the available training data or a deeper generator are presented.

4.3.1 Quantitative results

The deeper model and the model trained with a smaller dataset are compared with the previously presented models of the same type as a baseline in Tab. 4.5 below.

Name	Max. SSIM	Min. MAE-loss	Min. VGG-loss	Min. Fourier loss
maePFNET (Baseline)	**0.8665**	0.0463	**0.0207**	**0.0121**
Deeper (8 RRDB blocks)	0.8643	**0.0462**	0.0210	0.0125
Smaller dataset	*0.8613*	*0.04578*	*0.0219*	0.0125
maePF-RpGAN-SN (Baseline)	0.8590	0.0473	0.0212	0.0126
Deeper (8 RRDB blocks)	**0.8644**	**0.0470**	**0.0207**	**0.0124**
Smaller dataset	*0.8565*	*0.0484*	*0.0217*	*0.0128*

Table 4.5: Quantitative results comparing deeper models and models trained using only half of the available dataset. Baseline models are trained using a generator with four RRDB-blocks and the full dataset. Best results are marked in bold, worst are italicized.

Training a model of double depth meant increased training/inference time versus the shallower models. This is definitely not a trivial increase in training time, as the baseline models alone could take up to 12-15 hours of training to converge. Furthermore, the quantitative results do not show a clear increase in performance for the investigated metrics for the maePFNET model, only for maePF-RpGAN-SN. It is possible that both models could benefit from increased depth, but larger datasets my be needed to train them.

Decreasing dataset size seems to more clearly inhibit performance of the model. This is as expected, and indicates that using a larger dataset may lead to generally increased performance.

The inference time of the 8 RRDB-block deep generators was measured to be ∼10.90 ms on a GPU and ∼250.67 ms on a CPU.

4.3.2 Qualitative results

Qualitative results comparing the predictions of the models are shown in Fig. 4.12 and Fig. 4.13.

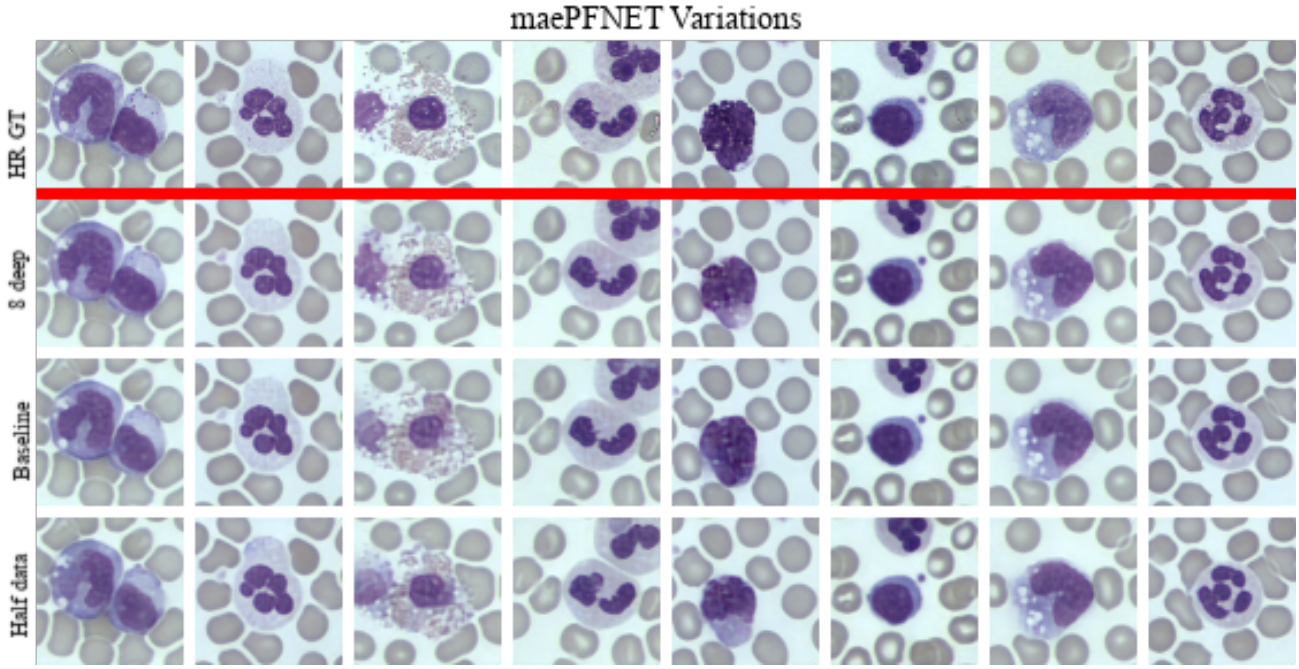


Figure 4.12: Comparison between three models trained with maePFNET. 'Baseline' is the same model as in Fig. 4.3, with four RRDB blocks in the generator, while '8 deep' is using a generator with eight RRDB blocks instead. 'Half data' is the same network as 'Baseline' except it has been trained using only half of the dataset. All networks use identical discriminators.

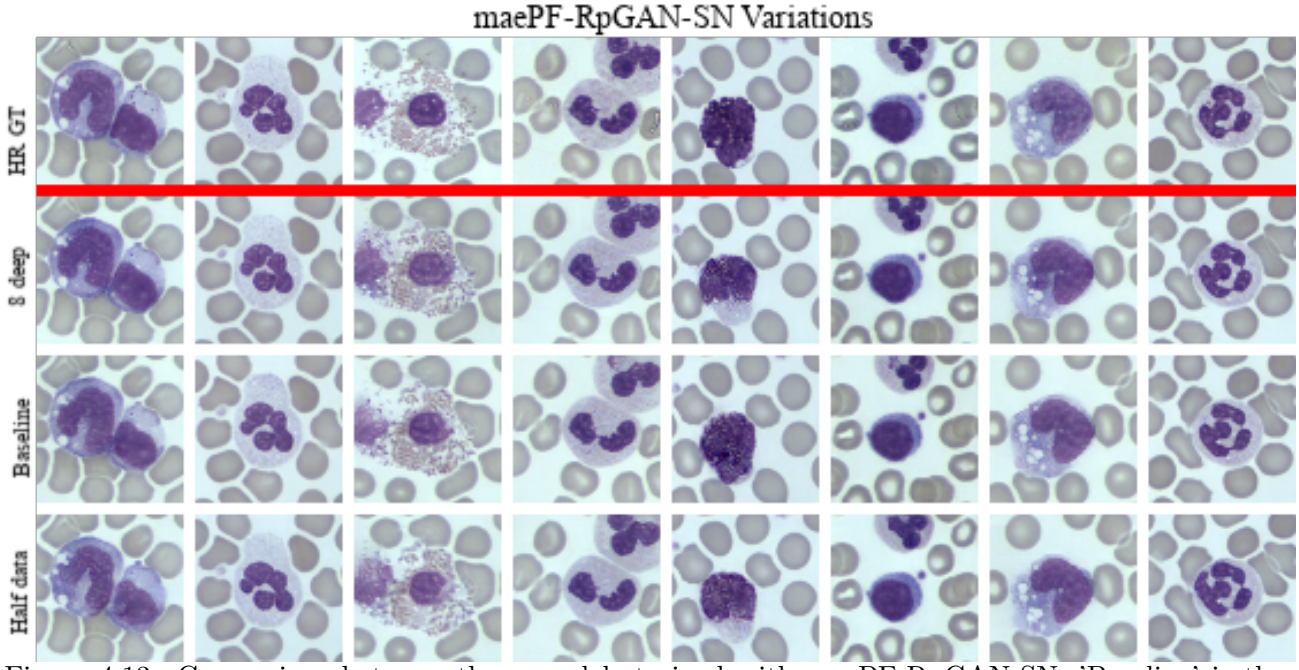


Figure 4.13: Comparison between three models trained with maePF-RpGAN-SN. 'Baseline' is the same model as in Fig. 4.3, with four RRDB blocks in the generator, while '8 deep' is using a generator with eight RRDB blocks instead. 'Half data' is the same network as 'Baseline' except it has been trained using only half of the dataset. All networks use identical discriminators.

For both model types it seems that removing half of the data generally leads to slightly worse predictions, as expected. The deeper models are not as clear.

For the maePFNET, the 8 deep model is judged to produce slightly better results on all samples except the eosinophil smudge in column 3 and the basophil in column 5. In the smudge cell it appears to model the leftmost nucleus section less accurately than the baseline, but the granules are clearer.

It is difficult to judge which prediction is truly better. As for the basophil, the baseline is clearly more accurate, as the deeper model has given it a lighter tone than the GT. This does not necessarily mean that the baseline model is actually better in general, instead it could simply be a consequence of the dataset. Basophils are exceedingly rare, and as such will be harder for the model to accurately learn how to color. There is also some instability in the solutions in general, so perhaps choosing an earlier or later epoch would produce a better result. Overall, it is judged that the increased depth model creates the best predictions when using the maePFNET, but the difference is small.

Considering the maePF-RpGAN-SN, it is difficult to see any general improvement of prediction quality when using the deeper model. The deeper model performs both better and worse than the baseline on different samples. One can see a clear difference between the two models in the construction of the edge nuclei in columns 5 and 6 and, the nuclei of the cell in column 7, and, as in the maePFNET case, the basophil.

The deeper model correctly stains most of the edge nuclei where the baseline fails. However, so does the half data model. As the half data generator is the same one used in the baseline, this shows that the accurate staining of the edge nuclei is not related only to the architecture of the generator used, but also the specific training performed. Therefore it is not clear that this improvement is due to the increased depth.

The deeper model hallucinates an extra component of the nucleus in the cell in column 7, which the baseline does not. Here, it seems the deeper model performs worse.

As for the basophil the same arguments as for the maePFNET model could be made, which lead to the difference in performance between the deeper model and the baseline. However, in the case of the maePF-RpGAN-SN model, no judgement can be made on whether the deeper model actually increases the quality of the predictions.

Chapter 5

Discussion

5.1 General comments

All four models produce reasonable results, although of varying quality. None of the models manages to perfectly reconstruct the HR GT target, but bear in mind that this is a very tough goal to achieve. All of the input PLS images have been collected on dry slides using a 20x 0.4 NA objective, with which the actual stained cell appears as in the first column in Figs. 4.3 and 4.4. These images look even blurrier than the simplest model trained with the HR GT, so there is not only a staining process being modeled by the network but also a sharpening. All models have also learned to remove the 'dark shadow' artifacts present in the 20x images, producing more clean images.

Observation of the quantitative results show that in general, it is difficult to judge the images quality using only these metrics; We see for example that the RpGAN model has the lowest average SSIM score, despite producing the most sharp images. All in all, the SSIM-scores are also very similar, which increases the difficulty in really seeing the best model from these values. Due to this, most evaluation of results has been done by manually viewing the predicted images. The goal was to create virtually stained images that are indistinguishable from their respective GT, which has not been achieved. Despite this, the results are good enough to show the potential of the proposed solution, and provides a strong baseline for future work.

Most of the models produce images close to their GT, but the details are not always correct; for example, the nucleus shape of the monocyte in the third row of Fig. 4.3 has an added portion in the maePNET prediction, or the smudge cell has added sections in the maePF-RpGAN-SN prediction. It has been observed that these types of small errors in nucleus shape can appear and disappear between epochs of training, even far into the training. This suggests that the model may struggle to converge to a stable solution, or that perhaps the data is not perfect; there could be a dust particles or other disturbing objects present in the input images not present in the GT. The error analysis in Sec. 4.2.3 shows this phenomena, and that there are indeed samples in the dataset that create mismatches between input and GT. Even though they are estimated to be small in number, it is unknown how large of an effect they have on the general model performance. Furthermore, the error analysis of both the best and worst samples point to the performance of the network predictions being heavily tied to the quality of the input/GT data. In order to draw certain conclusions on the true capacity of the model architectures presented one will need to continue iterating over the dataset formulation, both increasing its size and its quality by removing artifacts/mismatches and adding more good samples. What can be concluded using the current results however is that as a proof-of-concept, the proposed solution of this thesis shows potential: the additional information from the PLS images along with a neural network are able to virtually stain blood cells, at least in certain cases.

Of all of the investigated models, the maePFNET is the formulation recommended for future use. While the GAN-formulation certainly merits further investigation, its current version is deemed too unreliable, and likely to hallucinate details that are not actually present in the true cell. The

maePFNET may produce less detailed outputs, but at least it is unlikely to add or remove large portions of the cells.

5.2 Generator Networks

In general the generator networks were simple to train and seem to produce their results more reliably than the investigated GAN formulation. Each added loss term clearly leads to increased performance, and the maePFNET-version is deemed the most well-rounded formulation of all models presented in this thesis with both high fidelity and reliability in its predictions.

5.2.1 maeNET

The maeNET suffers from a blurring of the image and does not reconstruct the high-frequency details correctly. This is expected when using only a pixelwise MAE-loss, as it is known to prefer blurry solutions as they are more likely to cause a smaller loss (especially when the matching is not always pixel-perfect).

5.2.2 maePNET

The addition of a perceptual loss in maePNET shows clear improvements in the sharpness of the predicted image, although it seems to still get some of the details incorrect. An example of this is the nucleus shape of the monocyte in the third row of Fig. 4.3.

5.2.3 maePFNET

The addition of the Fourier loss in maePFNET produces images with slightly better quality than maePNET. This is reflected both in better quantitative metric scores in Tab. 4.2 and increased perceptual quality in Figs. 4.3 and 4.4. It is worth noting that the differences between maePNET and maePFNET are very small, but nevertheless indicate that including the Fourier loss term leads to a stronger model. Additionally it ended up decreasing the number of training epochs required for model convergence, most likely due to a stronger loss signal from the added Fourier term.

In the extended investigations regarding increased model depth, the deeper 8 RRDB generator model with maePFNET-loss seemed to lead to improved general performance, and is the formulation recommended for use.

5.3 maePF-RpGAN-SN

The GAN-model is more complex and this is reflected in its results. The quantitative scores are lower than some of the simpler generator models, but despite this the GAN-model outclasses them when it comes to sharpness and details in their predictions. This is seen clearest in the smudge cells and granulated neutrophils of Fig. 4.3 and 4.4.

However, as already mentioned in the results section, it is not clear that these details are always the true details present in the cell, and the model also seems to more easily erase parts of the nucleus. This is an extremely worrying result when considering application in industry; if the model is able to remove parts of nuclei it could lead to further errors in classification and analysis which could mean illnesses going unnoticed or mistreated. It is unclear why this behavior has started appearing in the model. One hypothesis is that the adversarial training has simply allowed the generator to focus more on producing detailed images, but the generator is not able to 'see' the actual details present in the cells, so it starts hallucinating them instead. When it learns that it can increase scores by changing or hallucinating objects, perhaps it also opens the door for actions such as removing nuclei. Of course these actions should in theory be penalized by the other loss terms in the loss function, but it seems these behaviors are at least still present in the current models. Three potential reasons for the model to start hallucinate are 1) That the information necessary to reproduce the GT is simply not present

in the input data, 2) The model is not complex enough to model the transformation, 3) The models needs more training data to converge to the correct solution. If the reason is 1), and the information in the PLS images simply is not enough, then there is not much improvement to be made and one will have to rethink the gathered data or the GT used. This is explored in Sec. 4.2.5 where the simpler, 20x GT is used as a training target, but even in this case it seems the GAN-model ends up hallucinating details. If the reason is 2) or 3), then perhaps more hints on future work can be had by analyzing the results in Sec. 4.3 and improving the size and quality of the full dataset.

The hallucinations observed in some predictions of this model could also explain its lower quantitative results when compared with the maePFNET model. In some images, the GAN-model produces the most accurate and high-fidelity images of all of the presented models. If it did this for all of the samples, it should score the highest on its quantitative results. However, the samples where it hallucinates erroneously drag down its scores by a large amount, giving an average prediction result that is lower than that of maePFNET. Its behavior results in it either performing really well, as in Fig. 4.5 b), or really poorly, as in Fig. 4.6 d). Reliability seems to have been sacrificed in order to allow for some extremely well detailed predictions.

Because of the GAN-models tendencies to hallucinate details it is, at least in its current state, deemed a poor candidate to this virtual staining problem. Increased details are not inherently good, unless they are in fact present in the true cell. Instead, they become problematic, and could potentially lead to misdiagnosis if ever used clinically. The model could however potentially be improved upon by being given a larger and higher quality dataset, as well as further hyperparameter experimentation of the loss function. It may be that reducing the weight of the loss stemming from the discriminator could reduce hallucinations, while still producing a generally sharper image than that of the generator networks.

5.4 Choice of ground truth

A hypothesis that occurred during analysis of the main results presented in Sec. 4.2 was that perhaps the choice of using the more detailed and structured HR GT was too ambitious a goal. The idea behind the choice was that it would provide stronger supervision, and allow the model to produce stained images of even higher quality than if they were chemically stained and recollected using the original 20x objective. The expectation was that even if it would not quite reach this goal, the partial solution would still be better than 20x. However, a possible consequence of this setup is that the information required to recreate these ground truth images is simply not present in the input. The hope was that the PLS images would give enough information that the models would not have to resort to too much statistical modeling, meaning guessing, when staining the images, but the results indicate that this may be the case. Hallucinations, instabilities in predictions, and similar errors may be occurring simply because the setup is too ill-posed.

With this in mind, the extra models presented in Sec. 4.2.5 were trained using the 20x GT as their target to see how the models behave when given a simpler target. In general, the images are closer to their GTs than in the HR GT models, and perhaps this is a more reasonable setup for future work; It keeps the focus on only modeling the staining, and leaves out the sharpening/enhancing of the image that needed to be included when attempting to model the change of physical objective as well. However, even in this scenario the GAN-model seems to occasionally hallucinate nuclei, and the colors of the predicted cells seems further from the GT than in the HR GT case. It is difficult to know exactly why this is the case, and it is important to note that the loss function weights used in the two model types are the same. These were optimized and tested for use with the HR GT, and may not work as well when using the 20x GT. An interesting future experiment could be to create a new dataset, where unstained blood cells are collected along with their PLS images using a 100x objective directly, and then matching that with the 100x GT, keeping the two at the same magnification/resolution.

Bibliography

[1] RW Horobin. How romanowsky stains work and why they remain valuable — including a proposed universal romanowsky staining mechanism and a rational troubleshooting scheme. *Biotechnic & Histochemistry*, 86(1):36–51, 2011, https://doi.org/10.3109/10520295.2010.515491. URL `https://doi.org/10.3109/10520295.2010.515491`.

[2] Bernadette F. Rodak and Jacqueline H. Carr. *Clinical Hematology Atlas (Fifth edition)*. Elsevier Health Sciences, 2015. ISBN 978-0-323-32249-2.

[3] CellaVision. Leukocytes in peripheral blood. `https://www.cellavision.com/en/cellavision-cellatlas/leukocytes`. Accessed 6 December 2021.

[4] Yair Rivenson, Tairan Liu, Zhensong Wei, Yibo Zhang, Kevin de Haan, and Aydogan Ozcan. Phasestain: the digital staining of label-free quantitative phase microscopy images using deep learning. *Light: Science & Applications*, 8(1):23, Feb 2019. ISSN 2047-7538. URL `https://doi.org/10.1038/s41377-019-0129-y`.

[5] Nischita Kaza, Ashkan Ojaghi, Paloma Casteleiro Costa, and Francisco E. Robles. Deep learning-based virtual staining of label-free ultraviolet (UV) microscopy images for hematological analysis. In Natan T. Shaked and Oliver Hayden, editors, *Label-free Biomedical Imaging and Sensing (LBIS) 2021*, volume 11655, pages 16 – 24. International Society for Optics and Photonics, SPIE, 2021. URL `https://doi.org/10.1117/12.2576429`.

[6] Guoan Zheng, Christopher Kolner, and Changhuei Yang. Microscopy refocusing and dark-field imaging by using a simple led array. *Optics letters*, 36:3987–9, 10 2011.

[7] Guoan Zheng, Roarke Horstmeyer, and Changhuei Yang. Wide-field, high-resolution fourier ptychographic microscopy. *Nature Photonics*, 7(9):739–745, Sep 2013. ISSN 1749-4893. URL `https://doi.org/10.1038/nphoton.2013.187`.

[8] Chang Qiao, Di Li, Yuting Guo, Chong Liu, Tao Jiang, Qionghai Dai, and Dong Li. Evaluation and development of deep neural networks for image super-resolution in optical microscopy. *Nature Methods*, 18(2):194–202, Feb 2021. ISSN 1548-7105. URL `https://doi.org/10.1038/s41592-020-01048-5`.

[9] S. Hacking and V. Bijol. Deep learning for the classification of medical kidney disease: a pilot study for electron microscopy. *Ultrastruct Pathol*, 45(2):118–127, Mar 2021.

[10] Ida Wagnström and Cajsa Olofsson. Deblurring of cell images using generative adversarial networks, 2019. ISSN 1404-6342. Student Paper.

[11] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[12] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. ISSN 0893-6080. URL `https://www.sciencedirect.com/science/article/pii/S0893608098001166`.

[13] Scott C. Douglas and Jiutian Yu. Why relu units sometimes die: Analysis of single-unit error backpropagation in neural networks. *CoRR*, abs/1812.05981, 2018, 1812.05981. URL http://arxiv.org/abs/1812.05981.

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

[15] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016, 1603.08155. URL http://arxiv.org/abs/1603.08155.

[16] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, and Xiaoou Tang. ESRGAN: enhanced super-resolution generative adversarial networks. *CoRR*, abs/1809.00219, 2018, 1809.00219. URL http://arxiv.org/abs/1809.00219.

[17] Dario Fuoli, Luc Van Gool, and Radu Timofte. Fourier space losses for efficient perceptual image super-resolution, 2021, 2106.00783.

[18] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016, 1609.04747. URL http://arxiv.org/abs/1609.04747.

[19] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg, 2010. Physica-Verlag HD. ISBN 978-3-7908-2604-3.

[20] C. Darken, J. Chang, and J. Moody. Learning rate schedules for faster stochastic gradient search. In *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, pages 3–12, 1992.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017, 1412.6980.

[22] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12 (1):145–151, 1999. ISSN 0893-6080. URL https://www.sciencedirect.com/science/article/pii/S0893608098001166.

[23] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. URL http://jmlr.org/papers/v12/duchi11a.html.

[24] Geoffrey Hinton. Coursera online course: Neural networks for machine learning, lecture 6. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[25] Robin M. Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley - benchmarking deep learning optimizers. *CoRR*, abs/2007.01547, 2020, 2007.01547. URL https://arxiv.org/abs/2007.01547.

[26] 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, 06 2014.

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

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015, 1512.03385. URL http://arxiv.org/abs/1512.03385.

[29] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution

using a generative adversarial network. *CoRR*, abs/1609.04802, 2016, 1609.04802. URL `http://arxiv.org/abs/1609.04802`.

[30] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016, 1608.06993. URL `http://arxiv.org/abs/1608.06993`.

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015, 1505.04597. URL `http://arxiv.org/abs/1505.04597`.

[32] Vincent Billaut, Matthieu de Rochemonteix, and Marc Thibault. Colorunet: A convolutional classification approach to colorization. *CoRR*, abs/1811.03120, 2018, 1811.03120. URL `http://arxiv.org/abs/1811.03120`.

[33] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016, 1611.07004. URL `http://arxiv.org/abs/1611.07004`.

[34] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014, 1406.2661.

[35] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 2017, 1701.00160.

[36] Lars Ruthotto and Eldad Haber. An introduction to deep generative modeling, 2021, 2103.05180.

[37] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge?, 2018, 1801.04406.

[38] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2017, 1611.04076.

[39] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017, 1701.07875.

[40] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan, 2018, 1807.00734.

[41] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018, 1802.05957. URL `http://arxiv.org/abs/1802.05957`.

[42] Yusuke Horiuchi, Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Spectral normalization and relativistic adversarial training for conditional pose generation with self-attention. In *2019 16th International Conference on Machine Vision Applications (MVA)*, pages 1–5, 2019.

[43] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. *CoRR*, abs/1707.02921, 2017, 1707.02921. URL `http://arxiv.org/abs/1707.02921`.

[44] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. Deep multi-scale convolutional neural network for dynamic scene deblurring. *CoRR*, abs/1612.02177, 2016, 1612.02177. URL `http://arxiv.org/abs/1612.02177`.

[45] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks, 2018, 1807.02758.

[46] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. *CoRR*, abs/1802.08797, 2018, 1802.08797. URL `http://arxiv.org/abs/1802.08797`.

[47] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[48] Robert Fischer, Simon Perkins, Ashley Walker, and Erik Wolfart. Hypermedia image processing reference, 2003. URL https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm.

[49] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *CoRR*, abs/1405.3531, 2014, 1405.3531. URL http://arxiv.org/abs/1405.3531.

[50] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017, 1706.08500. URL http://arxiv.org/abs/1706.08500.

[51] Robert Fischer, Simon Perkins, Ashley Walker, and Erik Wolfart. Hypermedia image processing reference, 2004. URL https://homepages.inf.ed.ac.uk/rbf/HIPR2/close.htm.

[52] Open cv documentation: Template matching. URL https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html. Accessed 14 december 2021.

Appendix A

Image processing and examples

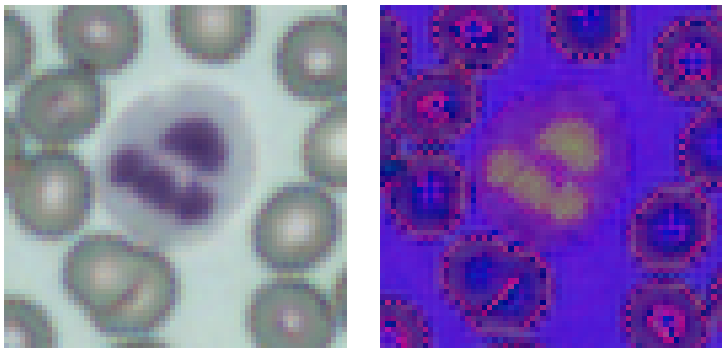
A.1 Image processing

To create the datasets to feed the neural networks with several image processing techniques were used. In this section of the paper the basic theory of the operations used are explained in more detail for later reference.

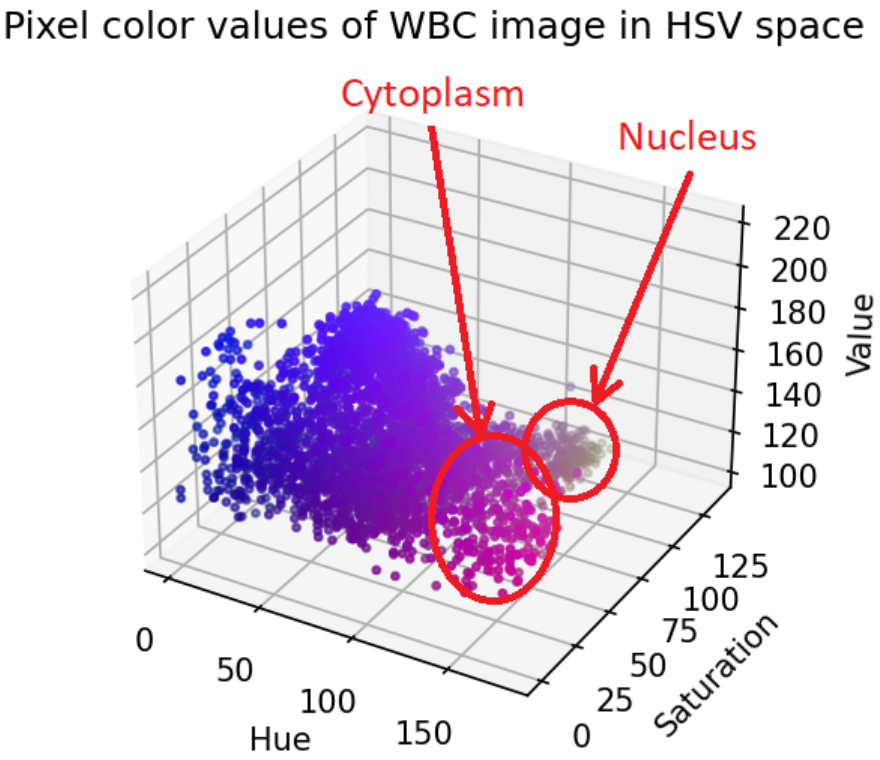
A.1.1 Color segmentation

In order to locate WBC:s in a colored blood smear a segmentation algorithm was needed. Since the WBC are colored in a well-defined blue-purple color compared to the RBC:s, using color segmentation was a natural choice. The raw image to be segmented is first transformed from the RGB colorspace to the HSV space. The HSV colorspace is designed to more closely align with the human perception of color. It is a cylindrical representation of color, with the angular dimension being hue, the radial dimension being saturation and the vertical being value (representing the amount of light in the color).

Analysis of a colored WBC shows the area of the HSV color space where the WBC and its nucleus is most represented. A 3D scatter plot of the HSV color values for all pixels is shown in Fig. A.1 c) with the areas representing the WBC circled in red.



(a) RGB (b) HSV



(c) Scatter plot

Figure A.1: a) An image of a WBC in RGB space. b) The same WBC after conversion to HSV space. c) A scatter plot of the values of the H, S, and V channels of the HSV image in b). The colors of the dots is decided by the corresponding color in the hsv image. The Nucleus can be seen as represented by bright white/grey and the cytoplasm by a pink/purple. The values on the different axis is in opencv's representation of the HSV space, where H is in the range [0, 180] and S and V are in the range [0, 255].

A range of values in the Hue, Saturation and Value channels can then be estimated and used to threshold an HSV color image into a binary image, where pixels with colors inside the range are set to 1 and those with color outside the range are set to zero, leaving a segmented image. The image can be further improved through a morphological closing operation [51] to reduce noise and connect parts of nuclei that have been detected individually. Fig. A.2 depicts an example image going through the steps described above.

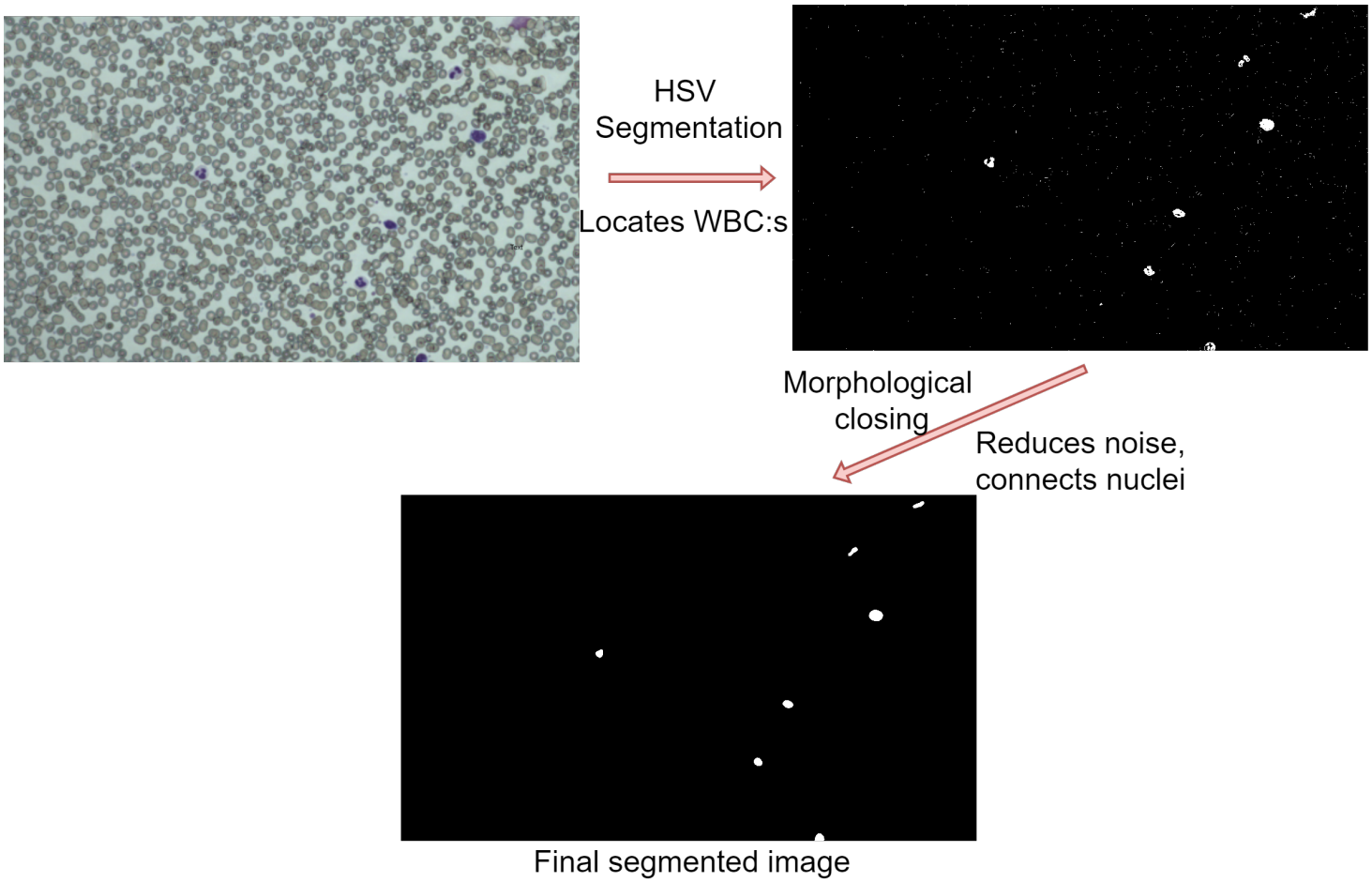


Figure A.2: Example of image being color segmented and closed to produce a clean, binary image of detected WBC:s.

A.1.2 Template matching

Template matching is a method to find a template image in a larger search image. The template is slid pixel-by-pixel over the entire search image, and for each position a correlation metric is computed. The position that produces the best correlation metric is then chosen as the matching position.

The correlation metric used in this project is the normalized correlation coefficient implemented in openCV:s matchTemplate function, and calculates the score at each position (x, y) according to the following equation[52]:

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x', y+y'))}{\sqrt{\sum_{x',y'}(T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x', y+y')^2)}}, \tag{A.1}$$

where T represents the template image, I represents the search image and R represents a matrix of results. The best match can be found by the position (x, y) of the maximum result in the result matrix R.

A.2 U-net vs ESR generator

The U-net generator architecture used in this test was based on that presented in the Pix2Pix paper by ... et al [33]. The general structure is the same as presented in Sec. 2.8, and is composed of an encoder-decoder structure. However, a key difference lies in the shape of the input to the model: in the classic Pix2Pix version, the input and output are both composed of a single image (either color or single channel). The input used in this thesis is instead composed of multiple images, giving inputs

with up to 225 channels. This results in the model becoming very large very quickly when following the pix2pix pattern of doubling the number of filters during encoding. An example architecture using the 138-channel 'full no HE' input is shown in Fig. A.3.

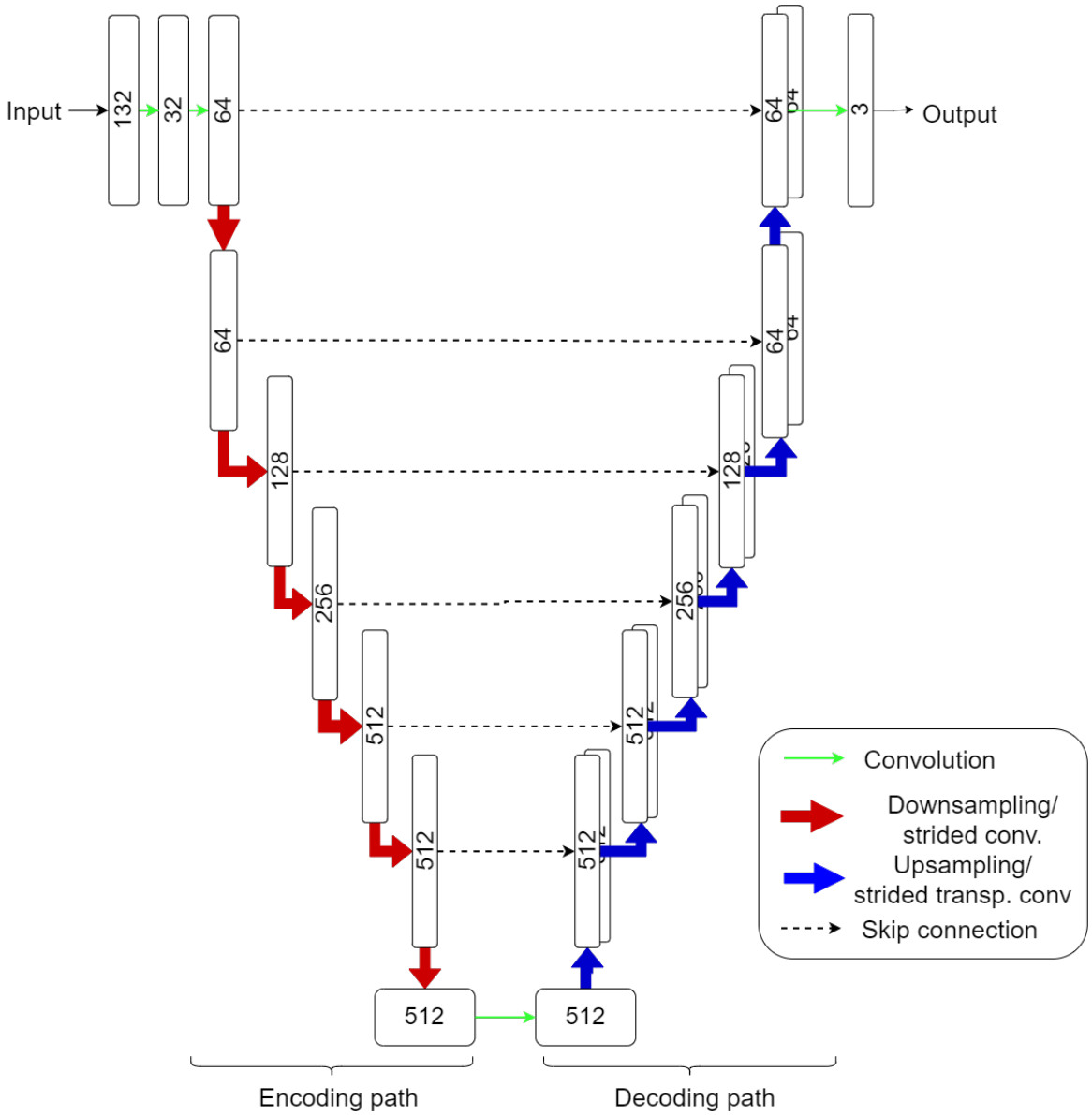


Figure A.3: Illustration of U-net generator. The number within each block defines the number of filters used. Each block other than the first applies batch normalization, and the first three blocks in the upsampling path apply dropout, as in [33]. The input shape is 64x64x132, and is reduced to 1x1x512 in the final step of the encoding path.

In this model, an extra convolutional layer has been added before the encoding path to reduce the number of channels in the first encoding step, reducing the number of trainable parameters. Despite this, the model ends up with over 30 million trainable parameters and ends up heavily overfitting to the dataset, generalizing worse while also being larger than the comparative ESR-based generator.

A.2.1 Quantitative results

The results shown in Tab. A.1 below clearly indicate increased performance from the ESR-based generator despite its much smaller complexity. It may be possible to construct different U-net based solutions for this problem that prove useful, but the results shown here were considered enough to limit further investigations of U-net in favor of ESR-based alternatives.

Name	Nbr trainable params.	SSIM	MAE-loss	VGG-loss	Fourier loss
maePFNET (ESR-based)	6515463	**0.8665**	**0.0463**	**0.0207**	**0.0121**
U-net	29279011	0.8344	0.0527	0.0251	0.0142

Table A.1: Quantitative results comparing U-net and ESR-based models. Both models are trained using the maePF-loss defined in eq. (3.4). Number of trainable parameters is included as well to illustrate difference in model sizes.

A.2.2 Qualitative results

A comparison figure between the maePFNET (ESR-based) and the U-net based generator is shown in Fig. A.4. The U-net model produces much worse results.

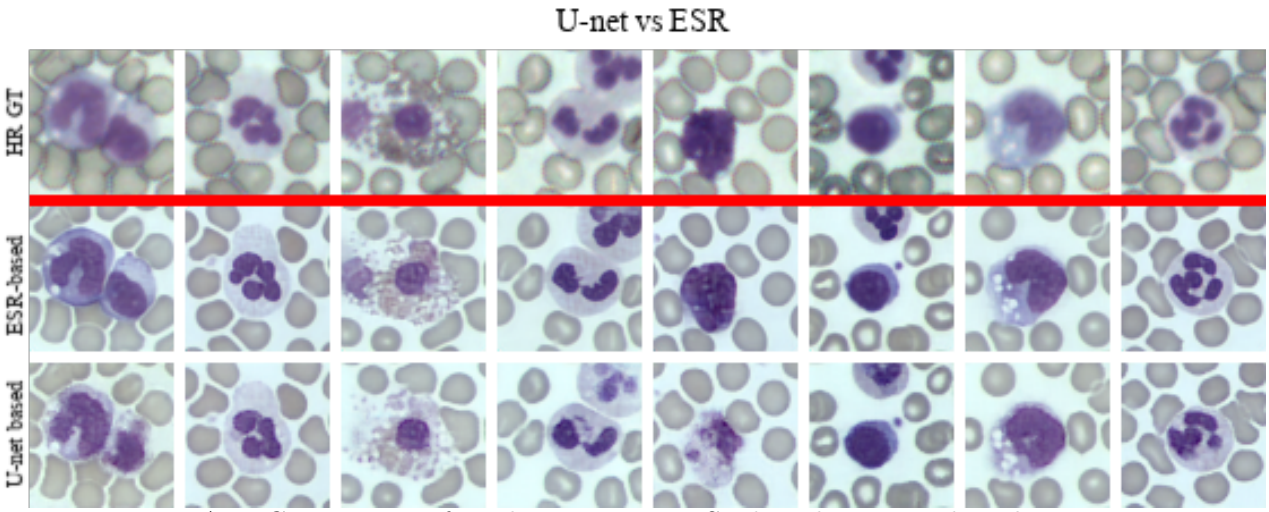
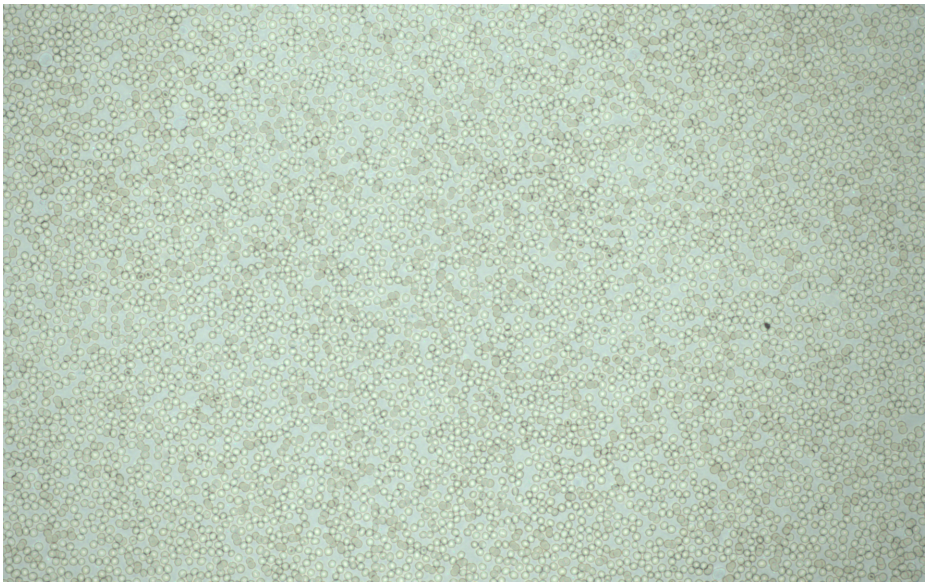


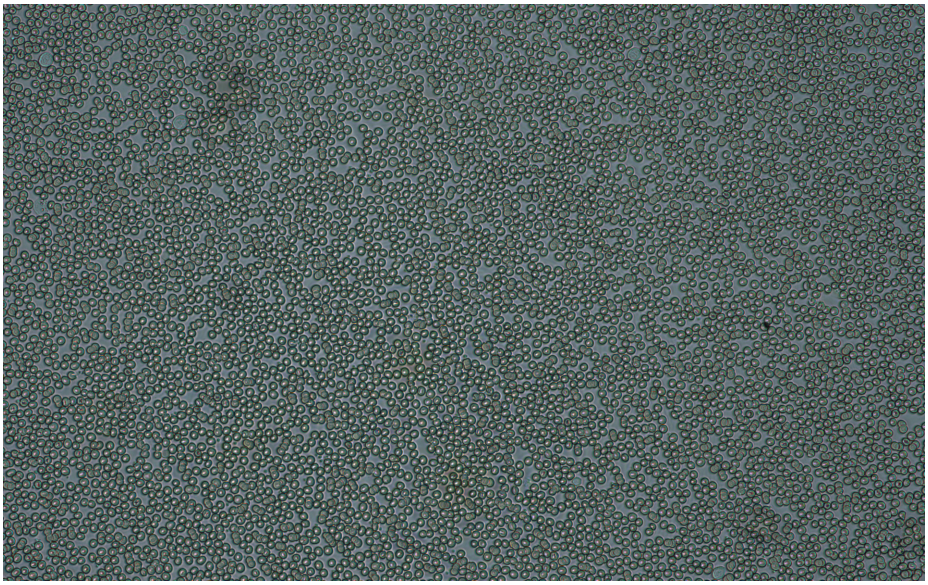
Figure A.4: Comparison of predictions using ESR-based or U-net based generator.

A.3 PLS example images

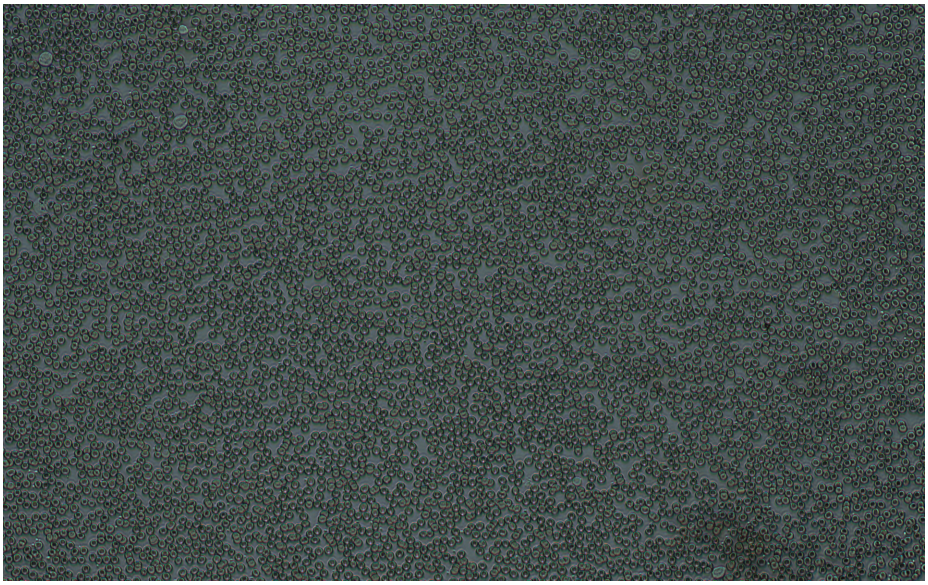
A.3.1 20x/0.4 NA images with different light angles



(a) Brightfield: All LEDs below 24°.

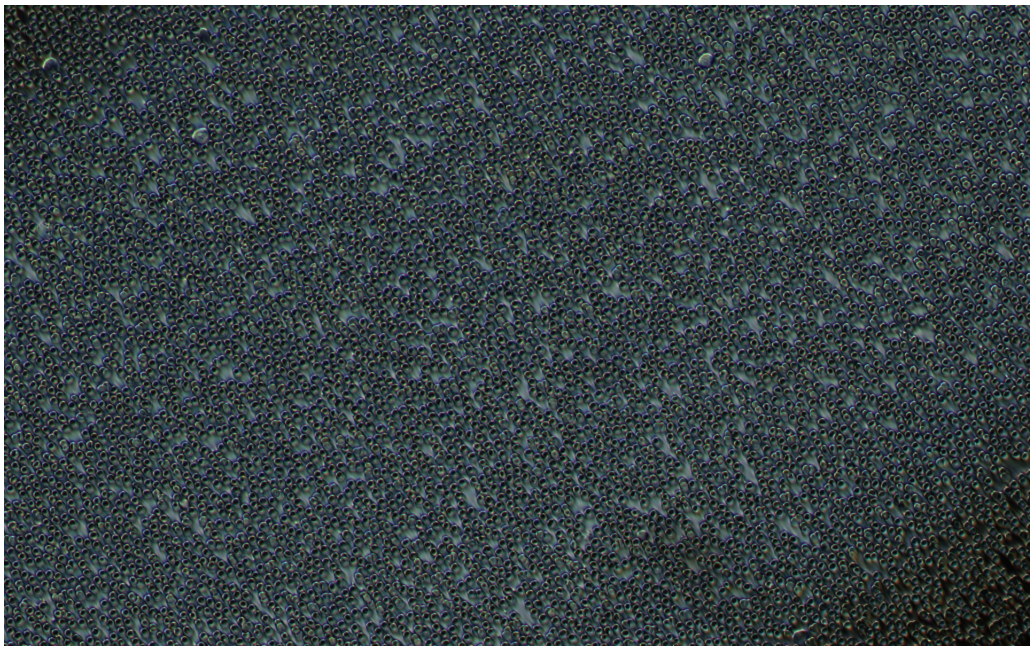


(b) Single LED at 8°angle.

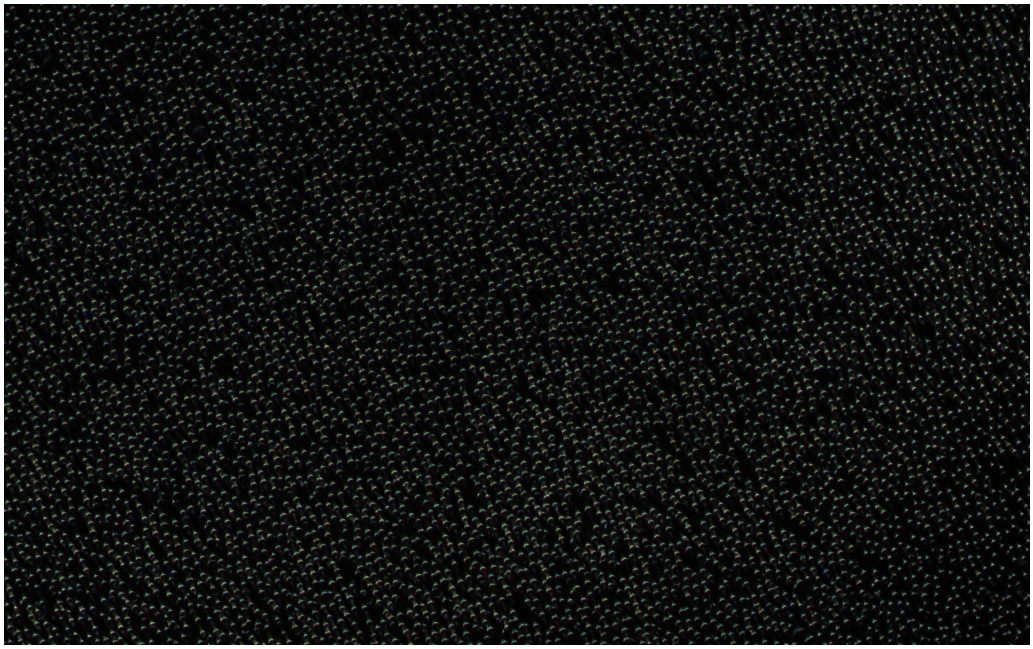


(c) Single LED at 10°angle

Figure A.5



(a) Single LED at 24°angle. Edge of dark field.



(b) Single LED at >24°. Dark Field.

Figure A.6

A.3.2 Full collected input stack

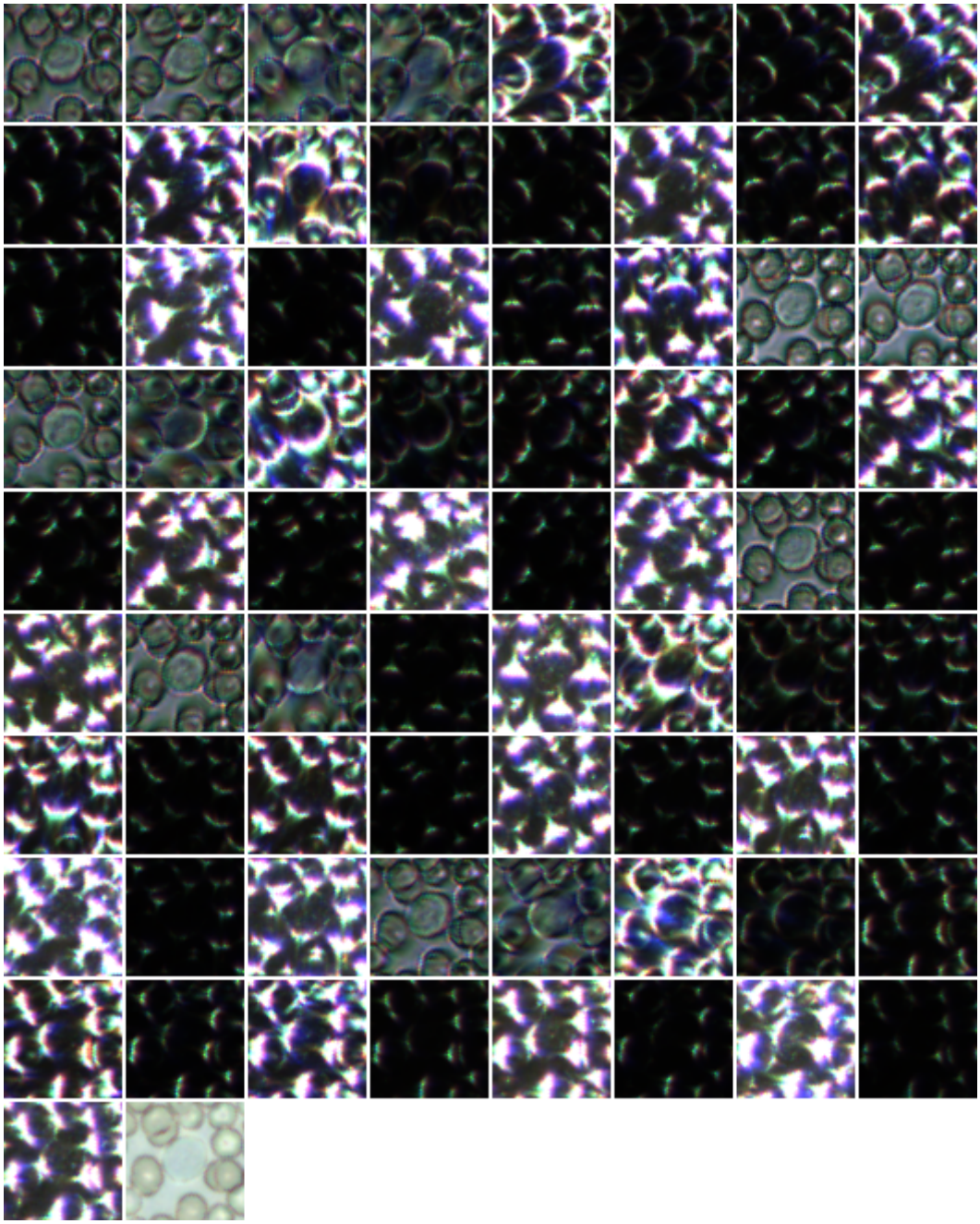


Figure A.7: Full input stack of 73 PLS images and one brightfield image of unstained white blood cell.