Mathematisch-Naturwissenschaftliche Fakultät

Computergrafik

Masterarbeit

CNN-based BRDF parameter estimation

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik
Computergrafik
Mark Benedikt Boss, mark.boss@student.uni-tuebingen.de, 2018

Bearbeitungszeitraum: 01.12.2017-01.05.2018

Betreuer/Gutachter: Prof. Dr. Hendrik Lensch, Universität Tübingen
Zweitgutachter: Prof. Dr. Andreas Schilling, Universität Tübingen

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Mark Benedikt Boss (Matrikelnummer 4067683), February 20, 2020

Abstract

The behavior of surfaces is an essential field in computer games and movies. An exact representation of a real-world surface allows for a higher degree of realism. Capturing or artistically creating these materials is a time-consuming process. Thus, in this thesis a method which utilizes an encoder-decoder Convolutional Neural Networks (CNN) to extract information of the Bidirectional Reflectance Distribution Function (BRDF) automatically is proposed. Opposed to previous means this method retrieves information of the whole surface as spatially varying BRDF-parameters with a sufficiently high resolution for real-world usage. The capture process for materials only requires five known light positions with a fixed camera position and thus can be acquired even in a mobile setup. This reduces the scanning time drastically and a material sample can be obtained in seconds with an automated system.

Kurzfassung

Das realistische Verhalten von Oberflächen ist ein wichtiger Aspekt in Computerspielen und Filmen. Eine genaue Repräsentation von realen Materialien ist essenziell, um ein hohes Maß an Realismus zu erzielen. Die Aufnahme oder manuelle Erzeugung von solchen Oberflächen ist ein aufwendiger Prozess. Um diesen Prozess zu verkürzen, wird eine Methode vorgestellt, die Parameter für die Bidirectional Reflectance Distribution Function (BRDF) automatisch mittels eines Encoder-Decoder Convolutional Neural Networks (CNN) extrahiert. Im Gegensatz zu vorherigen Verfahren ermittelt die vorgestellte Methode Informationen für jeden Punkt über die ganze Oberfläche. Dies geschieht mit einer ausreichend hohen Auflösung für reale Anwendungsfälle. Der Aufnahmeprozess benötigt nur fünf Lichtpositionen mit einem festen Kamerastandpunkt. Dadurch kann auch ein mobiler Aufbau konstruiert werden und die Aufnahmedauer für eine Oberfläche ist drastisch reduziert. Bei einem automatischen System kann diese Aufnahme in wenigen Sekunden ausgeführt werden.

Acknowledgments

I would first like to thank Prof. Hendrik Lensch of the computer graphics department at the University of Tübingen for giving me the opportunity to work on this thesis. I also want to thank Fabian Groh for being my supervisor and providing invaluable input throughout the whole thesis. Without the constant support of them, this thesis would not exist.

Further, I want to thank Raphael Braun for his previous work on the light stage and Sebastian Herholz for providing additional insights into working with Mitsuba and rendering in general.

The computational resources support this work at the computer graphics department at the University of Tübingen.

Lastly, I must express gratitude to my parents and my girlfriend for providing me with support and continuous encouragement throughout my years of study and this thesis in particular. This work would not have been possible without them.

Thank you.

Contents

List of Figures

List of Tables

Acronyms

2D two-dimensional. 41, 42, 44, 55–57

3D three-dimensional. 42–45, 55, 56, 68, 77

BRDF Bidirectional Reflectance Distribution Function. 17–21, 33, 34, 37, 39, 40, 45, 46, 52, 67, 68, 75, 77, 78

CNN Convolutional Neural Network. 18, 20, 21, 29, 31, 32, 39, 40, 70, 71

COBYLA Constrained optimization by linear approximation. 44

GAN Generative Adversarial Network. 21, 32, 33, 40, 65, 67, 68, 70, 71

GPU Graphical Processing Unit. 20

HDR High Dynamic Range. 41, 42, 52

HSV Hue Saturation Value. 49, 77

LED Light-Emitting Diode. 23, 40–44, 78

MAE Mean Absolute Error. 21, 29, 48, 53, 57, 59–61, 63, 64, 67–70, 77

MERL Mitsubishi Electric Research Laboratories. 20, 34, 45

MLP Multi-Layer Perceptron. 24, 25

MS-SSIM Multi Scale-Structural similarity. 59

MSE Mean Square Error. 29, 57, 63

PCA Principal Component Analysis. 20

ReLU Rectified Linear Unit. 27–29, 33, 53

SGD Stochastic Gradient Descent. 29, 30

SSIM Structural similarity. 59, 63

1. Introduction

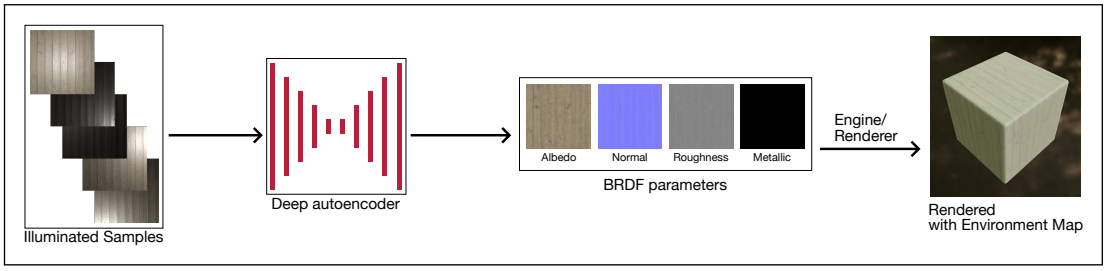
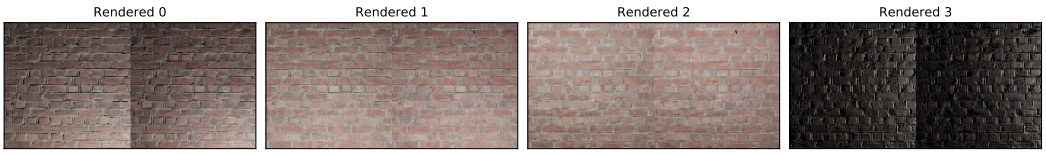


Figure 1.1.: The proposed BRDF estimation system. Here, several synthetic or real-world captured samples are passed to a deep autoencoder, which predicts BRDF samples. These samples can be used in an arbitrary game engine or renderer in every lighting condition.

With the advance of processing power and improvements in rendering algorithms, movies and video games approach photorealism. The 3D models of characters and scenes are incredibly detailed, and the behavior of light is recreated realistically in rendering algorithms. To achieve this high level of realism, objects and characters are often based on the real world. In the past artists created all 3D models and textures manually in a time-consuming process. With the advances in camera technology and processing power, an increasing amount of 3D models and textures are captured from real-world objects. This process is known as photogrammetry.

Most available photogrammetry software suits are only capable of capturing information about the height and color with the specific light condition at the time and location of the recording. Using these textures for rendering under different lightings leads to problems, as the texture already contains shadows and highlights. Removing these highlights and shadows is known as delighting. Nevertheless, delighted textures are not perfect as these only represent the color and height of an object. Information about the reflective behavior of the object is still missing. Artists often manually adjust the reflectivity to fit the object. However, this process requires an experienced artist and is again time-consuming. Another approach is to capture the object from a large number of different viewing angles and light positions. The captured images can then be used to fit a Bidirectional Reflectance Distribution Function (BRDF) to each pixel. As each different viewing angle needs different light positions, this process can be time-consuming. It can be sped up by using multiple cameras, but this increases the cost of the recordings drastically.



(a) Brick wall.

(b) Wood.

Figure 1.2.: Comparison of ground truth and predicted renderings. These images are taken from the trained CNN architecture from section 3.4. The ground truth of each pair is located on the left side and the prediction on the right side. These renderings are created with unlearned light positions.

Artists are capable of reproducing information about reflective behavior from a few images under different light conditions by leveraging their previous knowledge about similar objects. Neural networks are reaching human performance in many areas such as speech or image recognition in the recent years [45, 10, 39, 40]. In this thesis, the possibility of training a neural network to predict BRDF parameters for spatially varying materials from multiple input images is explored. Therefore, a system with an encoder-decoder Convolutional Neural Network (CNN) is proposed. A general overview of the proposed system is visible in fig. 1.1. Figure 1.2 displays two rendered predictions compared to the ground truth materials.

1.1. Problem Statement

In recent years scanning materials for video games and movies gained importance. Traditionally, all surfaces were created by artists in a time-consuming manual process. In this process, reference footage is first gathered, and then the texture is created and evaluated under several light positions. Creating the seeming randomness of real surfaces is hard to achieve for even skilled artists. Scanning of real surfaces solves this issue. Hardware for scanning surfaces is expensive to develop and writing pipelines to process the scans automatically is equally labor and time intensive. Scanning all needed materials for production requires a team and probably travel costs. Thus, many companies use large libraries like `megascans.se` to solve this.

For a convincing material, a spatially varying surface is essential. The overall texture and color of surfaces change for nearly every material at least slightly. Apart from the general texture and color of the surface, most of them have spatially varying

surface irregularities. These irregularities vary between larger visible and micro irregularities. The larger ones change the surface structure visibly. The smaller ones change the reflective behavior. Smooth surfaces have fewer imperfections and thus provide a sharp reflection, while rough surfaces have many irregularities, and the reflection is blurred. The structure and roughness of a surface change spatially on a material and is not homogeneous. For example, areas of a surface are not worn-out uniformly. Regions which are used more frequently are often smoother and contain scratches. These irregularities are essential for a realistic appearance of surfaces.

As the spatially varying behavior is critical, the size of the output textures is essential. An output size of at least 512×512 pixels is desirable for efficient usage.

The estimation of the BRDF parameters should be possible with few input images. However, it is shown that estimating a full BRDF from few images is ill-posed [25, 32, 22, 24]. Finding constraints to limit the search space is one option to solve this. The constraint can be, for example, to limit the number of categories to only wood or metal textures or to constrain the BRDF to an analytical model with fewer parameters than a full BRDF.

Additionally, this work should lower the capture time for BRDFs in the light stage at the University of Tübingen (see section 2.1). As mentioned capturing BRDFs is a time-consuming process, and this work explores possibilities to reduce this process by allowing recordings of materials with few captured images.

1.2. Related Research

As the capture time for a full BRDF requires a sample from every lighting angle for every viewing angle, several methods to decrease the capture time are researched.

One possible way to achieve a faster capture time is to decrease the time to capture individual samples. Mukaigawa et al. [26] propose a system which does not rely on a mechanical drive for lighting changes, but still achieves a dense measurement. By leveraging a projector with an ellipsoidal mirror, the lighting angle can be changed in quick succession [26]. However, this method only allows capturing smaller samples due to the size of the mirror.

By combining fast capture and sparser sampling, an additional speedup can be achieved. For example, Dong et al. [5] design a two-phase capture setup. For the first phase, a condenser-based optical setup allows capturing the hemisphere of outgoing rays for a single point. This step needs to be manually performed by moving the device slowly across the sample. The second phase captures the sample from a fixed view position with a moving light and environment lighting. A mirror ball is used to capture the environment lighting. In this phase, 20-200 samples are generated and both phases a combined in the reconstruction process [5].

Most materials are related to each other. For example, the behavior between different woods is not changing too drastically. The sampling can be done sparser, and data from full scans can be used to reconstruct the dense BRDFs from few samples. Nielsen et al. [29] perform a Principal Component Analysis (PCA) on the Mitsubishi Electric Research Laboratories (MERL) BRDF dataset [25]. Initial random samples are taken, and optimal sampling positions are searched by taking the knowledge of the most influential regions from the processed BRDF dataset. With this knowledge a reduction of the dimensionality is possible. The result of this work are non spatially varying BRDF parameters [30, 29]. A different method of using previous captured full BRDFs is to specify a set of fundamental materials and combine them to arbitrary new materials. Goldman et al. [8] propose a system where sets of input images with fixed camera angle and changing environment lights are taken. By constraining the BRDF additionally to an analytical model, the Ward model [50], spatially varying parameters and normal maps can be estimated. In this method, the user provides the number of fundamental materials to combine. The optimization is performed in two stages: The first optimizes the BRDF parameters and the second the fundamental material mix and normal texture [8].

Aittala et al. [1] propose another method which extracts full BRDF information from two images taken by a mobile phone. To achieve this the phone is held perpendicular to the sample, and one picture is captured with the flash and one without. The method extracts the diffuse and specular albedo, anisotropy information, glossiness and the normal structure. The output is a small tile of the material with a resolution of 192×192 pixels. To fit the parameters the whole image of the sample is split into a grid. A single tile is selected arbitrarily and is matched to the other tiles in the grid. As each tile is spatially distributed the half vector for each tile is different due to the different lighting and viewing vectors. Fitting a master tile to matching tiles is only possible for materials which conform to the assumption of stationary materials. The definition of a stationary material is a material with a texture which repeats itself. For more homogeneous materials this method is not generating accurate results. The fitting itself is done in a multistage process. The first step is fitting the master tile to every other tile by using correspondences. An initial spatially varying BRDF is fitted to the master tile and is refined in further optimization stages.

In recent years, due to an advance in Graphical Processing Units (GPUs), neural networks and more specifically CNNs gained popularity and are applied to various problems. Several works leverage the learning capability of CNNs to estimate BRDF parameters. Li et al. [21] trained a network to estimate Ward model parameters from a single photograph. This is done in an encoder-decoder CNN with skip connections. However, due to the especially sparse input of a single photo with unknown lighting conditions, the network only produces spatially varying normal and albedo parameters. The specular and roughness parameters are homogeneous for the whole sample [21]. In a slightly different approach, Yu and Smith [51] propose a system which works on unlabeled data. The main contribution of this paper is a rendering framework for neural networks which is capable of backpropagating

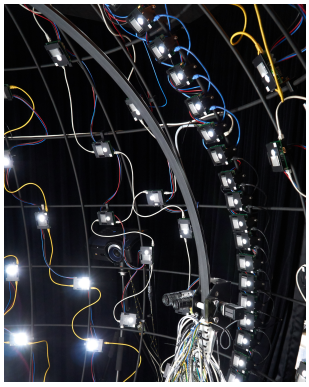
a loss. An example use case in the paper is the estimation of BRDF parameters. A CNN is used for this task. The network receives the image of the sample, the depth map of the scene, the view and light positions as input. The output of the network is the non-spatially varying BRDF parameters. The parameters can then be re-rendered by the network renderer, and the loss is calculated with the input image. The loss then propagates through the rendering implementation and the weights of the CNN are adjusted accordingly [51].

As the BRDF estimation can be seen as a mapping of input images to a different style of output images, it is worth to mention the work of Isola et al. [14]. Here a general framework for an image-to-image mapping is introduced. Isola et al. propose a U-Net [34] based architecture for the generator network which is modified to adhere to the guidelines of the DCGAN model by Radford et al. [31]. Additionally, a Mean Absolute Error (MAE) loss from the generator output to the ground truth data is introduced. This loss is added with a scaling factor to the Generative Adversarial Network (GAN) loss. The scaling factor increases the importance of the MAE compared to the GAN loss. This dual-loss concept is improving the feedback for the generator network, as it would otherwise only receive strictly binary input from the discriminator. By providing a dual loss, the training behavior stabilizes furthermore. According to Isola et al. [14], this produces sharper and more convincing results than with the U-Net architecture alone.

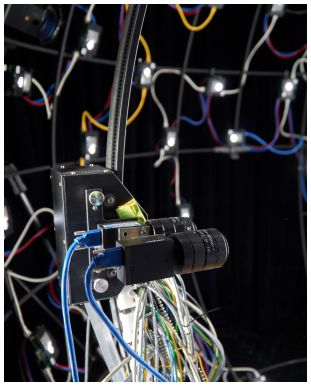
2. Background

This chapter provides a brief introduction of the hardware used to capture real-life samples, machine learning, and rendering.

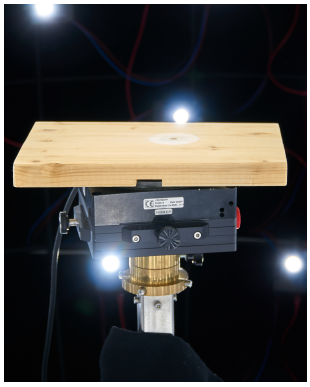
2.1. Hardware



(a) The main arc and the spherical gantry.



(b) The stereo cameras on the motorized camera arc.



(c) The turntable.

Figure 2.1.: Images highlighting the light stage setup.

The light stage is a spherical gantry consisting of 196 Light-Emitting Diode (LED) boards. It is about two meters in diameter. It features a vertical arc, called the main arc, which consists of 55 LED boards, and it is thus capable of producing the highest resolution in lighting angles. The main arc is pictured in fig. 2.1a. The remaining boards are distributed on the sphere. Each LED board contains 15 LEDs with different wavelengths. The brightest LED is emitting white light and can be considered the main LED.

In the center of the sphere is a motorized turntable, which can rotate a sample 360 degrees. This turntable is shown in fig. 2.1c.

Two cameras are positioned on a motorized arc near the main arc. Figure 2.1b displays the cameras on the motorized arc. The arc is fixed to the gantry, and the

cameras can be positioned freely between a horizontal and perpendicular position to the turntable surface.

2.2. Machine learning

The area of machine learning focuses on the design of algorithms that enable a computer to learn from data instead of relying on a hard-coded set of manually implemented rules.

2.2.1. Single-Layer Perceptron

The perceptron is an algorithm for a binary classifier. This means it maps a vector x with $x \in \mathbb{R}^d$ to an output value $f(x)$ which is a single binary value. This is expressed as:

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Where w is a weights vector with $w \in \mathbb{R}^d$, $w \cdot x$ is the dot product over the content of the vector $\sum_{i=1}^{d} w_i x_i$ and b represents a bias offset with $b \in \mathbb{R}$. The bias is used to shift the decision in a specific direction away from the origin. The binary output function is known as the Heaviside function [35].

By including the bias in the weight vector $w := (b, w_1, ..., w_d)^\intercal$ and adding a 1 to the input vector $x := (1, x_1, ..., x_d)^\intercal$ the calculation can be reduced to a single dot product $w \cdot x$.

2.2.2. Multi-Layer Perceptron

Regarding the Multi-Layer Perceptron (MLP) the perceptron model allows the output of more than one value. Additionally, different activations beside the Heaviside function are introduced. This allows for the MLP to express more complex problems than the linear decision of the single-layer perceptron. Thus, the perceptron is defined as:

$$y = f(w \cdot x) \tag{2.2}$$

Where $y \in \mathbb{R}^m$ is the output of the perceptron, $w \in \mathbb{R}^{n \times m}$ is the weight vector with added bias, $x \in \mathbb{R}^n$ is the input vector with an added one and f is the activation function. In section 2.2.2 this concept is visualized [36].

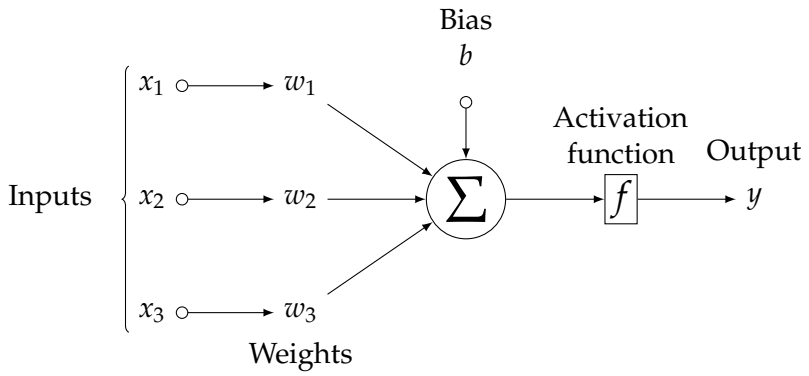


Figure 2.2.: Visualization of a perceptron.

The perceptron can be chained with two other perceptrons to form a MLP in the following way:

$$y = f_1(w_1 \cdot f_2(w_2 \cdot x)) \tag{2.3}$$

If a layer neither receives the network input nor is the final output layer, it is referred to as a hidden layer. The MLP consists of at least three layers. In most deep learning frameworks this is called a fully connected layer due to the connection of each output to every input in the dense weight matrix. Figure 2.3 illustrates this connection. Each circle in this diagram represents a perceptron (see section 2.2.2).

2.2.3. Backpropagation

As modern network architecture can contain millions of weights, these parameters w_i need to be set efficiently. The output of the network y is compared to the desired value $\hat{y}$ by a loss function E. This error between the desired value and actual output can be contributed to a single weight w_i using the partial derivative $\frac{\partial E}{\partial w_i}$. This expresses the amount of change needed for a specific weight to lower the error.

As this needs to be performed for all weights in the network, a technique, called backpropagation, developed by Rumelhart et al. [37] is used. First, the network is evaluated in a forward pass. An input x is passed through all neurons and weights to calculate the output of the network y. The error is then computed using a loss function which is expressible as an average of error functions $E = \frac{1}{n}\sum_x^n E_x$. This enables a gradient calculation after each evaluated training sample. The loss is then propagated from the output through the network. This is done using partial derivatives and repeatedly applying the chain rule of calculus. After this step, each neuron has an error value which represents its contribution to the network output [37].

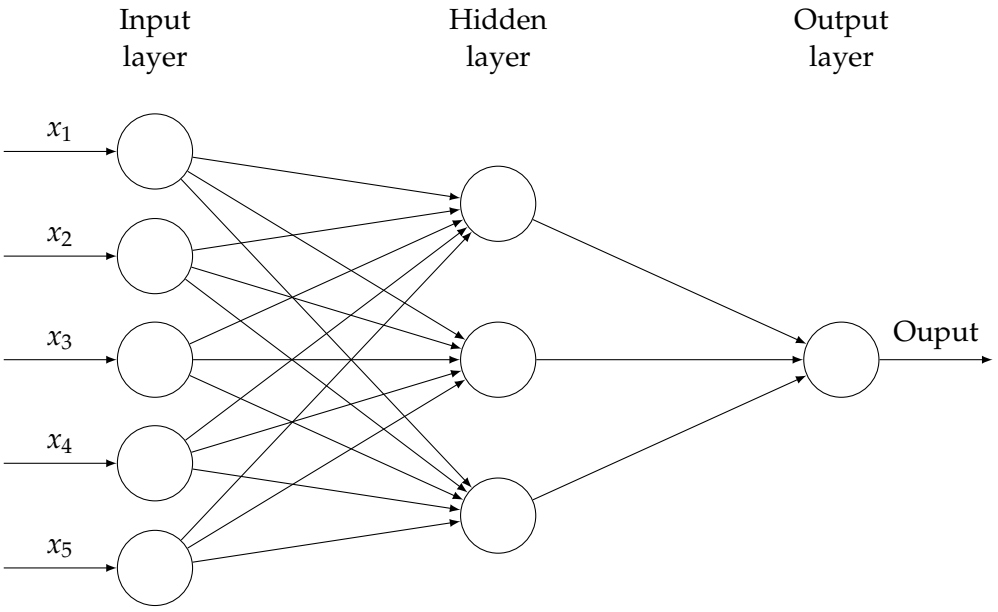


Figure 2.3.: A Multi-layer perceptron.

A simple example explains this process. The error function and the output is defined as:

$$\begin{aligned} E &= \frac{1}{2}(y-\hat{y})^2 \\ y &= \varphi(\underbrace{wx+b}_{:=a}) \end{aligned} \tag{2.4}$$

where φ is the sigmoid function. The sigmoid function has a particular simple derivative: $\text{sig}'(t) = \text{sig}(t)(1-\text{sig}(t))$.

It is assumed that only one weight $w = 1$ exists, the bias is set to $b = -3$, and that the input is $x = 3$ and the desired value is $\hat{y} = 1$. The forward pass results in $y = \frac{1}{2}$ and $a = 0$.

$$
\begin{aligned}
\frac{\partial E}{\partial y} &= y - \hat{y} \\
&= \frac{1}{2} - 1 = -\frac{1}{2}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial a} &= \frac{\partial E}{\partial y}\frac{\partial y}{\partial a} = \frac{\partial E}{\partial y} y(1-y) \\
&= -\frac{1}{2}\frac{1}{2}\left(1 - \frac{1}{2}\right) = -\frac{1}{8}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial w} &= \frac{\partial E}{\partial a}\frac{\partial a}{\partial w} = \frac{\partial E}{\partial a} x \\
&= -\frac{1}{8} 3 = -\frac{3}{8}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial b} &= \frac{\partial E}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial E}{\partial a} 1 \\
&= -\frac{1}{8}
\end{aligned}
$$

2.5

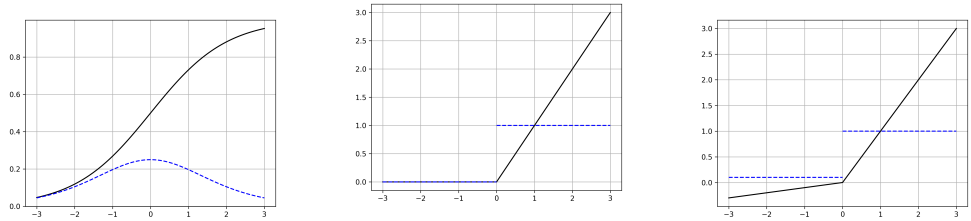
The values $\frac{\partial E}{\partial w}$ and $\frac{\partial E}{\partial b}$ can now be used by various gradient optimizer methods described in section 2.2.7.

2.2.4. Activation Functions

An activation function is responsible for mapping the output of the dot product from the input and weights to a new output value (see section 2.2.1). A simple activation function, the Heaviside step function, is already introduced in eq. 2.1.

An essential element of these functions is that they need to be nonlinear. If a linear function is used, this will result in a linear combination of every input. Thus, the network can be simplified to just a single neuron. By using a nonlinear function, a neuron cannot be linearly combined with another neuron, and the final output is dependent on every neuron.

Commonly used functions are Sigmoid, ReLU and the leaky ReLU. They are defined as followed:



(a) Sigmoid

(b) Rectified Linear Unit (ReLU)

(c) Leaky ReLU with $\alpha = 0.1$

Figure 2.4.: Example of different activation functions. The blue dashed lines represent the derivative and the black line is the evaluated function.

Sigmoid function - see fig. 2.4a

$$\begin{aligned} f(x) &= \frac{1}{1+e^{-x}} \\ f'(x) &= f(x)\,(1-f(x)) \end{aligned} \tag{2.6}$$

The sigmoid function was one of the earliest and for a long-term widely used activation functions. This is due to its simple derivation and general properties. However, the sigmoid function tapers off for large absolute values. The gradient of these values is weak. This is referred to as the vanishing gradients problem. The learning progress for large values is slow due to the small changes in the gradient. The tangens hyperbolicus is a similar function to the sigmoid function with nearly the same properties. It can even be expressed as a scaled sigmoid $\tanh(x) = 2\,\mathrm{sig}(2x) - 1$. The output range is between -1 and 1.

ReLU - see fig. 2.4b

$$\begin{aligned} f(x) &= \arg\max_0 x \\ f'(x) &= \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases} \end{aligned} \tag{2.7}$$

Recently the ReLU function became a popular activation function. As the function is linear for positive values, the gradient is constant in these areas. The constant gradient prevents a slow-down in training. Thus the vanishing gradients problem is avoided. Negative values result in the gradient and output being zero [27]. This is desirable because the activations in the network are sparse. However, this can lead to a problem of dead neurons. Dead neurons are neurons which only calculate negative outputs and thus only produce output with the value zero. The neuron is not learning and not contributing to the network.

Leaky ReLU - see fig. 2.4c

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise, } \alpha = 0.01 \end{cases}$$
$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x < 0 \end{cases} \tag{2.8}$$

The leaky ReLU technique is a possible solution for the dead neuron problem. As a small slope exists for negative values, a gradient exists. Neurons cannot die because of this change [23]. For the α parameter different values can be used. Applying different α values is often referred to as a parametric leaky ReLU. Commonly a value of 0.01 is used for leaky ReLUs.

2.2.5. Pooling and Stride

It is common to decrease the width and height dimensions in networks to compress an input in several stages. One way to achieve this is a max pooling operation. For max pooling, a window is moved spatially across the output of a layer. The maximum value is the only value used as the output of the operation. As only one value is used as the output from the window, this reduces the spatial dimensions by the size of the max pool kernel. For example, a kernel of size two is decreasing the resolution by half. For a CNN a possible variation is to move the convolution filter with a stride. This reduces the resolution, too. It is found that this produces results equal to a max pooling operation, but with increased performance and training speed [43].

2.2.6. Loss Functions

As previously mentioned in section 2.2.3 the error functions need to be expressible as an average of summed error functions: $E = \frac{1}{n}\sum_x^n E_x$. This work focuses on image generation. Common approaches to calculate an image metric are the Mean Square Error (MSE) and MAE. The MSE is defined as $E(x) = (x - \hat{x})^2$ and the MAE as $E(x) = |x - \hat{x}|$, where x is the predicted value and $\hat{x}$ is the actual value. These errors can be expressed as a sum of error functions and are usable as a loss function.

2.2.7. Optimizer

The Stochastic Gradient Descent (SGD) algorithm can be seen as a group of algorithms. Their general concept is to provide a method to calculate the gradient on smaller batches, rather than the whole dataset. These batches are called mini-batches. This is

especially useful in the area of machine learning as the datasets are large and cannot be stored in memory.

The SGD algorithm generally converges to a global minimum for a convex cost function, and a local minimum is found at least in most cases [2].

However, as steep gradients exist in several functions, finding a minimum can be difficult. Even small step sizes can result in oscillations. Therefore, several concepts are researched to circumvent this.

One of the first concepts, the momentum method is introduced by Rosenblatt and Malsburg [36]. The concept is to take previous gradients into account. If a gradient points in the same direction as previously, an accelerated step size can be used. If a gradient changes its direction, the step size should be dampened [36]. This leads to a faster convergence and less oscillation. The method is defined as followed:

$$\begin{aligned} \Delta w_{i+1} &= -\eta \nabla E(w_i) + \alpha w_i \\ w_{i+1} &= w_i + \Delta w^{i+1} \end{aligned} \tag{2.9}$$

Where Δw is the weight update vector, η is the learning rate or step size and $\alpha \in [0,1)$ is the exponential decay rate [36].

Kingma and Ba [17] proposed the Adam optimizer which extends the concept of momentum. The Adam optimizer provides an adaptive learning rate for each parameter.

For this, the exponential decaying average of past gradients

$$m_{i+1} = \beta_1 m_i + (1-\beta_1) \nabla E(w_i) \tag{2.10}$$

and the exponential decaying average of past squared gradients

$$v_{i+1} = \beta_2 v_i + (1-\beta_2)(\nabla E(w_i))^2 \tag{2.11}$$

are stored.

The parameters β_1, $\beta_2 \in [0,1]$ represent the exponential decay rates for the first and second moment respectively and η represents the learning rate.

The weights are updated as followed:

$$w_{i+1} = w_i - \frac{\eta}{\sqrt{v_{i+1}} + \epsilon} m_{i+1} \tag{2.12}$$

where ϵ is a configurable parameter to counter division by zero [17].

2.2.8. Convolutional Neural Networks

CNNs are a form of neural networks which are particularly suitable for visual-based learning tasks [41, 18, 45, 11]. This is due to the convolution operation being used as a shape detection method. Therefore, various filters are created. The values of the filters are learned as weights by the network. By applying the filters to a specific pixel and its surrounding pixels, an output strength for the activation function can be calculated. The filter is then slid over the input with a stride size [20]. An example for this is visualized in fig. 2.5. The filter element in the middle is applied to all valid elements on the left. This means the filter cannot be applied to the outer border of the input. Elements on the border are thus ignored and not visible in the result on the right. The output on the right is also referred to as an activation map.

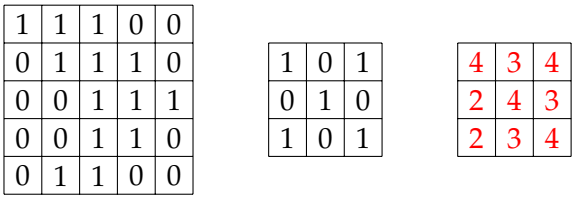


Figure 2.5.: Example of CNN filters. On the left is the exemplary image. The filter operation in the middle is performed with a stride of 1 and only on valid pixels. The result of this operation is on the right highlighted in red.

The output size $w' \times h'$ of an input $w \times h$, with w defining the width and h the height, is determined by the size of the stride and the selected padding method.

Several of these filters are created in each convolution layer. The values for the filter kernel are learned by the network and are called weights. Most CNNs decrease the size of the image in each layer with either strides or max pooling (section 2.2.5). With this method, different scales of features and shapes are learned.

By combining several layers with pooling operations, architectures such as auto encoders are created. In this encoder, a network compresses the input to a very sparse representation. An example is provided in fig. 2.6.

The input marked as red receives a 16x16 RGB image. Four convolutions are applied with a stride or max pooling of size 2. This halves the resolution in each layer. The last layer only consists of 512 outputs. By applying a fully connected layer (section 2.2.2) an output (in fig. 2.6 marked in blue) with only two classes is calculated. This is applied to multiple tasks. For example, this can be used to distinguish if an image contains a cat or dog. Another popular task is to read handwritten numbers [20]. Here, the last layer consists of ten outputs. One for each number. The highest value in the output corresponds to the detected number.

When a stride or a max pooling is applied after a convolution, it can be visualized as a downsampling operation. The information is condensed. The inverse operation

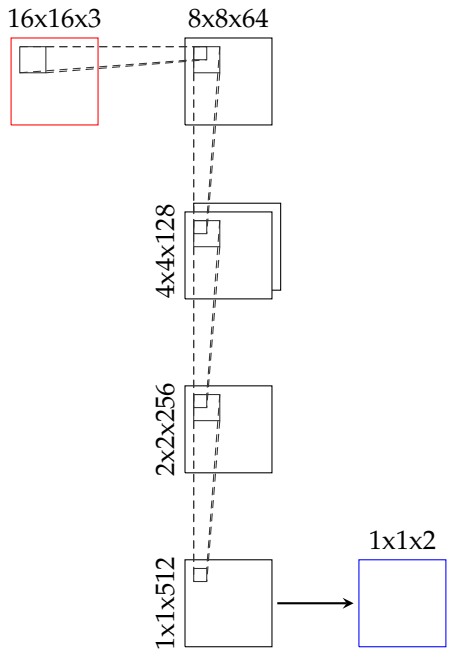


Figure 2.6.: Example of a CNN encoder.

is called a transposed convolution or often wrongly named deconvolution. A deconvolution is defined in the area of signal processing, and the transposed convolution performs a different operation. A convolution applies a kernel on the input, and the result is the output of the used filter. The transposed convolution reverses this concept. This can be done by swapping the forward and backward pass of a convolution layer (see section 2.2.3).

Finally, the concept of skip connections is an important concept in many network architectures. The skip connection appends the output of one CNN layer, referred to l_1, to the output of another layer, l_2. Here, the output size of both layers needs to be identical. For example, for a $w \times h \times c$ output the width w and height h dimensions need to match. The following layer of l_2 now filters over the joined input of l_1 and l_2. This can be helpful for larger networks, as information can flow more freely through the network.

2.2.9. Generative Adversarial Networks

One recently popular concept is GANs. They were introduced by Goodfellow et al. [9]. The general concept is that two networks learn in parallel and against each other. One network, the generator, tries to create convincing outputs. The other network, the discriminator, tries to learn if an image is from the ground truth set or if it is a generated one. As both networks learn at the same time, the discriminator tries to

achieve a better classification rate and the generator optimizes to generate better outputs. Thus, both networks learn in a competition to improve in their respective tasks. The discriminator network can be thought about as a complex loss formulation which gets trained in conjunction with the network. The training is converged if the discriminator is unable to classify the ground truth and predictions reliable. More specific, the classification rate achieves only 50% accuracy. The GAN loss is defined as:

$$\min_G \max_D L(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\log D(x)\right] + \mathbb{E}_{x\sim p_z(z)}\left[\log(1-D(G(z)))\right] \tag{2.13}$$

The general concept behind eq. 2.13 is that the loss formulation is based on the cross-entropy loss:

$$L(p,q) = -\sum_i p_i \log q_i \tag{2.14}$$

Where p represents a true distribution and q denotes an estimated distribution.

Several additions to this concept are researched recently. One of them is the concept of conditional GANs (cGAN) [9]. Here the discriminator not only learns whether the output is from the ground truth data, but whether the output is viable for the input. This is done by passing the discriminator network pairs of input and output images.

Radford et al. [31] introduced a set of rules for the generator network to stabilize the learning process for GANs. This architecture is called Deep Convolutional Generative Adversarial Network (DCGAN). The generator and discriminator networks are based on convolution layers. The other guidelines are:

1. Only used strides instead of pooling operations
2. Apply batch normalization in the generator and discriminator
3. ReLU activation function in all layers of the generator except the output which uses tanh
4. Leaky ReLU in all layers of the discriminator

Additionally, Radford et al. [31] propose the use of the Adam optimizer with a learning rate of 2e−4 and a β_1 of 0.5 to stabilize the training.

2.3. Image Synthesis

One of the central concepts of computer-guided image synthesis is rendering. Here a brief overview of the rendering equation, BRDFs, the analytical Cook-Torrance BRDF model and the evaluation of physically based point light sources is discussed.

2.3.1. Rendering Equation

Kajiya [16] defines the rendering equation as:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o) L_i(x, \omega_i)(\omega_i \cdot n) d\omega_i \tag{2.15}$$

Where L_o is the total out-going radiance for the point x with the direction ω_o, L_e is the emitted radiance, and L_i is the incoming radiance at the location x with the negative incoming light direction ω_i. The f_r term describes the BRDF. Lastly, the $(\omega_i \cdot n)$ is the cosine of the incident angle, which acts as the weakening factor for the incoming light. This is a consequence of light being distributed over a small area when it hits the surface perpendicular and over a larger area for non-perpendicular angles.

2.3.2. Bidirectional Reflectance Distribution Function

The BRDF represents the reflection behavior of surfaces under arbitrary incident and out-going angles. The BRDF can range from its purest four-dimensional form to a higher multi-dimensional function. The four-dimensional form of a BRDF is defined only for all incident and outgoing angles. This can be extended to, for example, six dimensions by adding spatial variance over the surface of the sample. Other extensions are the wavelength of the light or the time.

The BRDF is defined as the ratio between the reflected radiance L_r in direction ω_r and the incoming radiance L_i from direction ω_i [28]. This is expressed as:

$$f_r(x, \omega_i, \omega_o) = \frac{dL_r(\omega_o)}{L_i(\omega_i) \cos(\omega_i \cdot n) d\omega_i} \tag{2.16}$$

To achieve energy conservation the following equation applies:

$$\int_\Omega f_r(x, \omega_i, \omega_o)(\omega_i \cdot n) d \leq 1 \tag{2.17}$$

Measuring, storing and evaluating BRDFs is complex. In theory, a measurement for every possible viewing and lighting angle needs to be taken. In case of a spatially varying BRDF, the values for each point on the surface need to be stored. When the material is then, for example, rendered by ray tracing, the renderer needs to look up the stored value for the ray's incident and reflection angle. The time to capture a BRDF is therefore quite long. For example, the acquisition time for a single sample from the MERL dataset takes three hours [25]. The amount of data stored to represent one sample is significant. For the MERL dataset, a single sample takes 330 high dynamic range images, which are combined from 18 10-bit images [25]. Thus,

many analytical models have been proposed which reduce the file sizes for each material.

A popular model for approximating the behavior of specular highlights is the Cook-Torrance microfacet model [4]. The general concept behind this model is to describe microstructures in a statistical process, which is driven by a roughness parameter. This parameter describes how many microfacet normals are aligned with the surface normal. The microfacets itself are approximated as a small perfect mirror. For perfectly smooth surfaces the reflection of a light source is visible as a sharp image of the light source. All microfacets are aligned with the surface normal. However, most objects are not perfect mirrors and have tiny imperfections on the surface. The resulting reflection is blurred. The microfacet normals are now distributed around the surface normal. For example, fig. 2.7 visualizes these imperfections and highlights the microfacet normals h.

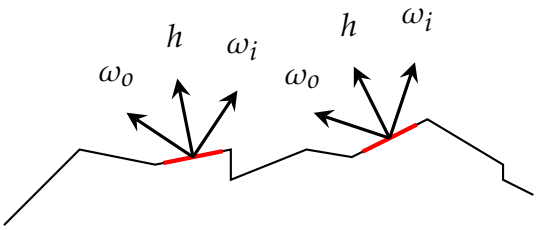


Figure 2.7.: Microfacet visualization. The vector h is the half vector between ω_i and ω_o. It is additionally the microfacet normal in this case. If the half vector is aligned with the microfacet normal a reflection is visible.

In general, a highlight is achieved when the half vector between the incident and the outgoing ray is aligned with the surface normal. A material with a small roughness value has a narrow normal distribution function. Most of the microfacet normals align with the surface normal. A rough material has a large roughness value, and the normal distribution function is broad. The microfacet normals are scattered around the surface normal direction. Thus, only some of the light rays are reflected back to the eye. This cause highlights to blur and produce a less sharp highlight.

The Cook-Torrance microfacet is defined as:

$$k_{\text{spec}} = \frac{D(\alpha, \omega_i, \omega_o, n) F(\omega_i, \omega_o) G(\alpha, \omega_i, \omega_o, n)}{4(\omega_o \cdot n)(n \cdot \omega_i)} \tag{2.18}$$

Where D is the normal distribution function, F is the Fresnel term, G is the geometric attenuation term, and α is the roughness parameter.

The normal distribution function describes how many of the microfacet normals are facing in the current half vector direction. Therefore, this function defines the shape and size of the highlight. Figure 2.8 visualizes the behavior of the normal distribution function with different roughness values.

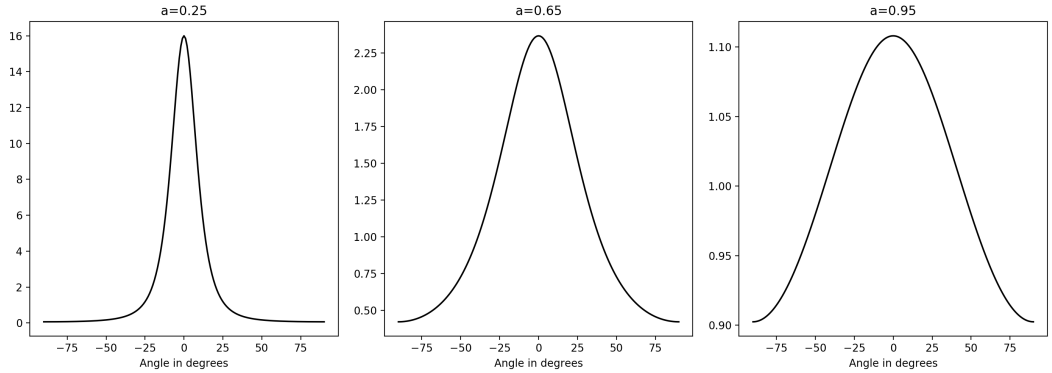


Figure 2.8.: Example for the normal distribution function with different roughness values. The variable a represents α here.

The Fresnel term represents the amount of light that is reflected rather than refracted for a given material.

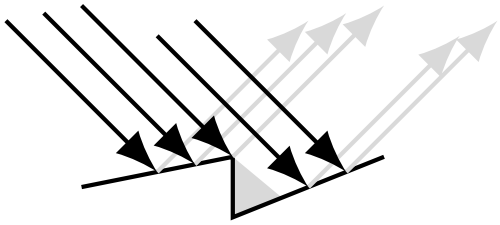


Figure 2.9.: Microfacet shadowing visualization. Incoming arrows represent light rays and outgoing arrows represent the reflected light. Some facets produce a shadow.

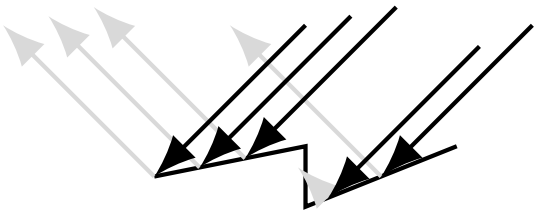


Figure 2.10.: Microfacet masking visualization. Incoming arrows represent light rays and outgoing arrows represent the reflected light. Some facets block the reflected rays.

Lastly, the geometric attenuation preserves the energy conservation. Due to the microfacets, some areas are self-occluded. This is visualized in fig. 2.9 and is referred to as shadowing. Additionally, some areas are not visible from the view direction. This is visualized in fig. 2.10 and is referred to as masking.

Several functions and approximations can be used to describe the normal distribution function, Fresnel term, and geometric attenuation function. In section 3.2 the choice for each of these functions is explained in more detail.

The original formulation of the Cook-Torrance microfacet model only describes the specular reflectance lobe of the BRDF. However, an extension is also mentioned which adds a diffuse component [4].

$$k_{\text{diff}} = \frac{1}{\pi} c_{\text{diff}} \tag{2.19}$$

In eq. 2.19 c_{diff} is the RGB value of a single pixel from the diffuse color map, which has a range between 0 and 1.

A more complex model is the diffuse term of the Disney BRDF [3]. This term takes into account that rougher materials are often slightly darker than smoother ones. It is defined as:

$$\begin{aligned} k_{\text{diff}} = & \frac{c_{\text{diff}}}{\pi}(1 + (F_{D90} - 1)(1 - \cos\theta_l)^5)(1 + (F_{D90} - 1)(1 - \cos\theta_v)^5) \\ & \text{with } F_{D90} = 0.5 + 2\cos\theta_d^2\alpha \end{aligned} \tag{2.20}$$

In eq. 3.7 c_{diff} is the RGB value of a single pixel from the diffuse color map, α represents the roughness value with range 0 to 1, θ_l and θ_v are the angles of incidence for the light and view ray, respectively, and θ_d is the difference between the angle of the light and the half vector. The half vector is defined as $h = \frac{l+v}{\|l+v\|}$.

The specular and diffuse terms are combined in eq. 2.21.

$$f_r = (k_{diff} + k_{spec}) \tag{2.21}$$

2.3.3. Point Light Evaluation

A point light is the most straightforward definition of a light source. The point light source is defined by a single point in space and an intensity value I_e. It emits the same amount of light in every direction.

As the parametrization of the point light source uses the radiant intensity as the unit, it needs to be converted to radiance unit to be used in the rendering equation. This conversion is done by dividing with the squared distance from the light source at the position y to the point x on the surface.

The light source is thus defined as:

$$L_i = \frac{I_e}{\left\|y - x\right\|^2} \tag{2.22}$$

3. Implementation

This section covers the first steps toward the usage of real-world acquired samples with the light stage in section 3.1. Additionally, the implementation details for the BRDF model described in section 2.3 are discussed in section 3.2 and the dataset for the neural network is described in section 3.3.

Furthermore, two neural network versions are proposed in section 3.4 and section 3.5.

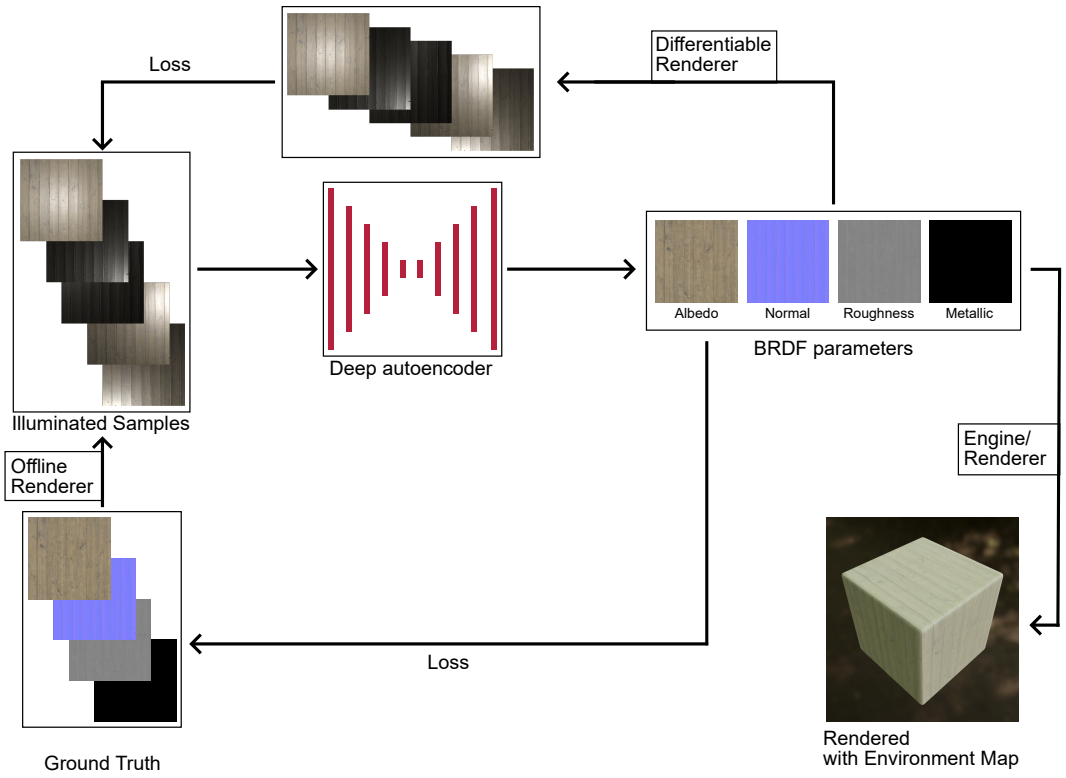


Figure 3.1.: Abstract overview of the proposed CNN.

The first version is an encoder-decoder CNN. An abstract overview is visualized in fig. 3.1. The specific details are described in section 3.4. As the proposed system is currently evaluated on a synthetic dataset, the samples are first rendered offline. The illuminated samples are then passed to the encoder-decoder CNN. The BRDF parameters for the Cook-Torrance model described in section 3.2 are then predicted. The predicted parameters are compared to the ground truth dataset samples. Addi-

tionally, the samples are rendered with a differentiable renderer and compared to the illuminated input samples. By using a differentiable renderer, the error from this comparison can be propagated back to the network. Finally, the predicted parameters can be used in an arbitrary engine or renderer under any lighting condition.

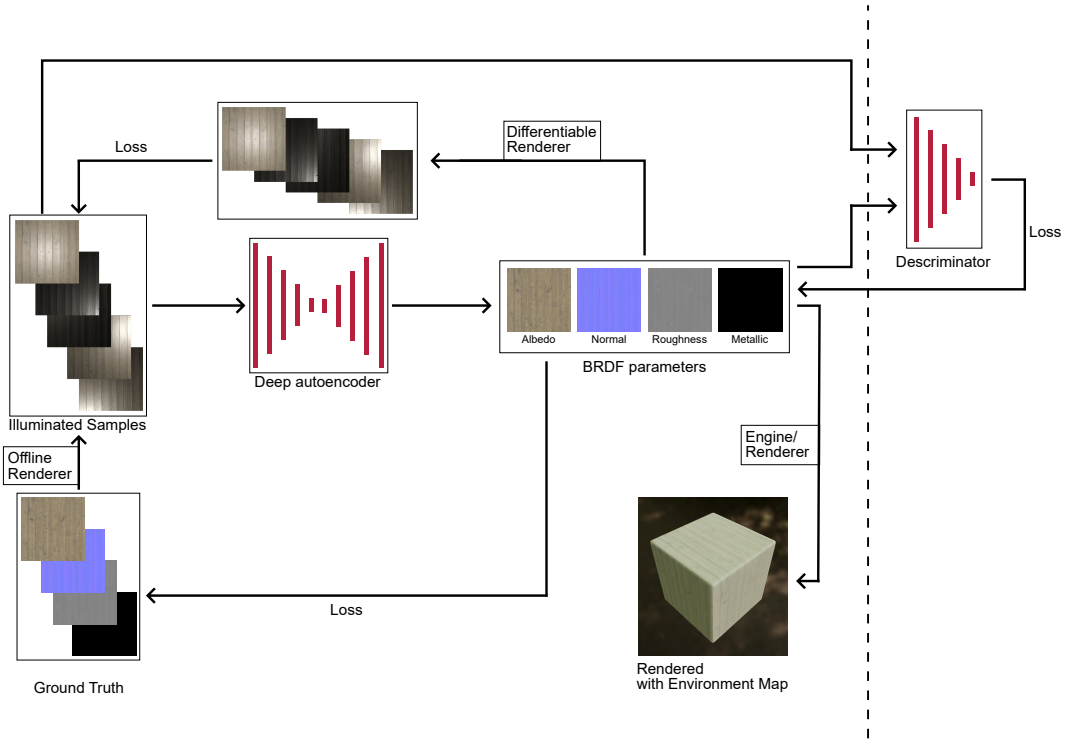


Figure 3.2.: Abstract overview of the proposed GAN.

The proposed CNN version is now extended to a GAN. Figure 3.2 provides an abstract overview. Detailed information are described in section 3.5. The general concept is identical to the CNN version. However, in fig. 3.2 the dashed line indicates the extension of the discriminator network. This discriminator network is trained to evaluate if a prediction is possible given the network input. This decision is added as an additional loss. Section 3.5.2 describes the exact definition of the loss.

3.1. Camera Pose Estimation

As a BRDF describes the behavior of a surface for different viewing angles, reflection angles and wavelengths, information about the camera pose, lighting position and light wavelengths are required. To simplify the BRDF estimation only white LEDs are used in the light stage (section 2.1). The LED positions are known, and the cameras are calibrated. The remaining unknown component is the camera pose in the LED coordinate system.

The camera pose can either be measured by an external device or estimated by using available information in the camera frame. External measurement requires additional hardware, and the resulting position would be in an own coordinate system. However, this would require alignment with the LED coordinate system. Another method is using markers on the gantry of the light stage. But as the sample is often lit with just one LED, the camera needs a wide aperture to maximize gathered light. This results in the background, where the LEDs are located, to be blurred. Tracking out of focus markers is error-prone, and thus this method is not feasible as well. Instead of markers, the visible, active LEDs in the frame can be used to estimate the camera pose. As white LEDs are behind a small diffusion lens, the visible highlights of the active LEDs are large. By capturing High Dynamic Range (HDR) images, the highlights of the lit LED are not blown out, and a brighter center is visible. This center is used for the position estimation.

By using following equation, sub-pixel accuracy is achieved:

$$\omega = \sum_{x}^{n} \sum_{y}^{m} p_{x,y} \tag{3.1}$$

$$c = \frac{\sum_{x}^{n} \sum_{y}^{m} \begin{pmatrix} x \\ y \end{pmatrix} p_{x,y}}{\omega} \tag{3.2}$$

In eq. 3.1 ω is the summed weight for each pixel coordinate x, y and its value $p_{x,y}$. The two-dimensional (2D) center position is calculated in eq. 3.2, where c is the center, x, y are the pixel coordinates of the n x m sized frame and $p_{x,y}$ is the pixel value again.

To automate this process, the highlights can be extracted automatically. When only a single LED is active and visible in the image, the brightness difference to all other visible elements in the frame is large. Binary thresholding can be applied with an empirically selected threshold value. The resulting thresholded image is then cleaned up with an erosion operation followed by a dilation. The resulting image is now sparse, and a labeling algorithm is executed to extract all active elements of the thresholding. The center of each label can now be calculated with eq. 3.2. This process is visualized in fig. 3.3.

With three well-distributed correspondences, a Perspective-n-Point algorithm can be used to estimate the camera pose. OpenCV implements such an algorithm [1]. However, the camera is configured with a smaller field of view (see the leftmost image in fig. 3.3) because this maximizes the size the sample covers in the frame. As a consequence, only a few LEDs are visible in the image. Additionally, the distribution of these visible

[1] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

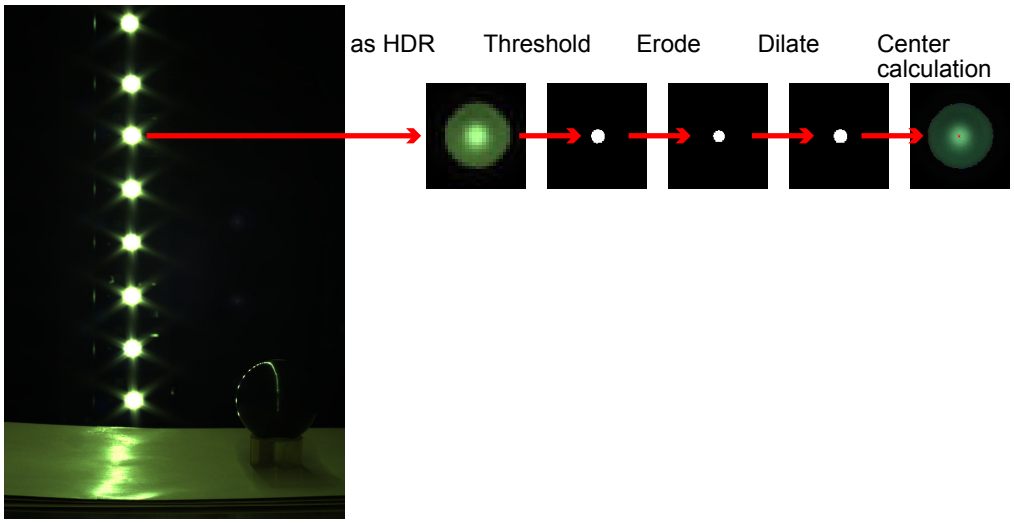


Figure 3.3.: Example for the camera pose estimation pipeline.

LEDs is just in a line. A robust position estimation requires spatially distributed correspondences. Thus, a Perspective-n-Point algorithm cannot be applied directly. To bypass this problem an additional step is introduced. A sphere with a known radius is placed at the sample location. Single LEDs which produce a visible reflection from the camera perspective are activated, and HDR images are captured. Afterward, the center is calculated with the method described previously.

Sometimes additional reflections are visible in the background on reflective elements (see fig. 3.4). The unwanted reflections are not visible in every image, and the location of them are scattered in the frame. By gathering all visible reflection center locations from all images, a true center is calculated. A median is calculated for every reflection coordinate in the x and y-axis independently. With sufficiently many captured images this corresponds to the center of the sphere. The distance between the sphere center and each visible reflection in every image is calculated, and only the reflection with the closest distance is considered. All other reflections are disregarded.

The next step is to fit the visible reflections on a simplified three-dimensional (3D) model of the Lightstage. The known parameters are:

- The sphere radius
- The 3D LED location
- The visible 2D reflection points on the sphere

The missing parameters are:

- The camera position (3D vector)
- The camera rotation (represented as 3D Euler angles vector)

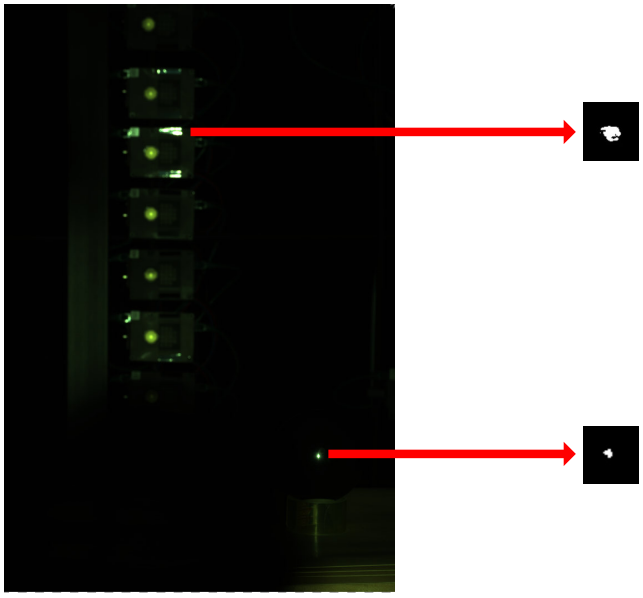


Figure 3.4.: Example for multiple highlights in the image. After the thresholding operation a second highlight is still present. The lower right highlight is the wanted highlight and the one highlighted above is an unwanted.

- The sphere position (3D vector)

Additionally, a model needs to be established which sets all parameters in relation. The result is a nonlinear equation with nine unknown parameters. The model calculates an error, and this error is minimized, which is known as optimization. This is done numerically by Ceres [2]. The Ceres solver is a nonlinear optimization library with support for unconstrained and bound constraint optimization.

A previously unmentioned parameter is the 3D reflection on the sphere. This parameter is estimated in a separate optimization step inside the Ceres optimization. The reflections on the sphere are highly specular, and the center is treated as a perfect mirror between the view direction and the LED position. Additionally, the sphere is treated as a level surface at the point of the reflection.

Figure 3.5 illustrates the model. The reflection half vector $H(\vec{\omega_i}, \vec{\omega_o})$ is calculated as followed:

$$H(\vec{\omega_i}, \vec{\omega_o}) = \frac{\vec{\omega_i} + \vec{\omega_o}}{\left\|\vec{\omega_i} + \vec{\omega_o}\right\|} \tag{3.3}$$

When a perfect reflection is occurring the surface normal is equal to the half vector $H(\vec{\omega_i}, \vec{\omega_o})$. To calculate the surface normal, the reflection point $\vec{p}$ and the sphere center

[2] `http://ceres-solver.org/`

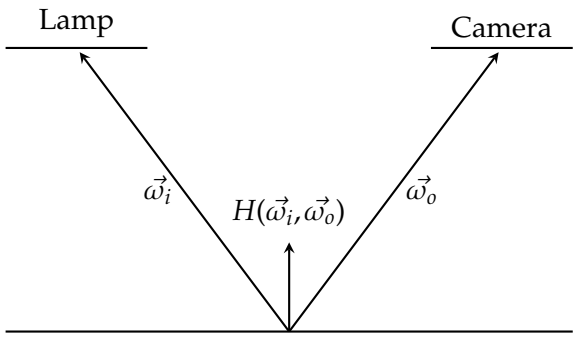


Figure 3.5.: Diagram of the reflection on the sphere.

$\vec{c}$ is used as followed:

$$\vec{n} = \frac{\vec{p} - \vec{c}}{\left\|\vec{p} - \vec{c}\right\|} \tag{3.4}$$

This leads to the first constraint of the optimization: $H(\vec{\omega_i}, \vec{\omega_o}) = \vec{n}$.

A second constraint is that the reflection needs to be on the surface of the sphere. For every point (x_0, y_0) the equation $(x_0 - c_x)^2 + (y_0 - c_y)^2 = r^2$ applies. With (c_x, c_y) representing the circle center in x- and y-axis and r representing the radius.

Constraint optimization is a specific optimization problem and is solved here with nlopt [3] and the Constrained optimization by linear approximation (COBYLA) algorithm [15]. The nlopt library is a nonlinear optimization library which implements a large number of different algorithms and is callable from various languages such a C, C++, Julia and Matlab. Ceres is not used for the sphere position estimation because, at the time of writing, it does not support equality constraint optimization. The nlopt library, on the other hand, supports equality constraint optimization with the COBYLA algorithm.

It is worth mentioning that the above optimization is executed in 2D, but the camera pose needs to be optimized in three dimensions. As the reflection has to be on a plane with the camera and the light, all coordinates can be projected on this plane. When the resulting plane is then transformed on an axis in the coordinate system of the LEDs, one coordinate can be discarded. As the plane and the transformation to the axis is known, the reflection can then be projected back to 3D.

The resulting 3D point is then transformed to 2D coordinates by the known intrinsic of the calibrated camera. The difference between the real recorded reflection coordinates and the estimated coordinates is calculated and used as the error in Ceres. Ceres

[3] `https://nlopt.readthedocs.io/en/latest/`

then minimizes the error by altering the unknown parameters (Camera position, rotation, sphere position).

With sufficiently many images containing reflections distributed on the visible surface of the sphere, the optimization results in a stable and accurate position.

3.2. Bidirectional Reflectance Distribution Function Rendering

In this section, the specific implementation details are discussed. In section 2.3 the Cook-Torrance model is described. However, the exact definition of the required normal distribution function, Fresnel term, and geometric attenuation function is left for this section. Furthermore, the exact details of the rendering implementation are presented here.

BRDF implementation details

The concept of Cook-Torrence microfacet model is introduced in section 2.3.2. The specific implementations for the normal distribution function, Fresnel term, and geometric attenuation are discussed in this section. To achieve a quick evaluation of the BRDF, common approximations for these functions are selected. The implementation is based on the Frostbite Engine [19].

As mentioned in section 1.1, this project aims to allow BRDF parameter estimation from input images with a neural network. To achieve this, the number of different materials is limited. As a limitation, only isotropic materials are estimated. Additionally, the choice of using the Cook-Torrance model is inherently a limitation, as the possible parameters compared to a full recorded BRDF like the MERL dataset are drastically limited [25].

The samples gathered from several resources for the dataset in section 3.3 are defined with the following parameters:

b **- Base color:** RGB - Values: $[0-1]$

n **- Normal:** 3D scalar - Values: $[0-1]$

r **- Roughness:** Scalar - Value: $[0-1]$

m **- Metallic:** Scalar - Value: $[0-1]$

The base color map represents the general color of an object without lighting information. The normal map encodes small surface normals in an image by mapping a normal vector (XYZ) to RGB values in an image. As an 8-Bit RGB image cannot capture the negative values, the value n is scaled by $2n-1$. The roughness map approximates even smaller scale surface irregularities. In the grayscaled map a

value of 1 is a rough surface, and 0 is a perfect mirror surface. This information is used in eq. 2.18 and results in smoother, diffuse highlights for a rough surface and sharp, perfect reflections for a smooth surface. The metallic map is used to determine the specular and diffuse color. Ideally, only 0 or 1 is used as a value for the metallic map, but sometimes values in between are necessary to express coated or rusted metals.

The roughness value is reparametrized as $\alpha = r^2$. This perceptually linearizes the roughness value [3].

In this definition, the base color map is not technically a diffuse color representation but a mix of the diffuse color for non-metallic objects and the Fresnel reflectance at normal incidence (f_0). The metallic map is used as a mask to determine which areas are metallic or non-metallic. This definition is often referred to as a base color texture map.

For non-metallic materials a base specularity is assumed. Often, a value of 0.04 or 4% reflectivity is assumed. The diffuse color c_{diff} and specular color c_{spec} is then defined as:

$$c_{\text{diff}} = (1-m)\,b \tag{3.5}$$
$$c_{\text{spec}} = (1-m)\,0.04 + mb \tag{3.6}$$

The diffuse term is defined as:

$$\begin{aligned} k_{\text{diff}} &= \frac{e_f\, s_l\, s_v\, (n \cdot \omega_i)\, c_{\text{diff}}}{\pi} \\ e_b &= 0.5\,\alpha \\ e_f &= (1-\alpha) + \alpha\,\frac{1}{1.51} \\ F_{D90}(\omega_o, h) &= e_b + 2(\omega_o \cdot h)^2\,\alpha \\ s_l &= (1-(1-n\cdot\omega_i)^5) + (1-n\cdot\omega_i)^5 F_{D90} \\ s_v &= (1-(1-n\cdot\omega_o)^5) + (1-n\cdot\omega_o)^5 F_{D90} \end{aligned} \tag{3.7}$$

This implementation contains a renormalization term which prevents the diffuse term to exceed 1. This can occur due to the Disney BRDF lacking energy conservation [19]. According to Burley, this is an intentional design choice. With this choice, artists can use the same diffuse color across all roughness values. The e_f variable defined in eq. 3.7 is responsible for the conversation of energy [19]. The scaling factor $\frac{1}{1.51}$ is due to the peak value of 1.5 in the Disney BRDF. The s_l and s_v variables are the Schlick approximation of the Fresnel term for the light and view direction, respectively [19].

For the specularity, the Cook-Torrance model from eq. 2.18 is used. For the Fresnel term F the Schlick approximation [38] is used:

$$f_{90} = \arg\max_{1} \arg\min_{0} \frac{(c_{\text{spec}} \cdot 0.3\vec{3}333)}{0.02}$$
$$F(\omega_i, h) = (1 - (1 - \omega_i \cdot h)^5)\, c_{\text{spec}} + (1 - \omega_i \cdot h)^5\, f_{90} \tag{3.8}$$

In this equation, h refers to the half vector between ω_i and ω_o. Furthermore $0.3\vec{3}333$ describes a three-dimensional vector with all elements being 0.33333. The definition of f_{90} contains an assumption from Engel [7, chap. 2.5] that no real-world material has a f_0 value lower than 0.02. It is assumed that values lower than this value are due to micro-occlusions, and thus the Fresnel value is gradually lowered [7, 19]. These micro-occlusions can be encoded in the specular value by artists. As no material in the dataset (see section 3.3) contain such values the term could be removed. However, as the dataset contains a large range of different materials from different artists and is intended to be extended, it is not removed.

For the normal distribution function, the GGX distribution function is used [47]. It is defined as:

$$D(\vec{n}, h, \alpha) = \frac{\alpha^2}{\pi((\vec{n} \cdot h)\alpha^2 - (\vec{n} \cdot h))(n \cdot h) + 1)^2} \tag{3.9}$$

Lastly, the height correlated smith version for the geometric attenuation function is selected [12]. According to Heitz, this is a correct approximation of microsurfaces, as it takes the height of the microfacets into account. It is defined as:

$$G(\omega_i, \omega_o, \vec{n}) = \frac{g_v + g_l}{2} \tag{3.10}$$
$$g_v = \vec{n} \cdot \omega_i \sqrt{(-(\vec{n} \cdot \omega_o)\, \alpha^2 + (\vec{n} \cdot \omega_o))(\vec{n} \cdot \omega_o) + \alpha^2}$$
$$g_l = \vec{n} \cdot \omega_o \sqrt{(-(\vec{n} \cdot \omega_i)\, \alpha^2 + (\vec{n} \cdot \omega_i))(\vec{n} \cdot \omega_i) + \alpha^2}$$

It is worth noting that this definition is a reformulation of the original definition of Heitz. This definition already contains the denominator for the Cook-Torrance equation eq. 2.18. Combining eq. 3.8, eq. 3.10 and eq. 3.9 results in k_{spec}:

$$k_{\text{spec}} = FGD \tag{3.11}$$

Renderer

The implementation of the online training/validation data rendering is done in Rust[4]. Rust can expose a C interface which can be accessed by Python and most other languages. This implementation is referred to as the Rust renderer.

The output of the renderer implementation is tested against the Mitsuba renderer. However, the Mitsuba renderer does not support the metallic model natively but supports rough conductors and a rough plastic model. These are comparable to metallic and non-metallic surfaces in the Rust renderer, respectively. Both models are then blended using the Blendmap plugin and the metallic map as the blend texture. For the rough plastic model, an interior index of refraction of 1.50043 is chosen as this represents the 4% base reflectivity of the above-described model. In fig. 3.6 the difference between the Mitsuba renderer and the Rust renderer are compared. Several approximations cause the small differences in the Rust renderer.

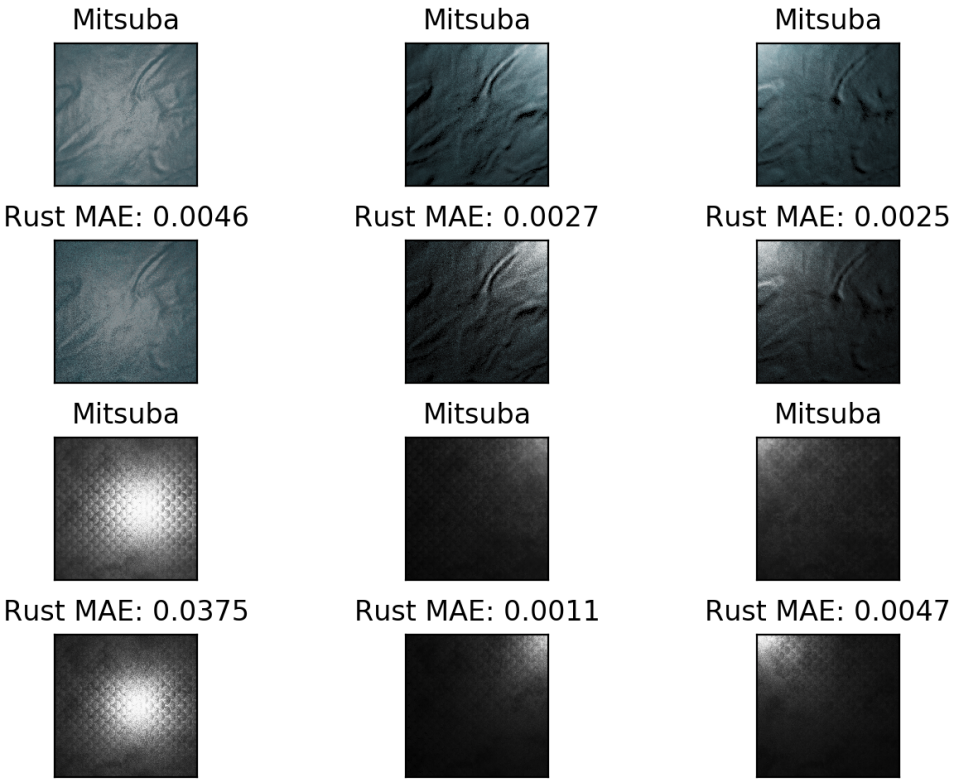


Figure 3.6.: Comparison between Mitsuba and the Rust implementation. The MAE for each image is annotated for the Rust renderer.

The rendering is additionally done inside of the network. Here, it is implemented with Tensorflow operations in python. The Tensorflow operations contain information about their gradients. This allows for a comparison of rendered images instead of

[4] `https://www.rust-lang.org`

just the texture maps for the error calculation. Thus, the error can be traced back to the predicted texture maps. Label-free training is a possibility with this system. It is worth noting that this is only possible because no light bounces are calculated. Additionally, flat samples from a perpendicular position to the surfaces are captured. This viewing angle prevents occlusion on surfaces. Both light bounces and occlusion prevent a proper derivation of the render equation. By using only one light source, the derivation is simplified compared to other setups.

It is either recommended to limit the network output for the predicted base color and roughness maps to $(0-1]$ or adding several division checks which prevent a division by zero.

3.3. Dataset

The dataset consists of 415 individual samples. The samples are taken from royalty-free assets from artists on `https://gumroad.com/`, `https://freepbr.com/`, `https://www.3d-wolf.com/` and `http://poliigon.com/`. Each sample consists of a base color, normal, roughness, and a metallic map.

3.3.1. Augmentation

As 415 samples is a small dataset for machine learning, several preprocessing steps are applied to increase the number of possible samples. Most maps are between a 2048×2048 and 5120×5120 pixels resolution. The network is trained with 512×512 pixels resolution images. In consequence, several samples can be generated from each sample.

The processing starts by taking a random square section of the image. The clipping is 2.5 times the size of the network input size. With the desired size of 512×512, the section needs to be 1280×1280 pixels. The cropped image is then rotated randomly, and the largest possible square is extracted from the resulting image again. The result is then resized with bi-cubic interpolation to the desired 512×512 pixels. The base color and roughness maps are then processed independently. For the base color map, a transformation to the Hue Saturation Value (HSV) color model is used to change the hue, saturation, and value randomly. The hue is limited to vary from -0.1 to 0.1 with a wrap around to 0 for values larger than 1 and 1 for values smaller than 0. The lightness and saturation are changed in a range from -0.2 to 0.2. Values are clamped to 0 to 1. Lastly, a contrast change is applied. However, these changes are only used if the metallic map indicates a non-metallic material. Otherwise, invalid values for the Fresnel term will be calculated. For the roughness map, a random value change between -0.4 and 0.4 is added, and the result is clamped to 0 to 1. The contrast is altered for the roughness map, too.

Each sample is processed ten times by this pipeline. This increases the training size from 415 sample to 4150 samples. 80% of the set are used for training, and the remaining 20% are for the validation step. Resulting in 3320 for the training and 830 images for the validation set.

With this preprocessing pipeline, the range of possible materials is significantly increased. The changes in possible roughness values are necessary for the network to learn specular behaviors.

3.3.2. Rendering

Currently, the rendering is based on a simple spherical geometry model. In future work it is recommended to take the exact geometry of the light stage into account (see section 2.1 and section 4.2).

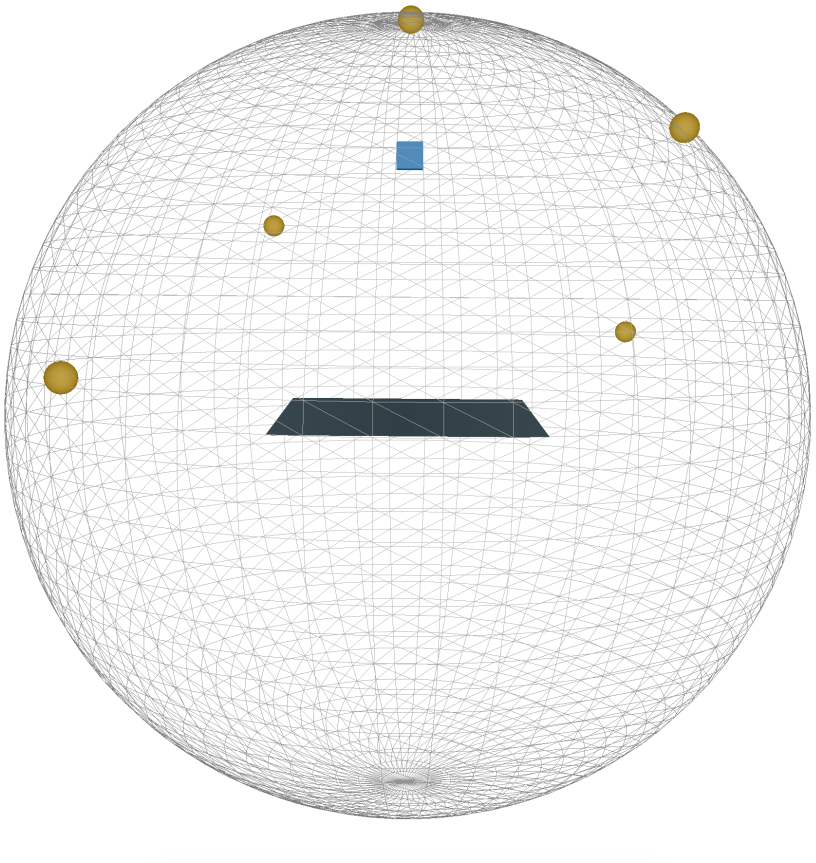


Figure 3.7.: Example of the current rendering geometry setup. The yellow sphere represent light positions. The blue cube represents the camera position and the gray surface is the sample.

The current geometry is visualized in fig. 3.7. The sample is 1×1 unit large and is centered in the origin of the coordinate system. The camera is placed 1 unit away from the sample and is directly perpendicular to the surface. The camera position is $(0,0,1)$.

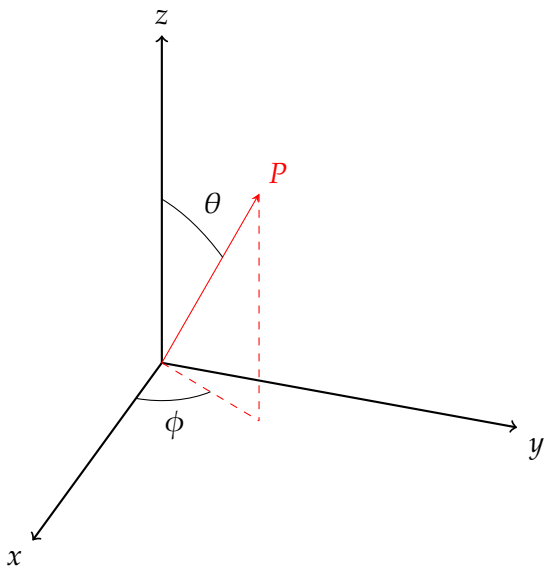


Figure 3.8.: The spherical coordinate system.

The five light source are positioned on a sphere with a radius of 1.5 units. The light positions are defined in spherical coordinates (θ, ϕ). Figure 3.8 visualizes the spherical coordinate system. The coordinates in degree are: $(0,0)$, $(80,45)$, $(80,225)$, $(45,330)$, $(60,120)$. These positions are selected empirically and should be replaced with the actual positions of the light stage setup (see section 2.1) in future iterations.

(a) Highlight surface normals **(b)** Highlight the base color

Figure 3.9.: Example for different light positions and resulting images.

The choice for the light angles is because of the first position $(0,0)$ providing flat

lighting without occlusions and shadowing. This light angle provides most of the information for the base color map and is visualized in fig. 3.9b. The other light angles are located at a flatter angle towards the surface. This produces visible information for the shadowing and general reflectance behavior of the surface. Figure 3.9a displays the visible structure of the surface.

The rendering can create a range larger than 0 to 1 output values due to harsh direct reflections. The large value range provides a problem while training the network. A normalization step is required to provide a roughly equal value range for the network input. Local tone mapping operators like the method by Reinhard et al. [33], defined as $x = \frac{x}{1+x}$, provide a way to reduce the range of an HDR image to $[0-1]$ [33]. However, diffuse, rough materials and highly glossy materials are now in the same value range. This obfuscates information about the specular material from the input. An improved approach is to use a function which can be applied globally for all samples. One solution is to clip the images to a 0 to 1 range. Information in the highlights is completely removed in this method. Another solution is to apply a function which reduces the range significantly like $x = \log(x+1)$. The variation can still be substantial after the log transformation, but a clipping operation can now be applied afterward. The highlights still lose information. However, due to the previous range reduction only particular highly specular materials are affected. This method provides the best results and is thus used for the network training.

3.4. Convolutional Neural Network

In this section, the process of designing the architecture of the Multi-Input-Multi-Output-Convolutional-Neural-Network (MIMOCNN) is discussed.

3.4.1. Architecture

The general idea behind most of this network is an encoder-decoder structure. In this architecture, the network first compresses and distills the general information about the BRDF and then decodes this information into spatially varying texture maps.

Preliminary tests use the aforementioned encoder-decoder setup which consists of convolutions and transposed convolutions (see section 2.2.8). However, the decoder is not capable of producing visibly pleasing output maps due to information about details getting lost. The U-Net architecture proposed by Ronneberger et al. utilizes skip connections between matching layers [34]. Figure 3.10 visualizes this structure. Skip connections are concatenations of tensors in a specific dimension. Mostly, the dimension of the channels is used for this. For example, the output from the first convolution, marked in blue, is added to the output of the third transposed convolution, marked in green. The result of the concatenation is a vector with the

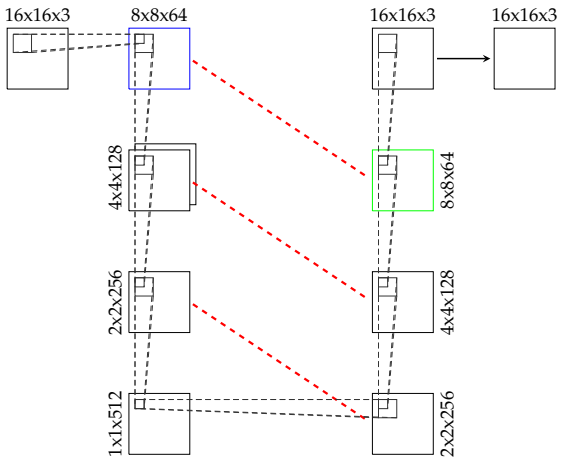


Figure 3.10.: Example of a U-Net architecture. The dashed red lines represent a skip connection.

shape $8 \times 8 \times 128$. The advantage of skip connections is allowing information from layers with less spatially compression being added back in the decoding step. The visible detail of the resulting output is greatly increased.

However, the U-Net architecture is intended for a single input image and a single output image. The input, on the other hand, consists of five images with different lighting conditions. A first approach is to join the different images in the RGB channel dimension, resulting in a single input of shape $512 \times 512 \times 15$. The output can be defined in the same way. The base color and normal maps consist of three channels and the roughness, and metallic maps use one channel each. This results in eight output channels. The general architecture is visualized in fig. 3.11. The Leaky ReLU [23] activation function is used throughout the network except for the last transposed convolution. As the input range of the images is between 0 and 1, the output should be in the same range. The sigmoid activation function fulfills this requirement and is used here. A batch normalization [13] is used throughout the entire network to allow for higher learning rates and improve the network initialization. A dropout [44] regularization is applied in the first four layers of the decoder network. Another change to the U-Net architecture is applied at this point. The original U-Net uses max pooling where this version uses strides [43]. This increases the training speed and convergence rate significantly. Additionally, the memory usage while using strides is drastically lower. The results are neither perceptually nor in the MAE loss worse. As the network contains a large number of weights due to the large input size, the reduction in memory usage is especially helpful and allows for higher batch sizes. Lastly, a kernel size of three is used for convolutions and transposed convolutions. The first tests do not provide satisfying results, and the number of filter outputs for each convolution needs a significant increase. Due to the size of the network, this approach is discarded and is not further explored.

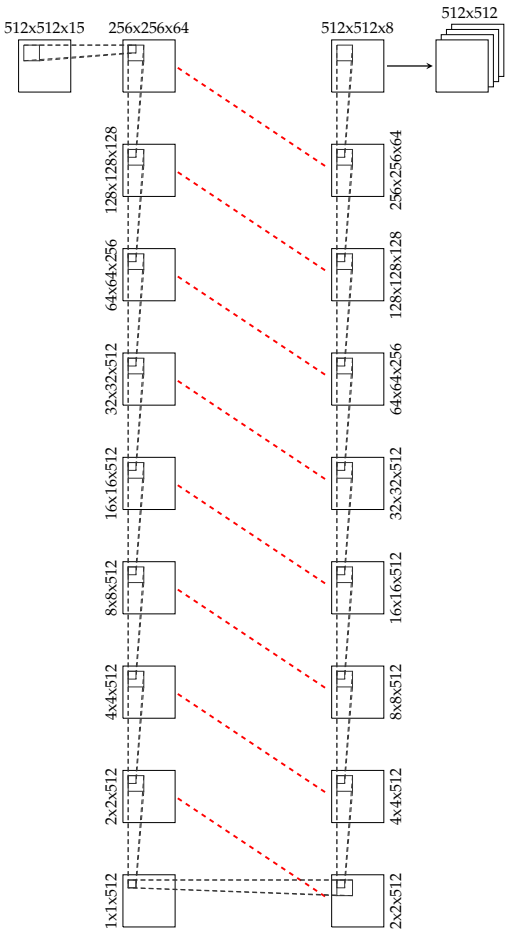


Figure 3.11.: Diagram of the first Multi-Input-Multi-Output-Convolutional-Neural-Network. Based on the U-Net architecture [34]. The kernel size for the convolution and deconvolution is three. Instead of a max pooling a stride is used. The dashed red lines represent a skip connection.

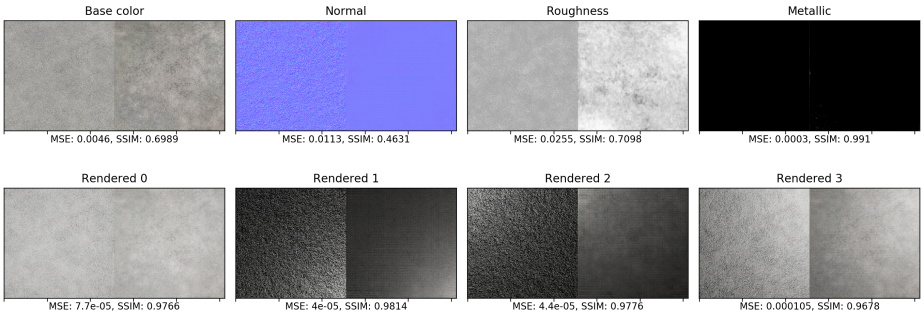
(a) Base color **(b)** Normal **(c)** Metallic **(d)** Roughness

Figure 3.12.: Results of the first CNN version. Notice the lighting information in the base color map and the central artifact in the normal map. Additionally, the roughness map is noisy and does not contain plenty of detailed information.

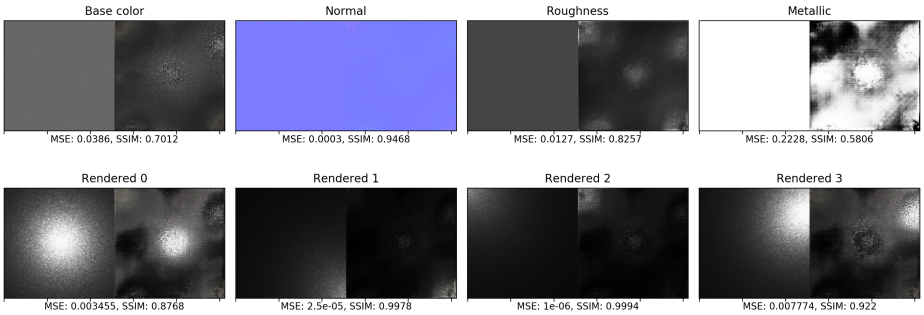
A second approach uses the U-Net architecture, but the input images are stacked in a new dimension. This dimension is further referred to as the depth. The output of the network does not utilize the depth dimension and produces an output with eight channels. The first layers of the network now utilize 3D convolutions with a stride of two in every dimension. Due to the stride, the width, height, and depth dimensions are halved in every step. Thus, the five input images are merged in three layers. As the skip connections from the initial layers now have an additional depth dimension, each dimension is individually appended to the matching transposed convolution result. The results of this architecture are visible in fig. 3.12. In specular materials, a bright spot from the perpendicular light direction (see section 3.3.2) is particularly noticeable in the base color map.

As the bright spot in the center may be a leftover due to the quick merging of several images, a third network architecture is tested which utilizes 3D convolution in the entire encoder. The depth dimension is not reduced in the encoder network. The merge of the depth dimension for the 2D decoder is done in several fully connected layers. The output of the last layer in the encoder is flattened and passed to the fully connected layer which reduces the output to a fifth of its size. This is then reshaped to the output shape of the last decoder layer with the depth dimension omitted. The previous method for skip connections is not feasible anymore, as all channels are appended for every depth dimension. This results in a large number of channels after each concatenation and thus a large number of weights in the following layer. As 3D convolutions require a large number of weights itself, the size of the network is not viable anymore. Instead of stacking the output, a trainable weight variable for each encoding layer and depth dimension is introduced. The output in each depth dimension is multiplied by the corresponding weight multiplicator and then summed up to a single depth dimension. Thereby, for each layer, only five additional merging weights are used. The size of the network is still large, and the training progress is slow. After 100.000 steps the results are not converged due to the ample search space.

To reduce the size of the network, 2D convolutions with reused weights are tested in another architecture. Each rendered image of a sample is passed to the same convolution layer. The weights of each convolution operation are used for all lighting conditions and are not specific to an individual condition as before. The results of each image are concatenated in a depth dimension. After the encoder, the depth dimensions are merged in a fully connected layer. The method to merge depth dimensions for skip connections is applied here, too. The training speed and general performance of the network improved compared to the previous version. However, the predicted normal maps are only capable of representing a completely flat surface. This is visible in fig. 3.13a. A possible explanation is that the error from a normal map is only visible in two of the five input images (see section 3.3.2). These images have a particular low lighting position, and they highlight the normal structure. In the other images, the surface structure is either not visible or not as pronounced. The normal map error simply gets shadowed by the error on the other maps. Maps such



(a) Rough plaster wall



(b) Galvanized steel

Figure 3.13.: Comparisons for the full two-dimensional convolution network with weight reuse.

as the base color map can produce a larger error in all images. Even with an increase in filter outputs for every convolution layer, the network is not capable of producing normal details. Additionally, the network is not able to remove the highlights of the light sources. In fig. 3.13b this is illustrated in the roughness, metallic and base color maps. In general, the results of this architecture are not detailed and blurry. The network is not expressible enough to estimate the inverse render function.

As the full 3D convolution encoder is too large and a full 2D convolution encoder not expressive enough, a combination is proposed in the final architecture. This architecture is referred to as the Multi-Input-Multi-Output-Convolutional-Neural-Network or MIMOCNN. The concept of using 3D convolutions to merge the different lighting conditions is applied here as well. However, it is not used in the encoder network but moved to a separate preparation stage. Here the depth dimension of the input is reduced with 3D convolutions and a stride of two in the depth dimension. The other dimensions are not reduced in this step. Afterward, the results are passed to the modified U-Net described at the beginning of this section. The resulting architecture is visualized in fig. 3.14. Exact configurations for each layer are listed in table 3.1. Layers named as *Conv3D-#* are 3D convolution, *Conv-#* define

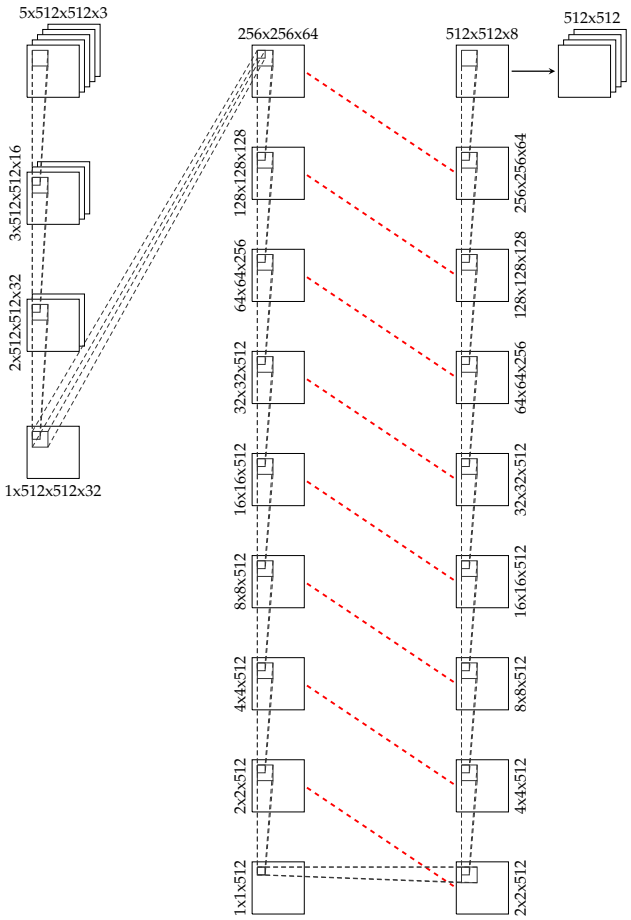


Figure 3.14.: Diagram of the Multi-Input-Multi-Output-Convolutional-Neural-Network. A U-Net inspired network. The kernel size of the convolution and deconvolution is three. Instead of max pooling, a stride is used. The dashed red lines represent a skip connection.

a 2D convolution, *TConv-#* are transposed 2D convolution, *Dropout* are the dropout operation with a keep probability of 0.5 and the *Concat x* layers represent the skip connections to the layer x. A batch size of four is used for the training, and it is performed on a single NVIDIA GeForce GTX 1080 Ti.

3.4.2. Loss Formulation

In this section, several loss formulations are discussed. One of the first tested losses is the MSE loss against the ground truth parameter maps. The MSE loss function produces slightly blurry images after the training. This lines up with the work of Zhao et al. [52]. The MAE loss is proposed in this work as an improvement compared to the MSE loss [52]. In the specific case of this thesis, it improves the

Layer	Kernel | Stride	Batchnorm	Activation	Output Size
Conv3D-1	$3\times3\times3$ | $2\times1\times1$		Leaky ReLU	$3\times512\times512\times16$
Conv3D-2	$3\times3\times3$ | $2\times1\times1$		Leaky ReLU	$2\times512\times512\times32$
Conv3D-3	$3\times3\times3$ | $2\times1\times1$		Leaky ReLU	$1\times512\times512\times32$
Conv-1	4×4 | 2×2		Leaky ReLU	$256\times256\times64$
Conv-2	4×4 | 2×2	✓	Leaky ReLU	$128\times128\times128$
Conv-3	4×4 | 2×2	✓	Leaky ReLU	$64\times64\times256$
Conv-4	4×4 | 2×2	✓	Leaky ReLU	$32\times32\times512$
Conv-5	4×4 | 2×2	✓	Leaky ReLU	$16\times16\times512$
Conv-6	4×4 | 2×2	✓	Leaky ReLU	$8\times8\times512$
Conv-7	4×4 | 2×2	✓	Leaky ReLU	$4\times4\times512$
Conv-8	4×4 | 2×2	✓	Leaky ReLU	$2\times2\times512$
Conv-9	4×4 | 2×2	✓	ReLU	$1\times1\times512$
TConv-1	4×4 | 2×2	✓	ReLU	$2\times2\times512$
Dropout	-	-	-	$2\times2\times512$
Concat Conv-8	-	-	-	$2\times2\times1024$
TConv-2	4×4 | 2×2	✓	ReLU	$4\times4\times512$
Dropout	-	-	-	$4\times4\times512$
Concat Conv-7	-	-	-	$4\times4\times1024$
TConv-2	4×4 | 2×2	✓	ReLU	$8\times8\times512$
Dropout	-	-	-	$8\times8\times512$
Concat Conv-6	-	-	-	$8\times8\times1024$
TConv-3	4×4 | 2×2	✓	ReLU	$16\times16\times512$
Dropout	-	-	-	$16\times16\times512$
Concat Conv-5	-	-	-	$16\times16\times1024$
TConv-4	4×4 | 2×2	✓	ReLU	$32\times32\times512$
Concat Conv-4	-	-	-	$32\times32\times1024$
TConv-5	4×4 | 2×2	✓	ReLU	$64\times64\times256$
Concat Conv-3	-	-	-	$64\times64\times512$
TConv-6	4×4 | 2×2	✓	ReLU	$128\times128\times128$
Concat Conv-2	-	-	-	$128\times129\times256$
TConv-7	4×4 | 2×2	✓	ReLU	$256\times256\times64$
Concat Conv-1	-	-	-	$256\times256\times128$
TConv-8	4×4 | 2×2		Sigmoid	$512\times512\times8$

Table 3.1.: Detailed final network architecture.

results, too. Zhao et al. additionally tested Structural similarity (SSIM) [49] and Multi Scale-Structural similarity (MS-SSIM) [48] and a combination of MS-SSIM with the MAE loss. These methods improved the visual quality of the image, but implementation and evaluation of these loss functions are left for further research. As the error is calculated for each texture map independently and then summed up, the error from each map is assumed to be equally visible in a rendered image. However, this is not true for the case of parameter maps in rendering. For example, an error in the base color map is directly visible in every lighting condition. Errors in the roughness map are only visible in grazing angles and do not contribute as much to the rendered output.

By adding the knowledge of rendering to the network and by comparing the rendered predicted texture maps to the input images, the importance of specific maps can be learned as well. For this, the rendering algorithm described in section 3.3.2 is implemented in Tensorflow. The error from the rendered predicted images to the input images can the backpropagated through the rendering algorithm. This is possible because every operation from Tensorflow in the rendering process contains its gradient. The MAE loss is then computed against the input images. The results of the network are mostly improved. Two problems arise with this approach:

1. The output is fitted to the five input images
2. The metallic and roughness maps can be used in conjunction to express specular behavior

Both problems are visible in fig. 3.15. As seen in fig. 3.15, the corners of the roughness map are brighter, and the corners of the metallic map are darker. This is due to the four light source being placed at the corners and one light source directly above the sample. The network in this example only trained against the five input images and no ground truth information is provided. Thus, the network tries to optimize only against the five input images. The error for these specific light configurations is low. New light and view configurations create a perceptually noticeable error. Additionally, it is obvious from this example that metallic and roughness can both be used in conjunction to express specular behavior.

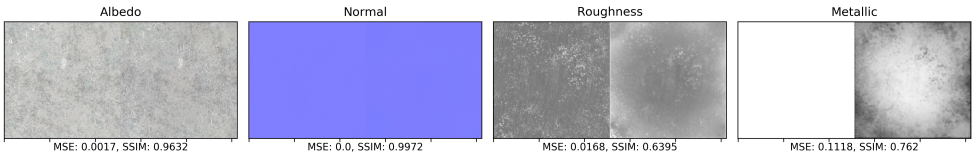


Figure 3.15.: Example of ambiguity of reflective behavior with roughness and metallic maps.

The first problem is fixed by adding additional input images with random lighting conditions. These images are not passed into the network. They are only used in the calculation of the loss. Thus, they are referred to as loss images. The predicted maps

are then rendered with the fixed light locations of the input images and the random ones of the loss images. The random light positions are currently sampled randomly from a uniform hemisphere distribution. In further works, the light positions need to be randomly selected from the possible locations of the light stage (see section 2.1). The MAE loss for each image is then summed up. Thereby the network is forced to not only optimize against a single lighting setup, but to random ones who generalize the results.

The second problem is more challenging to correct since determining if a material is metallic or not is difficult from a single viewing angle. A possible solution is to either pass in the ground truth knowledge of the metallic map or implement the support for different viewing angles in the renderer. As the implementation for different viewing angle is more complicated especially as the implementation needs to be done in Tensorflow operations, the solution of adding the metallic maps is chosen here. A MAE loss from the predicted metallic map to the ground truth map is added and merged with the loss to the input images and loss images.

Both of these solutions improve the output of the network drastically.

The previous loss formulation only calculates the loss for each image individually and sums the errors. Significant losses in a single image can be shadowed by smaller losses in other images. If an individual image contains a specific shadow due to the normal and the particular lighting condition, it can be invisible in other images. It is assumed that the learning behavior improves when the error is highlighted.

To test this hypothesis a new loss formulation, the max loss, is proposed. For this, the RGB channels on the rendered image from the predictions and the input images are summed up. For every rendered image from the predictions the MAE loss to the respective input image is calculated. The maximum error for each pixel between all these images is then searched. Equation 3.12 describes this process. The arg max operation returns the maximum error of the depth dimension d. The variables x and y describe the width and height dimension, and c is the channel dimension for the RGB values.

$$e_{\max}(p,l) = \frac{1}{xy} \sum_{x} \sum_{y} \arg\max_{d} \sum_{c} (= l_{d,x,y,c} - p_{d,x,y,c}) \tag{3.12}$$

However, the result from this loss function worsens the predicted output. Additionally, the training behavior of the network is not stable. This is most likely due to dropped paths to different images with this method.

Another evaluated loss is the mixed absolute relative loss. As the renderer produces images which exceed the 0 to 1 range, a method for preserving the unclipped range is evaluated. As relative small difference in large values are punished disproportionately with the MAE, a relative absolute error $e = \frac{\hat{x}-x}{\hat{x}}$ is preventing such harsh punishments. Here $\hat{x}$ represents the label and x the prediction. However, for dark areas, the relative

error produces large errors or can cause a division by zero. A combination between the MAE and relative absolute error can prevent this. The combination is defined as:

$$e_{\text{mix}}(p,l) = \lambda(l-p) + (1-\lambda)\frac{l-p}{l}$$
$$\lambda = \frac{v-l}{l} \tag{3.13}$$

With p as the prediction, l as the label and v as a configurable threshold. The threshold v is set to 1 as most values are within the 0 to 1 range. However, the error still does not enable unclipped input as the results contain visible artifacts. This is caused by the network still receiving unclipped values, and the weights inside the network are difficult to adjust to totally varying value ranges. Thus, the mix loss is not used. The final loss formulation is a MAE error with the logarithmic image transformation and a 0 to 1 clipping as described in section 3.3.2.

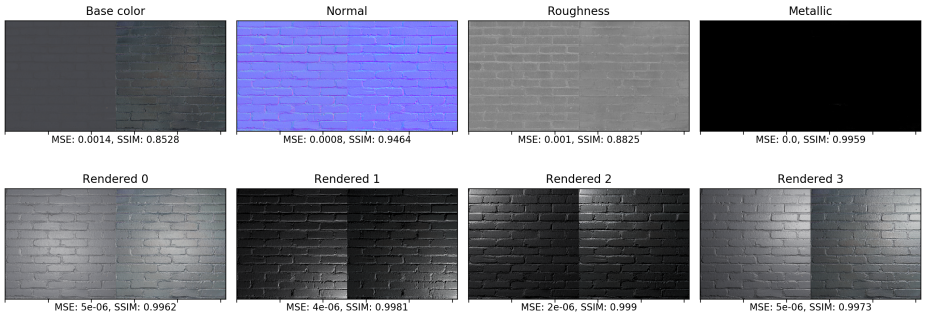


Figure 3.16.: Difference between error from texture maps and renderings. The ground truth is located left in each pair and the prediction on the right. Enlarged figure A.1a on page 80.

The error when comparing rendered images can be low, while the error when comparing the texture maps can be high. For example, in fig. 3.16 the error for a rendered image is low while the error in the base color map compared to other results is high (see section 3.4.4). With only a single viewing position the network is not capable of deciding whether the reflection is due to a lighter base color map or a smoother roughness map.

To guide the network the ground truth images are added to the training and the predicted texture maps are compared against the ground truth maps with a MAE loss. In the beginning, the knowledge of which texture map increases the rendered result the most is available due to the re-rendering. The rendering error acts as an importance sampling in the beginning. In later epochs, the error from the comparison against the ground truth texture maps helps to clear ambiguity in specular behavior. As the images are only used for the loss calculation, these inputs can be left out for the prediction of new materials.

3.4.3. Optimization

Several optimizers and strategies for setting a learning rate are explored. The Adam Optimizer with default parameters and a decreasing learning rate in later epochs produces the best results. Each epoch evaluates all samples, resulting in a step size of 830. The training is performed for 300 epochs. Initially, the learning rate is set to 1e−3. It is found that decreasing the learning rate in later epochs is improving the convergence. The learning rate is lowered as followed:

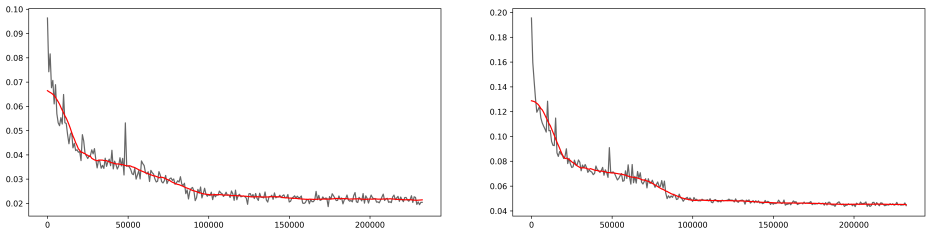
after 100 epochs: 1e−4

after 180 epochs: 5e−5

after 220 epochs: 1e−5

after 260 epochs: 5e−6

These parameters are gathered empirically. The error is demonstrated in fig. 3.17a and fig. 3.17b. As seen in both figures a decrease is not visible for the smoothed error anymore, and thus the training can is converged. In both charts, a drop in the loss is noticeable at the global step of 83000. This corresponds to the epoch 100. Here, the learning rate is reduced, and the error is reduced significantly afterward. Without lowering the learning rate, the error is not converging to this value.



(a) Total cost **(b)** Validation total cost

Figure 3.17.: Overview of the CNN training error. The smoothed error is represented as a red line.

Besides the Adam optimizer, several different approaches are tested. For example, cyclic learning rates with the momentum optimizer are tested [42]. In cyclic learning rates, the learning rate repeatedly changes between a minimum and maximum learning rate every x epochs. A simple approach for this cycle is to use a triangle function. The intuition behind this method is to start with a low learning rate. The initial direction of the gradient is determined. Then the step size increases to move faster to the minimum. The learning rate is lowered again to search for the minimum more reliably. The cycle is then repeated. The higher learning rates in each following cycle can prevent the network to settle for a local minimum instead of a global one. The higher learning rates can move the optimization out of a local minimum due to increased step sizes. However, in this specific task, cyclic learning rates do not

perform better than the Adam optimizer.

Another method is to decrease the learning rate automatically if the validation error is not decreasing in x epochs. This can be done by lowering the learning rate by 80% when the validation error is not decreasing anymore. The method provides good results, but the behavior is unpredictable while training and setting suitable parameters for the monitored epoch length x and the decrease percentage is difficult. Thus, the simple method of decreasing the learning rate after x epochs is preferred.

3.4.4. Results

The network is trained for 300 epochs, and after each epoch, an inference step against the validation dataset is performed. The epoch which produced the lowest validation error is saved and used for evaluation.

To evaluate the performance of the network, 26 materials are estimated. These samples are from completely new classes, which are neither in the training set nor the validation set. Some of these samples provide no resemblance to the samples in the dataset. The mean result across all samples are visible in table 3.2. By testing against completely unseen example, the generalizability of the network is demonstrated.

The rendered samples in this section are tone mapped and gamma corrected for visual comparison. Additionally, the exposure is increased by a factor of 32 before the tone mapping and gamma correction. The Reinhard tone mapping [33] is applied and afterwards gamma corrected with $x^{\frac{1.0}{2.2}}$.

Name	MAE	MSE	SSIM
Base color	0.0414	0.0033	0.8824
Normal	0.0176	0.0009	0.9542
Roughness	0.0396	0.003	0.8793
Metallic	0.0585	0.0472	0.9193

Table 3.2.: Comparison between 26 ground truth samples and CNN predictions. The values are the mean value across all images.

For the MAE and MSE a lower value and for SSIM a higher value is better. As seen in table 3.2 the mean results for the metallic and normal map are nearly identical to the ground truth. In particular, the MAE is interesting as this error can be thought about as the mean pixel value error in percent. For example, in the base color map, the network produces a mean value error of 4%. A predicted output can then be classified as either well- or ill-predicted. This can be done by using a threshold value. Figure 3.18 visualizes the well-predicted percentage with an increasing threshold level. For each sample, an 80% well prediction rate is achieved with a threshold of 0.6% for the metallic map, 2% for the normal map and 4% for the base color/roughness

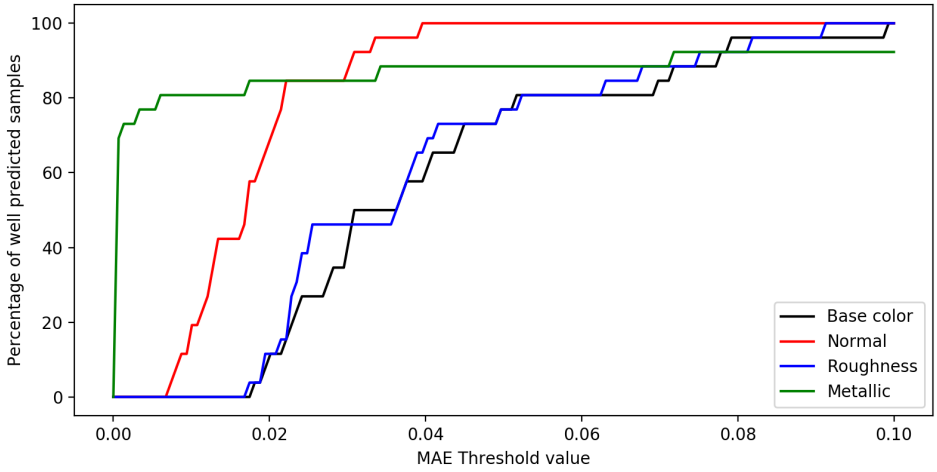


Figure 3.18.: Percentage of well predicted samples from the CNN for an increasing threshold.

map. For the normal and roughness maps, all samples are below an error of 4% and 9%, respectively. For the remaining maps, the distribution is larger.

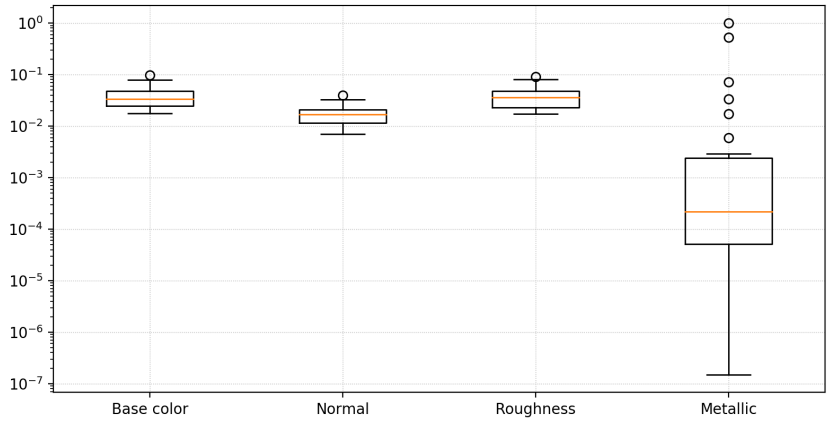


Figure 3.19.: Box plot for the mean error of the 26 CNN generated samples. The y-axis is scaled logarithmically.

The distribution of the MAEs is displayed in fig. 3.19. It is apparent that most texture maps provide a consistent error with few outliers. The metallic map is an exception. Here the error spans a substantial range. The largest outliers are for the sample visible in fig. 3.20. As seen in fig. 3.20 the network mostly classified the material as a non-metallic material. However, the rendering error is low and perceptually the difference between the ground truth and predicted re-rendering is close. Different view position would highlight the metallic behavior of materials more. As discussed in section 3.4.2 different viewing angles are left for further work.

In fig. 3.21 several samples from the 26 samples dataset are compared. Figure 3.21a,

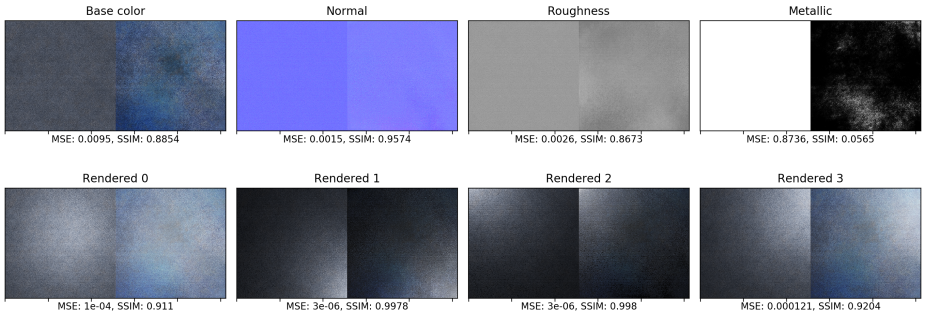


Figure 3.20.: Comparison for a grainy steel material. The ground truth is located left in each pair and the prediction on the right. Enlarged figure A.2a on page 81.

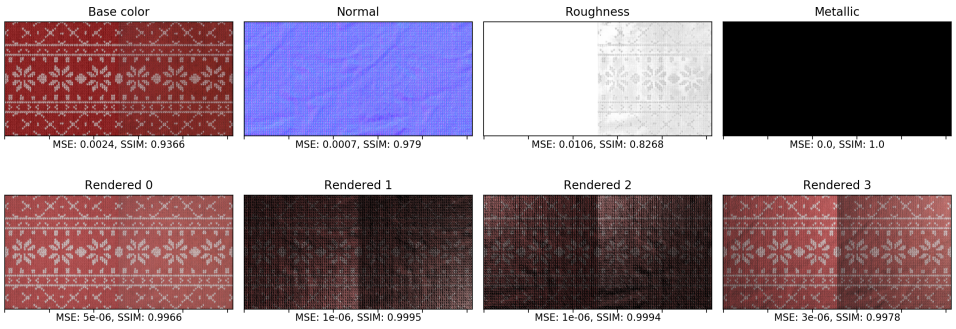
fig. 3.21c and fig. 3.21d display that the network is capable of learning normal maps with complex structures and fine details. Figure 3.21a demonstrates that the network tends to extract roughness information from the texture of the sample. As most samples in the dataset contain correlations between the overall texture, the base color map, and the roughness, this needs to be addressed in future research with an extended dataset. In fig. 3.21b the network can generate roughness and normal maps for even fine detail regions as the bricks. Figure 3.21d illustrates the reliable extraction of information from detailed surfaces like wood. The base color and roughness texture maps contain high detail and are predicted accurately.

As an additional example fig. 3.22 is shown. Here, the ground truth normal map on the left contains some unrealistic values around the structures. This sample is taken from the training dataset. The prediction done by the network removed these artifacts completely.

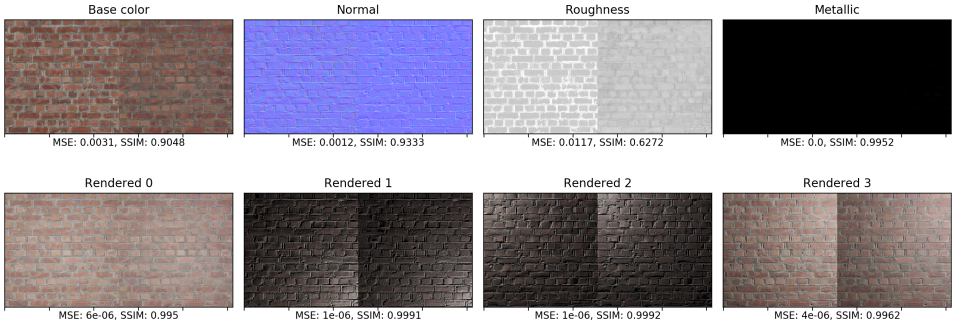
Another interesting example is shown in fig. 3.23. The rendered error is low even when the error for the base color and roughness map is high. Visually no difference exists in the rendered result. This displays how the maps can be used interchangeably to express material behavior. In this case, the base color and roughness maps are used to represent the surface reflectance. Splitting this is a difficult task with the limitation of five images and a fixed view position.

3.5. Generative Adversarial Network

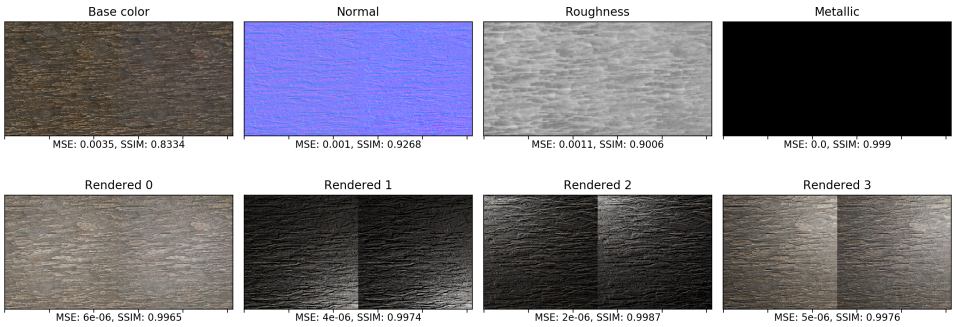
The architecture described in section 3.4 is extended to a GAN architecture in this section.



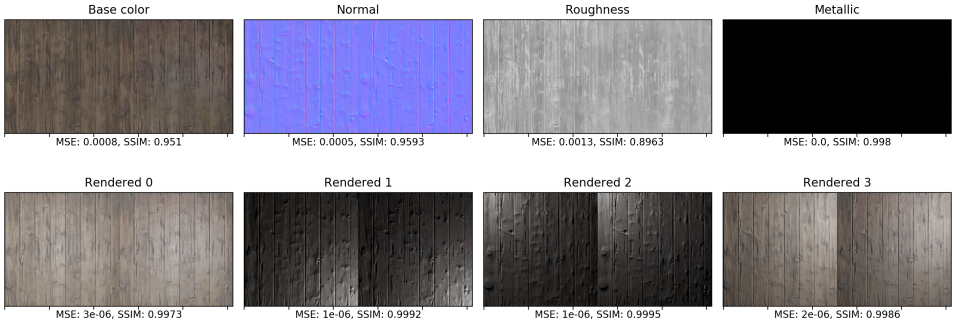
(a) Christmas Sweater. Enlarged figure A.3a on page 82.



(b) Brick wall. Enlarged figure A.4a on page 83.



(c) Tree bark. Enlarged figure A.5a on page 84.



(d) Old wood planks. Enlarged figure A.6a on page 85.

Figure 3.21.: Comparisons of different materials generated by the CNN. The ground truth is located left in each pair and the prediction on the right.

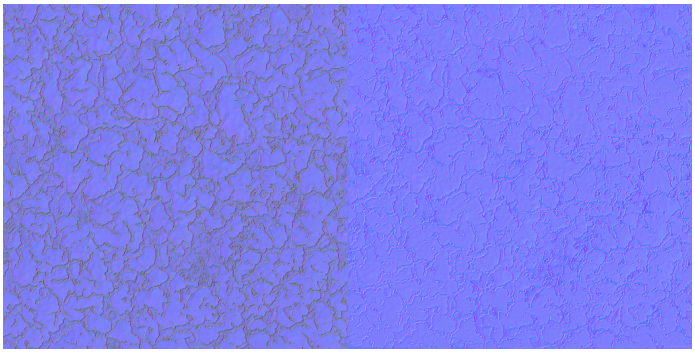


Figure 3.22.: Example for an improved normal map.

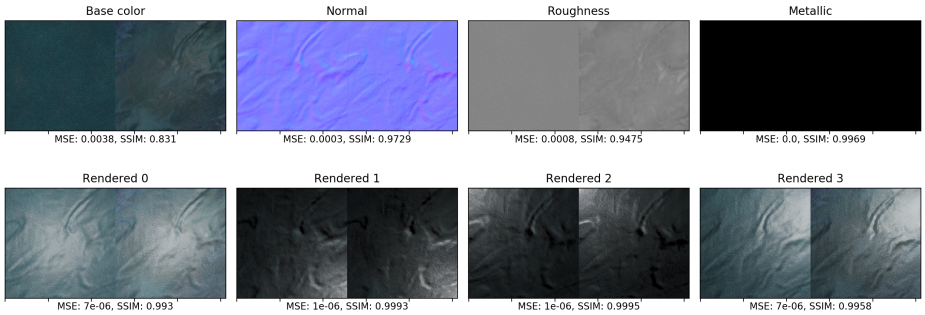


Figure 3.23.: Comparison for a leather material. The ground truth is located left in each pair and the prediction on the right. Enlarged figure A.7a on page 86.

3.5.1. Architecture

The concept of GANs is recently popular and lends itself to an image to image mapping. This concept is explored by Isola et al. [14]. The proposed framework of Isola et al. is applied to the model proposed in table 3.1 and fig. 3.14. Additionally, the changes for the DCGAN architecture from [31] are applied (see section 2.2.9). As the range of the generator output is changed, the output needs to be rescaled to $[0-1]$ for the rendering algorithm. However, the network in this architecture is not capable of learning, and the resulting predicted maps only consist of artifacts.

The DCGAN architecture changes are not producing reliable results. Even when the generator is trained on its own with a MAE loss, the network is not capable of learning accurate BRDF predictions. The original unmodified model in fig. 3.14 is used instead. This includes the output layer with a sigmoid activation function. A rescaling of the generator output for the rendering operations is not required anymore.

The discriminator network is modified, too. The network is now based on Zhu et al. [53], which is a follow up of Isola et al. [14]. The discriminator now utilizes instance

normalization [46] instead of batch normalization [14]. Batch normalization performs normalization on the batch and all spatial dimensions. Instance normalization on the other hand only performs normalization on the spatial dimensions. Each input image is treated by itself.

Layer	Kernel | Stride	Instance Norm	Activation	Output Size
Conv3D-in-1	$4\times4\times4 \mid 2\times1\times1$		Leaky ReLU	$3\times512\times512\times16$
Conv3D-in-2	$4\times4\times4 \mid 2\times1\times1$		Leaky ReLU	$2\times512\times512\times32$
Conv3D-in-3	$4\times4\times4 \mid 2\times1\times1$		Leaky ReLU	$1\times512\times512\times32$
Conv3D-ma-1	$4\times4\times4 \mid 2\times1\times1$		Leaky ReLU	$2\times512\times512\times16$
Conv3D-ma-2	$4\times4\times4 \mid 2\times1\times1$		Leaky ReLU	$1\times512\times512\times32$
Concat Conv3D-in-3 & Conv3D-ma-2	-	-	-	$512\times512\times64$
Conv-1	$4\times4 \mid 2\times2$		Leaky ReLU	$256\times256\times64$
Conv-2	$4\times4 \mid 2\times2$	✓	Leaky ReLU	$128\times128\times128$
Conv-3	$4\times4 \mid 2\times2$	✓	Leaky ReLU	$64\times64\times256$
Conv-4	$4\times4 \mid 2\times2$	✓	Leaky ReLU	$32\times32\times256$
Conv-5	$4\times4 \mid 1\times1$	✓	Leaky ReLU	$32\times32\times512$
Conv-6	$4\times4 \mid 1\times1$		Sigmoid	$32\times32\times512$

Table 3.3.: Detailed discriminator architecture.

The detailed discriminator architecture is illustrated in table 3.3. Where *Conv3D-in-1* is a 3D convolution which input is the five illuminated images from the sample. For *Conv3D-ma-1* the input is the four BRDF parameter maps (base color, normal, roughness and metallic). The *Concat Conv3D-in-3 & Conv3D-ma-2* layer joins the two outputs from the last input and BRDF map preparation layers respectively.

3.5.2. Loss Formulation

The proposed dual loss of Isola et al. [14] is used in this architecture. This dual loss is defined as the GAN loss defined in eq. 2.13 added to a direct loss compared to the ground truth. For the direct loss, the final loss formulation described in section 3.4.2 is used. The GAN loss with the direct MAE loss of the maps and renderings is combined with a scaling factor:

$$L(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{x\sim p_z(z)}[\log(1-D(G(z)))] + \lambda L_{\text{MAE}} \tag{3.14}$$

Where λ is the scaling factor and L_{MAE} is the loss function described in section 3.4.2. Isola et al. [14] proposes to set λ to 100 and it is applied here, too.

3.5.3. Optimization

The training is found to be unstable. By training the generator network beforehand, the combined training of the generator and discriminator improved and the results are more accurate. Thus, the generator network is trained for 100 epochs with a learning rate of 1e−3. The Adam optimizer with default parameters is used for the training process. A batch size of 4 is chosen here.

The discriminator and generator are trained afterward for 215 epochs over the whole dataset. An initial learning rate of 2e−4 is chosen. The learning rate is reduced after 145 epochs to 1e−4. The optimization is done by the Adam optimizer. However, the β_1 parameter is set 0.5. This change is proposed by Radford et al. [31] to increase the training stability. Due to the larger size of the networks, a batch size of 1 is selected.

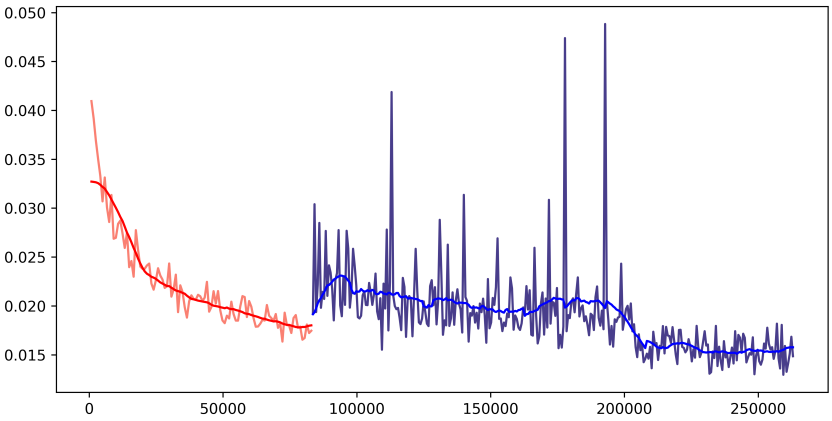


Figure 3.24.: The mean absolute error for the pre-trained generator and the GAN afterward.

In fig. 3.24 the loss against the rendered images and ground truth maps is displayed. The red lines represent the smoothed and non-smoothed MAE loss of the generator only training process. The blue line afterward illustrates the same error but in this case for the generator and discriminator learning in conjunction.

In fig. 3.25 the accuracy is visualized. An accuracy of 1 is the maximum achievable value. In the diagram, the blue line represents the smoothed and non-smoothed discriminator accuracy. This accuracy represents the ability of the discriminator to detect generated from ground truth samples reliably. As seen in fig. 3.25 the discriminator is not capable of detecting whether the sample is ground truth or generated in the end anymore. The red line represents the smoothed and non-smoothed generator accuracy. This accuracy represents the number of samples being classified as ground truth, even when they are generated. Interestingly the generator accuracy only achieves a value between 0.2 and 0.3 accuracy. The discriminator achieves 0.5 accuracy. This signifies that the discriminator is classifying a significant amount of ground truth samples as generated. An explanation for this behavior is that the dataset is generated from various artists (see section 3.3). Some of these

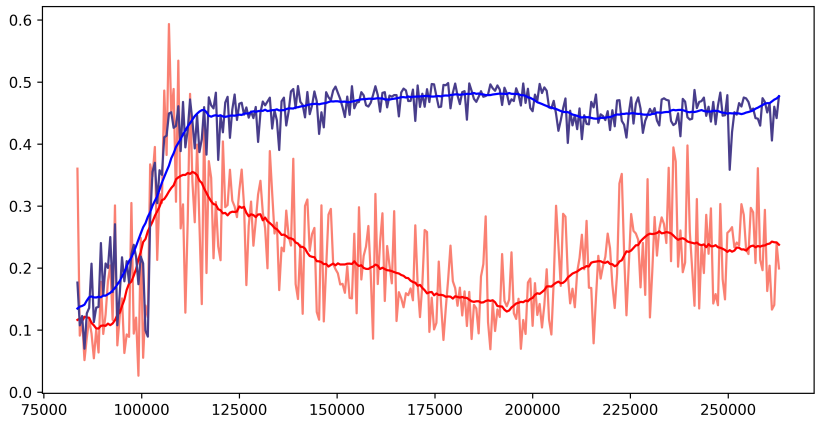


Figure 3.25.: The accuracy of the GAN.

samples can contain unrealistic values. The discriminator may determine these samples as not fitting in the dataset and thus as a generated sample.

3.5.4. Results

The rendered images are processed for clearer visibility in the same manner as the ones described in section 3.4.4.

Name	MAE	MSE	SSIM
Base color	0.0382	0.0031	0.9101
Normal	0.0207	0.0011	0.9435
Roughness	0.0675	0.0075	0.8574
Metallic	0.0855	0.08	0.9068

Table 3.4.: Comparison between 26 ground truth samples and GAN predictions. The values are the mean value across all images.

As seen in table 3.4 the results are mostly comparable to the CNN-version in table 3.2. The base color map is predicted more accurately by the GAN. The prediction for the roughness and normal maps are of the same accuracy. The metallic map is predicted slightly worse, but the prediction of the metallic map follows the dataset more closely regarding metallic maps being sparse (see fig. 3.27). The percentage of well-predicted samples is visualized in fig. 3.26. Compared to the CNN in fig. 3.18 the nearly perfect prediction is more common. 20% of the base color samples are predicted with a MAE lower than 0.02. For the roughness map, the rate is about 15% with the same threshold. The CNN is not capable of generating base color and roughness maps with an error that low. However, for the normal map the CNN is capable of estimating a large number of samples with a near perfect output. The CNN is capable of predicting 80% of the samples with a MAE of 0.02. The GAN on

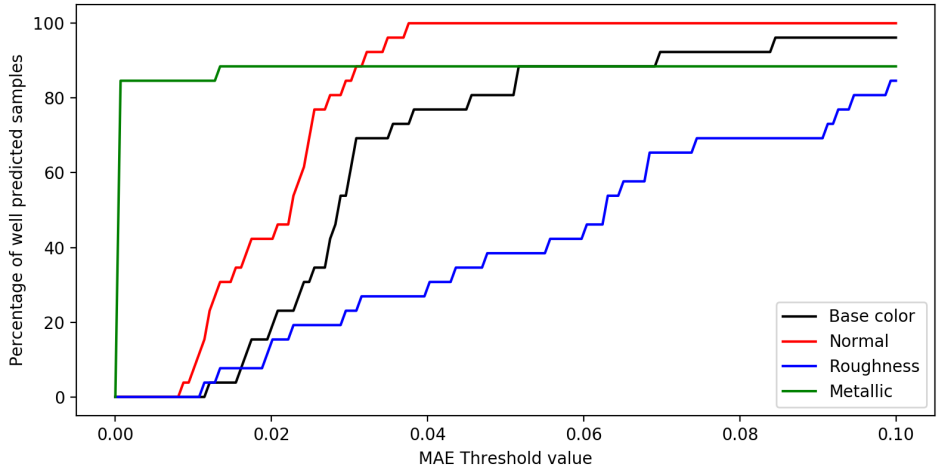


Figure 3.26.: Percentage of well predicted samples from the GAN for an increasing threshold.

the other hand only predicts 50% of the samples well with this threshold level, but the normal map from all samples is predicted with a lower threshold level than the CNN.

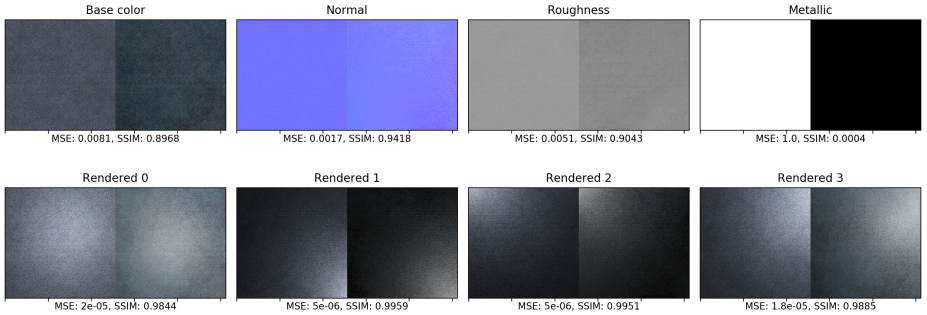


Figure 3.27.: Comparison of the GAN predicted grainy steel material. The ground truth is located left in each pair and the prediction on the right. Enlarged figure A.2b on page 81.

By visual comparison of the predicted samples of the GAN with the CNN, the performance can be analyzed more closely. The wrongly predicted grainy steel material in fig. 3.20 from the CNN is predicted also as a non-metallic material in fig. 3.27. However, the maps are more homogeneous and do not feature uneven textures like the base color texture in the CNN prediction.

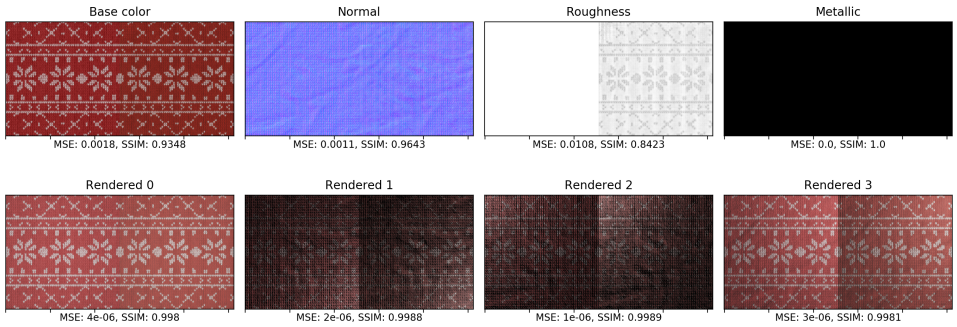
In fig. 3.28 the same samples as in the CNN comparison (see fig. 3.21) are displayed. The same capabilities of parameter estimation, as described in section 3.4.4, are noticeable here, but the results of the GAN approach are slightly sharper and provide higher detail.

3.6. Analysis

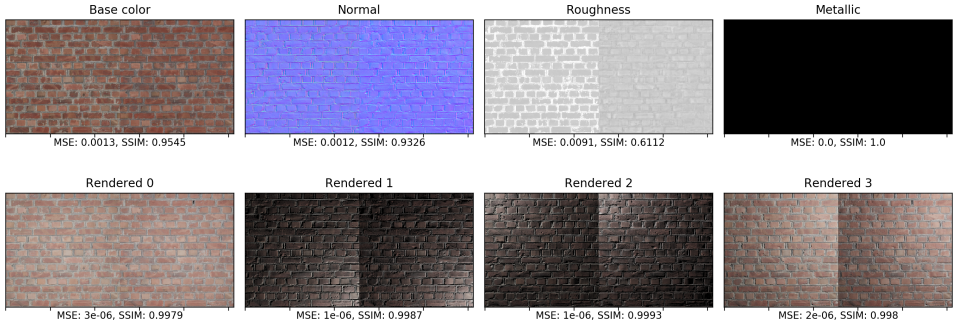
A saliency map is a technique of visualizing the inner working of a network. The saliency map is the gradient from the predictions of a network through all of its layers to the input. Thus, higher values represent a steeper gradient and in turn, results in a more significant information flow. In fig. 3.29 the gradient for each parameter map to each input image is visualized. Two representative samples, a tree bark, and a metal are selected. The tree bark is an interesting sample because of the detailed structure of the base color, normal and roughness maps. The metal is selected to highlight the detection of metallic surfaces.

In fig. 3.29a and fig. 3.29b the different patterns in the saliency maps are noticeable. Every sample and every texture map is handled in a slightly different manner by the network. All samples have in common that the network is evaluating connected patches. Additionally, it is visible that the network is receiving the strongest gradients from unlit regions. For example, in fig. 3.29b this is especially visible in the normal map, which produces a steeper gradient in all areas without a strong highlight. This is due to the images in the network are clipped (see section 3.3.2). The input images in the fig. 3.29 are not the same ones the network receives. The images in the comparisons are tone mapped with the Reinhard operation [33] and gamma corrected for better visual perception. The clipping results in a loss of information and feature extraction from other regions are more viable for the network.

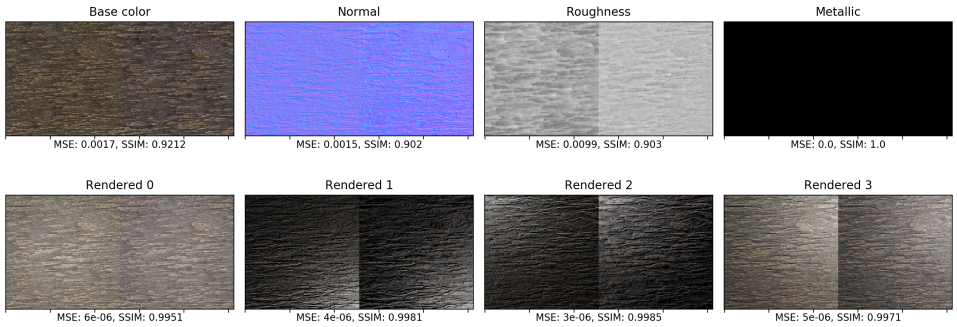
Another fascinating insight is that the gradient of the metallic map is only steep in a few select regions. The prediction of each pixel is thus dependent on non-local information from the input images. On the other hand, each pixel of the normal map is highly reliant on local information. In fig. 3.29a it is noticeable that the steepest gradients are on shadowed regions. This behavior is intuitive for humans as the shadowed areas in contrast to the lit regions provide a visual cue about the surface structure.



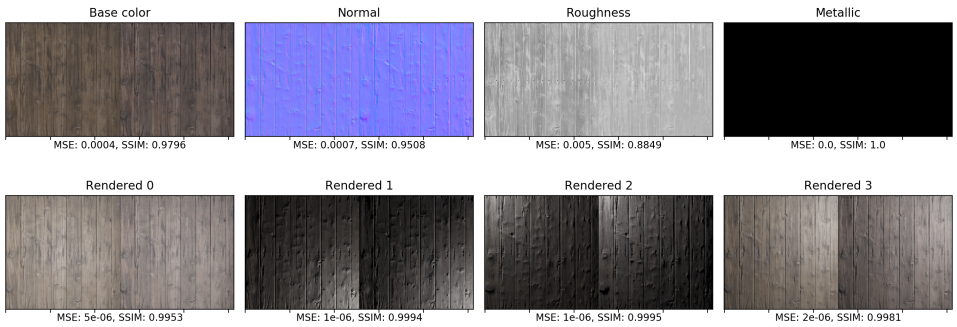
(a) Christmas Sweater. Enlarged figure A.3b on page 82.



(b) Brick wall. Enlarged figure A.4b on page 83.

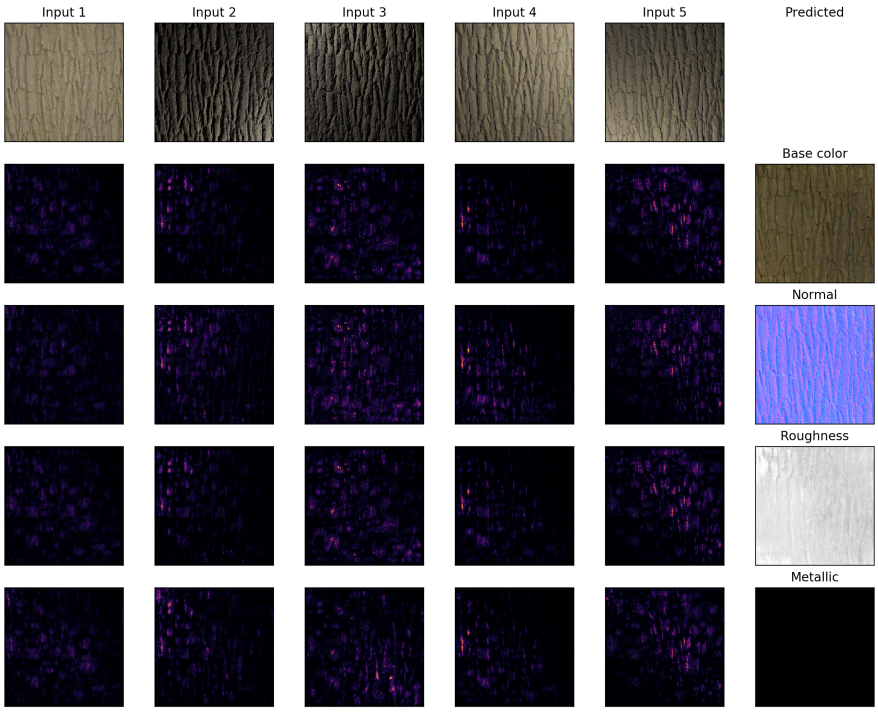


(c) Tree bark. Enlarged figure A.5b on page 84.

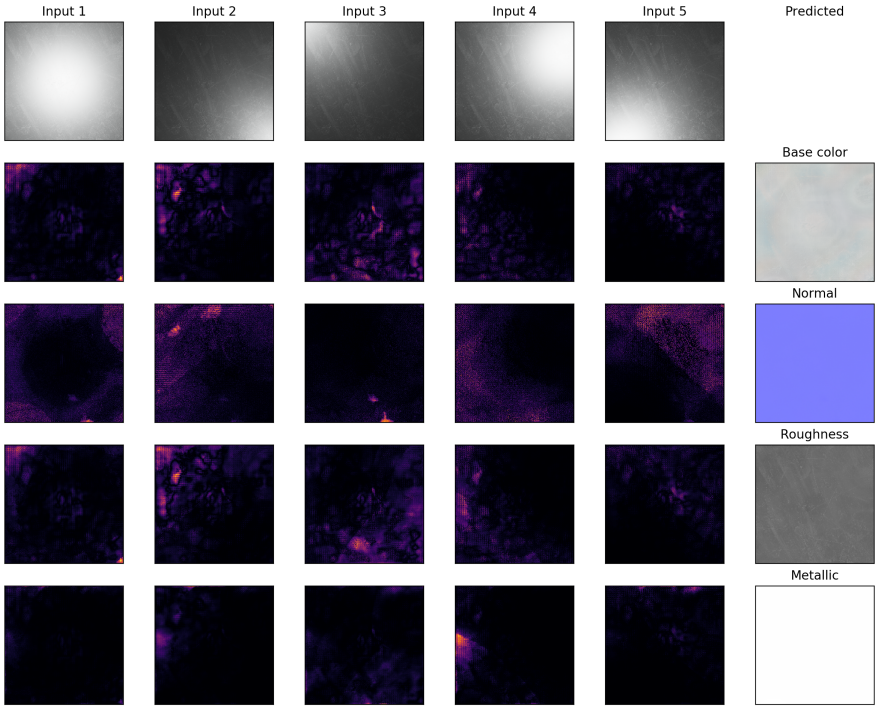


(d) Old wood planks. Enlarged figure A.6b on page 85.

Figure 3.28.: Comparisons of different materials generated by the GAN. The ground truth is located left in each pair and the prediction on the right.



(a) Tree Bark saliency maps. Enlarged figure A.8 on page 87.



(b) Metal saliency maps. Enlarged figure A.9 on page 88.

Figure 3.29.: Saliency maps for different samples. For each parameter map the saliency map for every input image is visualized.

4. Discussion

In this chapter, the results and contributions are discussed. The advantages and disadvantages of the proposed system are highlighted, and an outlook for future work is presented.

4.1. Conclusion

The main contribution of this thesis is a framework for an automatic BRDF estimation which only depends on five input images with a fixed viewing position. The resulting spatially varying BRDF parameter maps are estimated at a reasonably high resolution and can be used in video games and movie productions. Due to the low number of input images and the fixed viewing position, the material capture time is especially low. This is due to no complex setup with a motorized camera or light is necessary. The setup for training the network is flexible enough to allow for retraining with different light positions. This leads to the possibility of designing different scanning devices. In section 4.2 a potential portable setup is introduced.

The predictions of the proposed networks are accurate for most cases. In some cases, materials do not correspond to any other sample in the training set. Here, the results may vary. These corner cases will require manual human correction. It is possible that these cases will vanish with a more extensive dataset.

A differentiable rendering framework to be used inside a neural network is created. This framework is currently capable of rendering materials on a flat surface. However, enabling rendering of more complex surfaces is achievable by supporting a deformable plane, which fits the scene geometry. This can be used to render an arbitrary scene from a single perspective.

Additionally, first steps are taken at enabling accurate BRDF capturing in the light stage at the University of Tübingen. Here, the camera position is transformed into the coordinate system of the calibrated light locations on the spherical gantry. This is done without or few visible lights in the camera frame.

4.2. Future work

The proposed system is capable of extracting material information in a synthetic setup. Moving the system to samples acquired from a real-world scanning setup is an important area for future research. For this, the rendering setups need to be calibrated. The light sources need to emit the same radiance. Additionally, the camera light sensitivity and output range need to be adjusted to fit the rendering output. Both of these calibrations are necessary because the proposed network is calculating the error based on the renderings of the estimated materials. If the recorded sample is captured with a different intensity or color calibration than the renderer, the error will be corrected with changes to the predicted maps. For example, a color shift can be evident in the base color map or the textures can be lighter or darker. This error will be perceptible when these textures are used for renderings. Noise should be added to the synthetic training samples, as noise will be present in the camera images at least slightly. The network will handle camera noise better and provide more accurate predictions. Additionally, the light and camera positions need to be adjusted to fit the light stage. This change only requires retraining on the synthetic dataset.

(a) Perpendicular camera position **(b)** Different viewing position. **(c)** Perspective matched to perpendicular position

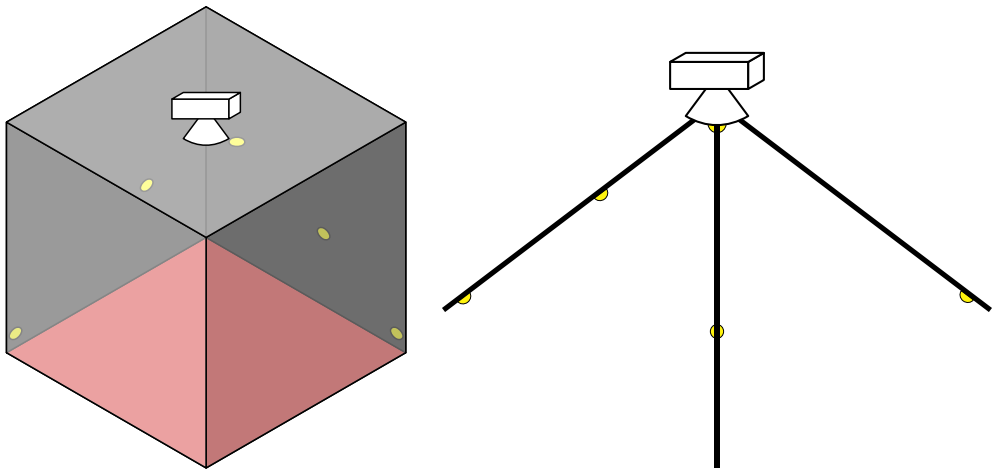
Figure 4.1.: Example for different viewing positions.

An additional improvement is the implementation of different viewing angles. Information about specular behavior is more apparent with different viewing positions, and the resulting prediction is improved. The rendering is capable of evaluating different viewing positions. However, the results are not applicable to real-world recorded samples. This is due to the renderer not taking geometry and perspective distortion into account. The resulting output image is still rectangular even when the viewing positions are at a flat angle. As a real-world recorded sample is distorted, it needs to be perspective corrected to correspond to a perpendicular camera position. In case of multiple images captured with different camera positions, it is not obvious if the network is capable of extracting reliable information from the input, as the perspective correction may cause a blurred image due to interpolation. For example, in fig. 4.1 a perpendicular view and light position is visualized in

fig. 4.1a. The second viewing position is taken from a flatter angle in fig. 4.1b. The sample in fig. 4.1c is manually perspectively correct in Affinity Photo[1]. The sample in fig. 4.1c is reduced in quality. This is especially noticeable in the top section of the image. Here, the perspective distortion is at its strongest. Another interpolation method may result in better result. A method for automatic extraction of the sample and removing the perspective projection needs to be developed and tested with the network.

Some of the predicted BRDF parameters are slightly incorrect regarding the color. This is due to the MAE loss being calculated directly on the RGB channels, and thus applies for color and lightness changes in conjunction. Changing the color space to a format where color and lightness are separated, can provide better results. The HSV or CIELAB is a suitable color space for this. For the CIELAB color space, the error metric is already defined as ΔE or the Euclidean distance [6]. This needs to be explored in future research.

Another addition is the extension of rendering basic 3D geometry with depth maps. Here, a surface is deformed in a way to represent the scene from a single viewpoint. Intuitively this can be thought about cloth being placed on objects. The fabric is distorted to capture the shape of the objects. It is possible that the network is capable of learning BRDF parameters and depth information in future research.



(a) A mobile scanning box **(b)** A mobile scanning tripod

Figure 4.2.: Example setup for mobile scanning units.

As the network is capable of learning the extraction of BRDF information with different lighting setups, different scanning setups are possible. Empirically it is found that two of the light source should be located at a flat angle and one from above. If this condition holds true, other scanning setups are possible. For example, mobile scanning units can be developed. In fig. 4.2 two possible setups are introduced. The

[1] `https://affinity.serif.com/de/photo/`

first mobile scanning configuration in fig. 4.2a displays a portable scanning box, where a box is equipped with five LEDs and it can be placed on a surface. The other setup is based on a tripod and is illustrated in fig. 4.2b.

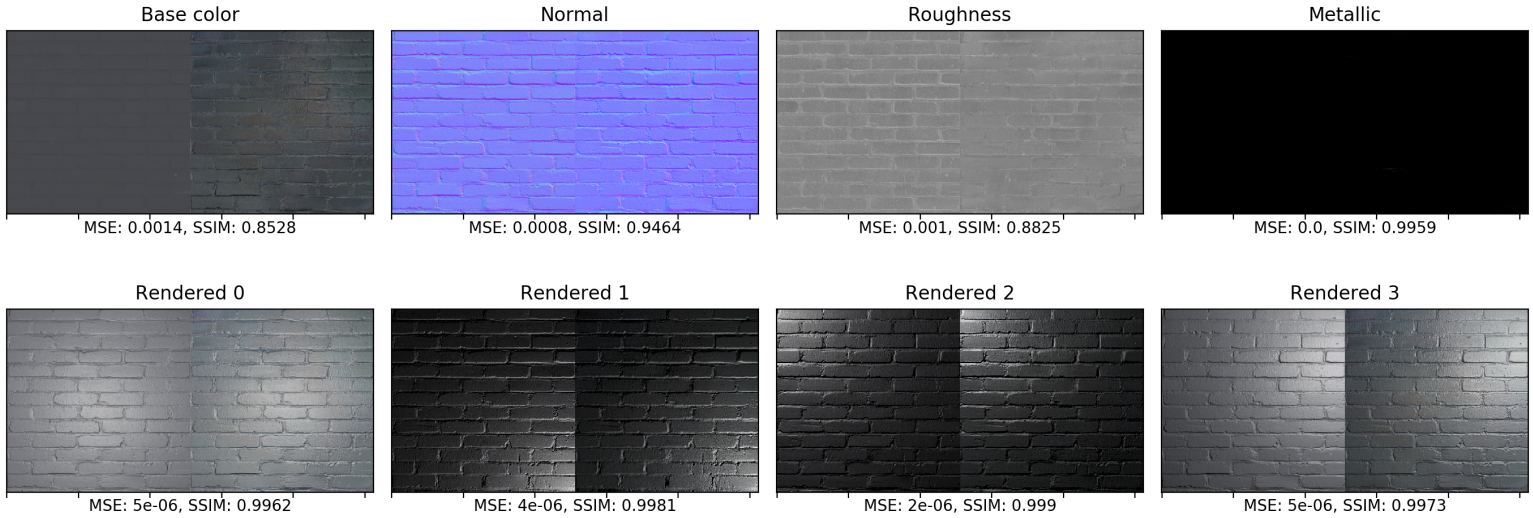
Another interesting approach is learning the best possible light position for material estimation. This can be done by allowing the network to optimize the light positions. As different light positions between each epoch are challenging to learn, alternating learning cycles between the network weights for the BRDF estimation and the light position optimization are proposed. An additional reduction of the learning rate for light positions optimization step is recommended. This only allows small positional changes. The first steps for the BRDF estimation may require higher learning rates to increase the convergence of the new light positions. With a sufficiently large number of epochs, an optimal light position is learnable.

An increase in the output resolution is another area of future research. A dedicated upscaling network is a possibility. With this technique, the complex estimation network is not increased in size further.

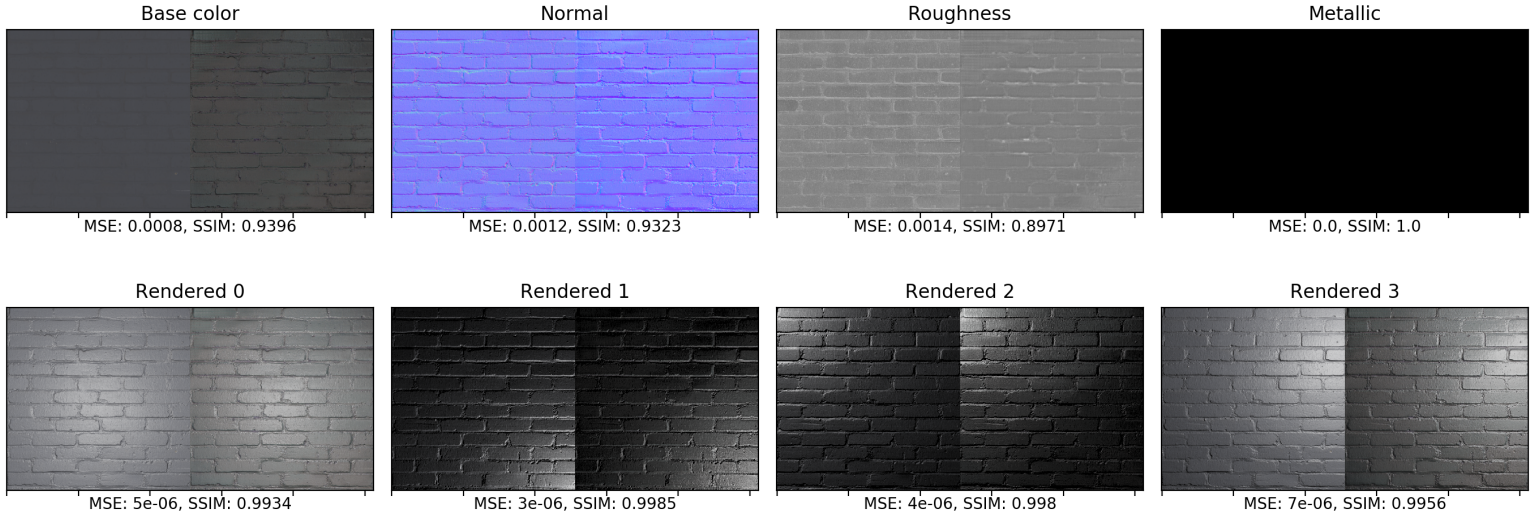
A. Appendix

Results

The results follow on the next page.

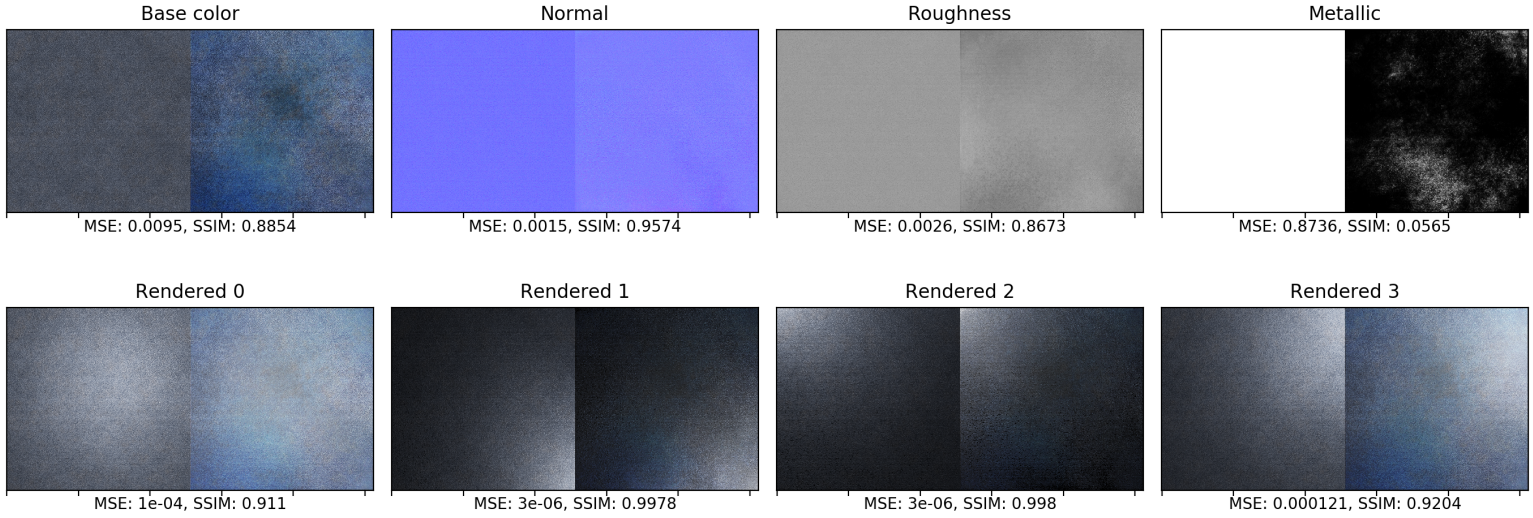


(a) Convolutional Neural Network

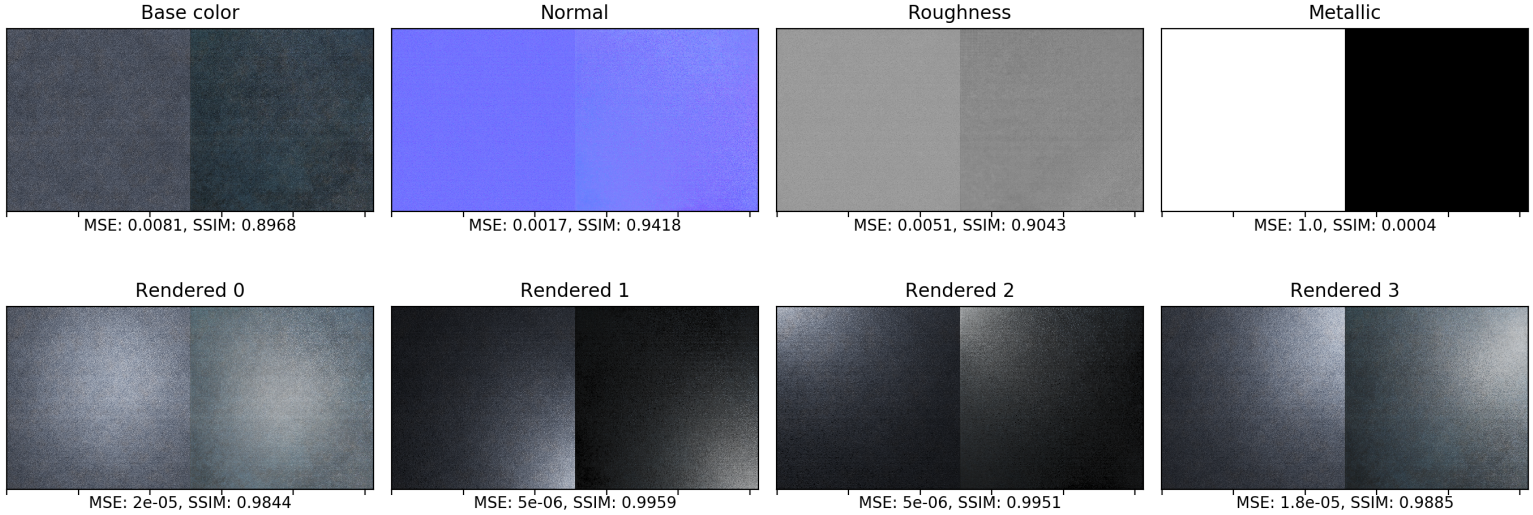


(b) Generative Adversarial Network

Figure A.1.: Painted brick wall. The left half of each pair is the ground truth and the right is the generated sample.

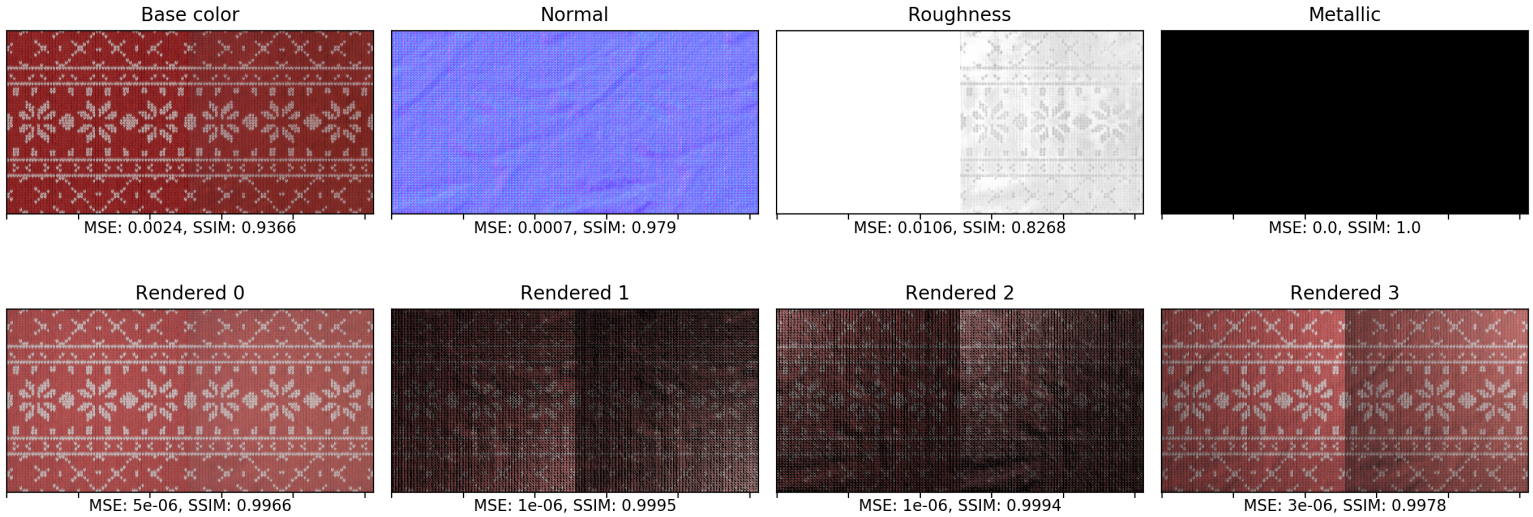


(a) Convolutional Neural Network

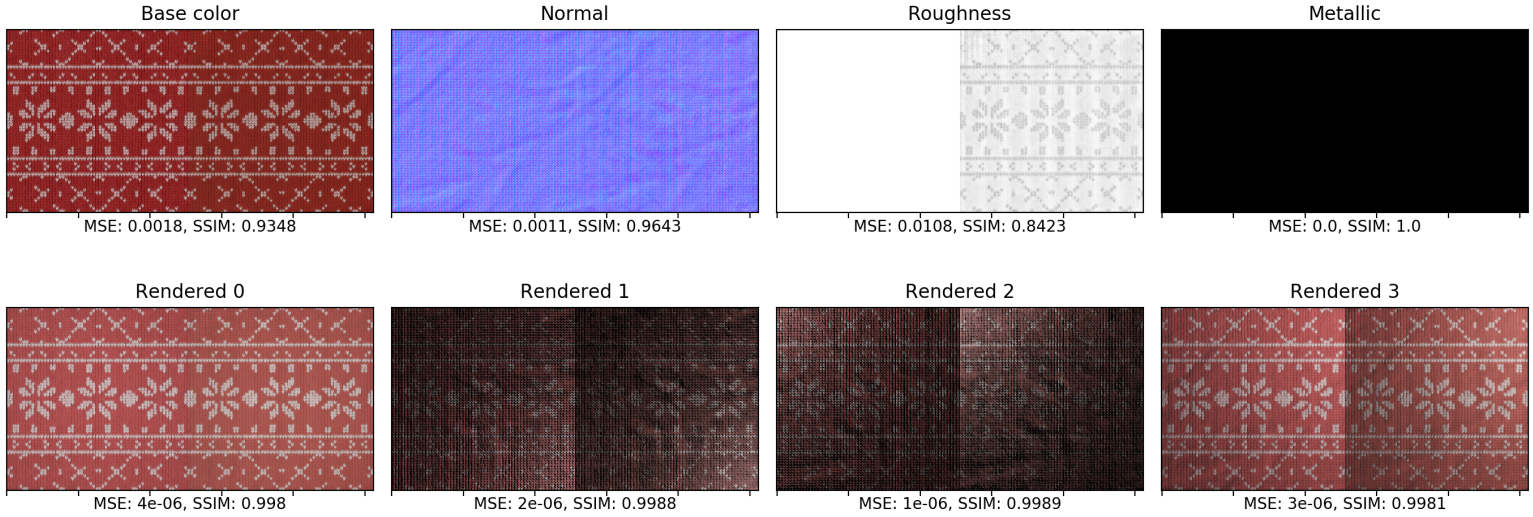


(b) Generative Adversarial Network

Figure A.2.: Grainy steel. The left half of each pair is the ground truth and the right is the generated sample.

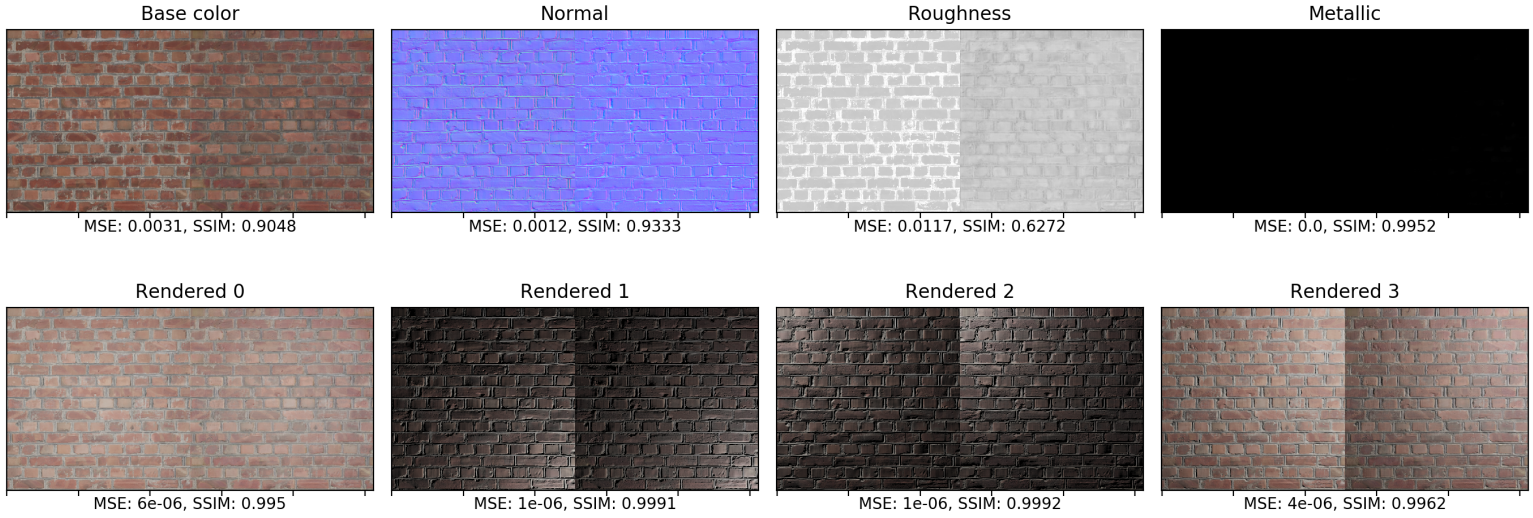


(a) Convolutional Neural Network

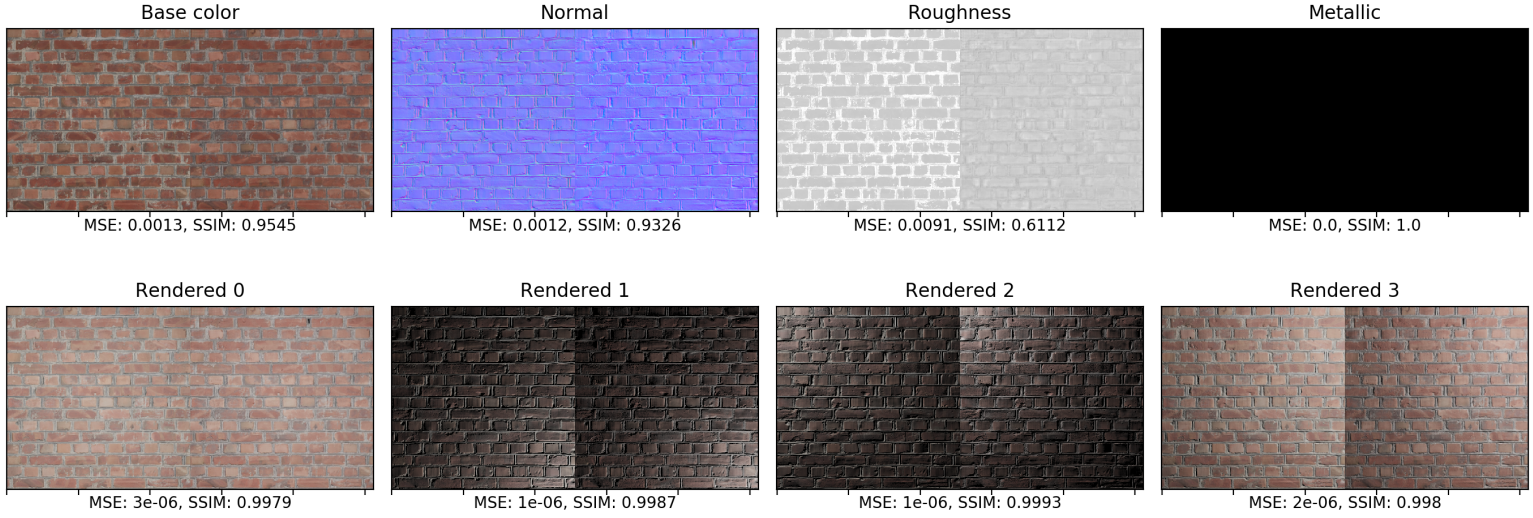


(b) Generative Adversarial Network

Figure A.3.: Christmas Sweater. The left half of each pair is the ground truth and the right is the generated sample.

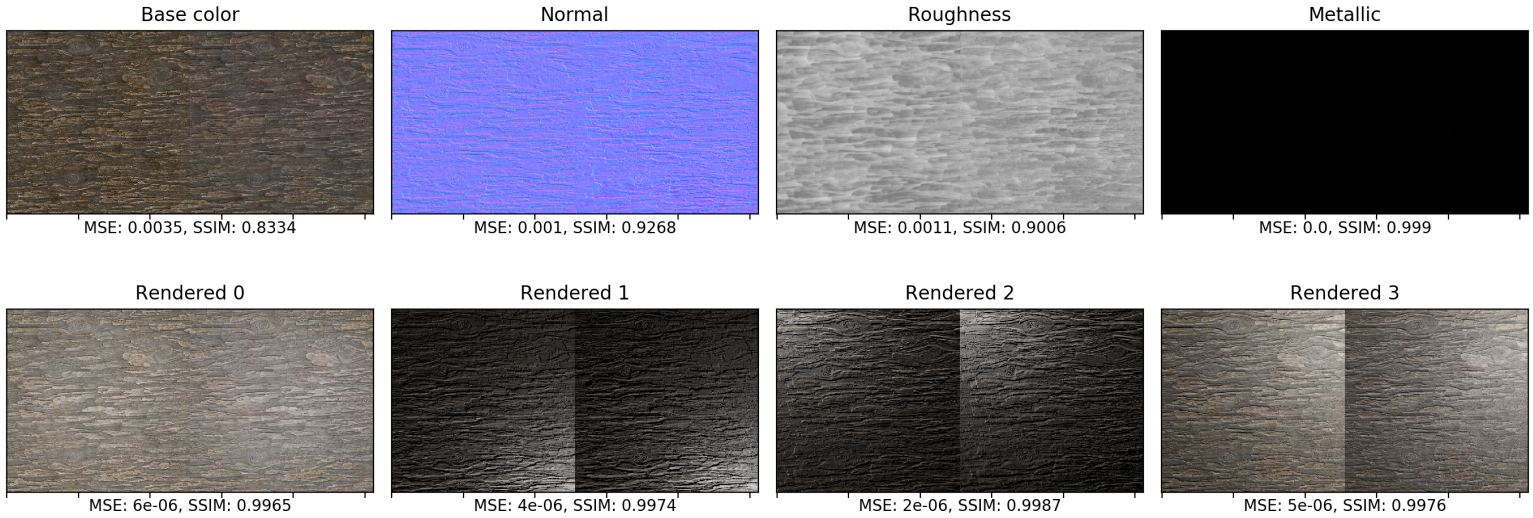


(a) Convolutional Neural Network

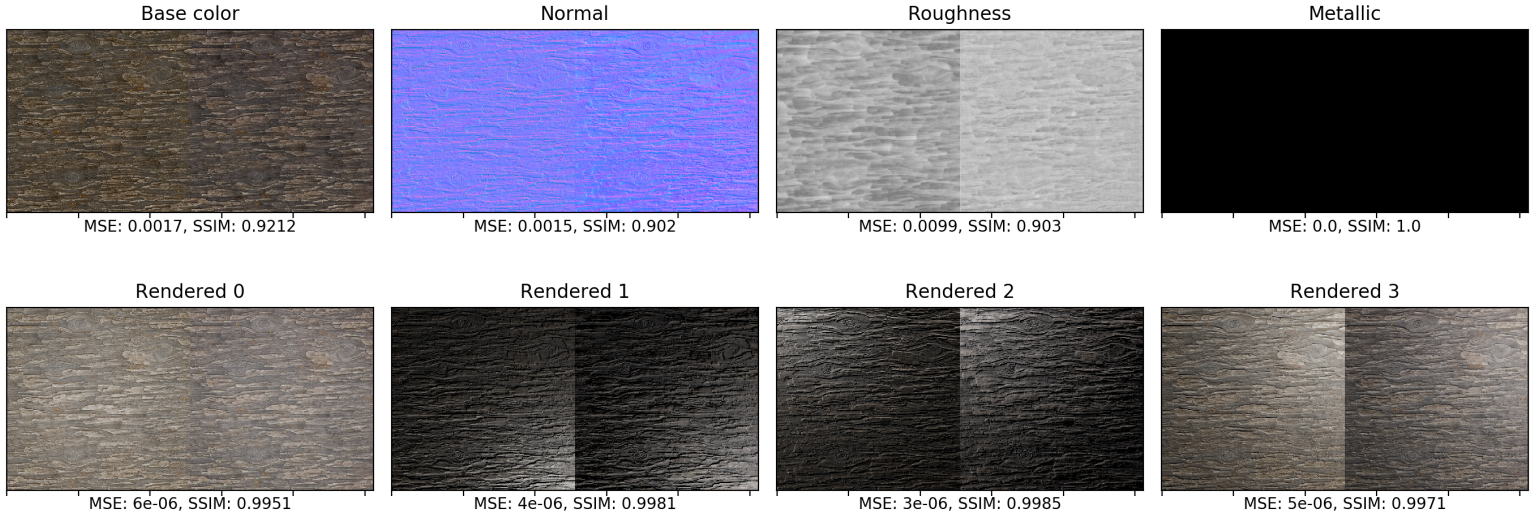


(b) Generative Adversarial Network

Figure A.4.: Brick wall. The left half of each pair is the ground truth and the right is the generated sample.

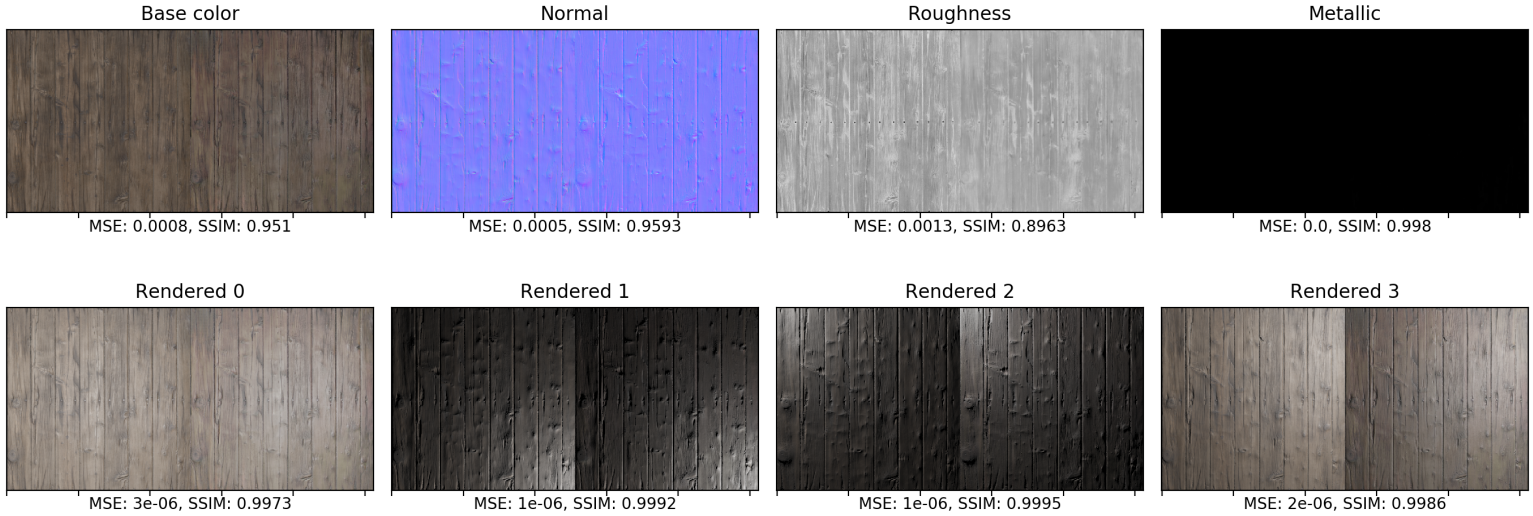


(a) Convolutional Neural Network

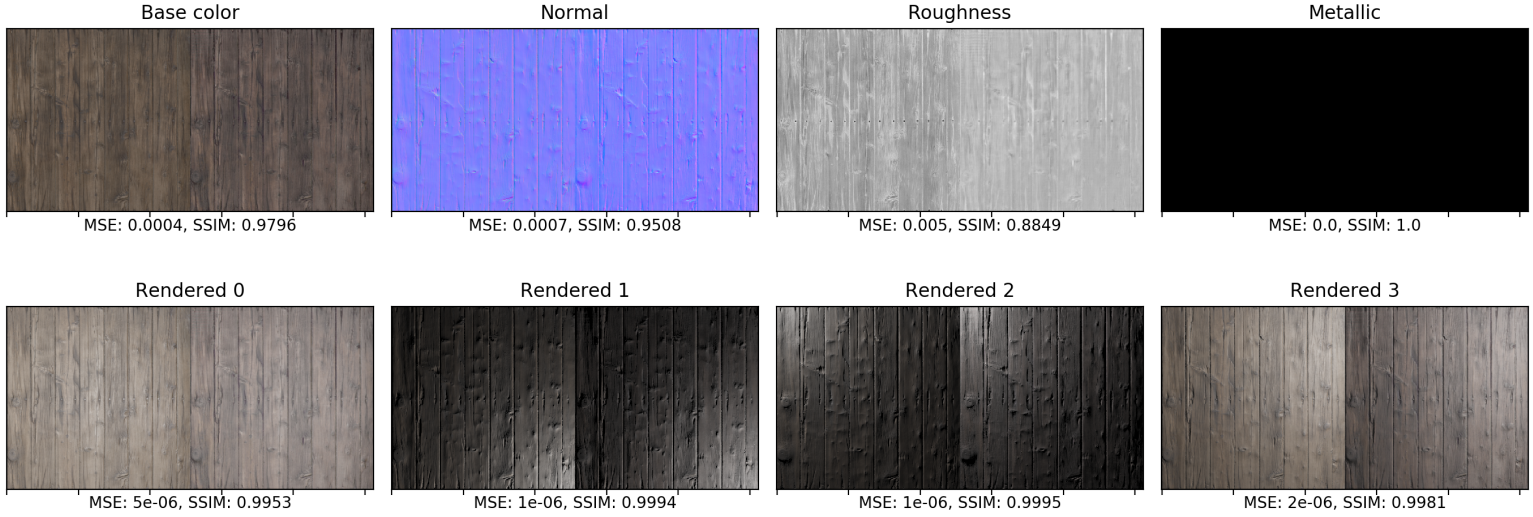


(b) Generative Adversarial Network

Figure A.5.: Tree bark. The left half of each pair is the ground truth and the right is the generated sample.

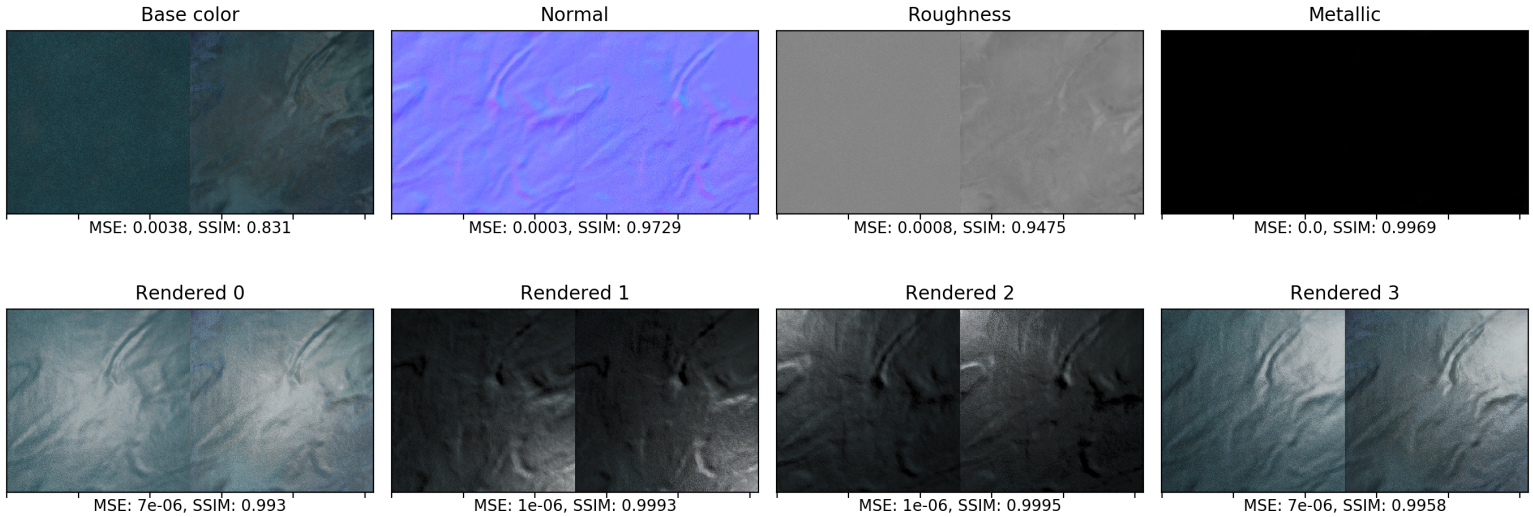


(a) Convolutional Neural Network

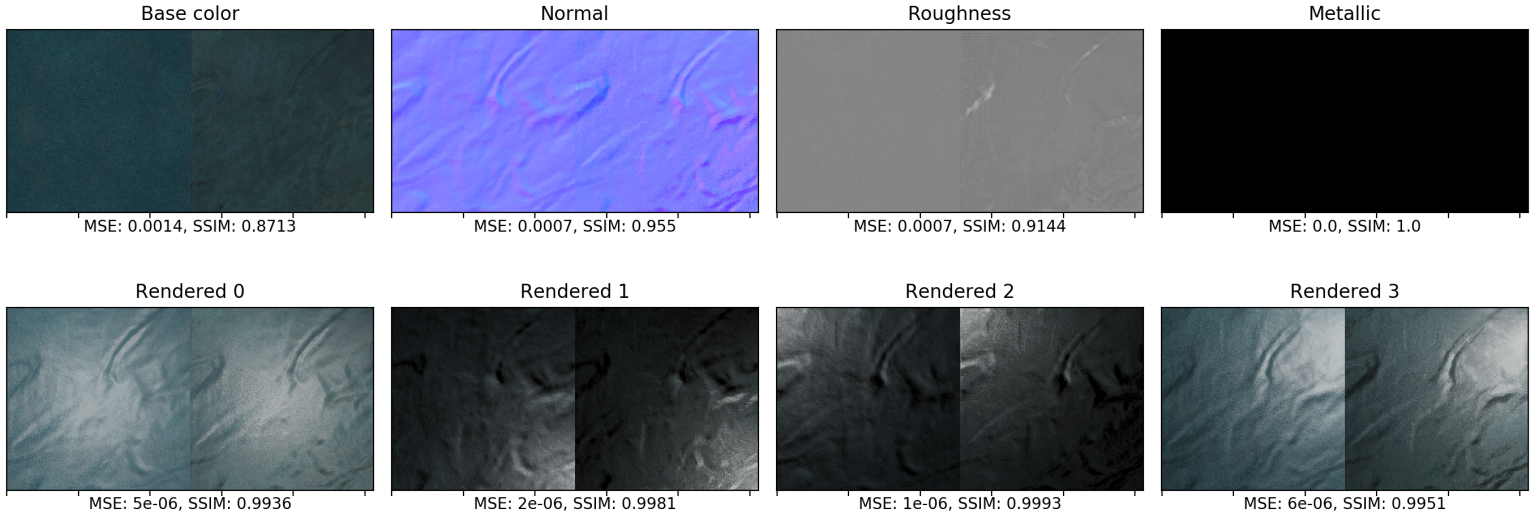


(b) Generative Adversarial Network

Figure A.6.: Old wood planks. The left half of each pair is the ground truth and the right is the generated sample.



(a) Convolutional Neural Network



(b) Generative Adversarial Network

Figure A.7.: Leather. The left half of each pair is the ground truth and the right is the generated sample.

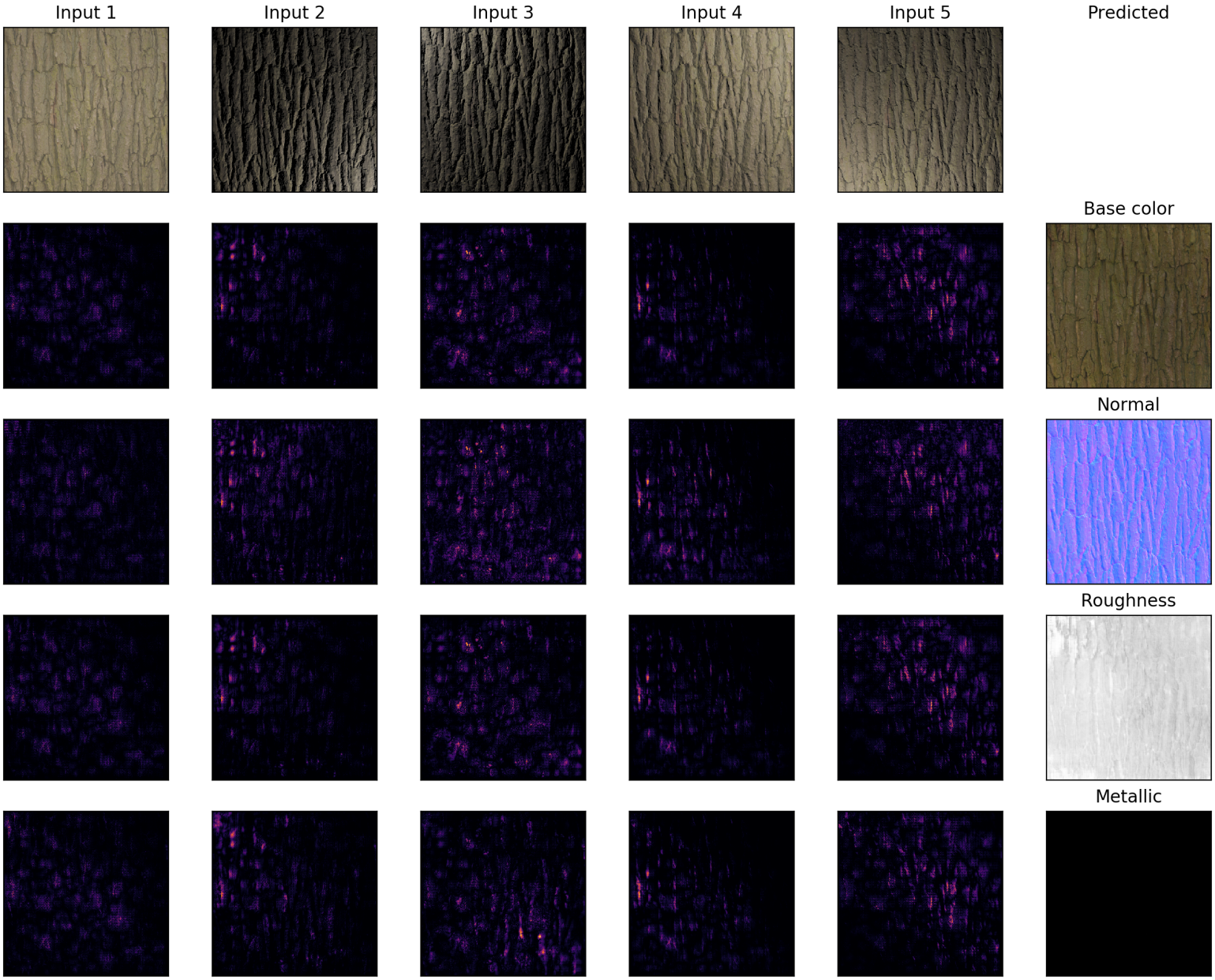


Figure A.8.: Tree Bark saliency maps

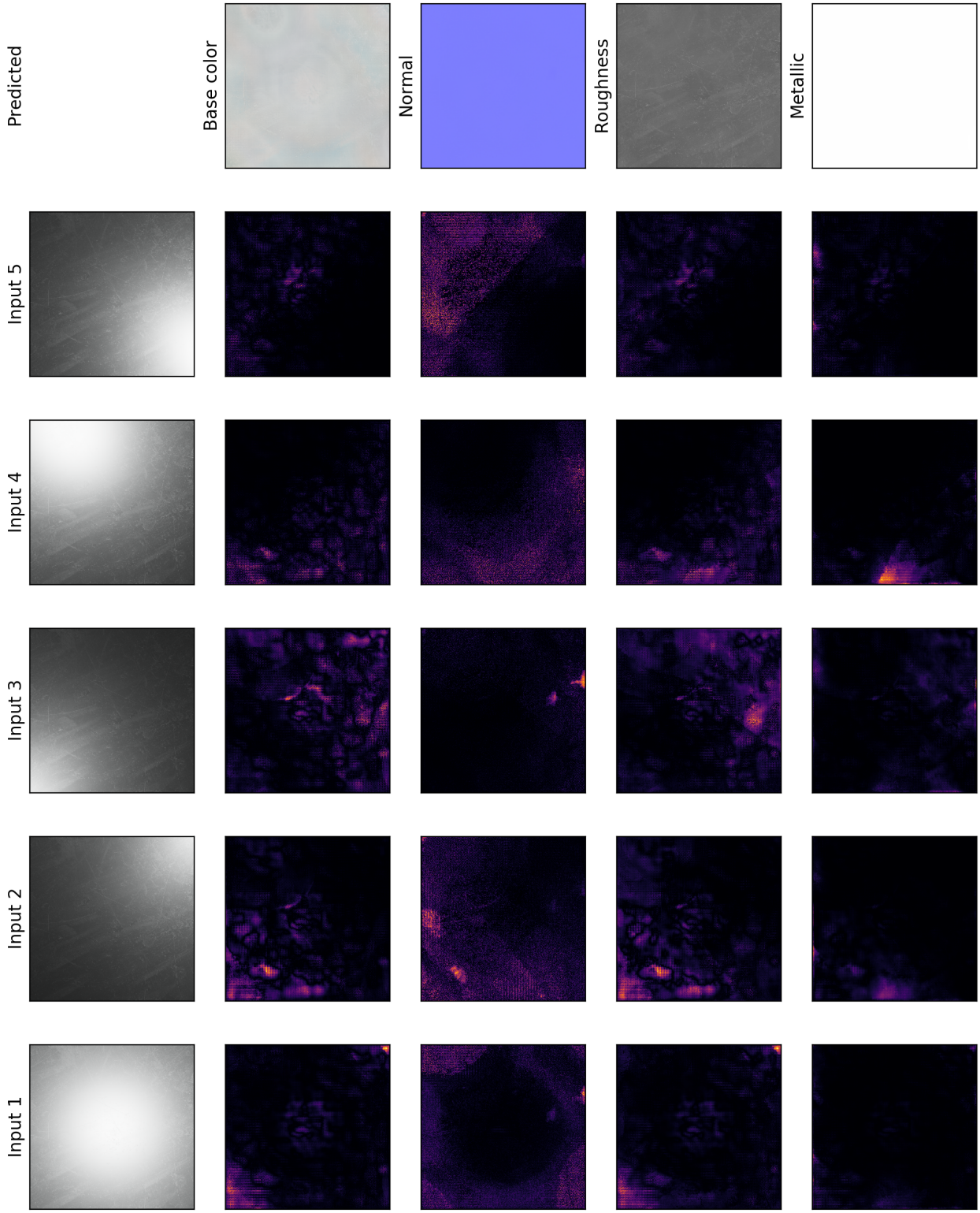


Figure A.9.: Metal saliency maps

Bibliography

[1] Miika Aittala, Tim Weyrich, and Jaakko Lehtinen. Two-shot svbrdf capture for stationary materials. *ACM Trans. Graph.*, 34(4):110:1–110:13, July 2015. ISSN 0730-0301. doi: 10.1145/2766967. URL `http://doi.acm.org/10.1145/2766967`.

[2] Léon Bottou. On-line learning in neural networks. chapter On-line Learning and Stochastic Approximations, pages 9–42. Cambridge University Press, New York, NY, USA, 1998. ISBN 0-521-65263-4. URL `http://dl.acm.org/citation.cfm?id=304710.304720`.

[3] Brent Burley. Physically based shading at disney. In *SIGGRAPH 2012*, volume Course Notes, 2012.

[4] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1):7–24, jan 1982. doi: 10.1145/357290.357293.

[5] Yue Dong, Jiaping Wang, Xin Tong, John Snyder, Yanxiang Lan, Moshe Ben-Ezra, and Baining Guo. Manifold bootstrapping for svbrdf capture. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 98:1–98:10, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0210-4. doi: 10.1145/1833349.1778835. URL `http://doi.acm.org/10.1145/1833349.1778835`.

[6] EN ISO 11664-4. Colorimetry - part 4: Cie 1976 l*a*b* colour space. Standard, 2011.

[7] Wolfgang Engel. *ShaderX7: Advanced Rendering Techniques*. Charles River Media, 2009.

[8] Dan B Goldman, Brian Curless, Aaron Hertzmann, and Steven M Seitz. Shape and spatially-varying BRDFs from photometric stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(6):1060–1071, jun 2010. doi: 10.1109/tpami.2009.102.

[9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA, 2014. MIT Press. URL `http://dl.acm.org/citation.cfm?id=2969033.2969125`.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep

into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

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

[12] Eric Heitz. Understanding the masking-shadowing function in microfacet-based brdfs. *Journal of Computer Graphics Techniques (JCGT)*, 3(2):48–107, June 2014. ISSN 2331-7418. URL http://jcgt.org/published/0003/02/03/.

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

[14] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks.

[15] M J. D. Powell. A view of algorithms for optimization without derivatives. *Mathematics TODAY*, 43, 01 2007.

[16] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM. ISBN 0-89791-196-2. doi: 10.1145/15922.15902. URL http://doi.acm.org/10.1145/15922.15902.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

[19] Sebastien Lagarde and Charles De Rousiers. Moving frostbite to pbr. In *Proc. Physically Based Shading Theory Practice*, 2014. URL https://media.contentapi.ea.com/content/dam/eacom/frostbite/files/s2014-pbs-frostbite-slides.pdf.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[21] Xiao Li, Yue Dong, Pieter Peers, and Xin Tong. Modeling surface appearance from a single photograph using self-augmented convolutional neural networks. *ACM Transactions on Graphics*, 36(4):1–11, jul 2017. doi: 10.1145/3072959.3073641.

[22] Feng Lu, Lei He, Shaodi You, Xiaowu Chen, and Zhixiang Hao. Identifying surface BRDF from a single 4-d light field image via deep neural network.

IEEE Journal of Selected Topics in Signal Processing, 11(7):1047–1057, oct 2017. doi: 10.1109/jstsp.2017.2728001.

[23] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.

[24] Stephen R. Marschner and Donald P. Greenberg. Inverse lighting for photography. 08 1998.

[25] Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A data-driven reflectance model. *ACM Transactions on Graphics*, 22(3):759–769, July 2003.

[26] Yasuhiro Mukaigawa, Kohei Sumino, and Yasushi Yagi. Rapid BRDF measurement using an ellipsoidal mirror and a projector. *IPSJ Transactions on Computer Vision and Applications*, 1:21–32, 2009. doi: 10.2197/ipsjtcva.1.21.

[27] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[28] Fred E. Nicodemus. Directional reflectance and emissivity of an opaque surface. *Applied Optics*, 4(7):767, jul 1965. doi: 10.1364/ao.4.000767.

[29] Jannik Boll Nielsen, Henrik Wann Jensen, and Ravi Ramamoorthi. On optimal, minimal BRDF sampling for reflectance acquisition. *ACM Transactions on Graphics*, 34(6):1–11, oct 2015. doi: 10.1145/2816795.2818085.

[30] Jannik Boll Nielsen, Henrik Aanæs, Jeppe Revall Frisvad, and Knut Conradsen. *On Practical Sampling of Bidirectional Reflectance*. PhD thesis, Technical University of Denmark (DTU), 2016.

[31] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *ArXiv e-prints*, November 2015.

[32] Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*. ACM Press, 2001. doi: 10.1145/383259.383271.

[33] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. *ACM Trans. Graph.*, 21(3):267–276, July 2002. ISSN 0730-0301. doi: 10.1145/566654.566575. URL `http://doi.acm.org/10.1145/566654.566575`.

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, 2015.

[35] F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*.

Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957. URL https://books.google.de/books?id=P_XGPgAACAAJ.

[36] Frank Rosenblatt and C. Van Der Malsburg. Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. In *Brain Theory*, pages 245–248. Springer Berlin Heidelberg, 1986. doi: 10.1007/978-3-642-70911-1_20.

[37] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, oct 1986. doi: 10.1038/323533a0.

[38] Christophe Schlick. An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum*, 13:233–246, 1994.

[39] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

[40] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017. URL http://dx.doi.org/10.1038/nature24270.

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

[42] Leslie N. Smith. No more pesky learning rate guessing games. *CoRR*, abs/1506.01186, 2015. URL http://arxiv.org/abs/1506.01186.

[43] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, 2014.

[44] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

[46] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. URL http://arxiv.org/abs/1607.08022.

[47] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, pages 195–206, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. ISBN 978-3-905673-52-4. doi: 10.2312/EGWR/EGSR07/195-206. URL http://dx.doi.org/10.2312/EGWR/EGSR07/195-206.

[48] Z. Wang, E.P. Simoncelli, and A.C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402 Vol.2. IEEE, 2003. doi: 10.1109/acssc.2003.1292216.

[49] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, apr 2004. doi: 10.1109/tip.2003.819861.

[50] Gregory J. Ward. Measuring and modeling anisotropic reflection. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, pages 265–272, New York, NY, USA, 1992. ACM. ISBN 0-89791-479-1. doi: 10.1145/133994.134078. URL http://doi.acm.org/10.1145/133994.134078.

[51] Ye Yu and William A. P. Smith. Pvnn: A neural network library for photometric vision. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[52] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging*, 3 (1):47–57, mar 2017. doi: 10.1109/tci.2016.2644865.

[53] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017. URL http://arxiv.org/abs/1703.10593.