Depth of field post-processing using neural networks

A master's thesis at the faculty of engineering

Otto Holmström
otto.holmstrom@hotmail.com

Lund University

Examiner
Andreas Langer

Supervisor
Carina Geldhauser

March 17, 2022

Abstract

It is possible to create images from a computer model closely resembling those taken with a physical camera. To improve the photo-realism and perceived quality of a rendered image, it is often desireable to add realistic effects that do not appear in computer graphics due to the camera model. One of these effects is depth of field, where a physical camera with a lens can only focus at one specific distance in a scene, making the rest of the scene appear blurry. This is opposed to a rendered image, where the entire scene appears sharp.

In this thesis work, it is investigated if a neural network is able to replicate the depth of field effect in computer rendered images, when given an image and the distance from the camera of objects in the image. Three neural networks based on the same structure are created and studied, and it is found that the simplest model fails to retain background information in the images while the most complex model manages to replicate the depth of field effect, with an average PSNR of 43.12 and average SSIM of 0.987.

Acknowledgments

The thesis work was carried out at ARM Sweden, over the period September 2021 to February 2022. I would like to thank my supervisor at ARM Alexander Hansson for his invaluable support throughout the thesis process. A big thanks also goes out to Mats Ekström for various creative ideas and continuous support, and to the machine learning engineers at ARM for their fruitful feedback in model design.

I would also like to thank my university supervisor Carina Geldhauser for her feedback in the thesis writing process, and for keeping me somewhat on track with the academic process.

Contents

Abbreviations

CoC - Circle of confusion

DoF - Depth of field

FLOPs - Floating point operations

GPU - Graphics processing unit

MSE - Mean squared error

NN - Neural network

PSNR - Peak signal to noise ratio

SSIM - Structural similarity

ReLU - Rectified linear unit

1 Introduction

Using computer graphics, it is possible to generate images from a software model that closely resemble images with a physical camera. These software models contain many abstractions, approximations and optimizations to render images in a reasonable time, and it is possible to tune the complexity of the model to balance rendering time against image quality. After generating an image, it is often desirable to improve the perceived quality of the content in a process called *post-processing*, by sharpening, blurring or denoising parts of the image to mention a few common post-processing effects.

Camera lenses can only focus at a defined distance at a time, and as such a photo taken of a scene with objects at different depths will not be able to focus on all objects at once. From this concept, the *depth of field* is the distance from the camera at which objects appear in focus, and the depth of field *effect* is the dynamic blurring of objects in a scene depending on the distance to the camera, also known as depth [1]. The physics behind depth of field are touched more upon in section 2.

Post processing effects can be complicated to compute for an entire image and in situations where the total image generation time is critical, rendering and post processing included, such as real time rendering in video games, post processing effects may be tuned down or completely disabled to reduce the total rendering time. Existing commonly available solutions show that neural networks have the potential to assist image rendering by sharpening or upscaling rendered images, improving the total rendering time, or improving the final image quality [2].

This thesis aims to investigate using a neural network model to emulate the depth of field effect in rendered images. For this, three neural networks of different complexities are created and trained on a dataset of images with and without depth of field, along with 3D scene information. It is found that the simplest model fails to preserve the background color in test images, while the most complex model is able to replicate the depth of field effect with very high fidelity, achieving an average PSNR score of 43.12 and average structural similarity of 0.987 over the test data.

1.1 Problem formulation

The thesis work stems from the question: can a depth of field effect be replicated, or approximated using neural networks?

To answer this question, there are three main points that are considered. Firstly, the type of data to be used in these neural networks is analyzed and the generation of such data is studied. Secondly, the type and structure of neural networks applicable to the problem are investigated. Finally, a comparison between different image metrics that can be used to evaluate the quality of the networks is made.

2 Theory

In this section, the physics behind depth of field are presented in the subsection *Camera optics and depth of field*. *Graphics rendering and post-processing* will elaborate on image rendering and the application of a post-processing effect on a rendered image. *Artificial neural networks* will touch on the theory behind the machine learning used in this work. *Image quality measures* will explain the mathematical models used to evaluate the quality and fidelity of the results from the machine learning model.

2.1 Camera optics and depth of field

In a pure mathematical setting, a camera is a tool that projects points in a 3D scene to a 2D plane (the image plane). The camera consists of two main elements that affect the way objects appear on the image plane: a *lens* that focuses light rays onto the image plane and an *aperture* that limits the number of light rays reaching the image plane. In figure 1, the basic structure of a camera is illustrated; light rays are emitted by the scene objects, pass through the aperture, and are focused by the lens to be projected onto the image plane.

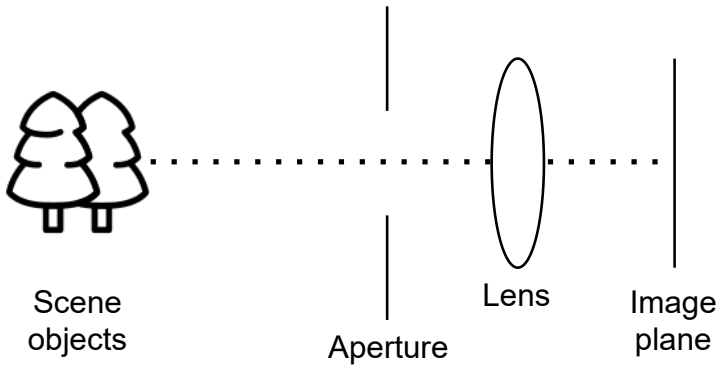
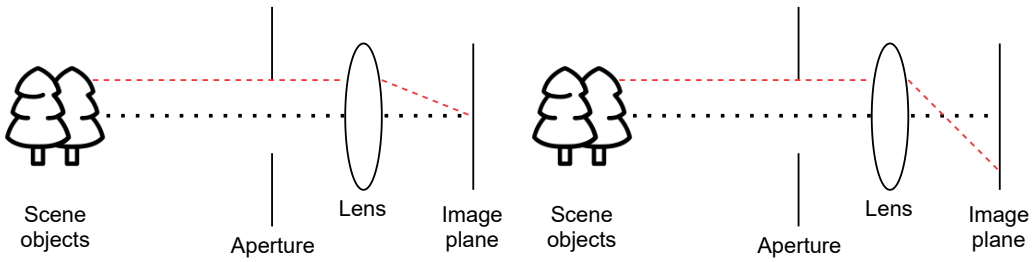


Figure 1: The main elements of a camera

2.1.1 Camera parameters

Different lenses focus light more or less strongly onto the image plane; to classify lenses, the *focal length* (a measure of distance) is used, where a lens with a shorter focal length bends light rays more sharply. This effect is illustrated in figure 2, where the focal length of the lens in figure 2a is longer than the lens in figure 2b.

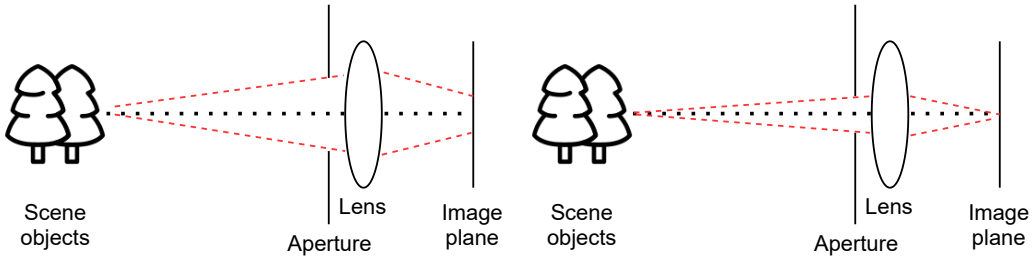


(a) A lens with a longer focal length (b) A lens with a shorter focal length

Figure 2: Comparing a light ray (in red) passing through two lenses with different focal lengths

The aperture of the camera is characterized by the size of the opening, and as most apertures are circular or near circular, the diameter of the circular opening is used to classify the aperture. In photography, the aperture is often written as a ratio dependent on a lens' focal length; for example, in a camera with a focal length $f = 50mm$ and an aperture diameter of $D = 25mm$, the aperture is denoted by the *f-number*: $N = f/D$. In this example, we obtain $N = 50mm/25mm = 2$, and the aperture would be denoted as $f/2$.

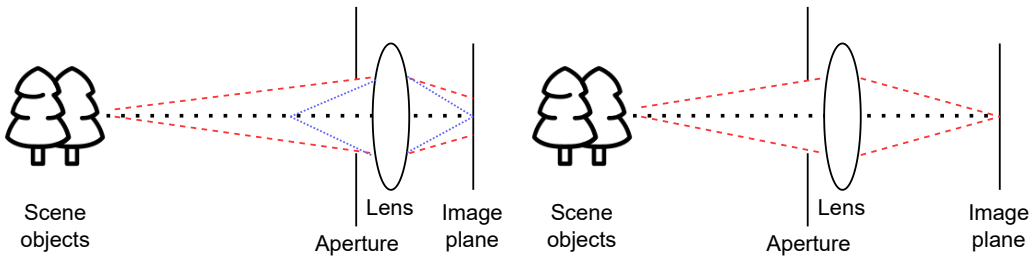
The aperture of a camera system influences how blurry out of focus points in the scene appear on the image plane. As light rays travel every possible path from a specific point among the scene objects, through the camera and onto the image plane, a smaller aperture leads to a smaller deviation in the path of different light rays, and therefore less blur. This is shown in figure 3, where the wide aperture in figure 3a leads to the scene objects appearing blurry as the light rays do not converge into one point on the image plane, while the scene object in figure 3b will not appear blurry as the light rays do converge into a single point.



(a) A scene captured with a wide aperture (b) A scene captured with a small aperture

Figure 3: Comparing light rays (in red) passing through different aperture sizes and the resulting perceived blur

It is also possible to adjust the depth at which scene objects appear in focus; this is called the *focus depth*, not to be confused with the focal length. By changing the distance between the lens and the image plane, as illustrated in figure 4, the distance at which diverging light rays are converged back into one point is changed. In figure 4a, this distance is short and as



(a) Short distance between the image plane and the lens.

(b) Longer distance between the image plane and the lens.

Figure 4: Adjusting the distance between the lens and the image plane changes the focus distance.

such the focus distance is short, illustrated by the light rays in blue, while the scene points appear blurry. In figure 4b, the lens is moved further away from the image plane and as such the light rays from the scene objects converge into one point, meaning they are in focus.

2.1.2 Circle of confusion

In figure 3, it is shown how objects not in focus appear blurry on the image plane. To measure how heavily blurred objects become, the *circle of confusion* is used: a point out of focus will appear as a circle on the image plane, and the circle of confusion is then defined as the diameter of this circle. Every point among the scene object thus has a corresponding circle of confusion, becoming smaller the more in focus said point is. It is worth noting that the circle of confusion does not correspond linearly with the distance from the point of focus; instead, it can be calculated as a function of distance through the formula [3]:

$$\text{ø}(s) \approx \frac{f^2}{N} \cdot \frac{|d_f - s|}{d_f \cdot s} \tag{1}$$

where ø is the circle of confusion, f is the focal length, N is the f-number, d_f is the focus distance and s is the distance to the point in question.

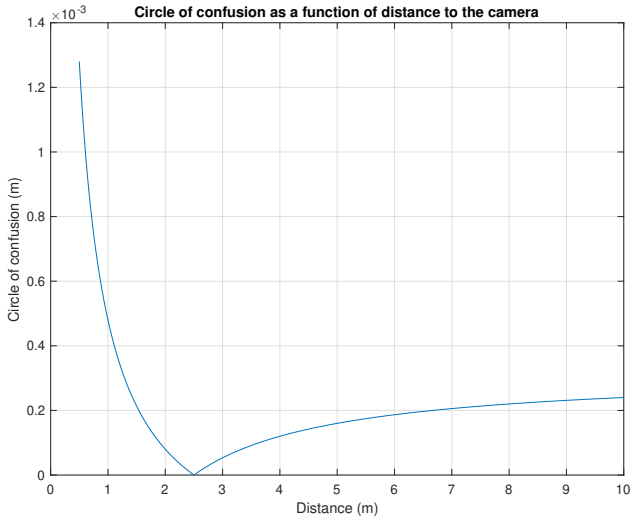


Figure 5: Circle of confusion as a function of distance

In figure 5, equation (1) is plotted with the parameters $f = 40$ mm, $N = 2$, and $d_f = 2.5$ m. This shows an important characteristic: *near-field* blur, caused by scene elements located before the focus depth is different from *far-field* blur, caused by scene elements located after the focus distance. In fact, the circle of confusion of objects in the near field tends to infinity, meaning that they become infinitely blurred as the distance to the camera approaches zero, while the circle of confusion of points in the far field tends towards the value $\frac{f^2}{N \cdot d_f}$ as the distance goes to infinity.

2.1.3 Depth of field

The concepts of focal range and aperture described in section 2.1.1, as well as circle of confusion in section 2.1.2 affect the depth of field in an image. An approximation for calculating the depth of field, meaning the distance between the nearest and furthest objects considered to be in focus, can be calculated as follows [4, p. 58]:

$$\text{DoF} \approx \frac{2u^2 N c}{f^2} \tag{2}$$

where u is the distance to the scene objects in focus, N is the f-number, f is the focal length and c is the minimum acceptable circle of confusion; c can be thought of as the largest diameter of a circle on the image plane that can still be considered as a point by an observer. The circle of confusion is therefore a subjective parameter that is not defined by the camera, but rather by the observer of the image.

Analyzing equation (2), the depth of field is proportional to the f-number, meaning that if the aperture is made wider, the depth of field is made more shallow. It is also observed that the effect is inversely proportional to the square of the focal length.

An example of depth of field can be seen in figure 6, where the face of the cat that is close to the camera appears in focus, while the tail of the cat or the background that is further away from the camera appears out of focus.

Figure 6: Depth of Field effect causes the background in the image to appear blurry

2.1.4 The pinhole camera

The pinhole camera is a basic camera model consisting of an infinitely small aperture and no lens. If an infinitely small aperture can be produced, only a single light ray per point among the scene objects is able to reach the image plane, leading to points at any distance from the camera being projected into single points on the image plane, meaning that all scene objects appear in focus, albeit inverted; this is illustrated in figure 7. Therefore, a pinhole camera does not exhibit a depth of field effect, which can also be seen through equation (2); as the aperture is infinitely small, the f-number N is infinite, leading to an infinite depth of field. In practice, a perfect pinhole camera is however impossible to build, partially since an infinitely small aperture is not achievable, but also because a small aperture does not let enough light pass through to create an image on a sensor. The pinhole camera is a useful model in computer graphics however thanks to its simplicity due to there being no lens needing to be modelled [5, p. 4-6].

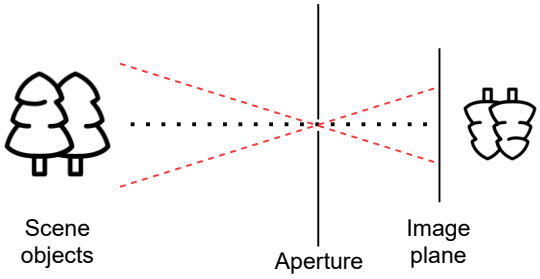


Figure 7: The pinhole camera model with example light rays being shown in red

2.2 Graphics rendering and post-processing

In a computer, there is a long flow of data from an abstract 3D model through the different computing elements to an image being presented, or *rendered* onto the screen. Depending on the context, which could be browsing a web page or playing a computer game, this flow differs; this section focuses on producing and outputting 3D content to a screen, such as in a video game.

2.2.1 Rendering

To generate an image to be rendered on a screen, data that represents the 3D scene goes through a *rendering pipeline* that processes the scene coordinates, projects them onto a 2D plane, applies the desired colors and any desired special effects.

In the context of computer games, where it is desired to produce a minimum of 30 images, often called *frames*, every second to create a smooth video experience, the rendering task is offloaded to a graphics processing unit (GPU), a device specifically designed to efficiently execute the mathematical operations involved in rendering scene objects. The result is the GPU producing a 2D image of the scene that can be displayed on the screen.

To render images, the GPU produces extra information about the scene objects. This is information like object depth or transparency, to make sure that objects behind a glass window are shown properly for example. When a 2D image is completed, this extra information is discarded as it is no longer needed, but it can be extracted if wanted, which is shown in figure 8.

Finally, the camera model used when rendering usually boils down to using a projection matrix to project 3D points onto a 2D plane [6]. This leads to the camera model behaving in a similar way to the pinhole camera in section 2.1.4, and therefore rendered images do not show any depth of field effect.

2.2.2 Post Processing

A rendered image lacks some effects seen in normal photographs, due to the simplified camera model. To increase the perceived image quality and increase the photo realism, these effects are reintroduced to a rendered image through mathematical approximations and algorithms. Examples of post processing effects are [7] ambient occlusion where areas

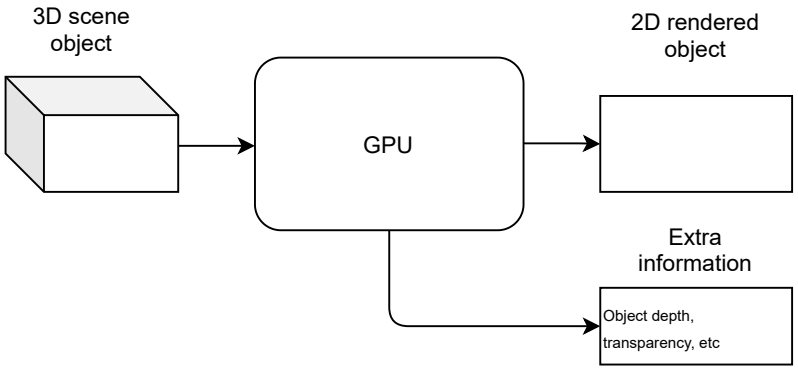


Figure 8: The GPU processes 3D scene data to produce 2D rendered objects

not directly exposed to ambient light such as corners are darkened, bloom that gives the appearance that light sources "glow", depth of field, and motion blur where objects in motion appear blurry.

When deciding on using post processing effects, it is necessary to determine the difficulty of creating said effect; while it can significantly improve the visual quality of an image, it also comes at the cost of computational power and excessive post processing can make a game feel "choppy".

2.2.3 Depth of Field as a post processing effect

As previously mentioned, depth of field is added to a rendered image as a post processing effect. The intensity of the effect for each pixel in the image is dependent on the camera parameters as well as the pixel depth, the latter of which can be obtained from the rendering pipeline. The depth of field algorithm can then be described as [8]:

1. Computing the circle of confusions of all pixels
2. Sample and blur each pixel using a 2D filter

In all, these steps involve multiple sampling operations per pixel, implying an increase to the overall rendering time of an image on the GPU. To achieve a photorealistic depth of field effect, it is necessary to blur each pixel at a strength relative to its circle of confusion, and there are no direct easy optimizations that can be made to the algorithm. Especially problematic is the near-field blur; observing figure 5, the circle of confusion increases rapidly the closer to the camera an object is, leading to a singular unblurred pixel affecting a big part of the image which could take a long time to calculate (compared to the whole depth of field algorithm). To prevent this a maximum circle of confusion limit can be applied, preventing this extreme blurring, at the cost of some visual realism.

Computing the circle of confusion

Recall that the circle of confusion is the diameter of the circle that a point in the 3D scene appears as on the 2D image plane. By using the depth information from the GPU, the circle of confusion can be computed through a camera model using supplied camera parameters of focal length and f-number, using equation 1. This results in an extra image channel where pixels with depth closer to the focus distance have a smaller circle of confusion.

Sampling, blurring and reduction of artifacts

To emulate the blur of a camera, a 2D circular filter with evenly spaced sampling points is used, as shown in figure 9. For each point in the 2D filter, the circle of confusion is extracted. If the circle of confusion of the sample point is large enough to overlap the current pixel, then an alpha value for the sample is computed as $\alpha_s = \frac{4}{\pi c^2}$, where c is the circle of confusion diameter, otherwise the sample point is not used as the object in this sample should not be blurred into the current pixel. This is illustrated in figure 10 Here, it is also necessary to check if the sample is in the foreground compared to the pixel; if an object is partially occluded by another, the object in the background should not be able to blur onto the other one where it is occluded. The computed alpha values are used to blend all samples together: samples with higher circle of confusion, corresponding to larger blur, have lower alpha values and less impact on the final color of the pixel. With the sampling step complete, there might be artifacts in the resulting image such as noise or repeating circular patterns instead of smooth blur. This arises from parts of the scene that are undersampled, which can be mitigated by running a denoising path on the blurred image.

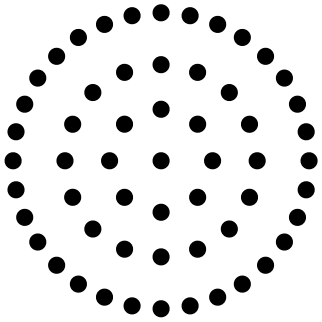


Figure 9: Example of a 2D filter with 49 sampling points that could be used for depth of field.

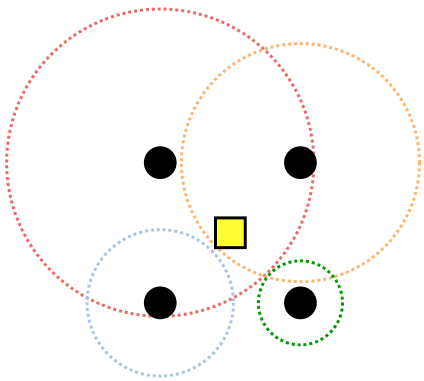


Figure 10: Usage of a simple 2D filter for depth of field. The circle of confusion in the top two sampling points cover the current pixel in yellow, and their color values are therefore blended onto the current pixel.

2.3 Artificial neural networks

An artificial neural network is a mathematical model, loosely resembling the structure of neurons in a brain. The model consists of many artificial neurons, organized into layers, where data flows from one layer to the next. The first known such model was proposed by Warren McCulloch and Walter Pitts in 1943 [9], and has since seen a lot of diverse research and applications. Neural networks are a subclass of machine learning algorithms, meaning that the model can compare a prediction it made on a certain input, and by comparing this to an expected prediction make changes to the model based on some metric.

2.3.1 Fully connected network layers

In the simplest form, a neural network is simply a discrete function that from some input x_n of size n, produces an output y_m of size m; it can be denoted $f : x_n \to y_m$. Writing x_n and y_m as column vectors, a simple *fully connected* neural network layer can be written in matrix form as $\boldsymbol{y_m} = \boldsymbol{W} \cdot \boldsymbol{x_n} + \boldsymbol{b}$, or

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & \ddots & & \\ \vdots & & & \\ w_{m1} & & & w_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \tag{3}$$

The matrix $\boldsymbol{W}$ contains the *weights* of the layer, and the matrix $\boldsymbol{b}$ contains the *biases* of the layer.

A simple full neural network can be created by chaining together a series of such matrix multiplications, and the network can then be written as

$$\boldsymbol{y} = \boldsymbol{W_n}(\boldsymbol{W_{n-1}}(\boldsymbol{W_{n-2}}(...(\boldsymbol{W_0} \cdot \boldsymbol{x} + \boldsymbol{b_0})) + \boldsymbol{b_{n-2}}) + \boldsymbol{b_{n-1}}) + \boldsymbol{b_n} \tag{4}$$

Figure 11 shows an example of a fully connected network with three layers, containing three, four and two neurons each respectively.

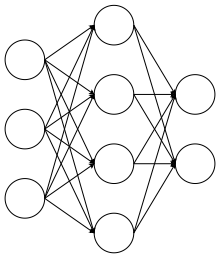


Figure 11: A simple fully connected neural network

2.3.2 Convolutional layers

A convolutional layer is, as the name implies, a layer in the neural network that performs a convolution operation on some input data. Convolutional neural networks (CNN), containing convolutional layers, see a wide range of applications in image processing and image classification. The convolutional operation can be applied to data of any input shape (1D, 2D, 3D), however with the use of images as input data, this section implies 2-dimensional convolutional layers.

Many well-known established neural network models consist partially or fully of convolutional layers, such as the "VGG" models for image recognition, or the "SRCNN" model for image super resolution [10][11]. It has been shown that convolutional neural networks are able to be trained to learn the different features of an image, for example edges, leading to a vast range of computer vision applications [12][13].

The layer consists of a kernel K that convolve on the input I to create an output O:

$$O = (I * K) \tag{5}$$

or in discrete form

$$O(i,j) = \sum_m \sum_n I(m,n) \cdot K(i-m, j-n) \tag{6}$$

In contrast from the fully connected layer, the number of parameters in a convolutional layer is limited to those in the kernel as opposed to a full matrix. The kernel can be chosen to be of any size and is reused over the whole input, which leads to lower memory consumption as less parameters need to be stored. Figure 12 shows an example of a convolutional operation.

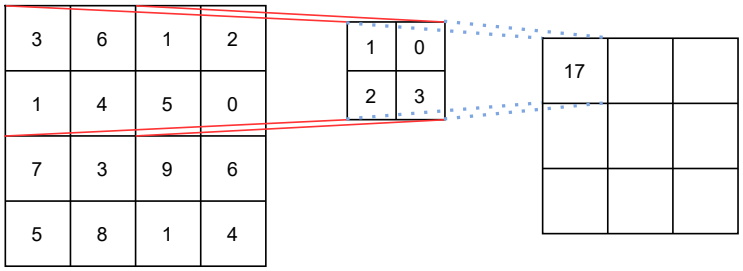


Figure 12: A 2x2 kernel applied on a 4x4 input to create a 3x3 output

Pooling

In the convolutional network, one wants to gradually reduce the size of the data flowing through the network; for example, in a model designed to accept images of size 200×200 containing written digits, with 10 outputs corresponding to digits 0 through 9, the input might be gradually reduced to a size of 100×100, then 50×50. To achieve this, pooling operations are used, where the input is divided into small sections, and one value is extracted from each section. This value could be calculated in any way, but most often a maximum pooling is used, as shown in figure 13 where a 2×2 max-pooling operation is used to extract the maximum value of each section and create an output of quarter the size of the input.

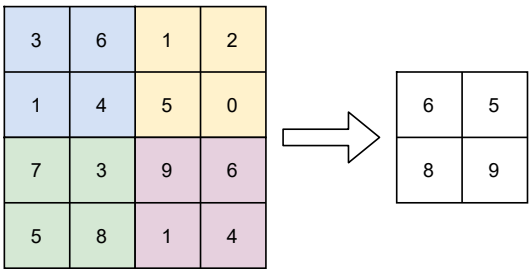


Figure 13: A maximum pooling 2x2 operation on a 4x4 input, generating a 2x2 output.

Activation functions

An activation function is a mathematical mapping altering the output of a neural network layer. One use of activation functions is to rescale an output, for example by using the function $\phi(x) = \frac{1}{1+e^{-x}}$ which maps the range $[-\infty, \infty]$ to $[0, 1]$.

A common activation function for convolutional neural network layers is the rectified linear unit (ReLU). The function is computed as follows:

$$\text{ReLU}(x) = \max(0, x) \tag{7}$$

, in short returning the input if it is positive, otherwise 0. The function was proposed to be used in neural networks by Glorot et al. in 2011, where it was shown to improve the training performance of some networks.

As discussed in section 2.3.3, the derivative of all functions in a neural network is a key parameter to the training process. This poses a problem for rectified linear units, as the derivative does not exist when $x = 0$. This is addressed by explicitly defining the derivative at $x = 0$ to be either 0 or 1.

2.3.3 Loss functions and backpropagation

The loss function of the neural network model is an abstraction for evaluating the performance of the network, given an input, a predicted output and a corresponding expected output. For use in neural networks, the loss function is any differentiable function C that maps the predicted output y and expected output t to a single number.

Given a loss function, the neural network weights are updated using the derivative of the loss function on the principle of gradient descent; the model is optimized to find a minima in the loss function. In the simplest case of a fully connected network layer, the new weights w'_{ij} are obtained by subtracting the loss function derivative from the old weights w_{ij}.:

$$w'_{ij} = w_{ij} - \gamma \frac{\partial C(y, t)}{\partial w_{ij}} \tag{8}$$

Here, γ is a parameter called the *learning rate*, a small value that dictates the speed at which the weights are adjusted. The loss function derivative in equation 8 is complicated to

calculate as is, but can be split up using the chain rule in a process called backpropagation:

$$w'_{ij} = w_{ij} - \gamma \frac{\partial C(y,t)}{\partial y} \cdot \frac{\partial y}{\partial w_{ij}} \tag{9}$$

The derivative $\frac{\partial C(y,t)}{\partial y}$ is easy to compute as the loss function C is known in advance. The derivative $\frac{\partial y}{\partial w_{ij}}$ can be calculated once again using the chain rule: if the neural network is described as in equation (4), it is split up as

$$\frac{\partial y}{\partial w_{ij}} = \frac{\partial y_n}{\partial y_{n-1}} \cdot \frac{\partial y_{n-1}}{\partial w_{ij}} \tag{10}$$

where y_k is the output of layer k. Equation (10) is then repeated recursively through the entire model, hence the term backpropagation. These concepts equally apply to convolutional layers, where the only difference compared to fully connected layers is how the derivative $\frac{\partial y_n}{\partial y_{n-1}}$ is computed.

2.3.4 Training of neural networks

Using repeated backpropagation, a neural network will approach a minima of the loss function. It is important to remember that the loss function is only evaluated on the training data and that the quality of predictions on the training data does not have to be representative of the quality of prediction on other testing data. For example, a convolutional neural network designed to identify pneumonia in x-ray pictures instead learned to identify which physical machine in a hospital was used to take the x-ray, as patients with pneumonia are often scanned on the same x-ray machine [14]. The model will always find the most optimal way to minimize the loss function, and it is therefore imperative to consider all possible correlations in the dataset.

Two other common issues in training neural networks are *underfitting* and *overfitting* [15]. Underfitting occurs when the network cannot reach a low enough error on the training dataset. This could be due to the network not being trained enough to properly learn the characteristics of the data, or that the model does not have enough complexity and therefore lacks the capacity to fully learn what is intended. Overfitting on the other hand implies that the network has learned too much; it performs well on the training dataset but fails to perform on a testing dataset. This usually occurs with a complex model that is trained for a too long period of time, and thus learns all the desired characteristics of the dataset but also extrapolates more characteristics in the training dataset that are not desirable.

Finding the right balance between model complexity and prediction error on the test data while avoiding underfitting and overfitting is a long process, often including some trial and error. To help prevent overfitting, it is possible to randomly alter the input images by means of rotating, shifting, or zooming the input images, diversifying the training data. It has also been shown that including so called "dropout" layers that with a set probability remove some of the input data to the layer helps in reducing overfitting [16].

2.3.5 Skip connections and U-nets

Most commonly, neural networks are sequential, meaning that all data from one layer flows into the next until the output, in sequence. There is however no restriction on how information must flow through the model; for example, some network structures such as recurrent neural networks use inputs from previous iterations to be able to analyze the context of a word in a sentence, or to predict the next item in a sequence given a series of inputs [17].

In this thesis work, the idea of skip connections is central. The basic principle relies on branching the path of network operations into two different branches, where different operations are performed, to then combine these two paths together to form an output. Figure 14 illustrates this principle. The input data is fed into two different operations; these

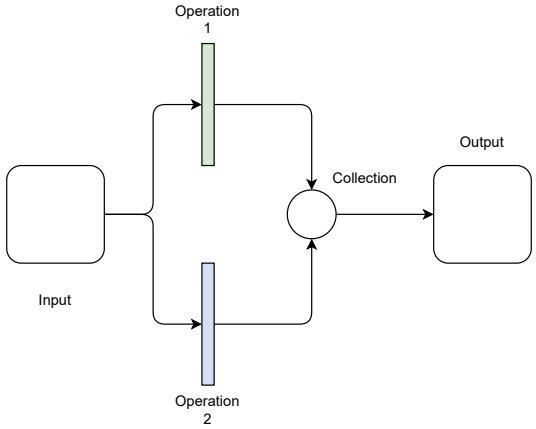


Figure 14: Branching logic in a neural network

can for example be different types of convolutional layers. The two paths are then merged with some operation, this can be an addition or concatenation for instance. From this collection, the output is obtained. A skip connection is obtained when one of the operations in figure 14 is not implemented; on one branch the input is processed through a series of neural network layers, and on the other branch the input "skips" over all the calculations before being combined with the processed input. As presented in section 2.3.4, a more complex model is harder to train and is more prone to overfitting, however the complexity of the network might be needed for it to perform well on the dataset. Introducing skip connections in a model has been proven to reduce the difficulty of training, and can lead to better performance [18].

U-nets

The u-net is a neural network architecture proposed by Ronneberger et. al in 2015, intended for image segmentation in the field of biomedicine [19]. It is constructed by a sequence consisting of downsampling operations, followed by convolution layers, the two repeated multiple times. The output from the convolutional layers is connected via skip connections to an upsampling of the downsampled and then convolved data, where another convolutional

layer is applied. The result of the series of downsample/upsample operations is a structure in the shape of an "U", hence the name. In figure 15, the structure of a u-net with two layers of depth is shown; it is however possible to design the network with any amount of depth.

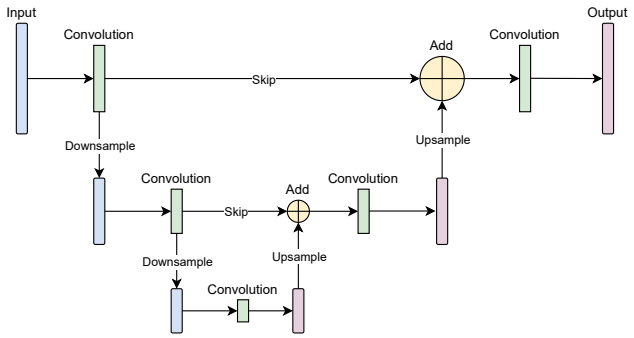


Figure 15: An example of a u-net with two layers of depth, using addition to recombine the skip connections

The downsampling operations remove information from the input data, and this is countered by using a large amount of feature channels, originally proposed to double for each downsample. In contrary, the amount of feature channels is halved with the convolution associated to each upsample. The result is a network architecture with high performance for image segmentation that requires a relatively low amount of data to train on [19].

Backpropagation with skip connections

The method of backpropagation through a skip connection is dependent on the type of collection used. In the case of addition, the operation can be written simply as

$$\boldsymbol{y} = \boldsymbol{y}_{\text{skip}} + \boldsymbol{y}_{\text{upsample}} \tag{11}$$

where $\boldsymbol{y}_{\text{skip}}$ is the contribution from the skip connection and $\boldsymbol{y}_{\text{upsample}}$ is the contribution from the upsampling operation. Relating this to the general formula for backpropagation presented in equation 10, one obtains

$$\frac{\partial y}{\partial w_{ij}} = \frac{\partial y_{\text{skip}}}{\partial w_{ij}} + \frac{\partial y_{\text{upsample}}}{\partial w_{ij}} \tag{12}$$

Backpropagation through skip connections is therefore extremely simple to compute.

2.4 Image quality measures

To evaluate the quality of a generated image compared to an expected image, image quality measures need to be used. By taking two images, one generated by a neural network and the other the predicted image, it is possible to quantify the difference between the two images as a single number, which can then be used as a loss function for a neural network.

Different quality measures perform different calculations to estimate the total error in the predicted image, and as such training a model with different image metrics will lead to differing results. It has also been suggested that simpler metrics like PSNR do not correlate well with subjective assessments of image quality, and that other algorithms are preferable to use for evaluating image quality [20][21].

2.4.1 MSE - Mean squared error

The mean squared error between an expected image I and a predicted image P is defined as

$$MSE = \sum_{i=0}^{m-1} (\sum_{j=0}^{n-1} [I_{ij} - P_{ij}]^2) \cdot \frac{1}{mn} \tag{13}$$

If the two images are identical, the mean squared error is zero. It has the advantage of being easy to compute, in terms of implementation and computational power.

2.4.2 PSNR - Peak signal to noise ratio

The PSNR of two images is closely linked to the mean squared error of two images, but also incorporates the maximum possible value in the image. It is defined as

$$PSNR = 10 \cdot \log_{10} \left(\frac{D^2}{MSE} \right) \tag{14}$$

where MSE is the mean squared error computed through equation (13) and D is the maximal possible value of the image; usually 255 if the image is in RGB format. A lower PSNR score corresponds to a lower image quality.

As the PSNR computes the logarithm of the inverted error, it is more representative of smaller variations than the mean squared error. For example, with $D = 1$, consider the two errors of $MSE_1 = 0.0001$ and $MSE_2 = 0.0002$. Computing the PSNR of these two errors, one obtains:

$$PSNR_1 = 10 \cdot \log_{10} \left(\frac{1}{0.0001} \right) = 40 \tag{15}$$

$$PSNR_2 = 10 \cdot \log_{10} \left(\frac{1}{0.0002} \right) \approx 37 \tag{16}$$

A small difference in mean squared error is thus much more pronounced in PSNR.

2.4.3 SSIM - Structural similarity index measure

SSIM was proposed in 2004 by Wang et al. as a new image quality metric, based on the assumption that subjective image quality is perceived not through the error of singular pixels but instead through the similarity of different structures in the image [22].

Three different characteristics, luminance (l), contrast (c) and structure (s), are computed over a small window of the image (typically 11×11) to get a local structural similarity. By applying this window to every pixel on the image, it is possible to create an average structural similarity of the entire image.

Given two windows $\boldsymbol{x}$ and $\boldsymbol{y}$ from two images, first the means (μ_x, μ_y), standard deviations (σ_x, σ_y) as well as the covariance of both windows are computed:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad \mu_y = \frac{1}{N}\sum_{i=1}^{N} y_i, \tag{17}$$

$$\sigma_x = \left(\frac{1}{N-1}\left(\sum_{i=1}^{N}(x_i - \mu_x)\right)\right)^{1/2}, \quad \sigma_y = \left(\frac{1}{N-1}\left(\sum_{i=1}^{N}(y_i - \mu_y)\right)\right)^{1/2} \tag{18}$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{19}$$

From this, the three characteristics can be computed.
Luminance:

$$l(\boldsymbol{x}, \boldsymbol{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{20}$$

Contrast:

$$c(\boldsymbol{x}, \boldsymbol{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{21}$$

Structure:

$$s(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{22}$$

The structural similarity is finally computed as

$$SSIM = l(\boldsymbol{x}, \boldsymbol{y})^{\alpha} \cdot c(\boldsymbol{x}, \boldsymbol{y})^{\beta} \cdot s(\boldsymbol{x}, \boldsymbol{y})^{\gamma} \tag{23}$$

The constants C_1, C_2, C_3 are given as

$$C_1 = (k_1 L)^2, \quad C_2 = (k_2 L)^2, \quad C_3 = C_2/2 \tag{24}$$

where L is the maximum value of a pixel (usually 255), $k_1 = 0.01$ and $k_2 = 0.03$ usually, and with $\alpha = \beta = \gamma = 1$, the formula becomes

$$SSIM = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{25}$$

When different types of noisy images are compared to a single noise-free image, the structural similarity index measurement can grade these differently while the PSNR score is identical, as proven by Wang et. al [22]. This implies that SSIM is more versatile at identifying different kinds of image noise or artifacts, and that it is better suited as a loss function for an image generating neural network.

2.4.4 Delta E*

Delta E is a metric for comparing the color accuracy between two images. It uses the images represented in the three-channel $L^*a^*b^*$ (CIELAB) color space, and is defined as the Euclidean distance between the points in the two images [23]:

$$\Delta E^* = \sqrt{(L_2^* - L_1^*)^2 + (a_2^* - a_1^*)^2 + (b_2^* - b_1^*)^2} \tag{26}$$

The CIELAB color space has the advantage of being defined by three channels more closely related to human color perception than RGB color: L^* represents luminance while a^* and b^* represent the opposite color pairs red-green and blue-yellow respectively.

Converting from RGB color to CIELAB color

To convert an image from RGB color space to $L^*a^*b^*$, first the image is transformed into a third color space, CIEXYZ, by following the following steps [24]:

A gamma correction is applied to the (R,G,B) channels separately:

$$[R', G', B'] = \begin{cases} [R, G, B]/12.92 & \text{if } [R, G, B] < 0.04045 \\ \frac{[R,G,B]+0.055}{1.055}^{2.4} & \text{else} \end{cases} \tag{27}$$

and then the conversion is done through a matrix multiplication

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \cdot \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} \tag{28}$$

From the XYZ values, the $L^*a^*b^*$ colors can be calculated as follows [23]:

First, divide the XYZ values by corresponding constants for RGB images:

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} \frac{X}{95.047} \\ \frac{Y}{100} \\ \frac{Z}{108.883} \end{bmatrix} \tag{29}$$

Then the $L^*a^*b^*$ colors are computed as

$$L^* = 116f(Y') - 16 \tag{30}$$
$$a^* = 500[f(X') - f(Y')] \tag{31}$$
$$b^* = 200[f(Y') - f(Z')] \tag{32}$$

where the function f is defined as

$$f(x) = \begin{cases} x^{1/3} & \text{if } x > (24/116)^3 \\ \frac{841x}{108} + \frac{16}{116} & \text{else} \end{cases} \tag{33}$$

Due to the $L^*a^*b^*$ color channels being more closely related to human color perception than RGB color, the Delta E is a good measure to evaluate the color accuracy of a generated image, given a reference; a Delta E* of 2.3 or higher implies a noticeable color difference [25]. Due to the complicated conversion from the RGB format, Delta E* is not suitable as a loss function; it can however be useful as an evaluation metric.

3 Method of work

The method of work was split up in three main categories according to the problem description in section 1.1. The section *data generation* elaborates on the process of creating a depth-of-field dataset, the section *network creation* explains the work and mindset behind the creation of the neural network models, and the section *model training* describes the work on loss functions as well as evaluating the model. Then, some alterations that were made to the model structures or training process are enumerated in the section *other analysis*. Lastly, the main delimitations made during the thesis work are listed in the section *delimitations*.

Three different models were created and to help differentiate between them, the codenames *M1*, *M2.1* and *M2.2* are given to each model.

3.1 Data generation

The training data used was developed over the whole period of thesis work, to fulfill the evolving requirements. At the beginning of the project, only a few images from a single scene with a low amount of blur were needed as a feasibility check. This evolved over time to a dataset with images from two different scenes, where the images contain both near field and far field blur.

For an image to be used in the model, the following three data elements are required:

- The *ground truth*: The image with depth of field post processing applied. This image has three channels for the red, green, and blue colors.
- The *color*: The image just before depth of field post processing is applied. This image also has three channels for red, green, and blue colors.
- The *depth map*: For each pixel in the image, a value between 0 and 1 is assigned, where pixels representing objects closer to the camera are assigned values closer to 1. The depth map contains just a single channel.

All used images were generated at a resolution of 1920×1080. This has the disadvantage of requiring more computational power than smaller images but allows to see more clearly the effects of varying depth in one single image. The final dataset consists of 181 unique generated images at the target resolution of 1920×1080 pixels with depth maps. From these, 158 are used as training data and 23 are reserved as testing data for evaluation.

3.1.1 Unity game engine

Unity is a modern game engine, providing tools for a developer to create games more easily for multiple platforms, such as computers and mobile phones [26]. It is highly customizable and allows for quick implementations of post processing effects, including depth of field. The Unity editor can be seen in figure 16.

Using scripts, it is possible to configure Unity to export image data before and after depth of field post processing has taken place, as well as the corresponding depth map. The

datasets used in the thesis work were generated by loading different scenes into Unity and exporting the needed data elements.

Figure 16: The Unity game engine editor window

3.1.2 Camera parameters

As shown in section 2.1.1, the aperture and the focal length of a camera are the main parameters for the depth of field in an image. To illustrate the most general type of depth of field, it was of interest to generate images that contain near-field blur close to the camera, far-field blur at a long distance from the camera and an area in focus at a medium distance from the camera. With some tuning, the internal camera was set to have a focal length of $f = 40$, a f-number of $f/1.4$ and a focus distance of $d_f = 2.5$.

While the focal length and focus distance are measures of distance, Unity does not provide a unit for these numbers. Figure 17 presents the same scene generated with differing focus distance: figure 17a has camera parameters set as above and shows both near-field and far-field blur, figure 17b shows only near-field blurring and 17c shows only far-field blurring.

In a more general setting, it could be possible to add the camera parameters as an input to the model and thereby obtain a fully generalized model for depth of field. This was opted against however, as this would probably have required a large amount of data with varying camera parameters, that would be unfeasible to generate in the time frame of the thesis work.

(a) Image with both near and far field blur

(b) Image with only near field blur

(c) Image with only far field blur

Figure 17: Different camera parameters give different depth of field

3.1.3 Data capture

Two different scenes were used for generating data, see figures 18 and 19 for examples of images from each scene. These scenes contain a relatively small number of objects each and using both scenes will help against the models overfitting on the elements on one scene. The captured images are in the RGB color format, however these are transformed into another color format, YUV, before use in the model. This color format consists of a luminance channel, Y, and two color channels, U and V. The conversion between RGB and YUV format can be simplified to a matrix multiplication [27]:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{34}$$

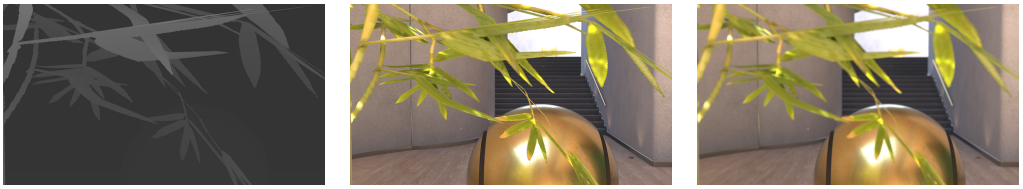
This 3×3 conversion matrix is invertible, so the operation of transforming YUV data into RGB data is also a matrix multiplication, using the inverse of the 3×3 matrix above.

Example data

Figure 18 shows one image of the dataset from the first scene, a small village with two houses and some rocks and figure 19 shows one image of the dataset from the second scene, an indoor room with palm trees and some golden spheres.

(a) Depth data (b) Color data (c) Ground truth

Figure 18: Example data from the first scene



(a) Depth data (b) Color data (c) Ground truth

Figure 19: Example data from the second scene

3.2 Network design

Three different neural networks were implemented, all based on the principle of u-nets described in section 2.3.5. One of these models had a depth of one, meaning one downsample/upsample pair, while the two others had a depth of two, implying two pairs of downsample/upsample operations. At the core of all three models is a "double convolution", consisting of a convolutional layer, followed by a rectified linear unit (ReLU; this layer maps negative inputs to 0 and positive inputs to themselves), followed by another convolutional layer and finally another ReLU.

Every model accepts an input of 4 channels, where the 3 first channels contain the image data, and the final channel is the image depth map. The networks have 3 output channels to produce a blurred output image.

Model M1: one upsampling/downsampling pair

The first model to be implemented in the thesis work was the simplest one in terms of structure and parameters. The input, consisting of 4 channels, is convolved into 32 channels, and then a maximum pooling downsampling operation is used to reduce the input resolution by 1/4. A total of three convolutional operations are applied on the downsampled data, increasing the total amount of channels to 64. This is then upsampled through bilinear interpolation back to the input resolution, where these layers are concatenated with a skip connection to the input. Finally, the data is reduced from 96 channels to 3 output channels, through 3 different convolutional layers. A diagram of the model can be seen in figure 20, and the model parameters can be seen in table 1.

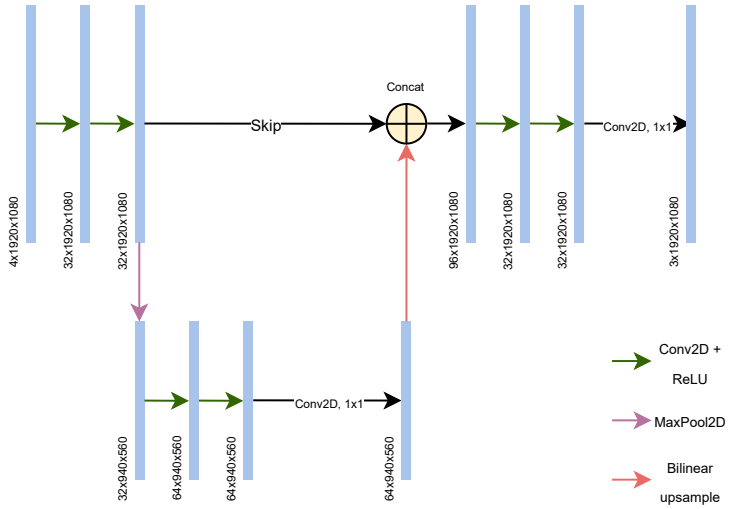


Figure 20: Diagram of the model M.1.

Table 1: Model M1 with depth 1

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
DoubleConv: 1-1                          [1, 32, 1920, 1080]       --
|    Sequential: 2-1                     [1, 32, 1920, 1080]       --
|    |    Conv2d: 3-1                    [1, 32, 1920, 1080]       1,184
|    |    ReLU: 3-2                      [1, 32, 1920, 1080]       --
|    Sequential: 2-2                     [1, 32, 1920, 1080]       --
|    |    Conv2d: 3-3                    [1, 32, 1920, 1080]       9,248
|    |    ReLU: 3-4                      [1, 32, 1920, 1080]       --
|-UnetDown: 1-2                          [1, 64, 960, 540]         --
|    Sequential: 2-3                     [1, 64, 960, 540]         --
|    |    MaxPool2d: 3-5                 [1, 32, 960, 540]         --
|    |    DoubleConv: 3-6                [1, 64, 960, 540]         55,424
|    Conv2d: 1-4                         [1, 64, 960, 540]         4,160
+----UnetUp: 1-6.                        [1, 32, 1920, 1080]       --
|        Upsample: 2-7                   [1, 64, 1920, 1080]       --
DoubleConv: 2-8                          [1, 32, 1920, 1080]       --
|    Sequential: 3-11                    [1, 32, 1920, 1080]       27,680
|    Sequential: 3-12                    [1, 32, 1920, 1080]       9,248
UnetOut: 1-7                             [1, 3, 1920, 1080]        --
|    Sequential: 2-9                     [1, 3, 1920, 1080]        --
|    |    Conv2d: 3-13                   [1, 3, 1920, 1080]        99
==========================================================================
Total params: 107,043
Trainable params: 107,043
Non-trainable params: 0
```

Model M2.1: two upsampling/downsampling pairs, maximum of 128 channels

The second model to be implemented builds on the first model, by having the same structure for the first upsampling/downsampling pair. After the first downsampling operation, the input is downsampled again to effectively 1/16 of the input resolution, however with a total of 128 channels. This data is then upsampled and convolved twice, each time with a skip connection from the corresponding input just before downsampling. There is once again a series of 3 convolutional layers that reduce the upsampled output down to 3 output channels. The model parameters can be seen in table 2, and a diagram of the model can be seen in figure 21.

Comparing the model with depth 1 and this model with depth 2, the first model contains 107403 parameters while this one contains 488419 parameters; adding one layer of depth to the network increased the complexity by almost a factor of 5.

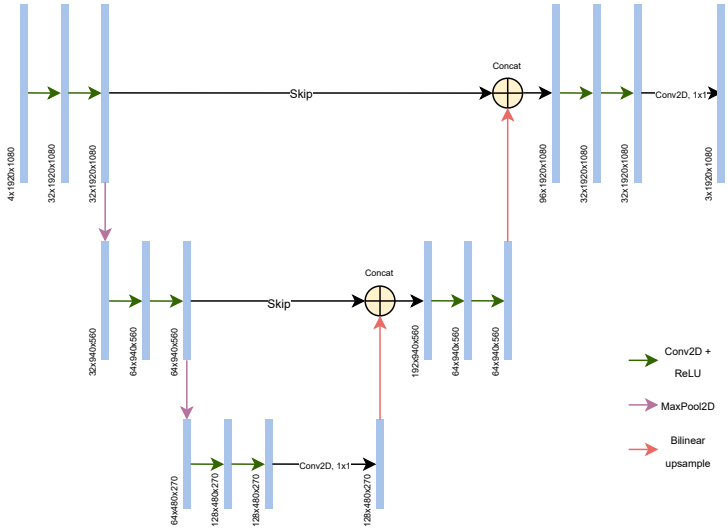


Figure 21: Diagram of the model M2.1.

Table 2: Model M2.1 with depth 2, 128 channels maximum

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
DoubleConv: 1-1                          [1, 32, 1920, 1080]       --
|    Sequential: 2-1                     [1, 32, 1920, 1080]       --
|    |    Conv2d: 3-1                    [1, 32, 1920, 1080]       1,184
|    |    ReLU: 3-2                      [1, 32, 1920, 1080]       --
|    Sequential: 2-2                     [1, 32, 1920, 1080]       --
|    |    Conv2d: 3-3                    [1, 32, 1920, 1080]       9,248
|    |    ReLU: 3-4                      [1, 32, 1920, 1080]       --
|-UnetDown: 1-2                          [1, 64, 960, 540]         --
|    Sequential: 2-3                     [1, 64, 960, 540]         --
|    |    MaxPool2d: 3-5                 [1, 32, 960, 540]         --
|    |    DoubleConv: 3-6                [1, 64, 960, 540]         55,424
|    |-UnetDown: 1-3                     [1, 128, 480, 270]        --
|    |    Sequential: 2-4                [1, 128, 480, 270]        --
|    |    |     MaxPool2d: 3-7           [1, 64, 480, 270]         --
|    |    |     DoubleConv: 3-8          [1, 128, 480, 270]        221,440
|    |    Conv2d: 1-4                    [1, 128, 480, 270]        16,512
|    +----UnetUp: 1-5                    [1, 64, 960, 540]         --
|    |          Upsample: 2-5            [1, 128, 960, 540]        --
|    DoubleConv: 2-6                     [1, 64, 960, 540]         --
|    |    Sequential: 3-9                [1, 64, 960, 540]         110,656
|    |    Sequential: 3-10               [1, 64, 960, 540]         36,928
+----UnetUp: 1-6.                        [1, 32, 1920, 1080]       --
|        Upsample: 2-7                   [1, 64, 1920, 1080]       --
DoubleConv: 2-8                          [1, 32, 1920, 1080]       --
|    Sequential: 3-11                    [1, 32, 1920, 1080]       27,680
|    Sequential: 3-12                    [1, 32, 1920, 1080]       9,248
UnetOut: 1-7                             [1, 3, 1920, 1080]        --
|    Sequential: 2-9                     [1, 3, 1920, 1080]        --
|    |    Conv2d: 3-13                   [1, 3, 1920, 1080]        99
==========================================================================
Total params: 488,419
Trainable params: 488,419
Non-trainable params: 0
```

Model M2.2: upsampling/downsampling pairs, maximum of 96 channels

The third and final model to be implemented copies the structure of the second model, with the only difference being that the amount of input and output channels to each convolutional layer has been multiplied by a factor of 3/4, except for the input and output layer. This reduces the total amount of parameters from 488419 to 275115; for instance, the maximum number of channels in the network is reduced from 128 to 96, while the image resolution and network structure is kept the same. A diagram of the model is shown in figure 22, and all the model parameters can be seen in table 3.

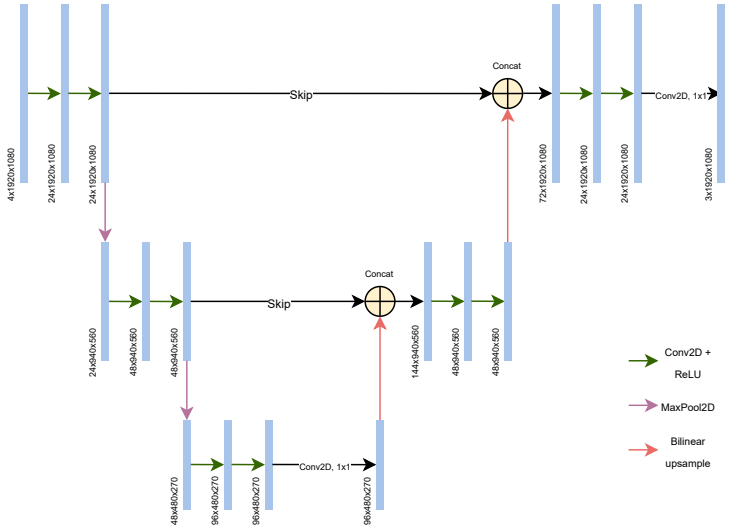


Figure 22: Diagram of the model M2.2.

Table 3: Model M2.2 with depth 2, 96 channels maximum

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
DoubleConv: 1-1                          [1, 24, 1920, 1080]       --
|    Sequential: 2-1                     [1, 24, 1920, 1080]       --
|    |    Conv2d: 3-1                    [1, 24, 1920, 1080]       888
|    |    ReLU: 3-2                      [1, 24, 1920, 1080]       --
|    Sequential: 2-2                     [1, 24, 1920, 1080]       --
|    |    Conv2d: 3-3                    [1, 24, 1920, 1080]       5,208
|    |    ReLU: 3-4                      [1, 24, 1920, 1080]       --
|-UnetDown: 1-2                          [1, 48, 960, 540]         --
|    Sequential: 2-3                     [1, 48, 960, 540]         --
|    |    MaxPool2d: 3-5                 [1, 24, 960, 540]         --
|    |    DoubleConv: 3-6                [1, 48, 960, 540]         31,200
|    |-UnetDown: 1-3                     [1, 96, 480, 270]         --
|    |    Sequential: 2-4                [1, 96, 480, 270]         --
|    |    |     MaxPool2d: 3-7           [1, 48, 480, 270]         --
|    |    |     DoubleConv: 3-8          [1, 96, 480, 270]         124,608
|    |    Conv2d: 1-4                    [1, 96, 480, 270]         9,312
|    +----UnetUp: 1-5                    [1, 48, 960, 540]         --
|    |          Upsample: 2-5            [1, 96, 960, 540]         --
|    DoubleConv: 2-6                     [1, 48, 960, 540]         --
|    |    Sequential: 3-9                [1, 48, 960, 540]         62,256
|    |    Sequential: 3-10               [1, 48, 960, 540]         20,784
+----UnetUp: 1-6.                        [1, 24, 1920, 1080]       --
|        Upsample: 2-7                   [1, 48, 1920, 1080]       --
DoubleConv: 2-8                          [1, 24, 1920, 1080]       --
|    Sequential: 3-11                    [1, 24, 1920, 1080]       15,576
|    Sequential: 3-12                    [1, 24, 1920, 1080]       5,208
UnetOut: 1-7                             [1, 3, 1920, 1080]        --
|    Sequential: 2-9                     [1, 3, 1920, 1080]        --
|    |    Conv2d: 3-13                   [1, 3, 1920, 1080]        75
==========================================================================
Total params: 275,115
Trainable params: 275,115
Non-trainable params: 0
```

3.3 Model training

The neural networks as well as code for calculating the image metrics, training the models, and importing images are implemented using *PyTorch* (version 1.7.1), a machine learning framework [28]. Training was done on an AWS cloud computer [29], containing an NVIDIA Tesla M60 GPU on which the neural network code was run.

Training passes

All images are stored as `.exr` files in a high dynamic range format, at a resolution of 1920×1080. In the scope of this work, the data is first clamped to standard dynamic range, leading to input values in the range $[0, 1]$. Each input image consists of 4 channels, and with one floating point number taking up 4 bytes of space, the total size of one image comes out to be $4 \cdot 1920 \cdot 1080 \cdot 4 = 33.18$MB, however the total GPU memory usage during training is much higher (in the order of 3 to 4 gigabytes), and in order to lower the possibility of being limited by GPU memory, the batch size during training was set to 1. The weights of all convolutional layers in the three different networks were initialized using PyTorch default behavior: this is using a uniform distribution $\mathcal{U}(-1/\sqrt{a}, 1/\sqrt{a})$, calculated as

$$a = C_{in} \cdot k_w \cdot k_h \tag{35}$$

where C_{in} is the amount of input channels to the convolutional layer, k_w and k_h are the width and height respectively of the kernels [30].

The loss function

As presented in section 2.4.3, using MSE or PSNR as a loss function is less favorable than SSIM in image quality assessment, and the SSIM score can more accurately estimate the perceptual quality of an image. This motivated the choice of only using SSIM as a loss function, throughout the entire thesis work. Both MSE (and therefore indirectly PSNR) and Delta E* as described in sections 2.4.1 and 2.4.4 were computed and used to assess the training performance of the models but were never used as actual loss functions.

3.4 Other analysis

Outside of the model training pipeline described in section 3.3, a few changes to the training process or to the structure of the neural networks were tested but opted against.

Batching

When training neural networks, it is common to collect multiple inputs together to be used in the model at the same time in a so-called *batch*. In theory, as the GPU can compute the multiple samples in the batch at the same time, this can reduce the training time for the network. In this case, batching was not used due to the high resolution of the input images. With a batch size larger than 1, the GPU memory usage becomes too high for the models presented here.

Layer optimization

One optimization technique that was attempted was restricting the model to only taking the luminance channel as an input and having a blurred luminance channel as output which is then combined with the unaltered color channels. This approach did not yield any acceptable results; when an object is blurred due to being out of focus, both the color of the light and the intensity of the light coming from said object are affected, and thus depth of field cannot be recreated only by affecting the luminance.

Altering the loss functions

Besides structural similarity, some other image quality metrics were investigated as possible loss functions. Both PSNR and delta-E*, presented in sections 2.4.2 and 2.4.4 were briefly used as loss functions: training against PSNR led to output images containing less accurate depth of field blur than when training against SSIM, and training against delta-E* had an issue with convergence where no progress was made during training, as well as the transformation to $L^*a^*b^*$. color leading to much longer training times.

Another image metric studied was gradient magnitude similarity deviation (GMSD) [31]. This measure compares the difference between gradients in two images; as sharp gradients like edges are the most affected by blur, GMSD should be able to be used as a loss function. The result was however that while sharp edges in an image were blurred properly, parts of the image with less sharp gradients did not reach as good quality as with SSIM, leading to an overall lower image quality. On top of this, the algorithm for computing the GMSD is more complex than SSIM, which led to much longer training times.

3.5 Delimitations

To limit the scope of the project to the time frame of the thesis work, a few delimitations were made.

- Training data was chosen to be generated from two different scenes. Including multiple scenes into the training dataset could have led to more generalized models, at the expense of training time.
- While alterations were made between the three different models, the main structure was limited to only that of u-nets described in section 2.3.5. This is motivated by the fact that generative adversarial networks, a deep learning structure popular for creating new images similar to the training data, have been proven to perform well when using a structure with skip connections [32][33]. These models are able to generate realistic images, and therefore a similar network structure should be able to replicate the depth of field effect.

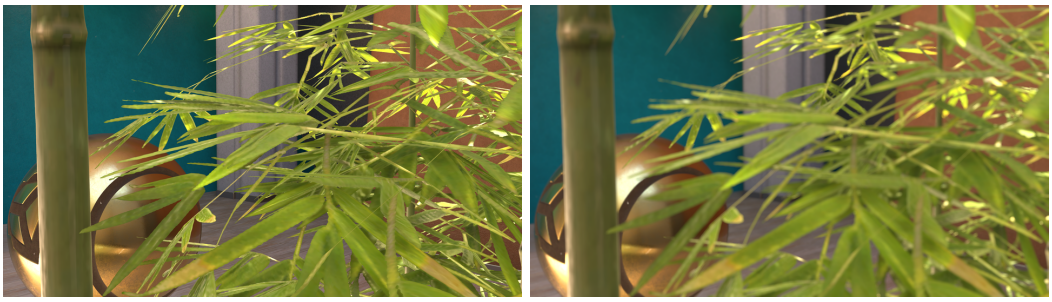
4 Results

This section will cover the evaluation and results obtained from all three models. The dataset consists of 23 test images for which SSIM, PSNR and Delta E* scores were calculated, and the maximum, minimum and average of these scores are shown. Moreover, all models obtained their better and worse results on the same images, and as presenting the entire testing data for each image is unfeasible, one better performing image and one worse performing image is chosen and their results presented[1]. Finally, the model size, CUDA time (i.e. GPU computation time) and number of floating point operations (FLOPs) are displayed.

Figure 23 shows the input image as well as ground truth of an image that performed well, and figure 24 shows the input and ground truth of an image that performed poorly. For each image, the ground truth is presented again for side by side comparison.

(a) Input image (b) Ground truth image

Figure 23: Input image and ground truth for an image that performed relatively well across all models



(a) Input image (b) Ground truth image

Figure 24: Input image and ground truth for an image that performed relatively poorly across all models

[1]Note that not the best and worst performing images are chosen; instead images that more clearly show the strengths and problems of the models are picked.

4.1 Model M1: one up/downsampling pair

Table 4: Maximum, minimum, and average image scores for the u-net with depth 1

```
========================================
Test data metrics:
SSIM: Max=0.9974, min=0.9455, avg=0.9821
PSNR: Max=42.4776, min=23.4046, avg=36.2280
Delta-E*: Max=8.9531, min=1.2966, avg=3.6004
========================================
```

In table 4, the image scores for the smaller model over the test data is presented. Observing the average scores, the structural similarity is close to 1, meaning that the structures in the image produced by the network are very similar to those in the ground truth. The Delta E* is however higher than the limit where a noticeable difference is seen, meaning that there is a systematic difference in color between the expected and actual output. This can be clearly seen in figure 26a, where the background that is supposed to be blue has turned red. Observing figure 25a, this can also be seen in the top middle of the image, where the background has turned a shade of purple instead of the expected blue color. This effect of discoloration in the background persists through the entire testing dataset, which could be a result of the model not being complex enough to retain background information for the dataset.

Regarding the actual blur, in figure 25a the network manages to recreate the depth of field effect as seen in figure 25b: there is near-field blur on the fern in the foreground, then a section in focus that can be seen on the planks on the left side of the image, and finally the background is blurred as can be seen on the hut.

(a) Model output image (b) Ground truth

Figure 25: Comparing model output and ground truth for the image that performed relatively well in the u-net with depth 1.

(a) Model output image (b) Ground truth

Figure 26: Comparing model output and ground truth for the image that performed relatively poorly in the u-net with depth 1.

Computational performance

Table 5 shows the total number of FLOPs, total model memory footprint and image generation time for the model. The CUDA time is obtained as an average of 100 model runs to improve accuracy. It can be seen that because of the high number of channels in the model, the total (video) memory requirement is over 3GB, even though the model input is 33MB.

Table 5: Floating point operations and model size for the model M1

```
Total FLOPs (G): 129.30
==================================
Input size (MB): 33.18
Forward/backward pass size (MB): 2969.40
Params size (MB): 0.43
Estimated Total Size (MB): 3003.00
==================================
CUDA time total: 160.52ms
```

4.2 Model M2.1: two up/downsampling pairs, maximum 128 channels

Table 6: Maximum, minimum, and average image scores for model M2.1

```
========================================
Test data metrics:
SSIM: Max=0.9969, min=0.9412, avg=0.9870
PSNR: Max=48.5071, min=33.1776, avg=43.1241
Delta-E*: Max=3.9815, min=0.7148, avg=1.5387
========================================
```

Table 6 show the image scores over the test dataset for the u-net with a higher amount of feature channels and two upsampling/downsampling pairs. Here, the average SSIM is very close to 1, implying that the network was able to produce a good depth of field effect on average. The average PSNR value is also high, indicating that there is not a lot of noise in the output images, and the average Delta E* shows that there is no big noticeable difference in color between the actual model output and expected output. The maximum Delta E* seen in the training set is almost 4, meaning that for a few images there is a noticeable change in color between the expected and actual output.

In figure 28a, there is apparent near field blur on the palm leaves as desired, however the leaves on the left-hand side exhibit an unwanted discoloration around the edges, see figure 29; this is the most apparent error this model produces. Figure 27a shows a close to identical recreation of the depth of field in figure 27b, with obvious near-field and far-field blur.

(a) Model output image (b) Ground truth

Figure 27: Comparing model output and ground truth for the image that performed relatively well.

(a) Model output image (b) Ground truth

Figure 28: Comparing model output and ground truth for the image that performed relatively poorly.

Figure 29: Upper left part of the image in figure 28a, illustrating artifacts

Computational performance

In table 7, the computational requirement, memory requirement, and GPU computation time for the M2.1 model are shown. The computation time is calculated as an average of 100 image inputs. Here, the total memory requirement is closer to 4GB with an input image of resolution 1920×1080.

Table 7: Floating point operations and model size for the model M2.1

```
Total FLOPs (G): 234.49
==================================
Input size (MB): 33.18
Forward/backward pass size (MB): 3632.95
Params size (MB): 1.95
Estimated Total Size (MB): 3668.08
==================================
CUDA time total: 258.68ms
```

4.3 Model M2.2: two up/downsampling pairs, maximum 96 channels

Table 8: Maximum, minimum, and average image scores for model M2.2

```
========================================
Test data metrics:
SSIM: Max=0.9964, min=0.9363, avg=0.9853
PSNR: Max=50.4431, min=33.0318, avg=44.3002
Delta-E*: Max=4.2692, min=0.7800, avg=1.5085
========================================
```

In table 8, the image quality measures for the testing dataset are shown. The average structural similarity is once again close to 1, complemented with a high average PSNR, implying that on average the model can reproduce the expected depth of field with high fidelity. The average delta E* also shows that there is for the most part no noticeable difference in color between the expected and actual output; it reaches a maximum of over 4, implying that a few images in the dataset experience a noticeable color shift.

Figure 31a does show some color shift in the background compared to figure 31b with it becoming slightly darker. There are also some artifacts around the edges of the leaves in the top left part of the image, shown in more detail in figure 32, where the limit between the leaves and the background turns dark blue. Figure 30 shows the network producing a near identical depth of field effect, both in the near-field and in the far-field.

(a) Model output image (b) Ground truth

Figure 30: Comparing model output and ground truth for the image that performed relatively well.

(a) Model output image (b) Ground truth

Figure 31: Comparing model output and ground truth for the image that performed relatively poorly.

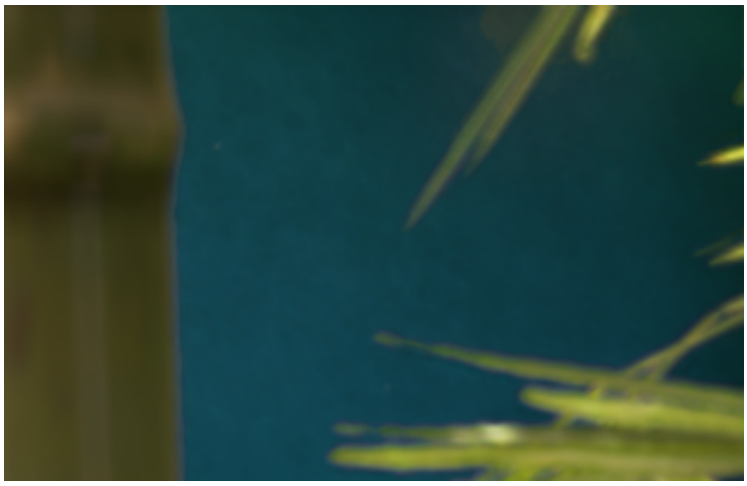


Figure 32: Upper left part of the image in figure 31a, illustrating some artifacts around the edges

Computational performance

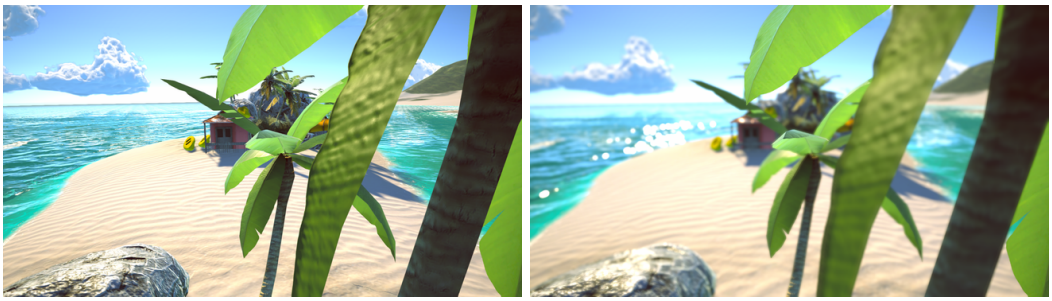
Table 9 shows the FLOPs, memory usage, and time taken for the model M2.2 to compute one image; the CUDA time is obtained as an average over 100 image inputs. The total memory usage is the lowest of the three studied models, under 3GB, however the computational complexity is higher than the model M1.

Table 9: Floating point operations and model size for the model M2.2

```
Total FLOPs (G): 132.47
==================================
Input size (MB): 33.18
Forward/backward pass size (MB): 2737.15
Params size (MB): 1.10
Estimated Total Size (MB): 2771.43
==================================
CUDA time total: 173.89ms
```

4.4 Studying a scene not used for training

In this section, an image from a scene (called "Boat attack") not used for generating training data is applied to the model M2.2 [34]. This scene contains elements not seen by the model during training, such as water, sand, and clouds. The input image as well as the ground truth is shown in figure 33, and the outputs are shown in figure 34.

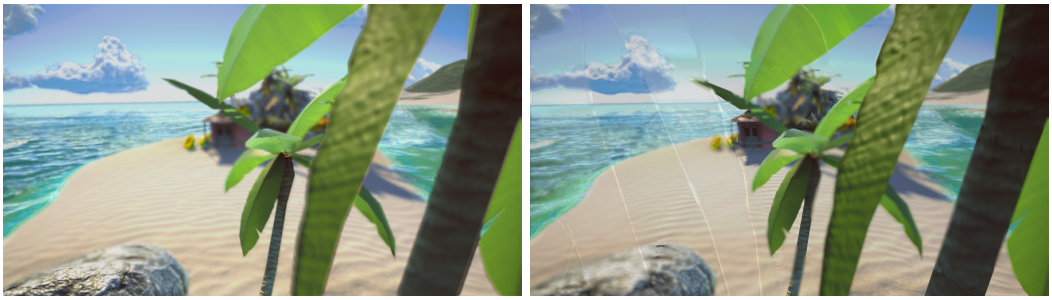


(a) Input image (b) Ground truth

Figure 33: Input and ground truth of the image from the boat attack scene

Observing figure 34a, it is seen that the model manages to create some near-field and far-field blur, while keeping a section of the rock on the bottom left in focus. The image quality is not very good however, with the scene in general appearing darker and artifacts appearing on the edges of objects which is especially visible on the palm tree in the foreground; this lower quality is also reflected in the image quality metrics presented in table 10.

Figure 34b uses the same input image shown in figure 33a, but the depth map has been mirrored along the vertical axis. This shows that the model takes the information contained in the depth map and blurs the input image accordingly.



(a) Model output image (b) Model output with the depth map mirrored

Figure 34: Model output of the image from the boat attack scene

Table 10: Image evaluation metrics for the model output in figure 34a.

```
========================================
Test data metrics:
PSNR: 29.6020
SSIM: 0.9292
Delta E: 4.8134
========================================
```

5 Discussion

The aim of the thesis work is to analyze the use of neural networks for replicating a depth of field effect. The results from section 4 are compared against one another and discussed to draw a conclusion from the problem formulation.

Results

The training process and data is kept the same for all three models, suggesting that the model with one downsampling/upsampling pair does not have enough complexity to be able to replicate the depth of field effect. This model also highlights the usefulness of measuring the delta-E*: while the average PSNR and SSIM scores suggest that the image fidelity is good, this is not the case when actually observing the images; there is a difference between the perceived quality and measured quality of the outputs.

Both models with two downsampling/upsampling pairs produce outputs with good measured quality and relatively few image artifacts. Comparing tables 6 and 8, the smaller model of the two produces ever so slightly better results than the other model, despite it being less complex. While the expected result is that the more complex model outperforms the other, a simple explanation could be the fact that the more complex model became slightly overfitted on the training data while this did not happen to the same extent to the less complex model.

Looking at figure 34b, it is seen how the model manages to extract which areas of the image to blur from the depth map. This implies that the model can generalize the depth of field effect to any image, and that it is not impossible to create a fully generalized model.

Computational performance

Comparing the computational performance of the three models, some interesting observations can be made. Firstly, comparing the models M2.1 and M2.2, these achieve near equal results on the test data while the M2.1 model is slower in time by a factor of almost 1.5, requires 900MB more memory and does 1.77 times more computations overall when compared to the M2.2 model.

Secondly, comparing the M1 and M2.2 models where the former does not recreate the depth of field effect accurately while the latter does, it is seen that these require close to the same computational performance; the model M2.2 requires 230MB less memory and requires only 2% more total operations with 8% more execution time. From this, it is possible to see that having two downsampling/upsampling pairs with skip connections instead of one helps in preserving the color accuracy which is the main struggle of the model M1.

Dataset

The dataset used in this thesis is limited, and as such it is not guaranteed that the models can be generalized to any arbitrary scene with good results. The main limitation in the dataset is not the number of images, but the fact that there are only two scenes with relatively few elements each from which the data is generated; one of the major issues that

arose during the project was bad color accuracy which can be attributed to these two scenes not being very diverse in terms of color. By increasing the number of scenes from which data is collected and picking scenes with different color profiles, better generalized results can be obtained. This is further reinforced by the result in figure 34a, where the general blur effect is applied to the image, but the overall image quality is quite poor.

Conclusion

Referring to the problem formulation in section 1.1 and using the results presented in section 4, the conclusion can be drawn that the depth of field effect can be replicated with high fidelity using neural networks. From the results, answers to the three smaller questions are formulated as follows:

- To train the neural networks presented in the thesis, a relatively small amount of data was needed. With 158 unblurred images and associated depth maps, the model can produce images nearly indistinguishable from corresponding ground truth images, given that the test data is taken from the same scene as the training data. These datasets can also easily be generated using game engines that can export blurred and unblurred images.
- The scope of the thesis was limited to only studying models with a u-net architecture, and this structure is applicable for the use case of replicating the depth of field effect.
- Only structural similarity was used as an actual loss function for training the neural networks in the thesis work, however the observed results show that SSIM is able to properly evaluate the quality of the generated blur.

6 Future work

In this section, different ideas that arose during the thesis work that were not investigated are listed.

Pre-processing using the circle of confusion

The circle of confusion is a direct measure of the strength of the depth of field effect for a point in the 3D-scene. Using equation (1), a possible improvement is to compute the circle of confusion from the depth map for every pixel in a pre-processing step. The circle of confusion is then used as an input layer instead of the depth map. This has the advantage of directly telling the network how heavily blurred each pixel should be; this information must now be extrapolated from the depth map by the models.

Optimization

During the thesis work, most of the effort was put into building and testing the models and very little time was left over for evaluation. Studying the values input to the final convolution where the number of layers is reduced to 3, some of the input layers are filled with values very close to 0, meaning they have little to no impact on the final output. Studying the models to figure out where the number of channels can be reduced would be a good first optimization step.

As discussed in section 5, having two skip connections instead of one made the difference between the models M2.2 and M1 being able to preserve the background color in input images and not being able to, so another interesting optimization could be to increase the number of skip connections even more while reducing the maximum amount of channels in the network.

Loss functions

In the thesis work, structural similarity was exclusively used as the loss function. It is however possible to use a weighted average of different metrics to use as a loss function, for example weighting together PSNR and SSIM to use in training:

$$e(\boldsymbol{x}, \boldsymbol{y}) = \frac{\alpha \cdot PSNR(\boldsymbol{x}, \boldsymbol{y}) + \beta \cdot SSIM(\boldsymbol{x}, \boldsymbol{y})}{\alpha + \beta} \tag{36}$$

References

[1] Salvaggio N, Shagam J. Basic Photographic Materials and Processes; 2019.

[2] NVIDIA DLSS;. Accessed 22 november 2021. Available from: https://developer.nvidia.com/dlss.

[3] Blahnik V, Schindelbeck O. Smartphone imaging technology and its applications. Advanced Optical Technologies. 2021;10(3):145–232.

[4] London B, Stone J, Upton J. Photography. Pearson; 2005.

[5] Forsyth DA, Ponce J. Computer Vision: A Modern Approach. 2nd ed. Prentice Hall; 2012.

[6] Viewing and Transformations in OpenGL;. Accessed 2022-01-20. Available from: https://www.khronos.org/opengl/wiki/Viewing_and_Transformation.

[7] Post-processing effects in Unity;. Accessed 2022-01-20. Available from: https://docs.unity3d.com/Manual/PostProcessingOverview.html.

[8] Next generation post processing in Call of Duty: Advanced Warfare;. Accessed 2022-01-20. Available from: http://www.iryoku.com/next-generation-post-processing-in-call-of-duty-advanced-warfare.

[9] McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics. 1943;5(4):115–133.

[10] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition; 2015.

[11] Dong C, Loy CC, He K, Tang X. Image Super-Resolution Using Deep Convolutional Networks; 2015.

[12] Sharif Razavian A, Azizpour H, Sullivan J, Carlsson S. CNN Features Off-the-Shelf: An Astounding Baseline for Recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops; 2014. .

[13] Xie S, Tu Z. Holistically-Nested Edge Detection. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV); 2015. .

[14] Zech JR, Badgeley MA, Liu M, Costa AB, Titano JJ, Oermann EK. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study. PLOS Medicine. 2018 11;15(11):1–17. Available from: https://doi.org/10.1371/journal.pmed.1002683.

[15] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016. http://www.deeplearningbook.org.

[16] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research. 2014;15(1):1929–1958.

[17] Rodriguez P, Wiles J, Elman JL. A recurrent neural network that learns to count. Connection Science. 1999;11(1):5–40.

[18] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition; 2015.

[19] Ronneberger O, Fischer P, Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation; 2015.

[20] Thung KH, Raveendran P. A survey of image quality measures. In: 2009 International Conference for Technical Postgraduates (TECHPOS); 2009. p. 1–4.

[21] Wang Z, Bovik AC. A universal image quality index. IEEE Signal Processing Letters. 2002;9(3):81–84.

[22] Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing. 2004;13(4):600–612.

[23] CIE Colorimetry 15. 3rd ed. CIE; 2004.

[24] Anderson M, Motta R, Chandrasekar S, Stokes M. Proposal for a standard default color space for the internet—srgb. In: Color and imaging conference. vol. 1996. Society for Imaging Science and Technology; 1996. p. 238–245.

[25] Sharma G. Digital Color Imaging Handbook. 1st ed. CRC Press; 2003.

[26] Unity;. Available from: `https://unity.com/`.

[27] BT.470: Conventional analogue television systems. ITU; 2005. Available from: `https://www.itu.int/rec/R-REC-BT.470/en`.

[28] PyTorch;. Available from: `https://pytorch.org/`.

[29] Amazon Elastic compute cloud;. Available from: `https://aws.amazon.com/ec2/`.

[30] PyTorch convolutional layer initialization source code;. Accessed 31 january 2022. Available from: `https://github.com/pytorch/pytorch/blob/72c972e1e1b4ad838de604e35269e200a70db5f2/torch/nn/modules/conv.py`.

[31] Xue W, Zhang L, Mou X, Bovik AC. Gradient Magnitude Similarity Deviation: A Highly Efficient Perceptual Image Quality Index; 2013.

[32] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative adversarial nets. Advances in neural information processing systems. 2014;27.

[33] Ledig C, Theis L, Huszar F, Caballero J, Cunningham A, Acosta A, et al.. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network; 2017.

[34] Boat attack demo scene source;. Accessed 9 february 2022. Available from: https://github.com/Unity-Technologies/BoatAttack.