JPEG-deblocking of Blood Cell Images using Deep Learning

Julia Hafsbrandt Fovaeus
Justin Ma

Master's thesis
2020:E77

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Abstract

This thesis investigates the use of convolutional neural networks as a reconstruction or *JPEG-deblocking* model for JPEG-compressed blood cell images, needed due to the well known block artifacts caused by JPEG-compression. CellaVision develops automated microscopy for blood analysis that detects and classifies blood cells from images. The automated analysis is carried out on high resolution microscope images, before the images are compressed to JPEG-75 format, where 75 is the quality factor. We investigate how hard the blood cell images can be compressed to still enable acceptable reconstruction quality for display to the user.

We propose a CNN-model that reconstructs blood cell JPEG-images of quality factor 50 and higher, to PSNR and SSIM values on average higher than JPEG-75 blood cell images. Using our method, 99.9% of the blood cell images are improved in terms of PSNR and SSIM. However, these metrics do not take into account opinions of professional laboratory technicians who are the main users of CellaVision's application. A comparison is made between a model predicting in the RGB colorspace and the YCbCr colorspace, the later being exploited by the JPEG-compression algorithm.

Results show that models trained on input images of higher or random quality outperform models trained on lower quality, even in the reconstruction of low quality images. Predicting from a higher quality factor is safer when considering quality criteria for medical images and their use in diagnosing, where image quality is critical. With our thesis we propose that CellaVision could store the images with JPEG-50 and still achieve a reconstructed image with a quality as good as JPEG-75, based on the SSIM and PSNR metrics, and thereby save 31% of storage space compared to today.

Acknowledgements

This thesis would not have been possible without the help from our colleagues at CellaVision, supervisor at the Department of Mathematics, and friends and family. Firstly, we would like to thank CellaVision and its people, who all welcomed us to their friendly community and amazing work environment, and made us a part of the company on the first day of work. We have enjoyed spending our time here and will especially remember the regular workouts and all the interesting discussions during lunch breaks.

A special thanks to our supervisors at the company, Martin Almers and Jesper Jönsson, who provided us with ideas, help and support. By being engaged very early on in our work and leading us in the right direction. We would not have gotten as far without your help. A big thank you to the master's thesis students, Hugo Sellerberg and Nellie Carleke, for helping us in the beginning with the setup, which lead us to avoid installation issues and such, which would have inevitably slowed us down. Also, thank you for the valuable discussions and for providing an enjoyable work environment.

Thank you, Mikael Nilsson, our supervisor from the Department of Mathematics, who provided valuable knowledge and great feedback during our virtual meetings. Your many years of experience in the image analysis field helped us do the right type of analysis for our results and ask the right questions to drive the work forward.

Lastly, we would like to give a big thank you to all our friends and families who supported us during this thesis. Special thanks to Kajsa Borg, who took the time to customize a color scheme for our figures and plots to be based on, and to Magnus, Carl and Alva Fovaeus, by proofreading our report and providing meaningful changes, bringing a higher academic level to our report.

Contents

Abbreviations

- **Adam**: Adaptive Moment Estimation
- **ANN**: Artificial Neural Network
- **CNN**: Convolutional Neural Network
- **MAE**: Mean Absolute Error
- **MLP**: Multilayered Perceptron
- **MSE**: Mean Squared Error
- **PSNR**: Peak Signal to Noise Ratio
- **ReLU**: Rectified Linear Unit
- **RGB**: Red, Green and Blue
- **SSIM**: Structural Similarity Index
- **YUV/YCbCr**: Luminance, Chroma Blue and Chroma Red

1 Introduction

1.1 Motivation

The amount of data being stored and transmitted daily have never been as great as it is today and long gone is the time when people got amazed by information being in the size of gigabytes. Data has become the new oil and is a source of competitive advantage that can fuel powerful machine learning algorithms with applications in various fields, such as business intelligence, educational technology and medical technology, and more. However, an emerging problem seems to be that our ability to produce data is greater than our ability to store it, and this seems consistent in all industries that work with data.

Our thesis was carried out at CellaVision, a medical technology company that produces systems for automation of blood analysis in hematology laboratories. Blood analysis is conventionally done manually by educated personnel finding, classifying and counting blood cells using a microscope. The distribution of different types of blood cells can be used as an indication of illness or level of infection. With CellaVision's system, the detection and classification of blood cells is done automatically with a digital camera microscope. After classification, the blood cell images are displayed on a screen to the laboratory technician for inspection. For each blood sample, hundreds of blood cell images are captured by the camera in CellaVision's machine, making storage saving an important aspect.

CellaVision uses JPEG-compression with a quality factor of 75% to compress cell images for storing [1]. While there is a loss in image quality in the form of compression artifacts, there is a gain in valuable storage space, a worthy trade off [2]. With a quality factor of 75%, the average disk space saved is roughly 97%. CellaVision wanted to investigate a method to reconstruct the compressed images using supervised deep learning methods. Moreover, if reconstruction of compressed images of quality factor 75% shows to be possible, it raises the question of how low quality factor one can use and still be able to reconstruct the images successfully, using a deep learning model. The goal is to achieve a reconstruction quality comparable to that of JPEG-75, that is used today. For a quality factor $Q < 75$, if it is possible to reconstruct a JPEG-Q image to a quality of JPEG-75, CellaVision could store all images in this lower quality Q instead, and then use the network to predict the restored image for display on the screen. In Table 1 the average file size for cell images of different quality factors and the disk space saved relative to JPEG-75 are compared.

Table 1: Comparing average file size for cell images compressed with JPEG varying the quality factor. The disk space saved is relative JPEG-75 which is the standard used today by CellaVision.

JPEG-quality	75	60	50	40	25
Mean size (kb)	10.42	8.04	7.19	6.43	5.28
Disk space saved (%)	0	22.8	31	38.2	49.3

All detection and classification in CellaVision's machines is carried out on the uncompressed image captured by the microscope, before it is compressed and stored on disk. However, the images displayed to the biomedical analyst are the color-normalized and JPEG-compressed images. When performing the diagnostic procedure, the analyst uses both distribution of different types of blood cells and visual inspection of the images [3]. Therefore, if they are to be reconstructed and displayed, some ethical considerations have to be evaluated. For instance, the structure of the cell nucleus, and the granules and vacuoles are all important aspects to look at during diagnostic [4]. Therefore, it is of great importance that these blood cell features are correctly reconstructed.

A deep learning model will predict pixel values to fill in the details discarded by the JPEG-compression algorithm. The lower the quality factor, the more details discarded and the more compression artifacts introduced, and consequently the more information has to be *made up* by the model. In Figure 1, a cell image is depicted comparing three levels of compression and the original image. In order to facilitate spotting the artifacts, which turn up as 8×8 blocks, the image is first transformed to grayscale and its contrast is adjusted using histogram equalization. It is clear that the first redundant information the JPEG-algorithm discards is the background. At a quality factor of 25%, the block artifacts are clearly visible in the cells themselves and a model would have to do a really good work reconstructing all the details needed for diagnosing. Details and structure of the cells can be of great importance here.

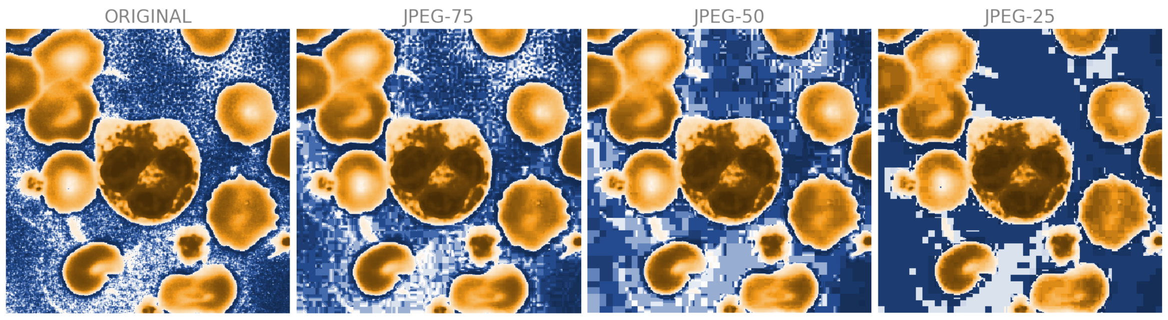


Figure 1: An example image of a blood cell, here in grayscale, visualized with a diverging color scheme. The image is displayed in three different JPEG-compression quality levels compared to its original. The image has undergone histogram equalization in order to better visualize the JPEG 8×8 block artifacts.

In a recent article from May 2020, Stanford researchers investigated the instability of reconstruction of medical images when introducing tiny perturbations [5]. Medical images are extensively used for diagnosing, so a deep learning model used for reconstruction needs to be able to distinguish perturbations introduced by the corrupting process - in our case the JPEG-algorithm, but could also be patient movement or noise - and small anomalies that can be important for diagnosing. In the article, the authors conclude that important details risk to be washed out by deep learning reconstruction models [5]. It is therefore crucial that the reconstructed images fulfill a quality standard based on human visual inspection of experts. The limit for CellaVision's present standard was set internally by engineers and consulting expert users.

1.2 Aim

There are two main objectives of this thesis and they are presented below.

- To which extent is it possible to reconstruct a JPEG-compressed blood cell image using a neural network?
- What is the lowest quality of a blood cell image for which the neural network is able to reach a reconstruction quality level comparable to the quality of JPEG-75?

2 Theory

"*Everything has been said before, but since nobody listens we have to go back and begin all over again - André Gide*"

2.1 Digital Image

A digital image can be encoded in many different ways, but what is common for all standard methods is that the image is represented by a rectangular array of picture elements, called *pixels*, arranged in m rows, n columns and c channels. The expression $m \times n$ is called the *resolution* of the image [2]. A *channel*, in the context of digital images, is an $m \times n \times 1$ array, representing the intensity of some image property such as color, brightness, or saturation to name a few.

The most common colored digital image is the RGB-image, which consists of three channels as can be seen in Figure 2, red, green and blue, while a grayscale image has just one channel - brightness. An example of an image with four channels is the CMYK-image, with the channels cyan, magenta, yellow and key (black) color.

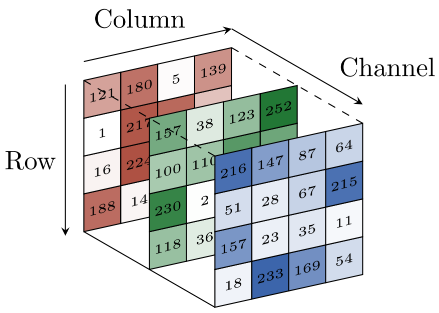


Figure 2: An illustration of an RGB-image with dimensions $4 \times 4 \times 3$. The channels represent red, green and blue. The set of three integer values in each channel axis is referred to as a pixel.

A pixel is essentially a set of numbers, representing the intensities of each channel for one image element. Each channel value is usually an integer in the range from zero to 255, for which zero represents zero intensity (black) and 255 represents full intensity, which for grayscale images means white and for RGB-images means red for the red channel. Everything in between represents a shade of the channel color. The range is not an arbitrary interval, since the number of integers in this range is actually a power of 2 ($256 = 2^8$). This is relevant, because computers are built to store data in zeros and ones, and therefore it is common to use bits as units for data [2].

A *bit* is the atomic unit for computer storage and can be either 1 or 0, a *byte* is a group of 8 bits. A byte can then represent a number in the range zero to 255, since it provides 256 (2^8) different combinations of zeros and ones [2]. In an RGB-image, consisting of three channels, each pixel can be represented as a 24-bit number (or 3 bytes), 8 bits for every channel. An RGB-image of a resolution of 360×360 can therefore at most occupy 388,800 bytes, or roughly 389 kB. If the resolution is doubled to 720×720, the RGB-image requires 1,555,200 bytes, which is four times as much the previous example. At this point, one may realize the need for compression methods when dealing with high resolution digital images.

2.2 Image Compression

Image compression is any type of data compression applied to digital images with the purpose of reducing their cost for storage and transmission. There are generally two types of compression algorithms: *lossless* and *lossy* compression. Lossless compression retains all the information of the original raw image, just encoding it smarter, making it possible to fully recreate the original image from the compressed image. Lossy compression, however, removes some information of the original image, making perfect recreation impossible. One way of looking at this is that lossless compression minimizes *redundancy* in the image, while lossy minimizes *irrelevancy*, in terms of what is relevant for the human eye to perceive [2].

JPEG

JPEG, an acronym for Joint Photographic Experts Group, is a widely used compression method for digital images on the internet [1]. The compression method can be both lossy and lossless, but the method most commonly used is the lossy, meaning that the smaller file size compromises the image quality. However, the quality loss is often not noticeable. This is because JPEG-compression retains the information that matters the most with respect to the sensitivity of the human visual system [2].

The human eye is more susceptible to contrast, sharp edges and, in general, changes in brightness rather than changes in color. The JPEG-compression makes use of this principle and primarily reduces the color information, while retaining most of the brightness information. This allows the image to be represented using fewer bits without losing much of its perceptual properties. JPEG works very well on continuous-tone images, for instance real life photographs, and is a flexible compression method because of its adjustable parameters, such as choosing to downsample a channel or not, and the quality factor. This allows the user to modify the compression ratio and achieve a desired compression and quality trade-off [2].

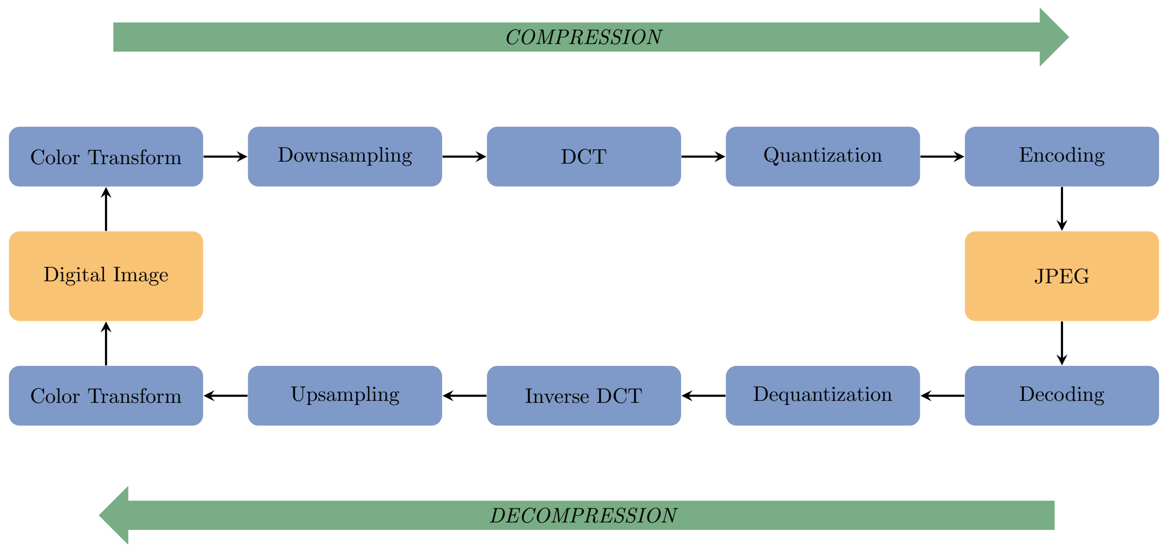


Figure 3: A flowchart of JPEG-compression.

There are several steps in JPEG-compression, as can be seen in Figure 3. These steps will not be discussed in detail, but a short explanation will be presented below [2]:

1. **Color Transform** - Color images, such as RGB- or CMYK-images, are transformed into YCbCr color space, which is a luminance/chrominance space, Y being a channel for brightness (luminance) and Cb and Cr are color channels for blue-difference and red-difference (chrominance). *This step is skipped for grayscale images.*

2. **Downsampling** - The chrominance channels are then downsampled by creating low-resolution pixels from the original ones. The luminance channel is not downsampled. For instance, if we started with a 1000×1000 color image, after this step we have 1000×1000 luminance pixels and 500×500 chrominance pixels, meaning each chrominance pixel coves the same area as a 2×2 block of luminance pixel. Information is lost here. *This step is skipped for grayscale images and is optional for color images.*

3. **Discrete Cosine Transform (DCT)** - The image is then divided in 8×8 blocks, some sort of padding is used if the resolution is not divisible by 8. The DCT is applied to each group of

8×8 blocks to create an 8×8 map of frequency components. Due to numerical preciseness, some information is lost here.

4. **Quantization** - Each of the 64 frequency components is divided by a separate number called its *quantization coefficient* (QC), and then rounded to an integer [2]. The higher the QC, the more information is lost and typically the high frequency components have larger QCs. This is where most of the information is lost. The quality setting in most JPEG-implementations controls the QC table.

5. **Encoding** - Some sort of arithmetic encoding is used to compress the data into a bitstream, commonly Huffman encoding [2].

6. ***DECOMPRESSION*** - The steps are then reversed in the same order to produce a JPEG-compressed digital image. Thus, JPEG is a symmetric compression method.

2.3 Image Reconstruction

The image reconstruction problem is to restore an original image $\mathbf{y}$ from a corrupted image $\mathbf{x}$. Corruptions that occurs due to image compression or involuntarily introduction of noise during caption. Image reconstruction belongs to the family of inverse problems, for which we want to find the mapping f in $\mathbf{y} = f(\mathbf{x})$. However, as the majority of interesting inverse problems, it is ill-posed. For a problem to be well-posed, the following requirements should be fulfilled; the solution exists, the solution is unique, and the solution depends continuously on the data [6]. For image reconstruction, as well as many other problems, the last condition is often violated. So, in order to find solutions to such a problem, it is often casted to an optimization problem on the form

$$\min_{\theta} \quad \mathcal{L}\left(\mathbf{y}, f(\mathbf{x}, \theta)\right), \tag{1}$$

where f is the mapping, θ are some parameters of this mapping and $\mathcal{L}$ is some error measure between restored and original image.

Historically, image reconstruction has been approached with inverse filter design and statistical noise estimation, for example using Gaussian Markov Random Fields [7]. However, in these days it is common to turn to machine learning techniques.

2.4 Machine Learning

The field of machine learning has expanded greatly during the 21th century due to growing interest generated from large projects such as GoogleBrain and OpenAI. The timing is no coincidence, and can be explained by the amount of data and the computational powers from GPUs that are available today. Initially, the research in artificial intelligence (AI) and machine learning (ML) were closely related, but during the late 1970s and early 1980s, AI research focused on using logical, knowledge-based approaches rather than learning algorithms. During that time, neural network research was abandoned by AI researchers, and the two fields started to diverge [8], but during the late 80s the field saw a return of neural network research because of its promising developments, mainly the backpropagation algorithms. Today, machine learning is considered a subfield of artificial intelligence.

Machine learning is the study of computer algorithms that improve through experience. The goal of any machine learning algorithm is to solve tasks, which have traditionally been hard for computers to solve, such as face detection, email spam filtering, product recommendations, fraud detection and so on. Mitchell provides a widely quoted definition of the algorithms studied in machine learning: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E"* [9]. Of course, many more tasks than stated above are possible, which makes the field of machine learning so valuable.

Supervised vs Unsupervised learning

In general, the algorithms of machine learning can be divided into two main categories, *supervised* and *unsupervised* learning. What makes them different is how they make use of the experience, which we from now on will refer to as the data.

In unsupervised learning, the training data consists of input $\mathbf{x}$ without any corresponding target $\mathbf{y}$. The goal of any unsupervised learning model may be to discover patterns, anomalies or estimate probability distributions of data. In supervised learning, the training data consists of examples of the input vectors along with their corresponding target vectors, also known as ground truth or labels [10]. The goal is to learn a mapping from some inputs $\mathbf{x}$ to some outputs $\mathbf{y}$. If the desired output consists of one or more continuous variables, the task is called regression, otherwise it is called classification.

When training a model in a supervised setting, the goal is to minimize some loss that we can measure. This is referred to as the performance from Mitchells definition. In a sense, all supervised learning models aim to solve the same broad optimization problem as seen in Equation 1, where $\mathbf{x}$ is the input vector and $\mathbf{y}$ is the target vector. As before, θ denotes the model's parameters, which in the context of ML are often referred to as the model's weights $\mathbf{w}$. The model that performs the mapping is denoted f and $\mathcal{L}$ is some loss function. The loss functions are discussed in more detail in Section 2.7.

In this thesis we aim to solve the inverse problem of reconstructing JPEG-compressed cell images with supervised deep learning methods.

Generalization

Supervised or not, ML models learn from experience. However, for a machine learning model to be useful, it has to perform well on unseen data. That is to say, data that is *not* included in its experience during training. We say that the model must be able to *generalize* [11].

To be able to measure a models ability to generalize, it is common to split the available data into three subsets; *training, validation* and *test* set. The *training* set is used to fit the parameters of the model, while the *validation* set is used to evaluate how the model performs on unseen data during training. How well the model performs on the validation set is what lets us decide on our hyperparameters and model structure. However, when done with training, the *test* set provides an unbiased evaluation of the final model.

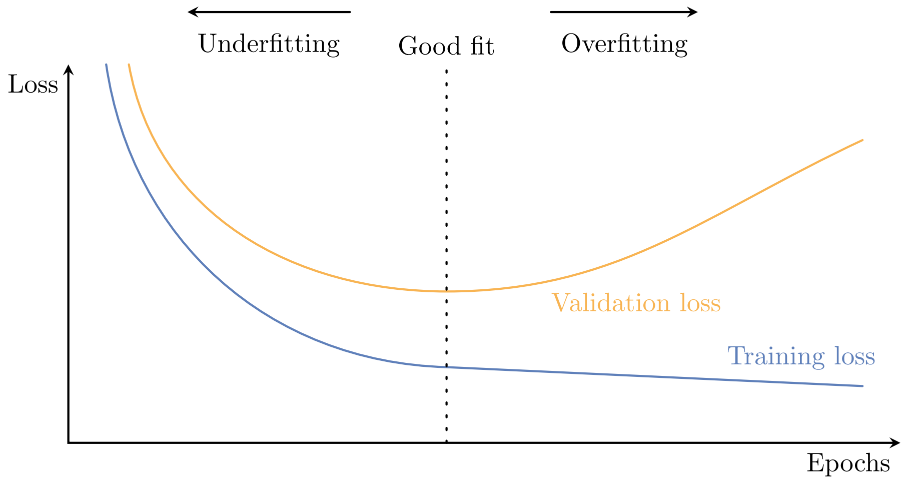


Figure 4: Illustration of a typical learning process. The training loss is able to decrease with increasing epochs, but at some point the validation loss is starting to increase, meaning that the model's ability to generalize decreases.

We can measure an ML algorithm's ability to generalize by comparing the training and validation loss. The training loss should be small, and the validation loss should be close to the training loss [11], as shown in Figure 4.

Underfitting occurs when the model is not able to obtain a low loss on the training set and overfitting occurs when the gap between the training error and the validation error is too large, as can be seen in Figure 4. This is illustrated further in Figure 5, where we can see how a under-, good- and overfitted model performs on some data points. One might think that the overfitted model provides the best fit to the data points, but in reality it would not perform well on unseen data.

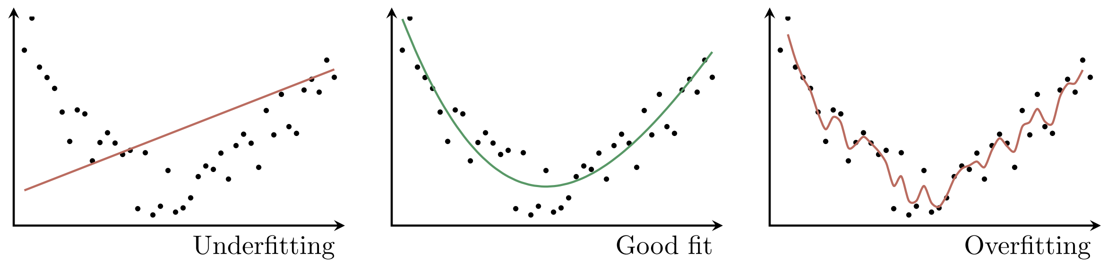


Figure 5: Illustration of the concepts of underfitting and overfitting. The black dots are data points to be fitted by a model in training, while the red and green lines are examples of a models attempt to fit.

2.5 Deep Learning

Inspired by the human brain and its complex network of connected neurons, Warren McCulloch and Walter Pitts, proposed the first model of artificial neurons for simple binary classification in their 1943 paper [8]. Initially, this simple model only considered binary inputs and outputs and had some restrictions on the possible weights, which needed to be set manually. Later on, this model was used in the development of the *perceptron* by Frank Rosenblatt in 1958 [8]. The perceptron became the first model that could learn the weights by itself. This was the beginning of what we at present time refer to as deep learning (DL), which has become a promising subfield of machine learning.

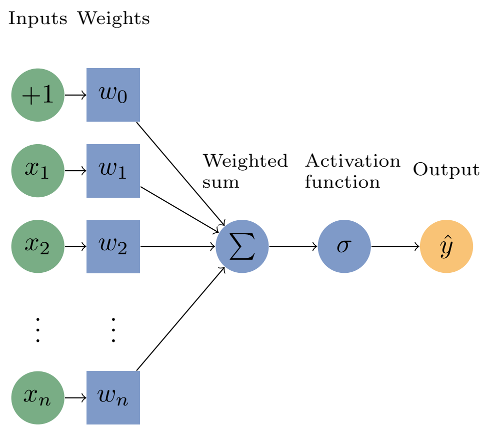


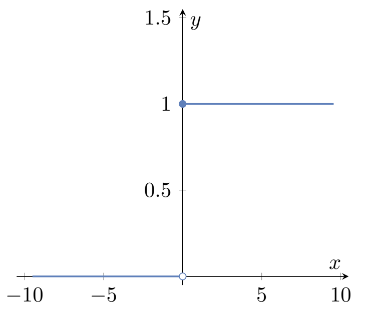
Figure 6: Illustration of the simple perceptron.

In Figure 6, the perceptron, a single-layer feedforward neural network is depicted. It consists of four main parts including input values $x_1, x_2, \dots, x_n$, weights $w_1, w_2, \dots, w_n$ and bias w_0, a weighted sum Σ and an activation function σ. The output of the perceptron can be formulated mathematically as

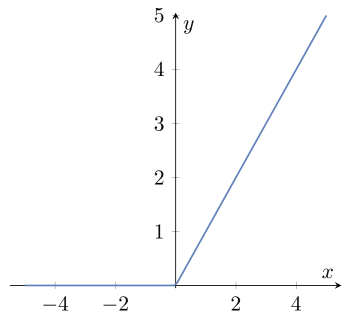
$$\hat{y} = \sigma\left(w_0 + \sum_{i=1}^{n} w_i x_i\right), \tag{2}$$

where the weights w_n and bias w_0 are trainable parameters, and $\hat{y}$ is the output.

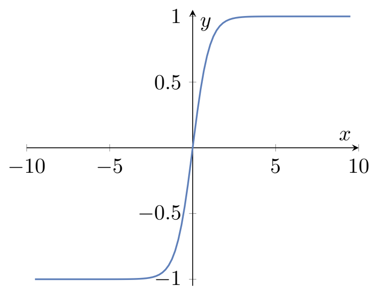
Activation functions



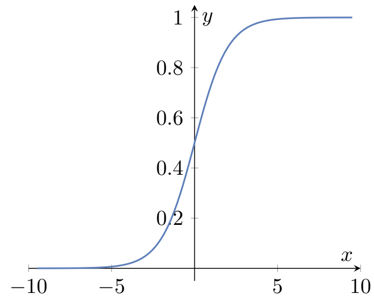
(a) Heaviside step function: $f(x) = H(x)$



(b) ReLU function: $f(x) = \max(x, 0)$



(c) Tanh function: $f(x) = \tanh(x)$



(d) Logistic function: $f(x) = \frac{1}{1+\exp(-x)}$

Figure 7: The most commonly used activation functions.

Some common activation functions are seen in Figure 7, Heaviside step function or also called threshold, rectified linear unit (ReLU), hyperbolic tangent (tanh) and logistic function [12, 13]. The activation functions serves the purpose of deciding how much an artificial neuron should be activated and further to introduce non-lineraity, which enables neural networks to do non-linear mapping.

Artificial Neural Networks

The perceptron is capable of solving simple binary classification problems using the threshold function as an activation function, but it is limited in its use due to its simplicity, and is not suitable for non-linear problems. In comes the multilayer perceptron (MLP), which is a class of feedforward artificial neural networks (ANN). ANN is the collective name for all types of learning models that are based on a collection of connected artificial neurons, similar to that of the perceptron, to learn and solve complex tasks such as classification and regression.

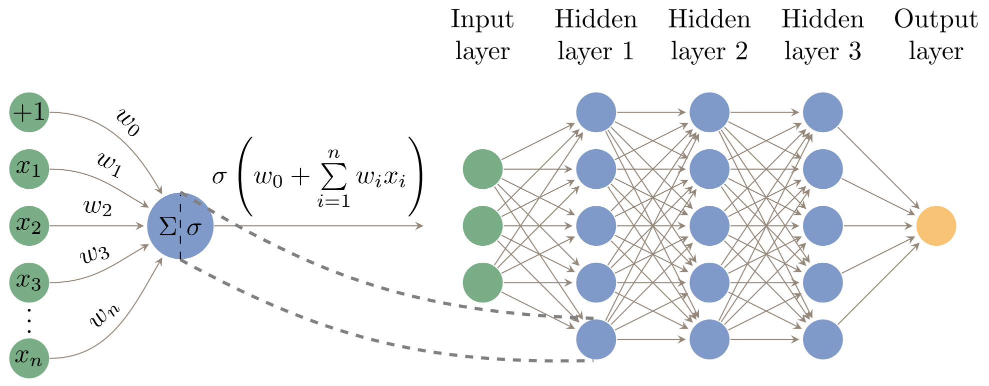


Figure 8: Multilayer perceptron with three hidden layers. Every node in the hidden layer consists of a perceptron. The figure is heavily inspired from [14].

The MLP, seen in Figure 8, consists of several connected perceptrons in an acyclic graph. The models are called feedforward because information flows forward, unlike recurrent neural networks, where the output are fed back as input. The name *deep learning* occurred as a result from the depth of these models. The purpose of the MLP is to approximate some, often non-linear function [11]. The network defines a mapping $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$ and learns the value of the weights $\mathbf{w}$ that result in the best function approximation.

Learning

As mentioned earlier, the goal of training a supervised ML model is to minimize the loss between the model's output and the ground truth with regard to some loss function, as formulated in Equation 1. First, the weights are initialized in some random manner, the network then makes a prediction on an input $\mathbf{x}$ obtaining the output $\hat{y}$. This is referred to as a forward pass. Next, the loss between the prediction and the ground truth is calculated, and finally the weights of the network is updated in a way that reduces the loss, which is referred to as backward pass. This is then repeated for a fix amount of time or until a certain condition is fulfilled, such as the validation loss reaches a minimum, see Figure 4. A backward and forward pass is called an *iteration or step* and an *epoch* refers to when the entire data set is passed [15].

The way a machine learning model updates its weights can be different, depending on the model being used. For a feedforward neural network, an algorithm called *gradient descent* is widely used. It works by taking a small step in the negative gradient direction of the loss function, which is a step to minimize equation 1 with respect to the weights of the network. A simple update rule using the gradient information of the loss function can be formulated as

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{L}}{\partial w_{ij}}, \tag{3}$$

where η is known as the learning rate, the indices i and j is to denote which weight parameter to be differentiated with respect to.

To efficiently derive the weight updates for the early layers in a neural network with a deep stack of hidden layers, the chain rule from calculus is used. This is called *backpropagation* and is the master algorithm behind training deep neural networks. However, in a deep neural network, a problem with vanishing gradients can occur. This might lead to no weight updates in the earlier layers and the network will eventually get stuck in training [15].

Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a kind of neural networks that are specialized in processing data that are sequentially related, such as time-series data or image data, and was first proposed by Yann LeCun in 1989. The name derives from the use of the mathematical operation called convolution, see Equation 4, and CNNs are simply defined as neural networks that use this operation in at least one of their layers [11].

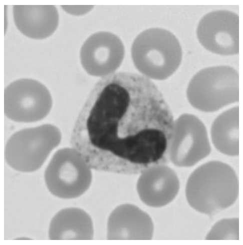
The convolutional operation, often denoted with an asterisk, is a commutative mathematical operation on two functions of a real-valued argument. In convolutional network terminology, the first argument is referred to as the *input.* The second argument is referred to as the *kernel.* The result or output is sometimes referred to as the *feature map* [11].

An intuitive way of thinking about the convolution operation, unrelated to images, is to think of hearing someone speaking through a wall. What you hear is not only the persons voice, but also the echo from the wall, and it can be formulated as a linear combination of the two signals, the voice and its echo. The wall is acting on the voice as a filter. If we call the voice $x(t)$ and the echo $w(t)$, we can express the sound $s(t)$ as the convolution $(x * w)(t)$ according to

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da. \tag{4}$$

2D Convolution

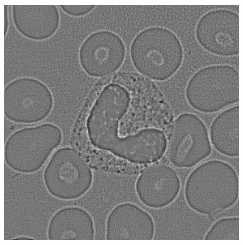
Analogously, a convolution of an image, which is in fact a 2D-signal, and a kernel will produce a filtered image. Again, the kernel is acting on the image like a filter, just like the wall on the person's voice. Kernels that act as filters on images can be used, for example to compute edge detection, image blurring or sharpening. An example can be seen in Figure 9.



(a) Original image

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

(b) Kernel



(c) Resulting image

Figure 9: Example of edge detection with convolution.

Often in a CNN the input is a multidimensional array of data and the kernel will usually be a multidimensional array of parameters that are adapted by the learning algorithm. The multidimensional arrays are referred to as *tensors*. For a two-dimensional image I as input, a two-dimensional kernel K is used, and Equation 4 can be formulated as

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n), \tag{5}$$

where $\mathbf{S}$ is the resulting two-dimensional feature map, i and j denotes which indices of the input image to convolve, and m and n are the rows and columns of the kernel. The convolutional operation in two-dimensions is illustrated in Figure 10, and can be thought of as a sliding kernel that is multiplied with different subsets of the input image I with the same dimensions as the kernel. A typical convolutional layer usually outputs a set of feature maps.

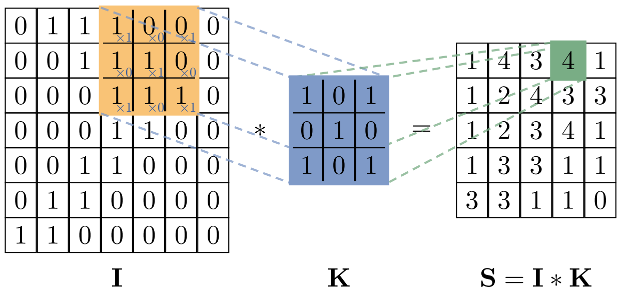


Figure 10: An illustration of a two-dimensional convolutional operation, the kernel matrix is sliding across the input image and computes the sum of the elementwise multiplications, as expressed in Equation 5. The stride of the sliding kernel can be set and does not necessarily have to be one. This figure is heavily inspired from [14].

Pooling, Padding, Batch Normalization, Dropout

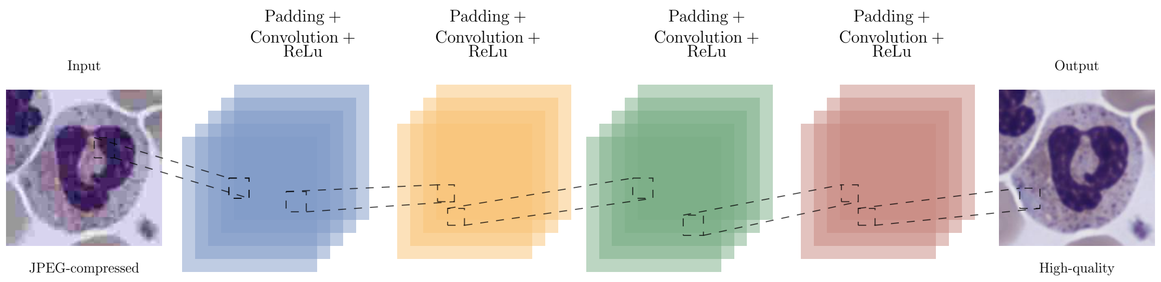

Input
Padding + Convolution + ReLu
Padding + Convolution + ReLu
Padding + Convolution + ReLu
Padding + Convolution + ReLu
Output
JPEG-compressed
High-quality

Figure 11: A very simple CNN with four convolution layers, where each colored square in a layers is a feature map. The input image is a hard JPEG-compressed image and the output is a reconstructed high-quality image.

The main principle of CNNs is the convolution operation, but it is usually combined with other types of layers with the purpose of streamlining, regularizing and for practical reasons. These layers include activation, pooling, padding, batch normalization, dropout and more. A typical activation function for CNN is the ReLU, see Figure 7b, with the purpose of introducing non-linearity. Pooling layers, such as max pooling, is used to downsample the resulting feature map, with the purpose to streamline the process. Paddings is used to make input and output dimension match, as the dimension of the output of a convolution is smaller, see Figure 10. The kernel can see more of the borders when padding is used, which can assist to avoid border problems. There are several kinds of padding like zero-padding and reflection padding. Batch normalization is an empirically proved method that makes CNNs faster and more stable by re-centering and re-scaling the input [16].

Hyperparameters

Apart from loss function and metrics, there are different choices one needs to make when designing a deep learning algorithm, these are usually referred to as hyperparameters. Model parameters are something that a model learns during training, while hyperparameters need to be set. Generally, one can say that model parameters can be updated with gradient descent while hyperparameters can not. Examples of hyperparameters are batch size, learning rate and number of epochs [15].

It is not memory-efficient to calculate the gradient for the whole training set, especially if the training set is large, therefore it is common to use mini-batches of data. A mini-batch consist of randomly drawn samples from the training set, and its size is adjusted with the batch size parameter which is typically between 8 and 128. After forward passing all samples in a batch, the gradient is calculated and an average is used for weight update. Hence, each weight update is made with respect to one batch at the time. This is called stochastic gradient descent (SGD) and there are other optimization methods that expands on SGD, such as Adam [17] which is used in this thesis.

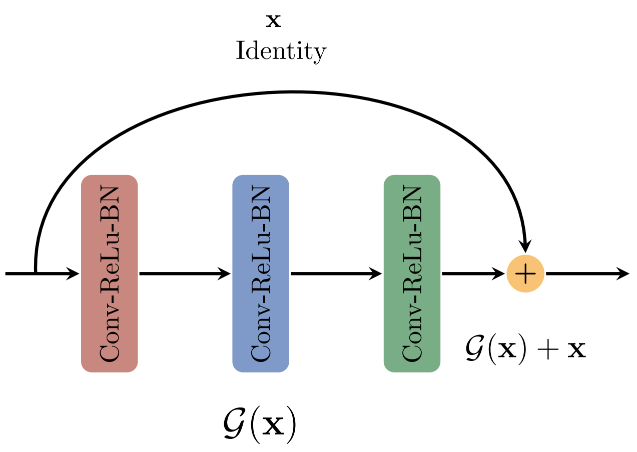
The advantages of using a small batch size are that it requires less memory, and the network will then update more often, because the weights update after each forward pass of the whole batch. On the other hand, the disadvantage is that a small batch size yields a less accurate estimate of the gradient. Setting the learning rate can be tricky, because a too high learning rate can make learning unstable, while a too low learning rate can make the model stuck in a local minimum. To stop the training, one usually provides a pre-determined number of epochs to run, or base the termination of training on some condition, this is called early stopping.

2.6 Network Architectures

There are many recent scientific papers related to our problem that use a variety of CNN structures, so for this thesis a number of different architectures were explored. This section aims to present their key ideas.

ResNet

As mentioned briefly in an earlier section, deep neural networks can be difficult to train due to the vanishing gradient problem. This was usually addressed by normalized initialization and intermediate normalization layers. ResNet was introduced in 2015 by Microsoft researchers to tackle the degradation problem by introducing a *deep residual learning framework* [18]. The idea was that instead of letting a few stacked layers directly fit a desired mapping, letting the layers fit a residual mapping, by introducing a skip connection.



$\mathbf{x}$
Identity
Conv-ReLu-BN
Conv-ReLu-BN
Conv-ReLu-BN
+
$\mathcal{G}(\mathbf{x}) + \mathbf{x}$
$\mathcal{G}(\mathbf{x})$

Figure 12: A residual block, which illustrates the concept of the deep residual learning framework [18].

Let the desired mapping be denoted as $\mathcal{H}(\mathbf{x})$ and the input as $\mathbf{x}$. Then, the stacked non-linear layers will fit the mapping of $\mathcal{G}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$. The original mapping is recasted to $\mathcal{G}(\mathbf{x}) + \mathbf{x}$. The idea is illustrated in Figure 12. The hypothesis was that is is easier to optimize the residual mapping than to optimize the original mapping [18]. The network used in this thesis has 5 residual blocks.

DenseNet

Densely Connected CNN (DenseNet) was first proposed in 2018, and expanded on the idea that CNNs can be more accurate with deeper layers containing skip connections. DenseNets are supposed to alleviate the problem with vanishing-gradient, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters [19]. Here we want to make it clear that DenseNet is also a fully convolutional network, hence the name *dense* does not refer to dense layers in the conventional sense, but to dense connections between convolutional layers.

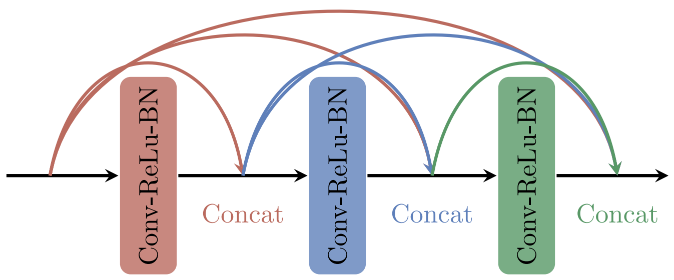


Figure 13: A 3-layer dense block. Each layer takes all preceding feature-maps as input.

The proposed architecture extrapolates on the idea of creating short paths from early layers to later layers, by connecting all layers directly with each other, to ensure maximum information flow between layers in the network. Figure 13 illustrates this idea. In contrast to ResNet, it does not combine features through summation before they are passed into the next layer: instead it combine features by concatenating them [19]. This makes the proposed DenseNet architecture differentiate between information that is added to the network and information and information that is preserved. DenseNet also has fewer parameters than ResNet.

SmallXception

This net is inspired by the Xception network, that was proposed as recently as 2017 by Chollet. Like the Inception Nets, it is built on the idea that channel wise correlation and spatial correlation can be separated. Thus, the net make use of *depth wise separable convolutional layers*, combined in residual blocks like ResNet [20].

U-Net

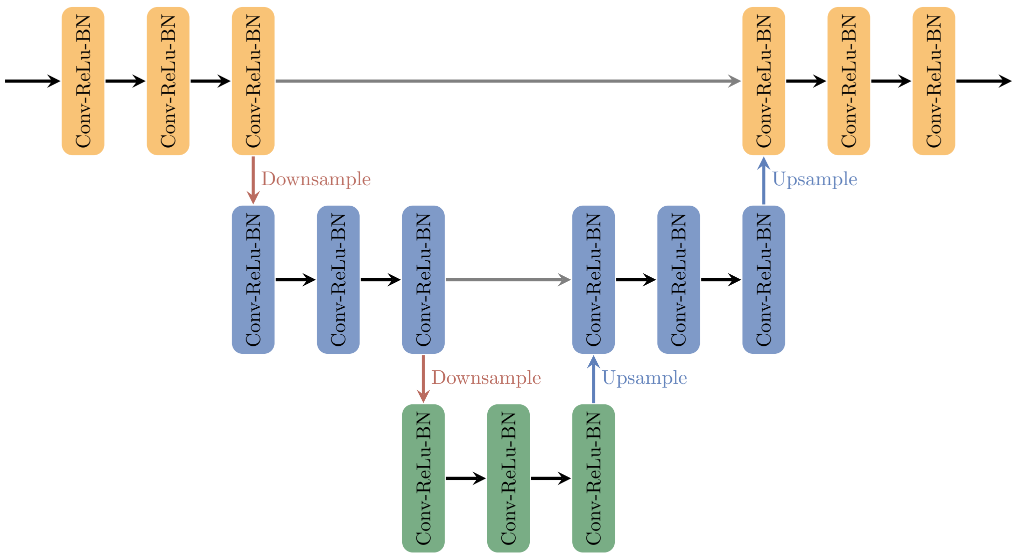


Figure 14: U-Net with two downsamples and upsamples.

The U-Net network architecture was first proposed in 2015 outperformed present state-of-art image segmentation methods at the time of release [21]. The U-Net structure proved to be successful conserving spatial dependence due to its skip connections and particular shape. The U-Net consists of an almost symmetrical contracting path and an expanding path. Down the contracting path, the number of feature maps increases at the cost of downsampling the layer input to lower image resolution using max-pooling layers.

The expanding path upsamples the feature-maps again to the original image size. On the expanding path, high-dimensional feature-maps are combined with low-dimensional features from the contracting path using skip-connections, which are the gray connecting arrows from left to right side in Figure 14. This enables learning of correlations between high resolution features, and low resolution ones as context and location.

DenseUNet

The DenseUNet network combines powerful features of U-Net and DenseNet. Instead of solely convolutional blocks on each level down the contracting path, DenseUNet makes use of dense-blocks as the ones used in DenseNet, on each subsampling level [19]. Denseblocks add internal short-memory skip-connections, additionally to the global skip-connections from the U-Net structure.

MemNet

MemNet is also built on the idea of dense connections, but goes a bit further. The MemNet consists of m so called memory blocks, which each contains k recursive units and a gate unit. Each recursive unit is a two layered convolution residual block. The difference compared to other networks is that is uses dense connections on two scales. Both internally in a memory block between the recursive units, as well as between each memory block. The authors refer to this as short-term and long-term memory connections. Each memory block ends with a so called gate unit, which combines the so-far extracted short-term and long-term features. The idea is similar to DenseUnet, but uses no subsampling [22].

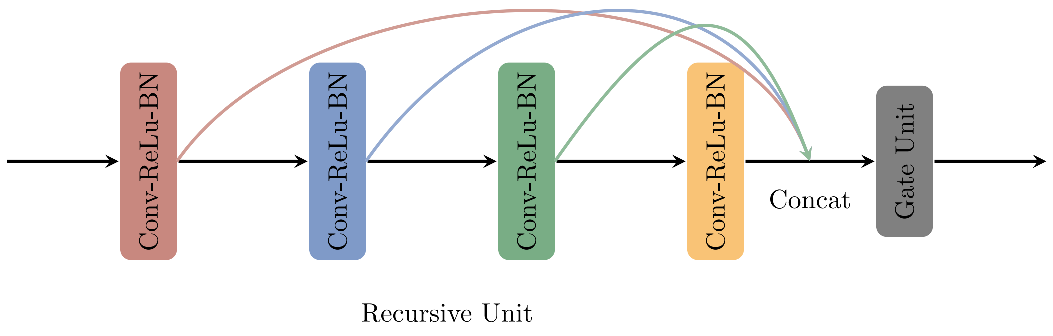


Figure 15: A memory block with four recursive units and a gate unit [22].

JJNet

This net is the same as ResNet but with the variation that the adding layer in the end of each residual block is displaced by a concatenation layer. So after each block, the input is concatenated to the output of three convolutions using a skip connection.

2.7 Loss Functions and Metrics

Choosing a loss function is a crucial step when designing a deep learning model, since the main driving force of improving a model is to minimize the loss. We want the most suitable loss for our model to successfully approximate the correct mapping from corrupted to original image. In this section, we present the loss functions explored in our thesis. Also, some of them were used as metrics for evaluation of results.

First, it is worth to mention again that the images used in this project are RGB color images. Among the conventional metrics for image quality, some are not originally defined for color images, so modifications were made to fit our case. In all definitions, let N be the number of pixels in the image. Then a three-channel RGB image is represented by $3N$ numbers.

Mean absolute error (MAE)

The mean absolute error between two images is defined as the ℓ_1-norm of their residual. So, for an image with N pixels it is

$$\text{MAE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{3N} \sum_{n=1}^{3N} |y_n - \hat{y}_n| \tag{6}$$

where y_n is the value of a pixel element n in the target image $\mathbf{y}$ and $\hat{y}_n$ is the value of a pixel element n in the predicted image $\hat{\mathbf{y}}$.

Mean squared error (MSE)

The mean squared error is defined analogously, as the ℓ_2-norm of the residual

$$\text{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2, \tag{7}$$

where y_n is the value of a pixel element n in the target image $\mathbf{y}$ and $\hat{y}_n$ is the value of a pixel element n in the predicted image $\hat{\mathbf{y}}$.

Peak signal-to-noise ratio (PSNR)

Another classical metric that is related to the MSE is the peak signal-to-noise (PSNR) metric [23]. It is measured in decibels (dB) and is defined as

$$\text{PSNR}(\mathbf{y}, \hat{\mathbf{y}}) = 10\log_{10}\left(\frac{\text{MAX}_I^2}{\text{MSE}(\mathbf{y}, \hat{\mathbf{y}})}\right), \tag{8}$$

where MAX_I is the maximum possible value in the image, hence depends on the range of the pixel values. In our project, all images are normalized and have the dynamic range of $[0, 1]$, so the PSNR simplifies to

$$\text{PSNR}(\mathbf{y}, \hat{\mathbf{y}}) = -10\log_{10}(\text{MSE}(\mathbf{y}, \hat{\mathbf{y}})). \tag{9}$$

High frequency + MAE loss

The idea of a loss function based on high frequency occurrence, is to give more importance to sharp differences in the image. The reason this can be of advantage is that the human visual system is more sensible to high contrasts and sharp edges. The computation requires a few steps. First, the residual image $\mathbf{y} - \hat{\mathbf{y}}$ is computed. The residual image is then convoluted with a Gaussian kernel $K(\hat{\mathbf{y}}, \mathbf{y}, \sigma = 3)$ of size 7×7 to produce the blurred residual image R^b

$$R^b = K(\hat{\mathbf{y}}, \mathbf{y}) * (\mathbf{y} - \hat{\mathbf{y}}). \tag{10}$$

A sharpened residual image can be obtained by subtracting the blurred residual image R_b from the original one, leaving only the pixels that correspond to high frequency areas. Finally, the ℓ_1-loss is computed for the sharpened residual image.

$$h_f = \frac{1}{M}\sum_{n=1}^{M}|(y_m - \hat{y}_m) - R_m^b|. \tag{11}$$

Since the convolution operation changes the dimensions of the residual image, $M < N$ where N is the total number of pixels of the input image.

As a resulting loss function, we weigh together the sharpened loss h_f and the basic ℓ_1 loss according to

$$\mathcal{H}_f = \alpha h_f + (1 - \alpha)\ell_1. \tag{12}$$

In our thesis, we set $\alpha = 0.99$.

Perceptual loss using pre-trained CNN

Instead of measuring the absolute difference between the predicted image and its target image using pixel-wise metrics, the idea is to measure structural differences. For extracting the structure of the two images, we use a famous pre-trained CNN called VGG16 [24, 25]. This CNN is already trained for classification of images, for which it extracts features. Both images are given as inputs to the VGG16 up to some intermediate layer, and the output for each is the set of feature maps corresponding to the output of the layer selected. An example is seen in Figure 16.

The perceptual loss value is then obtained by computing the ℓ_1 loss of the residual between the feature maps extracted from the predicted image and the feature maps extracted from the target image. If the images are identical, they will output the exact same feature maps and the loss will be zero [25].

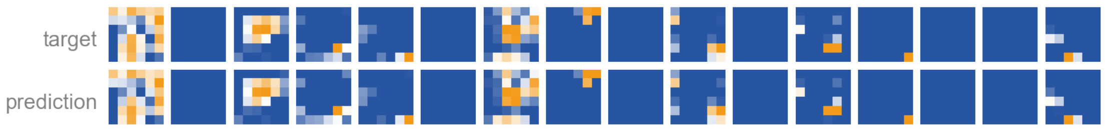


Figure 16: Sample of feature maps from VGG16 last convolutional layer. The top row is from the target image and the bottom row is from a reconstructed image. The two sets of extracted features look similar but are not identical.

In our thesis a weighted sum between the perceptual VGG loss and the MAE loss was used. The perceptual loss is denoted $p(\mathbf{y}, \hat{\mathbf{y}})$ and is calculated on the output from the last convolutional layer with 512 feature maps of size 6×6. The resulting loss function used is

$$P = \beta p(\mathbf{y}, \hat{\mathbf{y}}) + (1 - \beta)\,\mathrm{MAE}(\mathbf{y}, \hat{\mathbf{y}}). \tag{13}$$

In our thesis, we set $\beta = 0.9$.

Structural similarity index measure (SSIM)

The structural similarity metric was first proposed in 2004. Instead of comparing pixel-wise errors like the previously presented loss functions, this metric compares estimated statistical properties of two images. It is presented as a perceptual image quality assessment, that should better capture structural distortion [26]. The SSIM metric is defined as

$$\mathrm{SSIM}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{(2\mu_{\mathbf{y}}\mu_{\hat{\mathbf{y}}} + c_1)(2\sigma_{\mathbf{y}\hat{\mathbf{y}}} + c_2)}{(\mu_{\mathbf{y}}^2 + \mu_{\hat{\mathbf{y}}}^2 + c_1)(\sigma_{\mathbf{y}}^2 + \sigma_{\hat{\mathbf{y}}}^2 + c_2)}, \tag{14}$$

where μ is the estimated expected value, σ^2 is the estimated variance and $\sigma_{\mathbf{y}\hat{\mathbf{y}}}$ is the correlation between the two images. The terms $c_1 = (k_1 L)^2$ and $c_2 = (k_2 L)^2$ are two constants to avoid division with zero and L is the dynamic range of pixel values. Default values proposed by Wang are $k_1 = 0.01$ and $k_3 = 0.03$ [26].

The SSIM-metric is originally defined on gray scale images, so before computation our RGB-images are converted to grayscale.

3 Methodology

3.1 System overview

To implement and train the models used in this thesis, we had access to five stationary computers with graphics cards NVIDIA GeForce GTX 1660 TI, GTX 1650 and GTX 1050 TI. All models were implemented and trained in Tensorflow Keras Deep learning library (v2.0.0). The GPUs used CUDA toolkit 10.0 together with NVIDIA Deep Neural Network library 7.6.5 (cuDNN) and Python 3.7.7.

3.2 Data

The data set consisted of 60,296 color images in BMP-format with the resolution 360 × 360 pixels. Images from three different CellaVision systems were included in the data set, these were DC-1 (white images), DM96 (yellow images) and DM1200 (green images). In Figure 17 some images are shown. The distribution was roughly 50% white/pink images and 50 % green/yellow images.

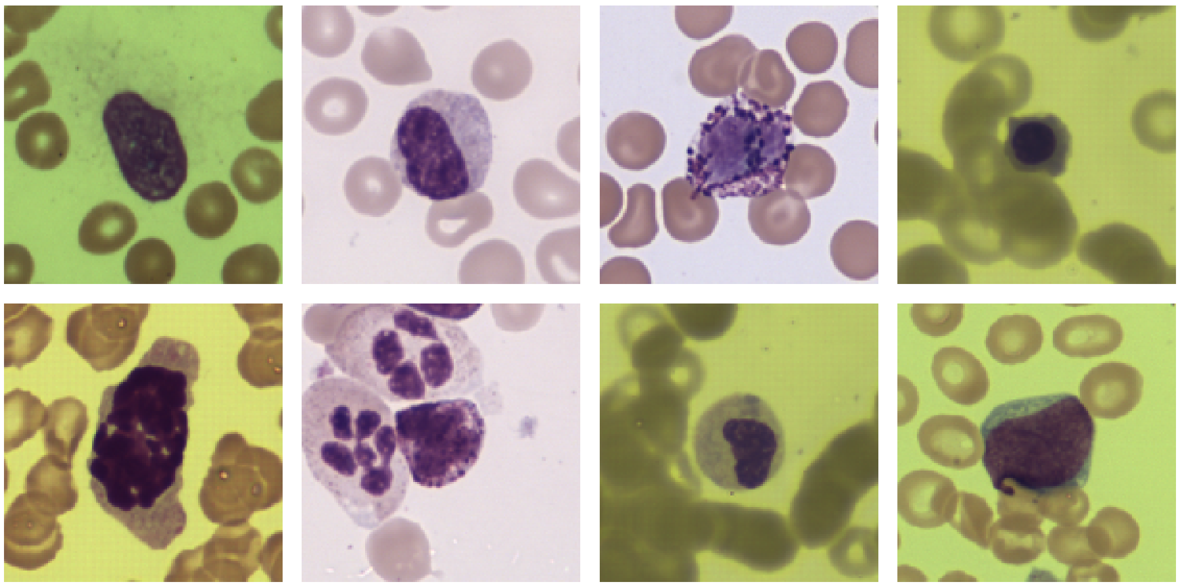


Figure 17: A small sample of our data, images of white blood cells. The variation in color tone depends on the system's optics chain. For the vast majority the cells are centered.

Splitting the data

First, all high resolution BMP-images were randomly divided into training, validation and test set. The validation set was used for model evaluation and hyperparameter tuning. The test set was used at a final stage for an unbiased evaluation, and hence not touched until a final model and hyperparameters were decided upon.

To create two randomized data sets, each with their own subsplits of training, validation and test set, two random seeds were chosen and given as input to Python's built-in pseudo-randomizer. The seeds were chosen arbitrarily and were set to 100 and 101, thus the data sets were referred to as data100 and data101. The reason for having two data sets is that the latter is used to confirm the results on the first data set, that is to say to make sure that the results do not depend on the specific data set. For testing and running certain experiments, a smaller data set Tinydata was created using seed 101. The subset sizes are presented in Table 2 below.

Table 2: The data sets that were used in this thesis. Total number of images were 60,296 and a 60/20/20 split were used for data100 and data101.

data100, 101

Set	Size
Training	36179
Validation	12059
Test	12058

Tinydata

Set	Size
Training	9600
Validation	2399

The BMP-images in these subsets were used as targets in the deep learning models to compare with the output predictions.

Compressing the data

After dividing into subsets, the BMP-images were compressed using Python PIL image library to JPEG-qualities 10, 25, 40, 50, 60, 75, 80 and 90. These were the inputs fed to the deep learning models.

3.3 Preprocessing and Data Flow

All images fed to the models were read batch wise by Tensorflow Keras ImageDataGenerator to tensors of shape [n, 360, 360, 3], where n is the batch size. Before training, the image pixel values were re-scaled and casted from integers in the interval [0, 255] to floats in [0, 1]. All training was carried out on a 192 $\times$ 192 pixel center-crop of the images to increase training speed. Also, as can be seen in Figure 18 which is the center-cropped images from Figure 17, a center-crop of this size captures almost always the most important part of the image - the cell. The images were *not* resized but center-cropped. This, because we wanted to alter the input quality as little as possible. A resizing algorithm would have altered the image quality by downsampling. This was not desirable, because the aim was for the model to learn the difference between JPEG and BMP-quality.

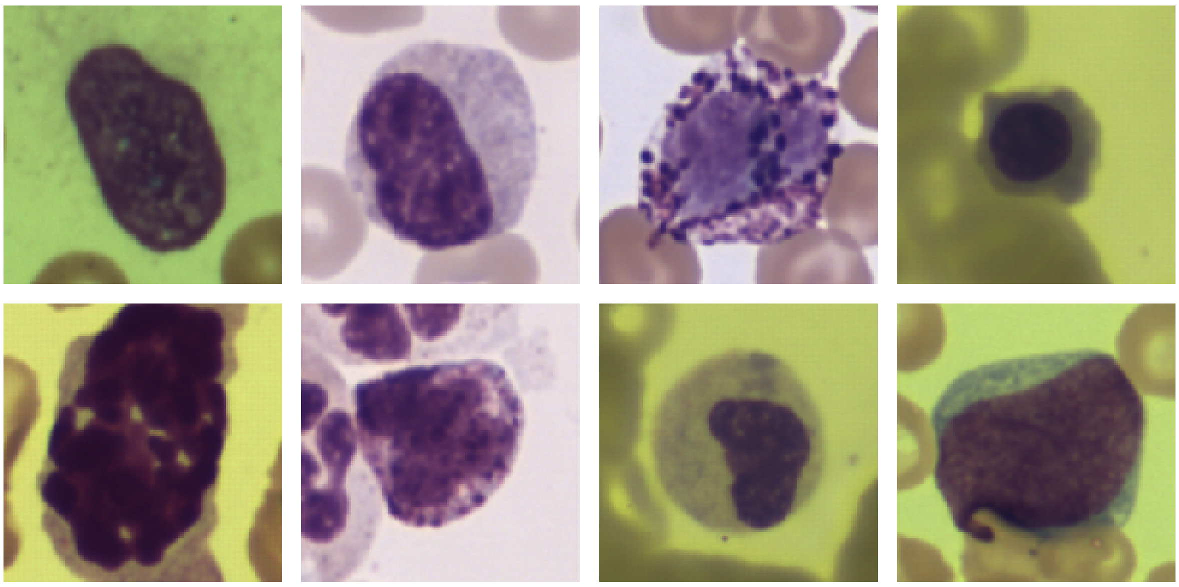


Figure 18: Center-crop of the small sample of our data presented earlier in Figure 17. Most of the uninformative background is cut out so training time is significantly decreased and images are more information dense. The center-crop was done with the code from this github [27].

Data augmentation

The images were all microscopic, and as a consequence the camera angle is consistent throughout the data set; the image is always taken from above. Therefore, there is no point in introducing alternative camera angles in the data augmentation step. Augmentations that are reasonable are flips around the vertical and horizontal axes, together with slight changes of image brightness. Introducing variations that are not probable to exist in a new sample of data will worsen the results. This was concluded empirically, and afterwards similar results were found in an article discussing the same issue for self driving cars [28]. So when augmentations were used, these were: horizontal flip, vertical flip and brightness range 80% - 120%.

3.4 Image Reconstruction

The objective was to implement an image-to-image CNN which takes a JPEG-image of a blood cell as input, and gives its reconstructed image as output, which should be similar to the original, uncompressed BMP image. What we essentially aim to do here, is having the network to predict the *JPEG-noise* introduced by the JPEG-compression algorithm.

This idea is illustrated in Figure 19. The JPEG-image is fed to the network, and in a final addition layer, the input image itself is added to the network's prediction, producing the restored image. This approach follows the hypothesis related to the invention of the residual block; that it is a simpler problem to fit a residual mapping, rather than the mapping itself. In other words, that it is easier to predict the difference rather than to generate the uncompressed image itself.

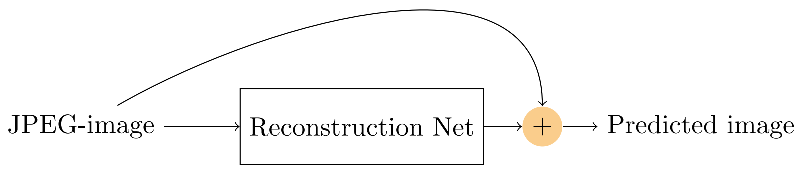


Figure 19: Rough sketch of the network structure used in our experiments. The input image is added to the prediction of the residual, producing the output image.

An example of how an input, prediction of noise and resulting prediction would look like is illustrated in Figure 20.

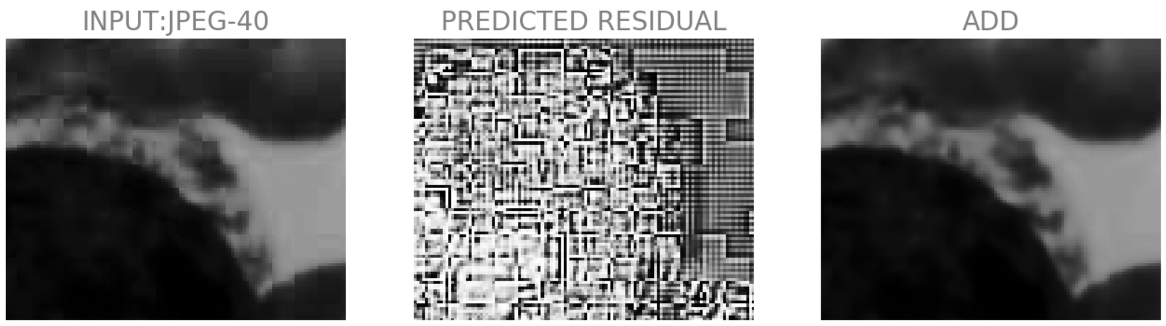


Figure 20: The middle image is a gray scale image of the JPEG-noise predicted by the model, when given as input the left column image. To the right is the resulting image when adding the predicted JPEG-noise to the input image in the last layer of the network.

In order to find a suitable structure, several network architectures were trained with different loss functions and evaluated with the metrics PSNR and SSIM. The best candidate was later trained on the big data set to produce the final results. Different quality factors for input images were considered for training.

Before training, metrics for JPEG-images were calculated and summarized in four plots. These measurements were taken into account for comparison with the results from our models, and worked as our baseline for results that we wanted to beat. The results are displayed using four violin plots, one for each metric represented on the y-axis and different JPEG-qualities on the x-axis. The violins are simply distributions [29], illustrating frequency for that specific JPEG-quality and metric-value. The violin plots of Figure 21 show both mean value and distribution of values. In light blue, metrics for the full image size are visualized and in blue we have corresponding measurements for the center-cropped image used in training. The center-cropped images contain less proportion of the uninformative, and probably easily compressed, background, and in the figure one can note that these images get a lower quality score in all categories.

It was concluded that in order to at least reach the quality of JPEG-75 based on these metrics, the reconstructed images should fill the requirements summarized in Table 3.

Table 3: When evaluating reconstructed images they should on average perform better than or equal to the average values for JPEG-75, seen in the table.

192 × 192		360 × 360	
SSIM	$\geq 98.43\%$	SSIM	$\geq 98.33\%$
PSNR	≥ 39.34 dB	PSNR	≥ 40.37 dB
MAE	$\leq 7.72 \cdot 10^{-3}$	MAE	$\leq 6.89 \cdot 10^{-3}$
MSE	$\leq 1.21 \cdot 10^{-4}$	MSE	$\leq 9.4 \cdot 10^{-5}$

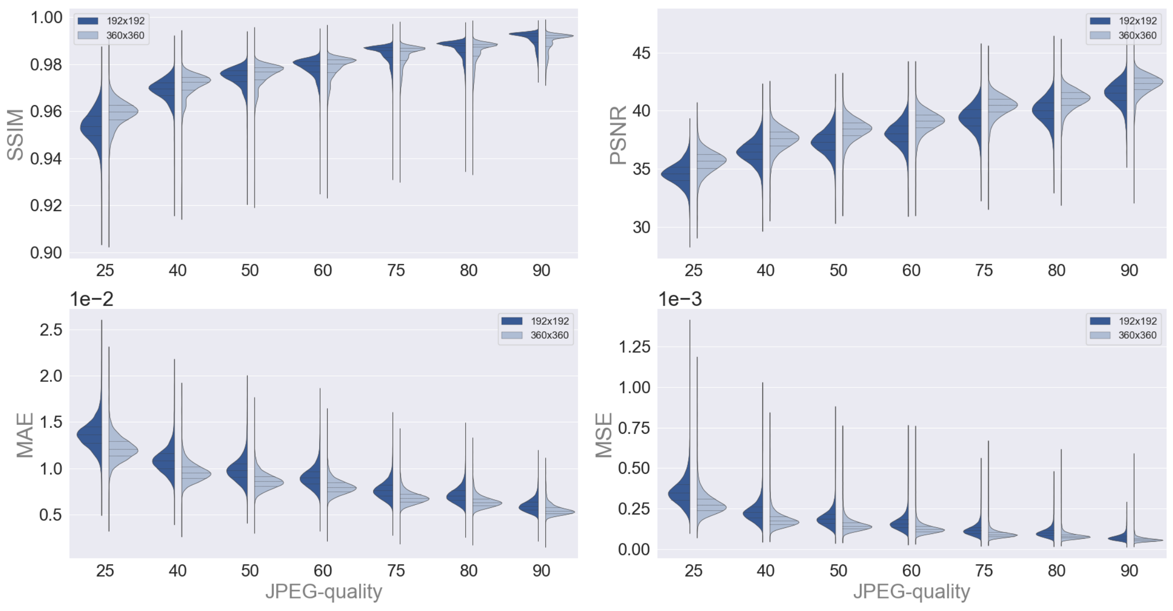


Figure 21: Violin plots of the metrics MAE, MSE, PSNR and SSIM, comparing JPEG-images of size 360 × 360 (light blue) and the training image size 192 × 192 (**blue**) center-cropped images. The metrics are computed with respect to the original images from the data100 validation set for varying quality factor. The variance of all but the PSNR-metric decreases with higher quality factor.

Finding preferred network architecture and loss function

Six different network structures and four loss functions were trained and evaluated on the smaller data set named Tinydata. The network architectures evaluated were U-Net, ResNet, JJNet, DenseUnet, Xception, and MemNet. The numbers of weights and their file size are presented in Table 4. The loss functions evaluated were

1. Mean absolute error
2. Mean squared error
3. $\mathcal{H}_f$: weighted sum of high frequency loss and ℓ_1
4. P: weighted sum of perceptual loss and ℓ_1

Table 4: Network structures trained on TinyData for the reconstruction task

Network	UNet	ResNet	JJNet	DenseUnet	Xception	MemNet
No. Parameters	888,003	114,691	244,771	198,115	11,183	84,403
Filesize (kB)	2131	1513	1729	2696	666	1688

Table 5: Configuration for loss-function experiment

Training set size	9600 images
Validation set size	2399 images
Traning on quality	JPEG-75
Image size	192x192
Batch size	8
Epochs	30

The results of this training are summarized in appendix A.

The DenseUnet was looking promising, so further training was carried out on the tiny dataset with input images of JPEG-50 and JPEG-25 format. However, after all networks were trained, JJNet was chosen as the preferred one and was also trained again on the tiny dataset but varying the last activation function tanh and no activation.

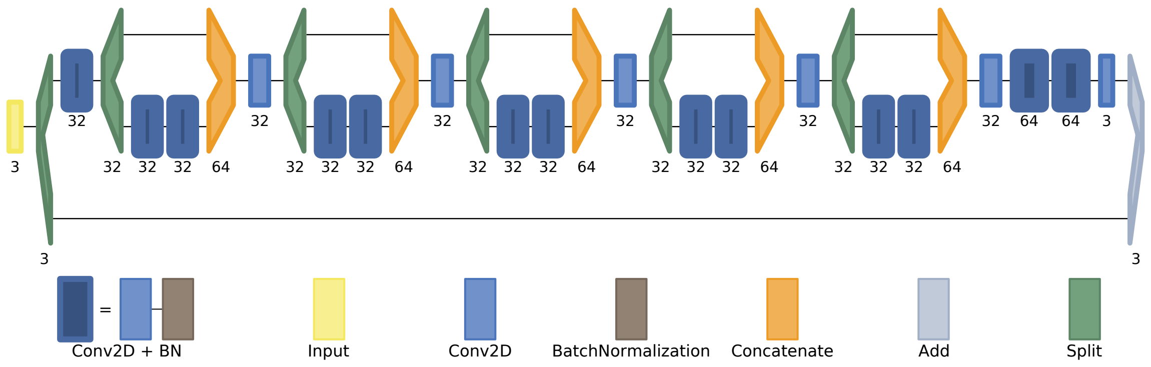


Figure 22: JJNet network structure. It is a basic ResNet structure with five residual blocks, but the adding layer is replaced by a concatenation layer. The blue layer denotes the input image, which is added to the network output in the last layer.

Training on big data set

JJNet with no activation function in its final layer and with the MSE loss function was chosen for successive training on the large dataset. This, because JJnet performed best in terms of PSNR and SSIM when predicting on all qualities. The good results for the lower quality JPEG was an important condition for selecting this model, since the aim was to be able to reconstruct lower quality images to a quality comparable with JPEG-75. The results that motivated further training with JJNet are found in appendix A.

When training on the large dataset, data augmentation techniques described above were applied in order to improve generalization. Training was first carried out separately for qualities 25, 50, and 75. As a last experiment the model was trained on images of uniformly random JPEG-quality, ranging from 25 to 100. The hyperparameters for training the JJNet on the big data set are presented below in Table 6. The Adam optimizer was used with default settings in Tensorflow Keras [30].

Table 6: Hyperparameters when training JJNet on data100.

Training set size	361079
Validation set size	12059
Testing set size	12058
Image size	192x192
Batch size	8
Loss function	MSE

To prevent overfitting, the model was trained until change of validation loss between epochs is negligible, according to a threshold $\delta = 10^{-8}$ and patience 50 using a Tensorflow callback. The patience refers to that the training is stopped if the validation loss not is lowered more than δ after 50 epochs. Thus, the models that reach the lowest validation loss may not train equal amount of epochs across all input quality factors.

Train again on different data set

Finally, the selected model was retrained with the same configuration as in Table 6 for the data set data101. This data set has the same size for all subsets as data100, but contains a different mix of the original 60,296 images. This was done to reassure that the results did not depend on the particular data set.

Model predicting in the YCbCr color space: exploiting JPEG properties.

As a final experiment the ResNet model was modified to predict the JPEG-noise in the YCbCr color space instead of RGB, similar to the approach in a recent work [31]. As said before, the JPEG-compression is carried out in YCbCr space which has one channel for luminosity and two color channels. The majority of information lost belongs to the color channels. Therefore, this network uses most of its computing power to reconstruct the color channels.

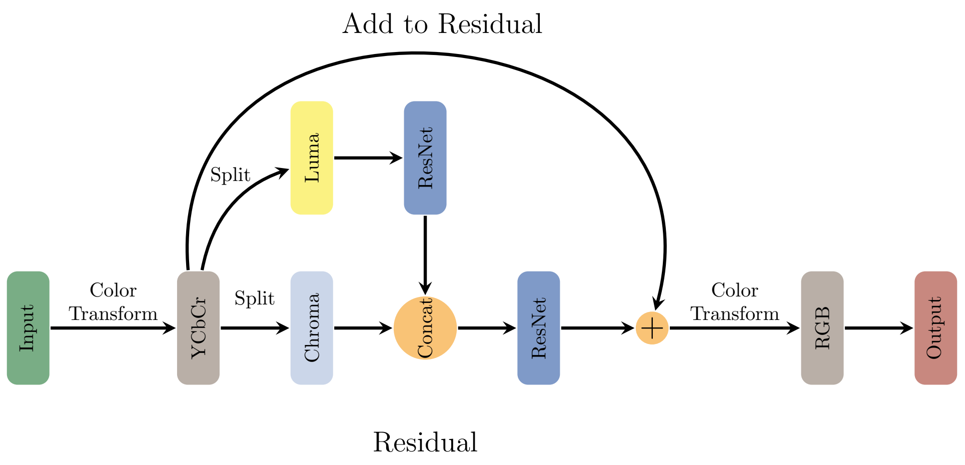


Figure 23: Network structure for the model predicting in YCbCr color space, which we call ResNetYUV. Inspired from Zini's paper [31].

Training was carried out on random JPEG-quality and JPEG-75, with the same hyperparameters and stopping condition as when traning JJNet.

3.5 Evaluation

Loss-minimization and metrics

After using the validation set to decide on a network structure and hyperparameters, the test set was used for a final, unbiased evaluation. Predictions were made by the selected, trained network on the images in the test set. For the resulting reconstructed images, metrics to compare the image quality to the corresponding original images were computed.

Metrics used for comparison were the MAE, MSE, PSNR and SSIM as described in the theory section. Statistics like the mean and standard deviation were considered and compared to the same metrics comparing the JPEG-image quality to the original image. The aim was to beat the image quality of JPEG-75, since this is the default compression level used in CellaVision's application.

Evaluation was made both over the training image size 192×192 pixels and the full image size 360×360. The larger images contain more information about the background, which is not as important as the cells themselves when measuring quality. Therefore this comparison was made to make sure the network had learned about the nucleus and not the background.

Relative reconstruction

The relative reconstruction, how much the metrics increase or decrease after reconstruction was calculated for the models considered. We check if any of the test images were actually worsened. Another investigation of this sort was the inspection of the worse JPEG-images in the test set. The worst images were selected considering their mean squared error to the original image. This check is interesting especially because we can conclude from Figure 21 that the quality assessment scores vary substantially between the JPEG-images even at a quality factor of 75.

Visual inspection

Reconstructed images were inspected and compared to input JPEG-images, as well as to the ground truth BMP-images. To better spot the differences and the JPEG-artifacts, the images went through histogram equalization before display. Also, visualizations of the predicted JPEG-noise were made to make model comparisons for some example images from the test set. The reason was to observe if the network learned to reduce the 8×8 block artifacts. However, no decision about model choice was based on visual inspection.

4 Results

In this section the most important results from the project are presented and briefly explained. To keep it concise, supplementary material and additional image samples are to be found in appendices.

4.1 JJNet trained on large dataset

In this subsection we present the results of training JJNet on the large dataset, details given by Table 6 in the methodology section. Figure 24 shows violin plots of metrics MAE, MSE, SSIM and PSNR from resulting predictions of three JJnet models trained on quality 50 (yellow), 75 (green) and random (black) respectively. On the x-axis we have the quality factor of the JPEG-images used as input. In the same figure, the results are compared to the metrics of corresponding JPEG images, represented by the blue violin. All metrics for predictions made on an input image of quality 50 by the model trained on JPEG-75 images scored higher in average than for JPEG-75 compared to the ground truth images.

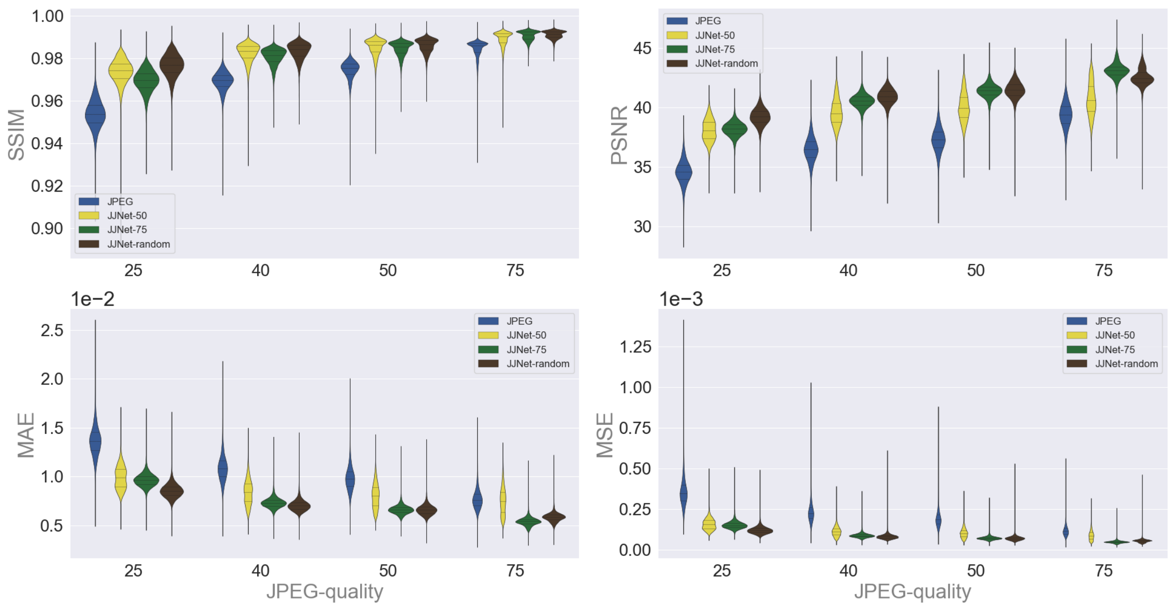


Figure 24: Comparing JJNet models trained on quality 50, 75 and random. The violin plots represents the distribution of the measured metrics SSIM, PSNR, MAE and MSE when comparing images predicted by the models and the original images. The models were let to predict on the input qualities 25, 40, 50 and 75. All images considered were of size 192 × 192.

For the models considered in Figure 24, the relative reconstruction in terms of the metrics is plotted in Figure 25.

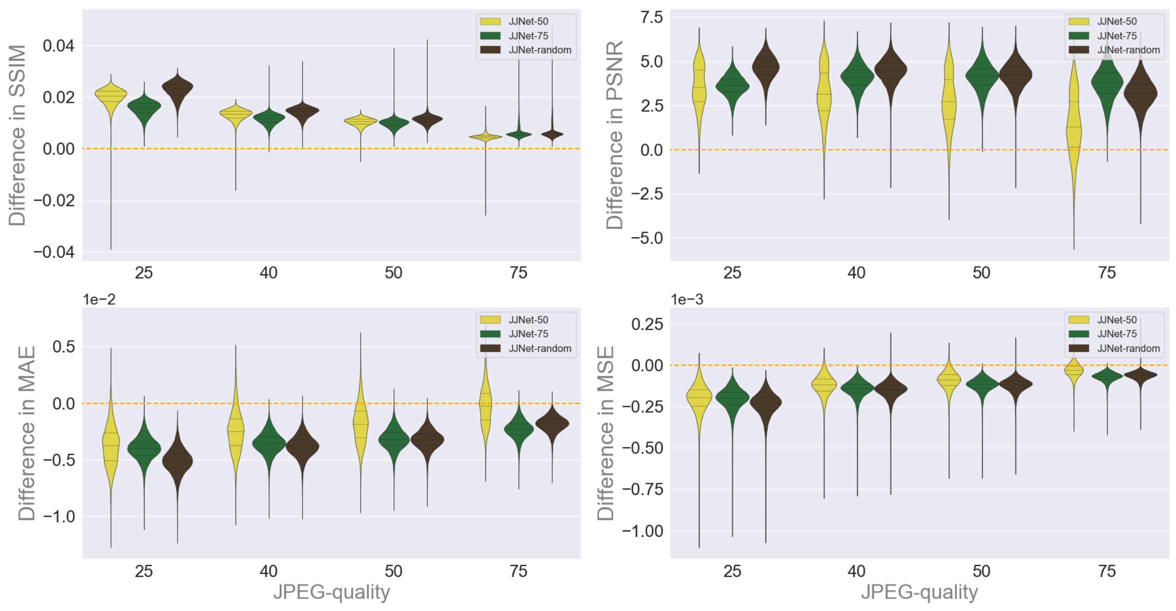


Figure 25: The violin plots represent the relative reconstruction in the units of the metrics SSIM, PSNR, MAE and MSE for the JJNet models trained on input qualities 50, 75 and random. The orange dashed line marks the line $y = 0$. For the quality scores SSIM and PSNR, positive difference marks an improvement. For the pixel errors MAE and MSE, a negative difference marks an improvement.

The models were trained until no improvement in validation loss was achieved for 50 epochs in a row. Table 7 summarizes the number of epochs run for each model.

Table 7: Number of epochs run until stopping condition was met for the models compared.

Model	JJnet-25	JJNet-50	JJNet-75	JJNet-Random
Epochs	268	72	73	299

4.2 ResNetYUV trained on large dataset

In Figure 26 the results for the metrics MAE, MSE, SSIM and PSNR are summarized in a violin plot. In the same figure the results are compared to the metrics of corresponding JPEG-images. All metrics for predictions made on an input image of quality 50 scored higher in average than for JPEG-75 In Figure 27 the relative improvement in terms of the metrics is presented in violin plots. For the SSIM-metric the tail is large but thin for the ResNetYUV-models. The outliers make up 0.4 % of all images evaluated and it was found that they come all from the DM96 system (yellow). A sample of the outliers can be found in Figure 28. The majority are cells that would be classified as smudge cells.

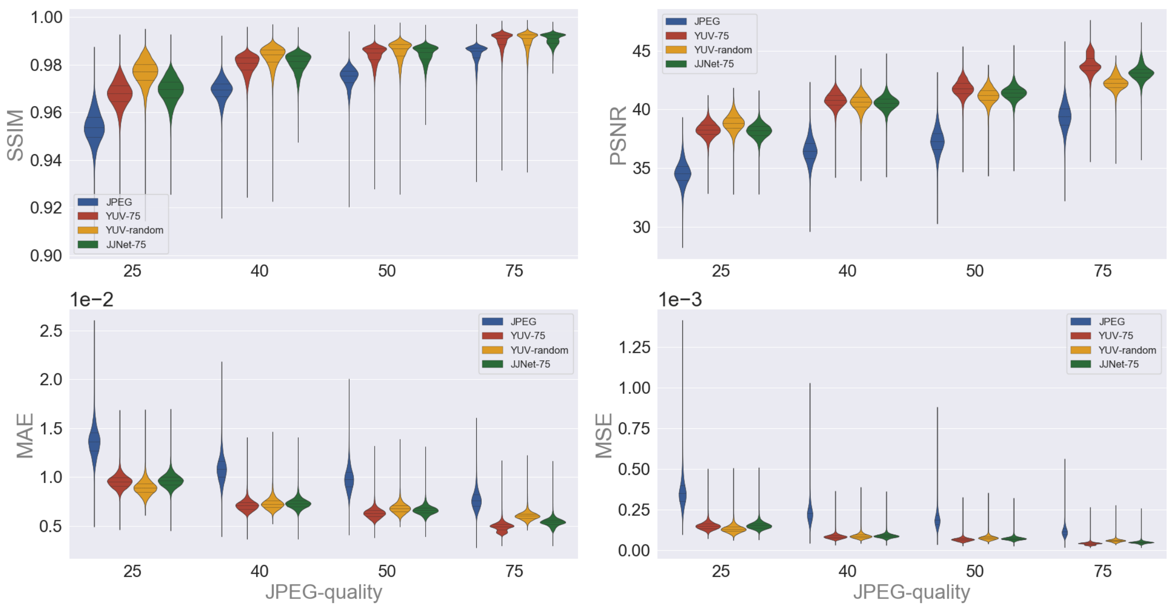


Figure 26: Comparing ResNetYUV models trained on quality 75 and random quality. The violin plots represents the distribution of the measured metrics SSIM. PSNR, MAE and MSE when comparing images predicted by the models and the original images. The models were let to predict on the input qualities 25, 40, 50 and 75. All images considered were of size 192 × 192.

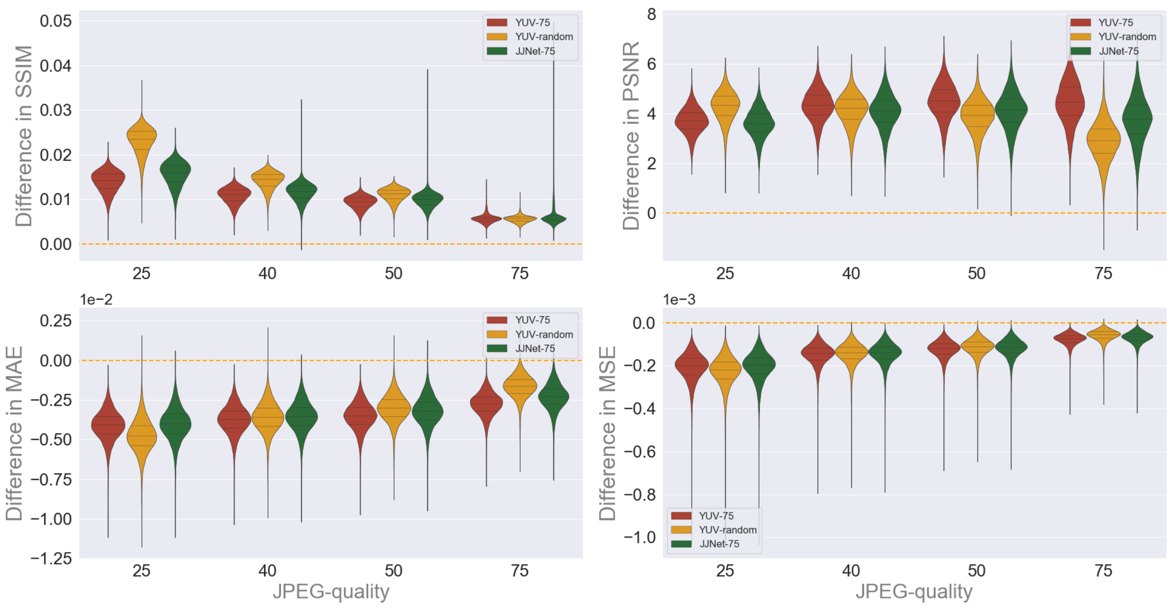


Figure 27: The violin plots represent the relative reconstruction in the units of the metrics SSIM, PSNR, MAE and MSE for the ResNetYUV models trained on input qualities 75 and random. The orange dashed line marks the line $y = 0$. For the quality scores SSIM and PSNR, positive difference marks an improvement. For the pixel errors MAE and MSE, a negative difference marks an improvement.

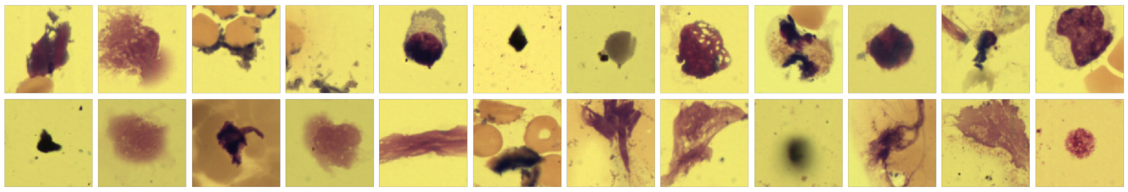
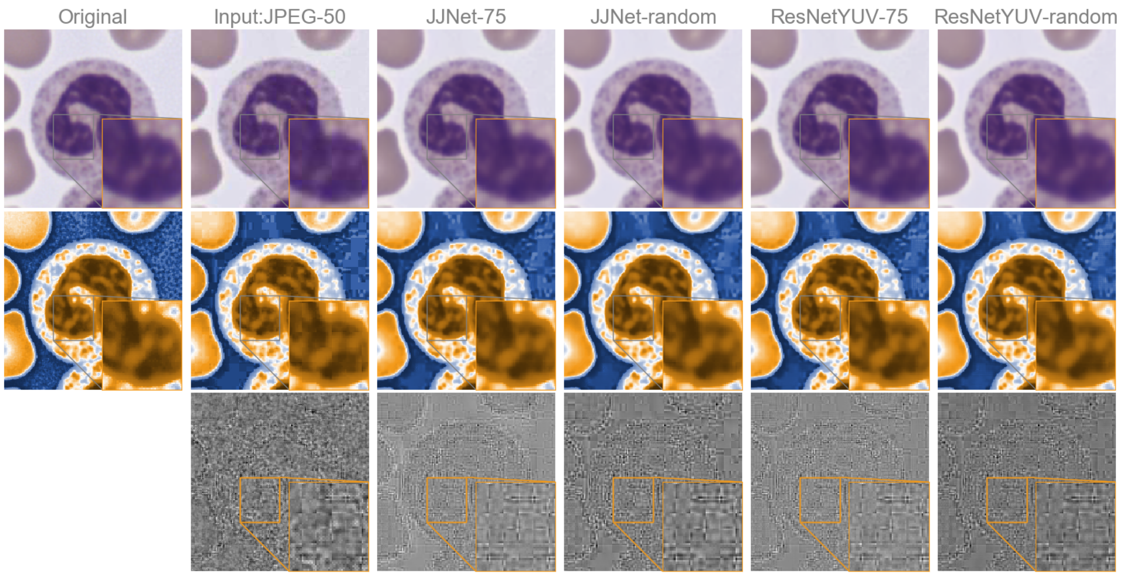


Figure 28: Sample of outliers for the ResNetYUV.



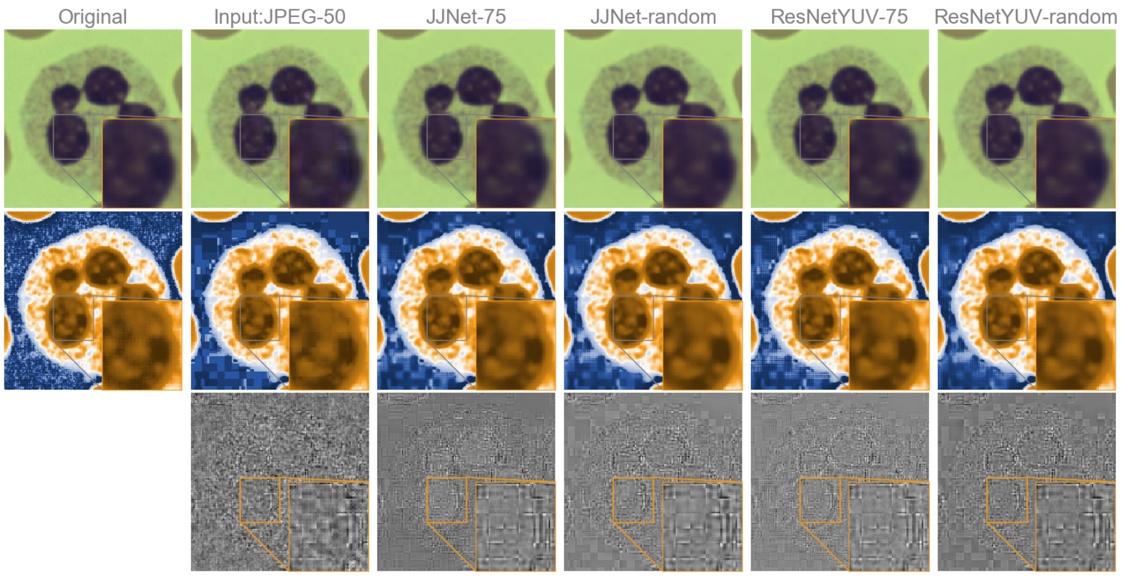


Figure 29: The first row contains the image predicted on, from the left we have: the original BMP image, the input image in **JPEG-50** format, prediction of our JJNet model trained on JPEG-75 quality images, prediction of our JJNet model trained on random JPEG-quality, prediction of JJNet model trained on JPEG-50 images, ResNetYUV trained on JPEG-75. In the second row the image has been converted to grayscale and then histogram equalized. In the bottom row we find the residual when compared to the original image. Hence, the models tries to predict the bottom row and add it to the input image.

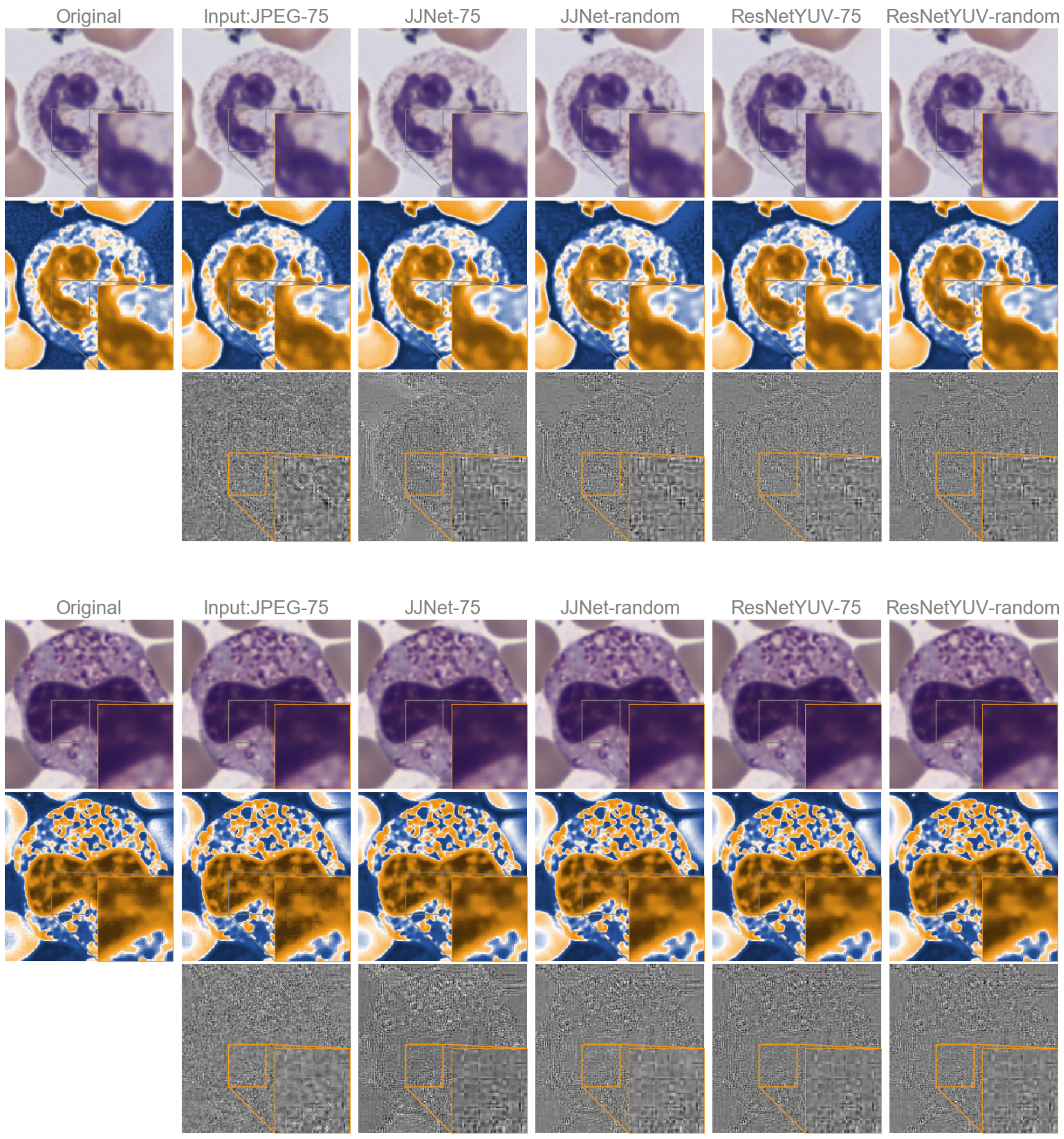


Figure 30: Just as on the previous page, these are model comparisons. The first row contains the image predicted on, from the left we have: the original BMP-image, the input image in **JPEG-75** format, prediction of our JJNet model trained on random quality images, prediction of ResNetYUV trained on JPEG-75 and prediction of ResNetYUV trained on random quality. In the second row the image has been converted to grayscale and then histogram equalized. In the bottom row we find the residual when compared to the original image. Hence, the models tries to predict the bottom row and add it to the input image.

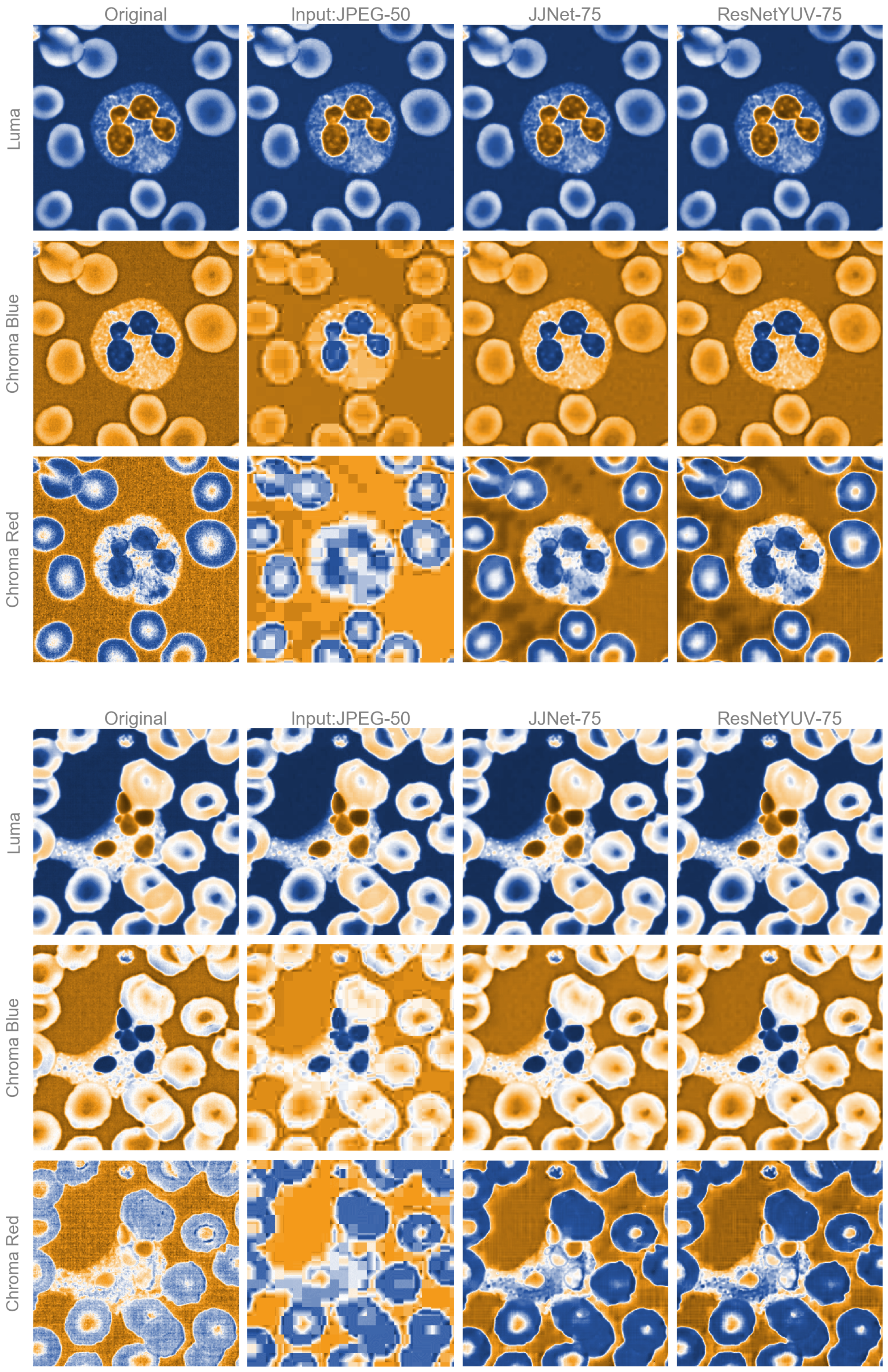


Figure 31: Two cell images shown in the YCbCr color space. There are almost no noticeable difference in the luminance space, while there are substantial differences in the chrominance space.

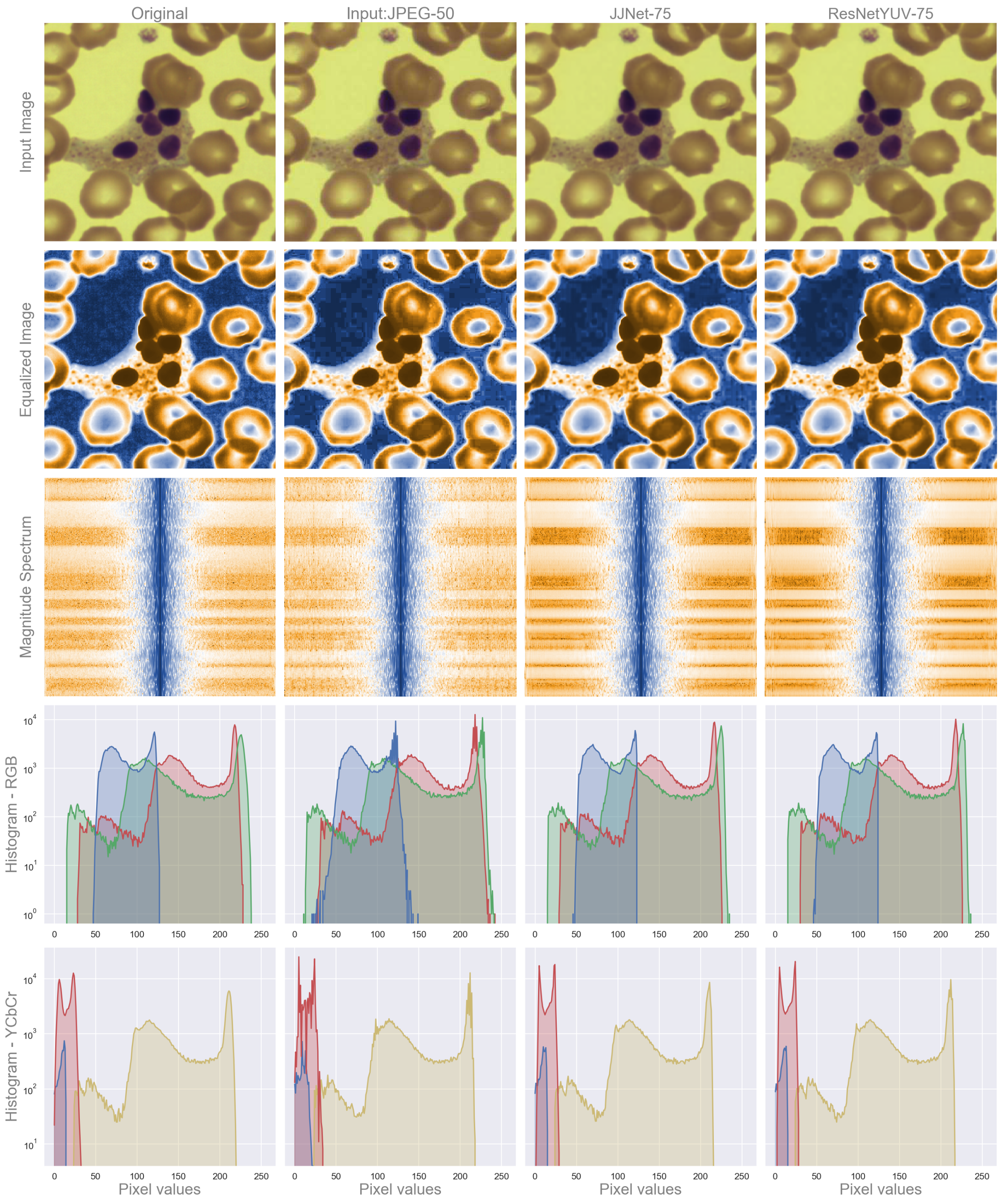


Figure 32: Comparing input, ground truth and predictions in equalized, magnitude spectrum and histogram in both RGB and YCbCr color space. The histogram is plotted in logarithmic scale.

In Figure 29 and 30 the predictions from four models are compared to the input and the original image, the residuals are also displayed in the third row. In the residual images, the 8×8 blocks are clearly visible. The differences are further displayed in Figure 31, where the images are converted to YCbCr color-space and visualized separately for every channel, we can see that luminance channels are barely affected by compression, while the chrominance channels are affected heavily. Lastly, in Figure 32 the magnitude spectrum and histogram for RGB and YCbCr are plotted, the color distribution are less smooth for the compressed image compared to the original. The models are able to smooth out the differences to some degree.

5 Discussion

5.1 General conclusions

It is possible to train a CNN to do JPEG-deblocking of cell images. At least 99.9 % of all images were improved with respect to MAE, MSE, PSNR and SSIM. The aim of the thesis was to investigate how low quality that could successfully be restored on average by a CNN. The results point at somewhere between quality 40 and 50, with respect to the concerned metrics. When comparing average file sizes in table 1 in the introduction section, we can conclude that even restoring JPEG-images of quality 60 would save 22.8% storage space on average, compared to what CellaVision is using today. Due to the problems of washed out details, briefly discussed in the introduction, it might not be worth taking the risk letting the model predict from a too low quality. Details that could be important for the following diagnostic could disappear or be distorted.

At the beginning stage of this project we implemented networks with a more generative approach to the problem. These models took a JPEG-image as input and generated a new image, not the residual mapping. Using this method, all network structures that we implemented failed to correctly predict the color, and often the output suffered from border errors. However, when we changed approach to letting the models predict the residual mapping between the JPEG-image and its original, results increased significantly and the need for sophisticated padding layers was eliminated. This confirms the hypothesis that predicting the residual mapping is a simpler problem.

The models were all trained on images of resolution 192 × 192 and center-cropped, which means that the training images have higher information density than the full size images. The laboratory technician working with the images does only care about the cell itself when performing the analysis. We noticed that when evaluating our images on the full image size, our results were better than when evaluating on the cropped images. Our hypothesis for this is when including more background in the images, it pushes the error down for the pixel metrics MAE, MSE and PSNR. This is also true when comparing JPEG-images with the originals. Since the background is basically only one color, it is easily compressed, and even though the models do not focus on reconstructing the background, they are doing a good enough work for the background's contribution to the mean error to be small.

5.2 Training input quality for reconstruction of low quality images

In terms of SSIM and PSNR on average, a JPEG image of a quality factor between 40 and 50 can be reconstructed to at least a quality comparable to JPEG-75, which is used today in CellaVision's products. This result were achieved by JJNet trained on 75 quality and JJNet trained on random input quality. The variance seems to be larger for the model trained on random quality, which should be expected.

When it came to which quality factor to train on, results showed that the model trained on images of quality 50 actually performed worse on all pixel errors (MSE, MAE, PSNR) than the model trained on 75 quality when reconstructing JPEG-50. However, the model trained on 50 achieved a SSIM score a little bit higher on average but with a higher number of outliers, see the violin plots in Figure 24. It seems like for this model structure and experiment setup, it is advantageous to train on higher quality images. We expected the model trained on JPEG-50 to perform best on quality 50 images, and that was true with respect to the SSIM score but not for the pixel errors. Morover, it has a different distribution which is visible in Figure 24. We think it can depend on the fact that reconstructing a JPEG-50 image is a harder problem than reconstructing a JPEG-75, and a softer stopping condition, different hyperparameters, or even a different model could have been preferable. We have to remember that the JJNet structure was selected because results showed that a model trained on JPEG-75 also performed well when predicting on JPEG-50.

At the same time, the model trained on random JPEG-quality shows better metrics results when predicting on JPEG-25 than the model trained on JPEG-75. Letting the network train on random input quality can be seen as augmenting the data, which apparently improves generalization.

Due to the very different distribution in the results for the JJNet-50 model, we investigated if it could be related to the different data sources. In the supplementary results, Figure 33 shows that for the model trained on quality 50 the results depend on which system the images are from. In terms of pixel errors MAE, MSE and PSNR, the model trained on quality 50 performs better on the images from the DM1200 and DM96 systems, that is to say the systems that produce the green and yellow images. However, in terms of SSIM these models achieved better results for the white/pink images from the DC-1 system. This could be partly explained by Figure 34 that shows that the difference in quality between the images from the different systems is also present in the compressed images.

5.3 Comparing models predicting in RGB or YCbCr space

When comparing our model JJNet with a residual net built to exploit the channel separation used by the JPEG-compression algorithm, we find that the predicted images look alike and results are similar. The mapping between the colorspaces RGB and YCbCr is linear, so the model should learn it without much effort. Nevertheless, when it comes to efficiency, we propose predicting in YCbCr to be able to focus the computing power on reconstructing the Cb and Cr channels. As seen in Figure 31.

Moreover, the outlier cells that produce the thin tail in the violin plots for the YCbCr-models are all yellow images from the DM96 system, which is underrepresented in the data. The amount of outliers is very low, and also, the of all outliers the majority are cells what would be classified as smudge cells. That means cells that are dying or breaking into parts and are not easily confused with other cell classes.

5.4 Further work

A big part of this thesis was to evaluate different types of CNN architectures, which is time consuming. With the developments of recent research, such as Graph Neural Networks (GNN) and randomly wired neural networks [32], the architecture choice can be automated. The number of parameters are also a concern for this application to be useful in practical situations. An analysis of running time and pruning should be done to optimize the network.

JPEG-compression is for certain a useful image compression method, but does not take the semantics of the images into account. Because cell images are very similar, we believe that a dimensionality reduction method based on learning could be useful for compression, such as an autoencoder. Our work could further be used as a final part of a decoder in an autoencoder. Assuming an autoencoder has a dense, flattened bottleneck layer, the encoded image can be stored in a vector containing roughly 1797 floating number of the datatype float32 and have the same size as an JPEG-50 image. Of course, this would only be useful if the decoding and reconstruction proved to be better than JPEG-50 quality.

References

[1] I. T. U. (ITU, "Iso/iec is 10918-1: Digital compression and coding of continuous-tone still image – requirements and guidelines," 1993. Available: `https://www.w3.org/Graphics/JPEG/itu-t81.pdf`.

[2] D. Salomon, G. Motta, and D. Bryant, *Data Compression: The Complete Reference*. Molecular biology intelligence unit, Springer London, 2007.

[3] B. Bain, *A Beginner's Guide to Blood Cells*. Wiley, 2008.

[4] B. Bain, *Blood Cells: A Practical Guide*. Wiley, 2014.

[5] V. Antun, F. Renna, C. Poon, B. Adcock, and A. C. Hansen, "On instabilities of deep learning in image reconstruction and the potential costs of ai," 2020. Available: `https://www.pnas.org/content/early/2020/05/08/1907377117`.

[6] J. Hadamard, "Sur les problèmes aux dérivées partielles et leur signification physique," *Princeton University Bulletin*, pp. 49—52, 1902.

[7] H. Rue and L. Held, *Gaussian Markov Random Fields: Theory and Applications*. Chapman & Hall/CRC, 2005.

[8] S. J. Russell and P. Norvig, *Artificial Intelligence: A modern approach*. Pearson, 3 ed., 2009.

[9] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[10] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

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

[12] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," *Proceedings of ICML*, vol. Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel, 2010. Available: `https://icml.cc/Conferences/2010/papers/432.pdf`.

[13] Y. A. Lecun, L. Bottou, G. B. Orr, and K. R. Müller, "Efficient backprop," 1998.

[14] P. Veličković, "Tikz," 2018. `https://github.com/PetarV-/TikZ`.

[15] F. Chollet, *Deep Learning with Python*. USA: Manning Publications Co., 1st ed., 2017.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. Available: `https://arxiv.org/abs/1502.03167`.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. Available: `https://arxiv.org/abs/1412.6980`.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. Available: `https://arxiv.org/abs/1512.03385`.

[19] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. Available: `https://arxiv.org/abs/1608.06993`.

[20] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR*, vol. abs/1610.02357, 2016. Available: `https://arxiv.org/abs/1610.02357`.

[21] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. Available: `https://arxiv.org/abs/1505.04597`.

[22] Y. Tai, J. Yang, X. Liu, and C. Xu, “Memnet: A persistent memory network for image restoration,” 2017. Available: `https://arxiv.org/abs/1708.02209`.

[23] R. Szeliski, *Computer Vision: Algorithms and Applications.* Springer London, 2011.

[24] X. Zhang, J. Zou, K. He, and J. Sun, “Accelerating very deep convolutional networks for classification and detection,” 2015. Available: `https://arxiv.org/abs/1505.06798`.

[25] J. Johnson, A. Alahi, and F. Li, “Perceptual losses for real-time style transfer and super-resolution,” *CoRR*, vol. abs/1603.08155, 2016.

[26] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: From error visibility tostructural similarity,” 2004. Available: `https://www.cns.nyu.edu/pub/eero/wang03-reprint.pdf`.

[27] R. Aliyev, “Keras center and random crop support for imagedatagenerator,” 2019. Available: `https://gist.github.com/rstml/bbd491287efc24133b90d4f7f3663905`.

[28] M. Cooper, “When conventional wisdom fails: Revisiting data augmentation for self-driving cars,” 2018. Available: `https://towardsdatascience.com/when-conventional-wisdom-fails-revisiting-data-augmentation-for-self-driving-cars-4831998c5509`.

[29] J. L. Hintze and R. D. Nelson, “Violin plots: A box plot-density trace synergism,” *The American Statistician*, vol. Vol. 52, No. 2, pp. 181-184, 1998. Available: `http://www.stat.cmu.edu/~rnugent/PCMI2016/papers/ViolinPlots.pdf`.

[30] T. K. P. library. Available: `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam`.

[31] S. Zini, S. Bianco, and R. Schettini, “Deep residual autoencoder for quality independent jpeg restoration,” 2019. Available: `https://arxiv.org/abs/1903.06117`.

[32] S. Xie, A. Kirillov, R. Girshick, and K. He, “Exploring randomly wired neural networks for image recognition,” 2019. Available: `https://arxiv.org/abs/1904.01569`.

A Results of evaluation of CNN structures and loss-functions

Table 8: JJNet (none activation) trained on tinydata-75 set with different loss functions. Evaluated on full size image 360x360.

Q input	loss / metric	MAE	MSE	$\mathcal{H}_f$	P
JPEG-75	MAE	0.005202	0.005098	0.008542	0.010175
	MSE	0.000048	0.000045	0.005262	0.001653
	SSIM	0.986610	**0.987806**	0.987498	0.979426
	PSNR	43.292477	**43.520752**	31.151243	33.367924
JPEG-50	MAE	0.006477	0.006403	0.009689	0.011212
	MSE	0.000072	0.000070	0.004993	0.001677
	SSIM	0.980852	0.981726	0.982134	0.972780
	PSNR	41.479839	41.622425	30.997332	32.893024
JPEG-25	MAE	0.009427	0.009409	0.012310	0.013836
	MSE	0.000148	0.000147	0.004536	0.001771
	SSIM	0.968668	0.969280	0.970455	0.957383
	PSNR	38.333157	38.385494	30.207113	31.712442

Table 9: DenseUnet trained on tinydata-75 set with different loss functions. Evaluated on full size image 360x360.

Q input	loss / metric	MAE	MSE	$\mathcal{H}_f$	P
JPEG-75	MAE	0.005422	0.005734	0.005760	0.014171
	MSE	0.000051	0.000057	0.000058	0.000324
	SSIM	0.985958	0.984753	0.985308	0.942267
	PSNR	42.970387	42.495281	42.443214	34.914772
JPEG-50	MAE	0.006801	0.007206	0.007246	0.014984
	MSE	0.000079	0.000089	0.000091	0.000362
	SSIM	0.979461	0.977618	0.978709	0.934879
	PSNR	41.066444	40.538464	40.483364	34.433105
JPEG-25	MAE	0.009846	0.010339	0.010374	0.017021
	MSE	0.000162	0.000180	0.000182	0.000470
	SSIM	0.966254	0.963242	0.965468	0.918965
	PSNR	37.957813	37.498360	37.464077	33.299122

Table 10: Xception trained on tinydata-75 set with different loss functions. Evaluated on full size image 360x360.

Q input	loss / metric	MAE	MSE	$\mathcal{H}_f$	P
JPEG-75	MAE	0.006262	0.006247	0.007111	0.948656
	MSE	0.000075	0.000074	0.000091	0.902291
	SSIM	0.983904	0.984214	0.986251	0.618318
	PSNR	41.312988	41.379795	40.546642	0.448263
JPEG-50	MAE	0.007877	0.007793	0.008458	0.947834
	MSE	0.000118	0.000115	0.000132	0.900775
	SSIM	0.976617	0.977121	0.980510	0.619807
	PSNR	39.357105	39.467072	38.923443	0.455588
JPEG-25	MAE	0.011215	0.011080	0.011549	0.947747
	MSE	0.000230	0.000225	0.000241	0.900738
	SSIM	0.961995	0.962809	0.968019	0.619482
	PSNR	36.467884	36.570087	36.285736	0.455777

Table 11: UNet trained on tinydata-75 set with different loss functions. Evaluated on full size image 360x360.

Q input	loss / metric	MAE	MSE	$\mathcal{H}_f$	P
JPEG-75	MAE	0.006855	0.006051	0.048337	0.024066
	MSE	0.000095	0.000065	0.115861	0.002092
	SSIM	0.955383	0.966090	0.886478	0.763669
	PSNR	40.357185	41.966438	20.888081	30.223175
JPEG-50	MAE	0.008655	0.007566	0.050914	0.024764
	MSE	0.000149	0.000101	0.119108	0.002146
	SSIM	0.942370	0.956861	0.872747	0.758365
	PSNR	38.374264	40.040775	20.710127	29.860464
JPEG-25	MAE	0.012149	0.010796	0.052711	0.024976
	MSE	0.000281	0.000200	0.113582	0.001895
	SSIM	0.919325	0.938162	0.851783	0.752064
	PSNR	35.627075	37.042557	20.140076	29.649017

Table 12: DenseUnet trained on tinydata-50 set with different loss functions. Evaluated on full size image 360x360.

Q input	loss / metric	MAE	MSE	$\mathcal{H}_f$	P
JPEG-75	MAE	0.005787	0.005579	0.006155	0.013731
	MSE	0.000059	0.000054	0.000083	0.000278
	SSIM	0.985002	0.985866	0.986618	0.971110
	PSNR	42.345886	42.750717	40.852112	35.566528
JPEG-50	MAE	0.007094	0.006822	0.007476	0.014357
	MSE	0.000088	0.000079	0.000111	0.000309
	SSIM	0.978725	0.979657	0.981877	0.964538
	PSNR	40.630436	41.055099	39.601498	35.111153
JPEG-25	MAE	0.010043	0.009722	0.010438	0.015898
	MSE	0.000170	0.000157	0.000195	0.000392
	SSIM	0.965869	0.966902	0.971548	0.950940
	PSNR	37.737923	38.083920	37.144844	34.090057

Table 13: DenseUnet trained on tinydata-25 set with different loss functions. Evaluated on full size image 360x360.

Q input	loss / metric	MAE	MSE	$\mathcal{H}_f$	P
JPEG-75	MAE	0.006855	0.005848	0.035370	0.024339
	MSE	0.000093	0.000060	0.025804	0.000718
	SSIM	0.983301	0.986121	0.956016	0.980282
	PSNR	40.427544	42.296913	16.121908	31.457451
JPEG-50	MAE	0.008656	0.006764	0.037031	0.024537
	MSE	0.000148	0.000079	0.025791	0.000746
	SSIM	0.975315	0.980737	0.948317	0.974821
	PSNR	38.419212	41.111084	16.119438	31.294344
JPEG-25	MAE	0.012148	0.008945	0.040631	0.024772
	MSE	0.000279	0.000134	0.026143	0.000807
	SSIM	0.959129	0.969495	0.932359	0.961603
	PSNR	35.654266	38.787029	16.048773	30.950073

Table 14: ResNet trained on tinydata-75 set with different loss functions. Evaluated on full size image 360x360.

Q input	loss / metric	MAE	MSE	$\mathcal{H}_f$	P
JPEG-75	MAE	0.017533	0.010621	0.130226	0.059120
	MSE	0.000540	0.000224	0.038239	0.005159
	SSIM	0.975525	0.977998	0.887520	0.970587
	PSNR	33.086910	37.267948	14.239019	24.623951
JPEG-50	MAE	0.017976	0.011544	0.129919	0.058985
	MSE	0.000572	0.000259	0.038070	0.005135
	SSIM	0.970875	0.972777	0.884669	0.965716
	PSNR	32.823353	36.544926	14.258619	24.615915
JPEG-25	MAE	0.019061	0.013539	0.129260	0.058942
	MSE	0.000637	0.000342	0.037910	0.005128
	SSIM	0.960926	0.961383	0.876967	0.954117
	PSNR	32.276459	35.147339	14.277790	24.526352

Table 15: DenseNet trained on tinydata-75 set different with loss functions. Evaluated on full size image 360x360.

Q input	loss / metric	MAE	MSE	$\mathcal{H}_f$	P
JPEG-75	MSE	0.006850	0.005651	0.005878	0.010891
	MAE	0.000093	0.000056	0.000096	0.000201
	SSIM	0.983302	0.985490	0.986446	0.975798
	PSNR	40.427036	42.609436	40.739090	37.276997
JPEG-50	MSE	0.008650	0.007077	0.007226	0.011923
	MAE	0.000148	0.000087	0.000124	0.000243
	SSIM	0.975317	0.978860	0.980486	0.970161
	PSNR	38.419350	40.680565	39.354683	36.411327
JPEG-25	MSE	0.012145	0.010251	0.010279	0.014379
	MAE	0.000279	0.000178	0.000211	0.000353
	SSIM	0.959127	0.964273	0.967032	0.954970
	PSNR	35.653793	37.553047	36.857132	34.687981

Table 16: MemNet trained on tinydata-75 set with different loss functions. Evaluated on qualities 25, 50 and 75 images of full size image 360x360.

Q input	loss / metric	MAE	MSE	$\mathcal{H}_f$	P
JPEG-75	MAE	0.0070141	0.0070466	0.0069026	0.0138406
	MSE	0.0000827	0.0000832	0.0000792	0.0003007
	SSIM	0.9857847	0.9858206	0.9873110	0.9810556
	PSNR	41.0996437	40.9613304	41.1519051	35.3439293
JPEG-50	MAE	0.0082995	0.0081888	0.0080126	0.0147284
	MSE	0.0001138	0.0001114	0.0001047	0.0003488
	SSIM	0.9800301	0.9799342	0.9822581	0.9730129
	PSNR	39.6191025	39.6462784	39.8984489	34.6901474
JPEG-25	MAE	0.0108490	0.0108901	0.0104284	0.0167932
	MSE	0.0001914	0.0001951	0.0001739	0.0004648
	SSIM	0.9683505	0.9680085	0.9719595	0.9566798
	PSNR	37.2544060	37.1804504	37.6432533	33.4264565

Table 17: JJNet (tanh activation) trained on tinydata-75 set with different loss functions. Evaluated on full size image 360x360.

Q input	metric \ loss	MAE	MSE	$\mathcal{H}_f$	P
JPEG-75	MAE	0.006850	0.005810	0.999585	0.009662
	MSE	0.000093	0.000059	0.999369	0.000150
	SSIM	0.983301	0.983998	0.168727	0.979090
	PSNR	40.427040	42.347717	0.002758	38.274952
JPEG-50	MAE	0.008651	0.007274	0.999682	0.010549
	MSE	0.000148	0.000092	0.999623	0.000182
	SSIM	0.975317	0.976472	0.167576	0.972767
	PSNR	38.419392	40.406876	0.001658	37.445156
JPEG-25	MAE	0.012145	0.010391	0.999904	0.012982
	MSE	0.000279	0.000185	1.000191	0.000280
	SSIM	0.959128	0.961250	0.165419	0.957670
	PSNR	35.653831	37.396099	-0.000804	35.566620

B Supplementary results

Comparing training on two dataset

Table 18: Mean values of metrics of predictions for JJNet evaluated on the test set data100 (360×360), comparing quality factor of images trained on. Compared to Table 3, JJNet-75 and JJNet-random fulfills the reconstruction requirements.

QF input	metric \ model	25	50	75	random
JPEG-75	MAE$\cdot10^3$	16.17	7.764	**5.378**	5.672
	MSE$\cdot10^5$	42.0	9.420	**4.864**	5.290
	SSIM	0.9841	0.9878	0.9896	**0.9902**
	PSNR	35.94	40.57	**43.18**	42.83
JPEG-50	MAE$\cdot10^3$	16.166	8.133	6.431	**6.251**
	MSE$\cdot10^5$	41.6	10.38	6.881	**6.480**
	SSIM	0.9829	0.9845	0.9838	**0.9855**
	PSNR	35.82	40.06	41.66	**41.93**
JPEG-25	MAE$\cdot10^3$	16.14	9.695	9.158	**7.996**
	MSE$\cdot10^5$	40.1	14.96	13.74	**10.55**
	SSIM	0.9794	0.9767	0.9720	**0.9792**
	PSNR	35.45	38.38	38.65	**39.82**

Table 19: Mean values of metrics of predictions for JJNet evaluated on the test set data101 (360×360), comparing quality factor of images trained on. Compared to Table 3, JJNet-75 and JJNet-random fulfills the reconstruction requirements.

QF input	metric \ model	25	50	75	random
JPEG-75	MAE$\cdot10^3$	6.205	7.369	5.732	5.580
	MSE$\cdot10^5$	7.380	8.675	5.519	5.220
	SSIM	0.98434	0.988454	0.988858	0.98916
	PSNR	42.476	40.880551	42.6328	42.8778
JPEG-50	MAE$\cdot10^3$	6.487	8.082	6.761	6.223
	MSE$\cdot10^5$	7.641	10.50	7.590	6.541
	SSIM	0.983122	0.984145	0.9829	0.98455
	PSNR	42.075	40.00	41.24	41.89067
JPEG-25	MAE$\cdot10^3$	7.566	10.12	9.940	8.262
	MSE$\cdot10^5$	8.052	16.50	14.50	11.32
	SSIM	0.9795	0.972310	0.97097	0.97776
	PSNR	40.70677	37.9403	38.43	39.51064

Correlation between systems

When separating the white/pink and green/yellow images, the violin plots for models trained on JPEG-50 quality take an interesting form.

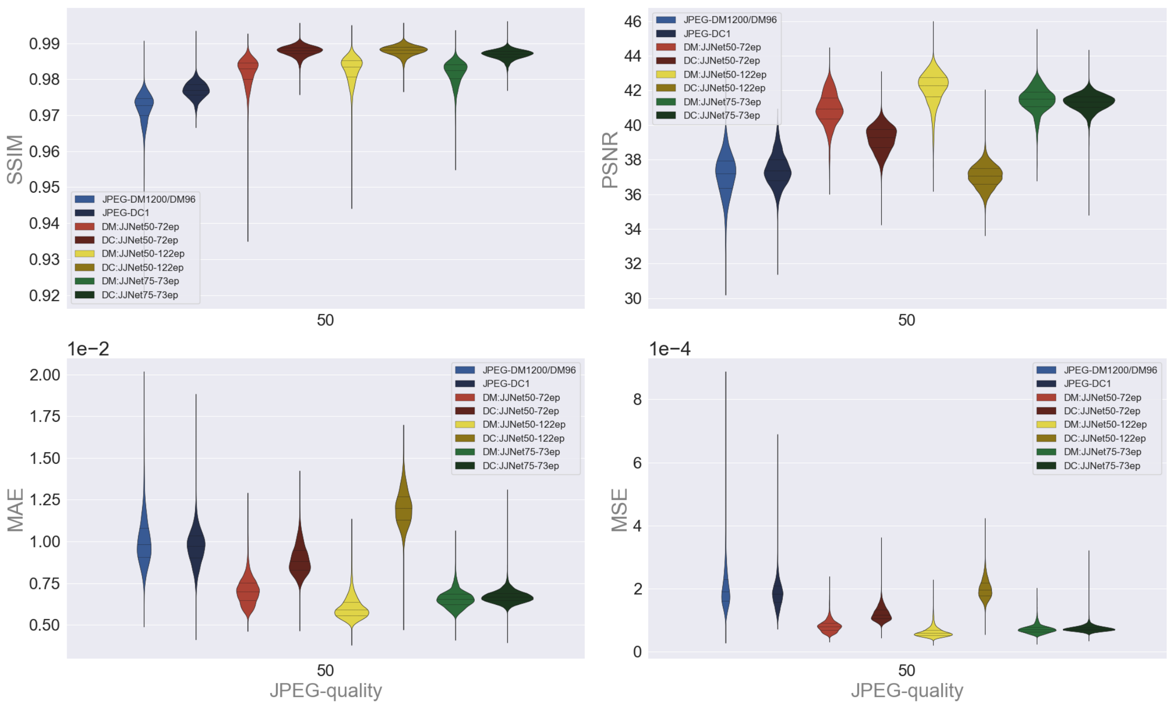


Figure 33: Model and camera system comparison with respect to the metrics SSIM, PSNR, MAE and MSE when predicting on quality 50. The darker color corresponds to white/pink images from the DC1 system and the brighter color corresponds to green/yellow images from the DM1200 and DM96 systems. In red is the JJNet model trained on JPEG-50 for 72 epochs with the early stopping condition. In yellow we have the same model trained for another 50 epochs, in total 122 epochs. In green we have as before the JJNet model trained on JPEG-75 for 73 epochs. For the model trained on JPEG-50, there is a clear difference between white and green images and the separation between the two groups increase with epochs trained.

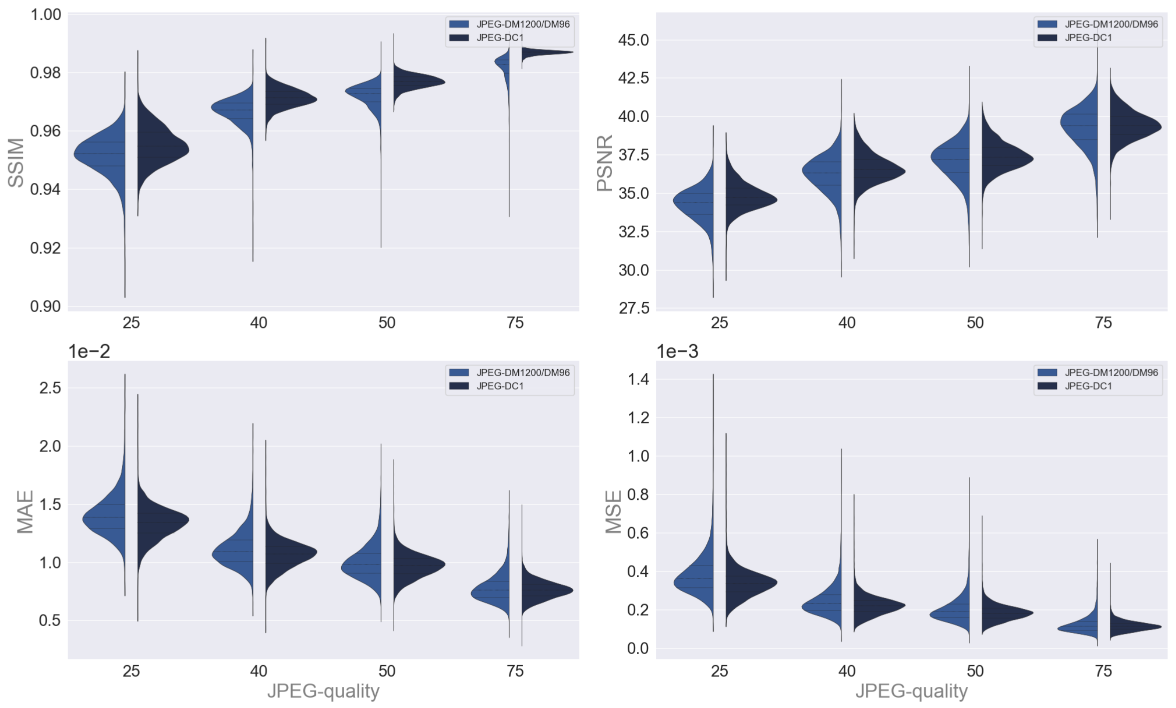


Figure 34: Comparing JPEG-compression quality with respect to the metrics SSIM, PSNR, MAE and MSE. In blue we have the evaluations of the green and yellow images from the DM1200 and DM96 systems, and in darkblue we have the evaluations of the white and pink images from the DC-1 system.

Master’s Theses in Mathematical Sciences 2020:E77
ISSN 1404-6342

LUTFMA-3433-2020

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

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