Learning Sample-Based Monte Carlo Denoising from Noisy Training Data

by

Andrew Tinits

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Monte Carlo rendering allows for the production of high-quality photorealistic images of 3D scenes. However, producing noise-free images can take a considerable amount of compute resources. To lessen this burden and speed up the rendering process while maintaining similar quality, a lower-sample count image can be rendered and then denoised after rendering with image-space denoising methods. These methods are widely used in industry, and have recently enabled advancements in areas such as real-time ray tracing. While hand-tuned denoisers are available, the most successful denoising methods are based on machine learning with deep convolutional neural networks (CNNs). These denoisers are trained on large datasets of rendered images, consisting of pairs of low-sample count noisy images and the corresponding high-sample count reference images. Unfortunately, generating these datasets can be prohibitively expensive because of the cost of rendering thousands of high-sample count reference images.

A potential solution to this problem comes from the Noise2Noise method [29], where denoisers can be learned solely from noisy training data. Lehtinen et al. applied their technique to Monte Carlo denoising, and were able to achieve similar performance to using clean reference images [29]. However, their model was a proof of concept, and it is unclear whether the technique would work equally well with state-of-the-art Monte Carlo denoising methods. The authors also do not test their hypothesis that better results could be achieved by training on the additional noisy training data that could be generated with the same compute budget that was previously allocated to generating clean training data. Finally, it remains to be seen whether the authors' suggested parameters are equally effective when Noise2Noise is used with different denoising methods.

In this thesis, I answer the above questions by applying Noise2Noise to a state-of-the-art Monte Carlo denoising algorithm called Sample-Based Monte-Carlo Denoising (SBMC) [19]. I adapt the SBMC scene generator to produce a dataset of noisy image pairs, use this dataset to train an SBMC-like CNN, and conduct experiments to determine the impact of various parameters on the performance of the denoiser. My results show that the Noise2Noise technique can be effectively applied to a state-of-the-art Monte Carlo denoising algorithm. I achieved comparable results to the original implementation at a significantly lower cost. I find that using additional training data can further improve these results, although more investigation is needed in this area. Finally, I detail the parameters that were necessary to achieve these results.

Acknowledgements

I would like to thank my advisor, Stephen Mann, for his help and support throughout the completion of my thesis. Thank you to my thesis readers, Morgan McGuire and Yuri Boykov, for their helpful comments. I would also like to thank the members of the Computer Graphics Lab for fostering a collaborative research environment.

I would like to acknowledge Compute Canada, NSERC, and David R. Cheriton for providing the resources that made this project possible.

Finally, thank you to my parents, Sabrina, Markus, Adam, Nick, Alex, Peter, Zameer, Chris, and Lucy for keeping me sane over the last few years.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ANN artificial neural network 14

BRDF bidirectional reflectance distribution function 10, 11, 13

BSDF bidirectional scattering distribution function 11, 13

BTDF bidirectional transmittance distribution function 11

CNN convolutional neural network iii, vii, 4, 8, 15–17, 21, 26

DSSIM structural dissimilarity index viii, ix, 40, 43, 44, 49

GAN generative adversarial network 18

GPU graphics processing unit 2, 29, 30, 36, 37

HDR high dynamic range viii, 21, 23, 25, 26, 28, 29, 36

MLP multilayer perceptron 14

MSE mean squared error viii, ix, 29, 31, 33, 35–37, 40, 43, 44, 47–49

SBMC Sample-Based Monte-Carlo Denoising iii, viii, ix, 8, 18, 21, 23–26, 28–31, 35–37, 40, 42–45, 48–53

SGD stochastic gradient descent 15

SSD solid-state drive 30

SSIM structural similarity index 40, 50

On turning back a man has more wisdom
than he had upon departing.
Wildly I sped on this empty errand,
treading the trail of the wind;
merrily I met the waves,
wandering on their wet trail.
I wanted to touch the wall
which stood behind heaven,
to search for the earth's end
and poke it with my fingers.

. . .

Yet, dear brothers, no regret
can stem from this voyage.
Knowledge must be held
as higher than a silver treasure,
more precious than heaps of gold!
Thus, on our errant route,
on that delusive pasture path,
we found many truthful tidings:
that the wide world has no end;
that Taara, in his wisdom,
fixed no limit anywhere,
set no impassable barriers.

And whatever gainful grain
I might have plowed or planted
on those foreign fields
will furnish us with food for thought
until we reach our life's end.

F. R. Kreutzwald and J. Kurman,
Kalevipoeg: An Ancient Estonian Tale,
Symposia Press, 1982, p. 212.

Chapter 1

Introduction

Monte Carlo rendering is an important concept in the field of computer graphics. Like other rendering methods, these algorithms take as input a description of a 3D scene, including object geometries, material and lighting properties, and camera parameters. The output is a 2D rendered image of the scene as viewed by the camera. Unlike many other rendering methods, the image produced is photorealistic. This means that the image appears exactly as it would if the same scenario were set up in the real world. This is achieved by Monte Carlo rendering being a physically-based rendering algorithm. This class of algorithms construct the image by simulating rays of light travelling around the scene while following the rules of physics as closely as possible.

In particular, Monte Carlo rendering is a randomized algorithm that samples the space of possible light paths through the scene to eventually converge on a solution. As more and more light paths are sampled, this process produces an image that is gradually refined towards the correct result. At the beginning, the resulting image appears noisy, and as the process continues this noise is reduced and the image becomes more and more clear (see Figure 1.1 for an example). Although this process converges in the limit to the correct solution, achieving a result with an acceptably low level of noise can often require many thousands of iterations. This computation can take many hours or even days for a single image on a modern computer. These long run times are due to the convergence properties of the algorithm, in which the remaining error is proportional to the reciprocal of the square root of the number of samples computed (see Figure 1.2) [43]. Because of this property, the majority of the error in an image will be removed early in the rendering process, while reducing the remaining error will often take a huge number of samples.

In contrast to many other rendering methods that prioritize speed while delivering ac-

Figure 1.1: A Monte Carlo rendered image at 8 samples per pixel (left) and at 8192 samples per pixel (right) [19]

ceptable visuals, Monte Carlo rendering prioritizes accuracy at the cost of speed. The emphasis on speed primarily comes from the world of real-time graphics, where dozens of new image frames must be produced and presented to the user each second to maintain the illusion of on-screen motion. Even with the fastest modern computer hardware, the physically-based rendering algorithms described above have traditionally been too slow to be usable for real-time graphics applications. The standard applications for physically-based rendering algorithms have been in offline rendering scenarios such as animated movies, visual effects for film and television, and advertising. Physically-based rendering algorithms are well suited for these tasks as they produce the most accurate and visually impressive renderings, and studios are generally willing to spend the extra computing time for these results.

This status quo has recently begun to change, however, as computers become ever faster and as algorithms continue to be improved. While performing physically-based rendering in real time has long been a dream of many in the computer graphics field, that dream has recently started to become a reality. In fact, the major graphics processing unit (GPU) producers have recently begun to incorporate computing units into their processors that are specifically tailored to the needs of physically-based rendering algorithms. This has led to the recent release of mainstream video games that make use of physically-based rendering algorithms to achieve effects that were previously difficult or impossible to produce with

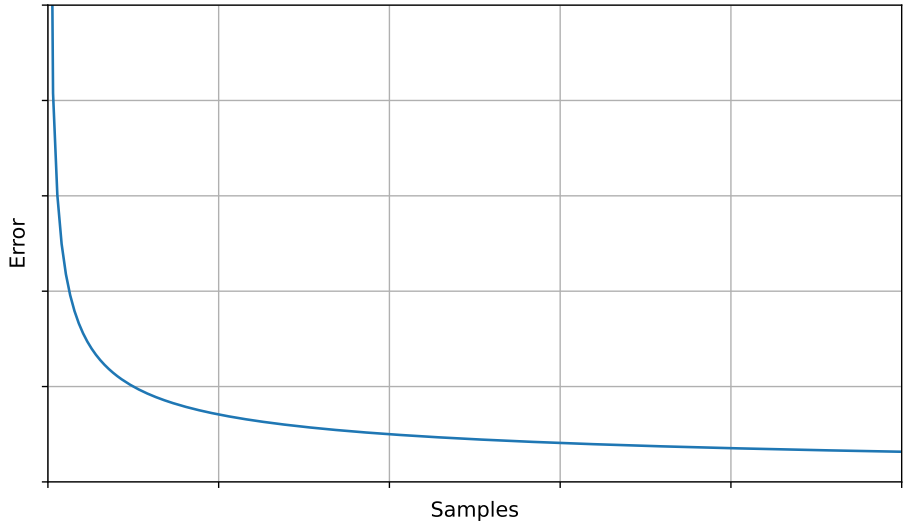


Figure 1.2: Error in Monte Carlo rendering is proportional to $\frac{1}{\sqrt{Samples}}$

traditional real-time rendering methods.

A large part of the recent success in adapting physically-based rendering methods for real-time applications has been due to the use of denoising. Denoising methods involve stopping the Monte Carlo rendering process early, and then applying post-processing techniques to reduce the remaining noise to an acceptable level without sampling more light paths. This results in an acceptable-looking image that can be produced in much less time, in some cases achieving real-time frame rates. Denoising methods are also useful in offline settings, as they can save costs and shorten the long rendering times (see Figure 1.3 for an example). Denoising does, however, often result in dropping one of the mathematical guarantees of the Monte Carlo rendering process, namely the unbiased nature of its results. This is often acceptable as the goal is generally to produce a noise-free image with the desired global illumination effects in a reasonable amount of time, not necessarily to produce a mathematically unbiased image.

Denoising can be accomplished by many different techniques, but the most common ones involve using either hand-tuned or learned filters to reduce the noise. A denoising algorithm takes as input the noisy output from the Monte Carlo renderer, and usually some additional features that the renderer provides such as geometry, material, and lighting properties. The denoiser will take this information for each pixel in the image, and generally

Figure 1.3: A Monte Carlo rendered image at 8 samples per pixel (left) and denoised with my denoising model (right) [19]

look at the neighbouring pixels as well, to determine a denoised value for the original pixel. As there is often a limited amount of information available, especially in low sample count renderings, performing this task successfully will generally require the use of prior knowledge about the properties of rendered images.

There has been a good amount of success recently in using machine learning techniques to learn Monte Carlo denoising programs rather than hand-tune their implementations. These methods have produced state-of-the-art denoisers that can surpass the performance of their hand-tuned counterparts [19]. Learning-based denoisers are trained on large datasets of rendered images and their associated feature data, traditionally containing pairs of low-sample count noisy images along with their corresponding high-sample count clean reference images. In this supervised learning arrangement, an adaptable denoising model is first constructed with parameters that can be adjusted to better fit the data. In recent state-of-the-art approaches, these models are generally deep convolutional neural networks (CNNs) [19]. The training data is then fed into the model one example at a time, and the model will attempt to denoise the noisy input image using its current parameter values. The result is compared against the clean reference image using a loss function, and then the parameters of the model are updated using an optimization algorithm to better fit the data (see Figure 1.4). The goal of this process is to produce a denoising program that both performs well on the training dataset, and can also generalize

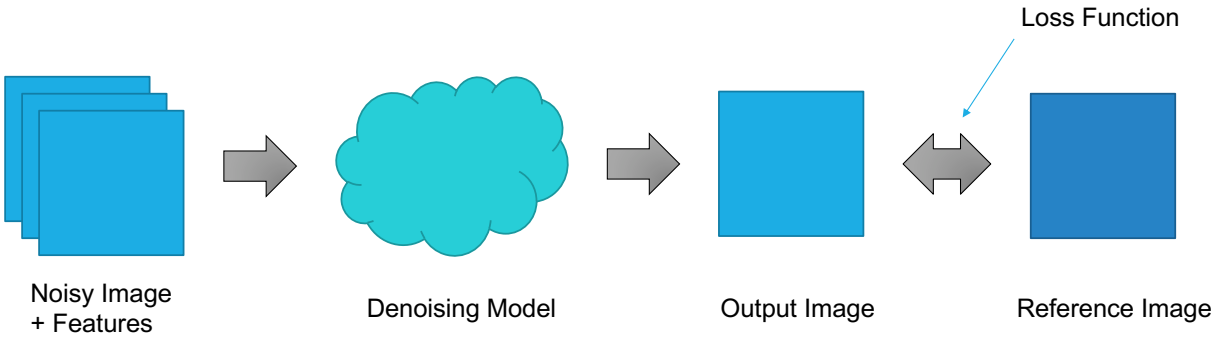


Figure 1.4: Overview of the supervised learning configuration for training a Monte Carlo denoising model

to perform well on new inputs that were not part of the training data.

This supervised learning process, while effective when all of the components above are in place, has one major real-world hurdle. Generating the required training data can take an immense amount of computing power, such that only the largest organizations with correspondingly substantial computing budgets can afford to construct these datasets [19]. Training deep learning models can often require tens or hundreds of thousands of training examples, and for each of those training examples a clean reference image is required. As described above, rendering a single clean reference image for a moderately complex scene can take many hours to days on a single computer, so generating tens to hundreds of thousands of them is a massive undertaking.

As an illustration, assuming that 100,000 reference images are required and that each of them takes one day to render on a single 32-core computer, then even with access to 1,000 computers in parallel, rendering these images would take over three months to complete. With the cost to rent a 32-core server currently at around $1 per hour [2], this equates to a cost of roughly $2.4 million. This amount is beyond the reach of most academic research groups and all but the largest private companies. Other limiting factors include the storage and transfer costs of these large datasets, which can reach several terabytes in size. Using current pricing from a major cloud computing provider [1], storing a 5TB dataset would cost around $100 per month, while transferring this dataset over the Internet would cost almost $500. Some organizations are fortunate in having access to data that can be repurposed for training Monte Carlo denoisers, such as animated films. These films can contain over 100,000 frames, which are often produced by Monte Carlo rendering. Training a denoising model using this data would require access to the uncompressed reference frames, as well

as noisy versions of those frames and the associated feature data. Unfortunately, the few organizations that have access to this data do not share it publicly.

There have been several publicly released denoising models that were trained with these types of datasets, such as those from Intel and Nvidia [3, 13]. However, the datasets themselves are themselves are rarely published. One exception is the dataset published by Xu et al., which features 1000 renderings of indoor scenes that can be divided into smaller patches for training [55]. The main issue with this dataset is that the images are generated by a proprietary commercial renderer [55]. Since different rendering systems can have different noise properties, models trained with this dataset could have trouble adapting to other renderers, and would likely need to be fine-tuned with additional training data from the target rendering system. Their dataset also includes a limited number of additional features [55], which may not meet the needs of every denoising model. In general, the difficulty of generating and distributing training data severely limits who can participate in research on these denoising methods, and hinders the ability of researchers to reproduce results in this field.

One potential solution to this problem comes from a method called Noise2Noise [29]. Noise2Noise is a technique where denoisers can be learned solely from noisy training data. With this technique, instead of consisting of pairs of noisy and clean images, the training set is made up of noisy pairs, where each member of the pair is a different noisy realization of the same image. The authors present some mathematical intuition that, under certain conditions, this process will still converge in the limit to the same solution as training with clean reference images [29]. The main requirement for Noise2Noise training is that the noise distribution must have zero mean, which is the case for Monte Carlo noise [29]. The error that is introduced by using noisy reference images approaches zero as the number of training examples increases [29]. This property is being used in the clean reference paradigm as well, since in reality there is no such thing as a perfectly clean reference image with Monte Carlo rendering. In practice there will always be some amount of noise left over when the rendering process is finished. The intuition for Noise2Noise is that each training step will, on average, direct the model output toward the mean of the underlying noise distribution, which corresponds to the clean reference image (see Figure 1.5) [29]. This method works well in practice, and the authors demonstrate several successful applications of their technique [29].

Although their method is applicable to any type of denoising, the authors of Noise2Noise demonstrate their technique on the particular case of Monte Carlo denoising [29]. However, their implementation is primarily a proof of concept and does not achieve state-of-the-art results or make use of some of the more advanced Monte Carlo denoising techniques. The authors do make a comparison between using noisy versus clean reference images, and find

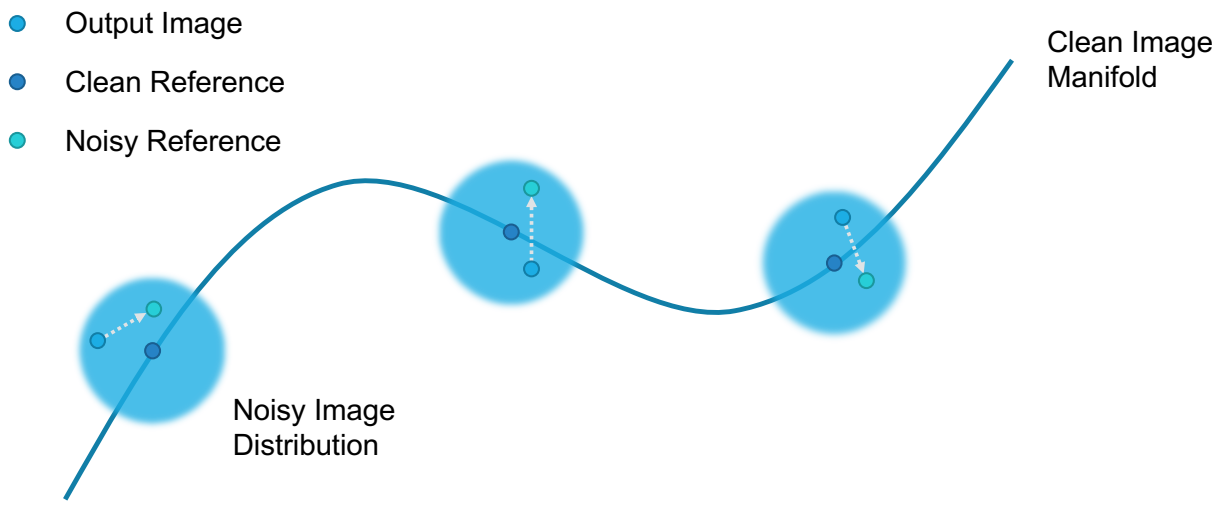


Figure 1.5: Overview of Noise2Noise training [29]

that they are able to achieve the same denoising performance with the noisy references. However, training with the noisy reference images took approximately twice as long to converge to this result [29]. The main benefit of using noisy references is that they are much faster to generate, with the authors noting a speedup of a factor of 2000 over the clean references [29]. The authors hypothesize that one could therefore produce a much larger quantity of noisy references than clean ones within the same compute budget, and in this way potentially achieve even greater performance than with clean references [29]. This hypothesis is supported by an experiment that the authors performed on a different denoising task [29].

Noise2Noise seems like a promising solution to the difficulties of training Monte Carlo denoisers on a reasonable budget. However, significant questions remain to be answered. First, it is important to establish whether Noise2Noise can be applied to state-of-the-art denoising techniques to achieve similar results on a smaller budget. Second, it remains to be seen whether using noisy references could allow for even better performance to be achieved by generating and training on significantly more example pairs than would be possible in the traditional supervised learning setting. Finally, the Noise2Noise paper makes a number of recommendations about parameters such as tone mapping and loss functions that the authors found to be helpful in improving the training process [29]. It is unclear whether these suggestions apply more generally to the use of Noise2Noise with other denoisers, or whether they are specific to the situation in that paper. The answers to these questions could greatly reduce the costs of achieving state-of-the-art Monte Carlo denoising results,

and offer clues as to how to improve these results even further. Reducing costs would also help to make research in this area accessible to more researchers, and make it easier to reproduce results in this field.

In this thesis, I answer the above questions by applying Noise2Noise to a state-of-the-art Monte Carlo denoising algorithm called Sample-Based Monte-Carlo Denoising (SBMC) [19]. This algorithm trains a CNN directly on the samples produced by the renderer to produce an effective denoiser that is good at handling low-sample count images [19]. SBMC is also notable for providing sample code as well as a set of test scenes that can be used for comparisons with other denoising algorithms. I adapt the SBMC scene generator to produce a dataset of noisy image pairs and their associated features, use this dataset to train an SBMC-like CNN, and conduct experiments to determine the impact of various parameters. This dataset can be generated for a computing cost of around $900 rather than millions of dollars.

My results show that the Noise2Noise technique can be effectively applied to a state-of-the-art Monte Carlo denoising algorithm to achieve results comparable to the original implementation at a significantly lower cost. I find that using additional training data can further improve these results, although more investigation is needed in this area. And finally, I detail the parameters that were necessary to achieve these results. Reducing the cost of achieving state-of-the-art results in Monte Carlo denoising has the benefit of making research in this field more accessible, and making it easier to reproduce results.

Chapter 2

Background

2.1 Monte Carlo Rendering

To introduce Monte Carlo rendering, it is first necessary to explain the basics of ray tracing. Ray tracing is a simple algorithm that forms the basis of most photorealistic rendering methods. The ray tracing process starts at the camera, which is a specification of how and from where the scene will be viewed. During ray tracing, rays are generated at the camera and sent into the scene. The next step is determining whether the ray will intersect with any of the objects in the scene. This step is performed by computing intersection tests between the ray and the various objects in the scene to find the closest hit. Once the closest intersection point is located, the next step is to determine which light sources are visible from that point. This question can be answered by constructing rays from the intersection point to the various lights in the scene, and computing the closest intersections in the same way as for the original ray. If the closest intersection along one of these shadow rays is the light source, then that light is illuminating the original intersection point. With this lighting information available, the ray tracer can now use the surface properties of the object at the intersection point to compute its shaded appearance. A final optional step in ray tracing is to trace additional rays from the original intersection point into the scene. These rays can be used to simulate indirect light transport, where light arrives at a surface after reflecting off or passing through other surfaces.

One of the simplest possible models of a camera is called a pinhole camera. This type of camera consists of a box with a small hole in one wall and a piece of film attached to the opposite wall, as shown in Figure 2.1. Light will enter the box through the hole and create an image on the film. Drawing lines from the corners of the film, through the pinhole,

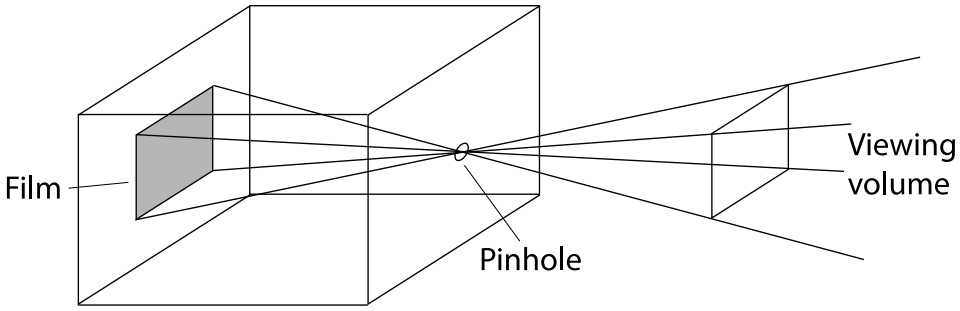


Figure 2.1: A pinhole camera [43, Fig. 1.1 (CC BY-NC-ND 4.0)]

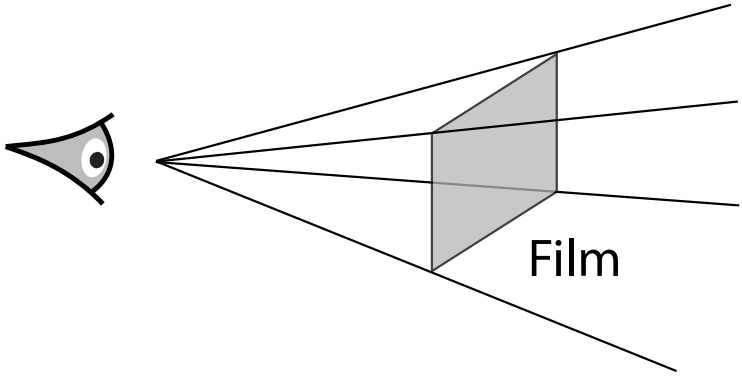


Figure 2.2: A pinhole camera with the film placed in front of the eye [43, Fig. 1.2 (CC BY-NC-ND 4.0)]

and into the scene shows that only objects within the viewing volume will appear on the film. It is convenient in ray tracing to instead place the film in front of the pinhole at the same distance away, as shown in Figure 2.2. Although this configuration is not physically realistic, it is an equivalent model that defines the same viewing volume. In this revised configuration, the pinhole is often referred to as the eye. The film can be logically divided into pixels, and goal of the ray tracing process is then to determine the colour value for each pixel in the film. This colour value is computed by generating rays that travel from the eye, through the corresponding pixel, and into the scene as described above.

Once a ray is traced from the camera to a surface, it is necessary to compute the amount of light energy that is scattered back along that ray towards the camera. As part of the scene description, objects will generally be assigned a material, which describes the scattering properties at every point on the surface. This surface description is called the bidirectional reflectance distribution function (BRDF) [43]. The BRDF at a point p gives

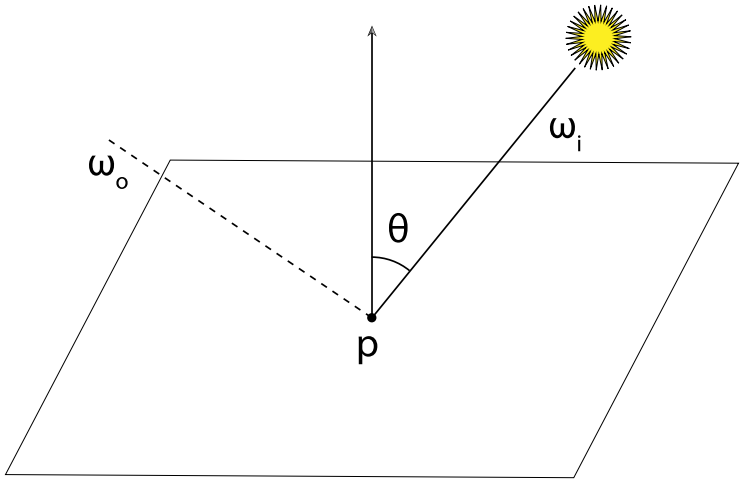


Figure 2.3: The geometry of surface scattering [43, Fig. 1.6 (CC BY-NC-ND 4.0)]

the amount of light energy reflected from an incoming direction ω_i to an outgoing direction ω_o and can be written as $f_r(\mathrm{p}, \omega_o, \omega_i)$ (see Figure 2.3) [43]. The amount of scattered light L is computed as $L = \sum_{\text{Lights}} f_r(\mathrm{p}, \omega_o, \omega_i) L_i$, where L_i is the amount of incident light from a particular light source [43]. L_i is zero when the shadow ray computation determines that the light source is not visible from the point p (see Figure 2.4). BRDFs can be generalized to transmitted light, called a bidirectional transmittance distribution function (BTDF), or to incorporate both transmitted and reflected light, called a bidirectional scattering distribution function (BSDF) [43].

To handle effects like specular reflection and transmission, it is possible to go beyond the single-bounce model and trace additional rays from the initial intersection point. For example, if the initial camera ray were to hit a mirror, then the next ray would start at the intersection point, and have the same direction as the original ray, but reflected about the surface normal at the intersection point. The entire ray tracing routine can then be recursively called with this new ray to find the light arriving along that path. The contributions from all of these rays are then summed to compute the full amount of light arriving at the camera. For a transmissive object, the subsequent ray would be refracted through the surface. It is also possible for the ray tracer to consider multiple effects of this nature, potentially spawning multiple rays at each intersection point. This recursive splitting process results in a tree of rays for each location in the image, originating at the camera and spreading throughout the scene to locate all of the light sources that contribute to that portion of the image (see Figure 2.5).

More generally, the amount of light leaving a point on a surface in a particular direction

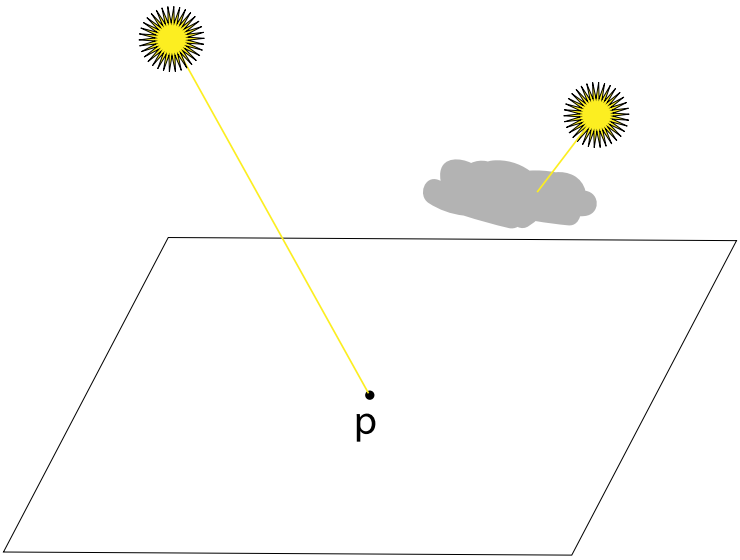


Figure 2.4: Shadow rays can determine whether a light source is visible from a point p on a surface [43, Fig. 1.5 (CC BY-NC-ND 4.0)]

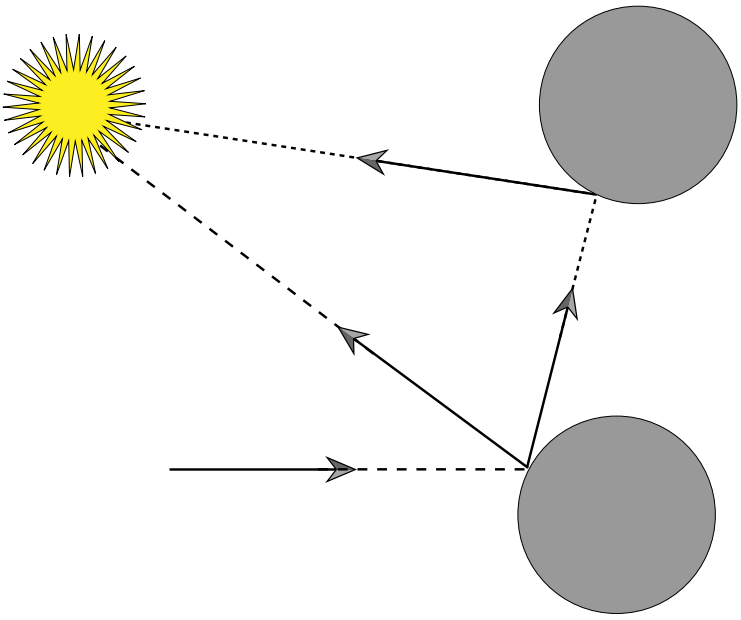


Figure 2.5: A tree of rays produced by recursively ray tracing a scene [43, Fig. 1.8 (CC BY-NC-ND 4.0)]

can be calculated by summing the light emitted by the object itself, along with the amount of light reflected off the object in that direction. This concept, based on the principle of conservation of energy, is formalized as the rendering equation [24]. In this equation, the left hand side is the outgoing radiance $L_o(\mathrm{p}, \omega_o)$ leaving point p in the direction ω_o [43]. This value is computed as the sum of the emitted radiance at that point and in that direction, $L_e(\mathrm{p}, \omega_o)$, and the incoming radiance from all directions on the sphere S^2 around p, multiplied by the BSDF $f_r(\mathrm{p}, \omega_o, \omega_i)$ and a cosine term [43]:

$$L_o(\mathrm{p}, \omega_o) = L_e(\mathrm{p}, \omega_o) + \int_{\mathrm{S}^2} f_r(\mathrm{p}, \omega_o, \omega_i) L_i(\mathrm{p}, \omega_i) |\cos \theta_i| \mathrm{d}\omega_i$$

Finding solutions to the rendering equation forms the basis for many photorealistic rendering algorithms. This integral equation is impossible to solve analytically except for simple scenes, so these rendering algorithms either make simplifying assumptions or use numerical integration methods [43]. The algorithm described above simplifies the problem by only evaluating the integral for directions to light sources and for perfect reflection and refraction. To compute global illumination effects such as diffuse inter-reflection, it is necessary to consider all of the possible directions that light can travel throughout the scene. Performing these computations on the full integral requires the use of numerical techniques such as Monte Carlo integration [36]. These algorithms evaluate the integral by taking many random samples over the various directions and weighting the samples according to the corresponding BRDF values [43]. Monte Carlo rendering methods of this type can produce photorealistic images, since they simulate the full range of possibilities for how light can travel through a scene.

2.2 Convolutional Neural Networks

The idea for neural networks was originally inspired by biological neurons in humans and animals [26]. The basic unit of a neural network is therefore the neuron, roughly modeled after its biological equivalent (see Figure 2.6). Biological neurons receive incoming signals on their dendrites, perform some signal processing in the cell body, and transmit output signals through the single axon. A simple approximation of the signal processing performed by the neuron is that the input signals are summed, and if the resulting value is above a certain threshold, then an output spike is sent along the axon [26]. The axon forms branches, which connect to the dendrites of other neurons to form synapses.

In the mathematical model of the neuron, the inputs that arrive at the dendrites are labeled x_i. The number of inputs can vary for different neural network structures. Each

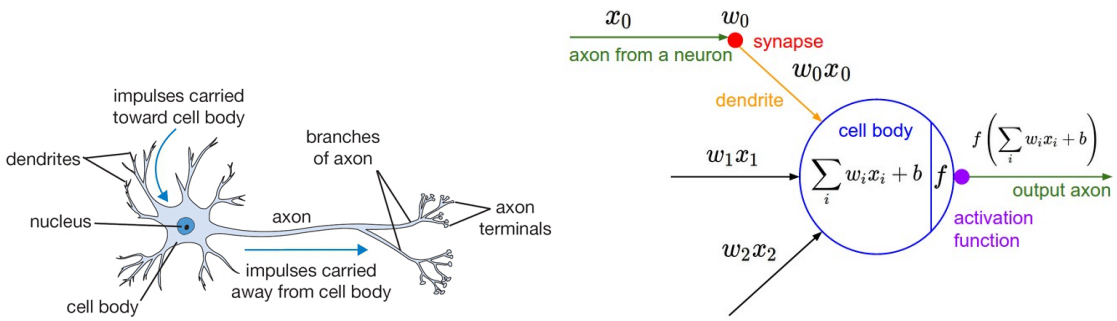


Figure 2.6: A biological neuron (left) and its mathematical model (right) [26]

input has an associated weight w_i stored in the neuron that represents the importance of that connection. These weights are learnable parameters of the neuron, and control the amount of influence that the connected neurons have on that neuron [26]. Input values are first multiplied by the corresponding weights, and then summed in the cell body along with a single learnable bias value b. The firing rate of the biological neuron is modeled by a non-linear activation function f, which is applied to the resulting sum to produce the neuron's output value. The activation function represents the frequency of spikes travelling along the axon [26].

Neural networks are formed from collections of individual neurons, where the neurons are connected together in a graph structure (see Figure 2.7). In this structure, the outputs of some of the neurons become inputs to other neurons. Cycles in the graph are not allowed, as they would form infinite loops of dependencies [26]. Neural networks are often organized into layers, where neurons in a particular layer are connected to those in the previous layer and the following layer, but there are no connections within the same layer. The most common type of layer is the fully-connected layer, where neurons in adjacent layers share all of the possible connections between each other. The layers are often arranged into a stack, with one or more hidden layers located between the input layer and the output layer. These types of neural networks are referred to as artificial neural networks (ANNs) or multilayer perceptrons (MLPs) [26].

The next question that naturally arises is how to train these neural networks to complete a desired task. This training is done by a method called *backpropagation*, which is a way of computing gradients by recursively applying the chain rule of differentiation [26]. During the training process, the neural network is first given an example input, and the corresponding output is calculated via a forward pass through the network. This output is then compared to the reference value using a loss function. Next, backpropagation is used to compute the gradient of the loss function with respect to the weights of the

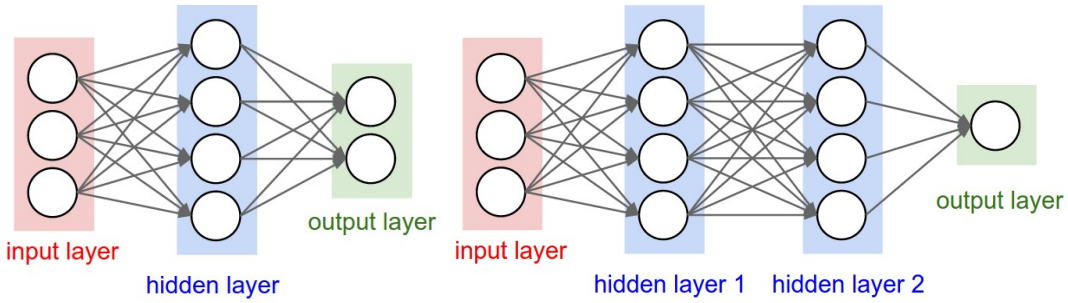


Figure 2.7: Neural networks with one hidden layer (left) and with two hidden layers (right) [26]

network. Finally, the network weights are updated to minimize the loss by using these gradients along with an optimization algorithm such as stochastic gradient descent (SGD). Backpropagation works by computing the gradients one layer at a time, moving backward from the last layer of the network to the first layer.

CNNs are a specific type of neural network architecture designed for image processing. The fully-connected networks described above do not scale well to images, as reasonably sized images have a large number of pixels, which would result in huge numbers of connections and weights for such a network [26]. CNNs differ from regular neural networks in that their layers have neurons arranged in three dimensions: width, height, and depth (see Figure 2.8). For an input image, this could correspond to the image's width, height, and number of colour channels, for example. Instead of being fully connected, the neurons in a layer are only connected to a small region of the preceding layer [26]. In addition, the layers in a CNN make use of parameter sharing, where the all of the neurons in a single two-dimensional depth slice share the same weights and bias. This parameter sharing scheme operates with the assumption that a feature that is useful at one point in an image is likely to be useful at other points as well [26]. Together, these optimizations drastically reduce the number of parameters in a CNN and allow them to be much more efficient during training and inference.

CNNs are constructed out of several types of layers. Each layer transforms an input 3D volume to an output 3D volume of neuron activations [26]. Input layers are similar to those used with regular neural networks, and in this case hold the pixel values of the input image. Convolutional layers, the core building block of CNNs, compute the output of neurons that are connected to local regions in the input. These layers can be viewed as a set of filters with a specified receptive field that are passed over the input image [26]. Pooling layers perform a downsampling operation along the spatial dimensions to reduce

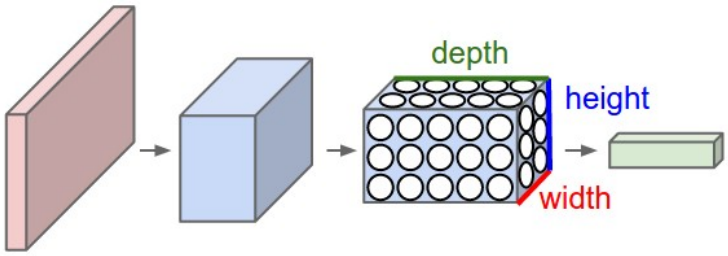


Figure 2.8: A CNN showing the spatial arrangement of neurons within one of the layers [26]

the size of the volume. Finally, fully connected layers are available and operate similarly to those described above, providing connections to all of the neurons in the previous layer. CNNs consist of a stack of these layers, which transform the input image from the original pixel values to the desired outputs.

Chapter 3

Related Work

3.1 Monte Carlo Denoising

Monte Carlo denoising methods can be separated into two main categories: traditional algorithms, and machine learning-based methods. Traditional Monte Carlo denoising algorithms generally rely on hand-tuned regression models that operate over local pixel neighbourhoods. Machine learning-based methods, on the other hand, have models with adjustable parameters that are learned from training data. These two types of denoising methods are briefly reviewed below. For a more comprehensive review, see the survey papers by Zwicker et al. [62] and Huo and Yoon [22].

One of the earliest works in traditional Monte Carlo denoising is the energy preserving non-linear filter by Rushmeier et al [49]. McCool et al. followed this with a method based on anisotropic diffusion that was the first to make use of auxiliary features such as depth maps and normals to improve filtering performance [34]. These works led to later non-linear image space filtering methods, including the cross-bilateral filter [18, 42, 56], the non-local means filter [12], and the edge-avoiding filter [17]. A subsequent thread of work involved combining these non-linear filters with auxiliary feature information to help guide the filtering [41, 47, 38, 48]. The preceding methods were later classified as zero-order regression models [10]. First-order regression methods are also popular [9, 37, 10], and there are some methods that employ higher-order regressions as well [39].

The use of machine learning for Monte Carlo denoising was initiated by Kalantari et al., who used a multi-layer perceptron to predict the parameters of a cross-bilateral filter [25]. Bako et al. improved on this by using a deep CNN to predict filter kernels

separately for each pixel rather than using the same filter weights for the entire image [7]. This kernel-prediction idea is used by many later works [52, 19, 40, 6, 30, 35, 31]. Other recent works, however, take the simpler approach of predicting the pixel radiances directly [13, 55, 58, 57, 54, 20, 32]. Gharbi et al. propose another alternative with SBMC where splatting kernels are used to allow individual pixels to determine their contributions to their neighbourhoods [19]. SBMC also considers the individual Monte Carlo samples separately, which results in improved performance at low sample counts [19]. This concept is extended by Munkberg et al., who partition the samples into layers and filter them separately to improve efficiency [40], and also by Lin et al., who add information about individual Monte Carlo paths to improve denoising performance [31]. Other recent works make use of different machine learning architectures, such as generative adversarial networks (GANs) [55], self-attention [59], and ensembles [61]. Many recent works also focus on real-time denoising and denoising of video sequences [13, 21, 35, 40, 23].

3.2 Weakly-Supervised Image Denoising

In the field of machine learning, there are several different levels of supervision that are commonly used. Fully-supervised learning is the most familiar case, where the machine learning model is trained with high-fidelity reference data. On the other end of the spectrum is unsupervised learning, where the model is trained on example input data only, without using any reference data at all. In the middle is semi- or weakly-supervised learning, where the model is trained with partial or lower-quality references. These approaches are used when it is costly or difficult to acquire full reference data. Noise2Noise is an example of weakly-supervised learning, as it uses noisy reference images for training, which do not have the same quality as clean references [29].

Noise2Noise is not the first work to recognize that clean reference images are not necessary for image denoising. Outside of machine learning methods, there are classic algorithms such as NLM [12] and BM3D [16] that denoise a single noisy image without requiring any additional information. These methods look at statistical information from the rest of the image to determine how to denoise a particular pixel. Within machine learning, Noise2Noise is the first denoising method that can learn from noisy images without requiring prior knowledge of either the noise distribution or the distribution of the clean images [29]. Other methods that can train on noisy images, such as AmbientGAN [11] and GradNet [20], require either an explicit statistical model of the noise (AmbientGAN), or additional regularization that makes assumptions about the clean results (GradNet). In the case of Monte Carlo denoising, the noise distribution cannot be characterized analytically,

so methods with the former requirement cannot be used [29].

Since the release of Noise2Noise, several works have extended the idea so that only individual noisy images are required for training, rather than the noisy image pairs required by Noise2Noise [29]. These works include Noise2Void [27], Noise2Self [8], and Laine et al. [28]. These extensions are useful in settings where noisy image pairs are difficult to obtain, such as in biomedical imaging. However, these methods generally come with reduced performance when compared to the original Noise2Noise method because they have less information available during training [27]. It therefore makes sense to use the Noise2Noise method with Monte Carlo denoising, since noisy image pairs are simple to obtain in this scenario. To my knowledge, no other works besides Noise2Noise itself have evaluated Noise2Noise-style training for Monte Carlo denoising.

Chapter 4

Procedure

4.1 Scene Generation

To train a supervised machine learning model, several key ingredients are needed. The first is a dataset of training examples, consisting of pairs of inputs and their corresponding targets. Next is a model that will attempt to produce the appropriate target output for a given input. This model will have parameters that can be adjusted to better fit the training data, and ideally will allow the model to perform well on inputs that it has not previously been exposed to. The final ingredient is a training algorithm that will feed the training inputs to the model, compare the outputs to the corresponding targets, and update the model parameters accordingly. This section and the following two sections will describe the creation of a training dataset. The next section will discuss the model and its tunable parameters. The final section of this chapter will discuss the procedure for training the model.

In the case of Monte Carlo denoising, the training dataset will generally consist of pairs of noisy low-sample count images and their corresponding clean reference images. The noisy input images are often accompanied by additional feature data that is collected during the rendering process for each image. However, in the Noise2Noise paradigm used here, the input and target images are both noisy realizations of the same image [29]. In other words, they are renderings with the exact same inputs and parameters, but with different random seeds. These images are used to train a model that will take noisy renderings and their corresponding feature data, and produce an approximation of the corresponding clean reference image.

Generating a dataset of rendered images requires a set of scenes to render. In computer graphics, a scene is a digital description of an environment, specifying its geometry, materials, textures, and lighting information. The scene file can also include a description of a camera used to capture an image of the environment, by defining its parameters and orientation. This scene file is processed by a rendering program, which generates an image of the scene as viewed by the camera. These scenes can be manually created in 3D modelling software, and often incorporate assets that are available from various free or commercial libraries. These assets represent the commonly used elements of a scene, such as object geometries, materials, textures, and environmental lighting.

The scenes that make up the training set should include a wide variety of objects, materials, and light transport scenarios. This will ensure that the resulting model can perform well by generalizing to a diverse range of real-world scenes. Training deep CNNs often requires tens to hundreds of thousands of training examples, so a similarly large number of scenes will be needed. Manually creating the scenes with 3D modelling software is therefore not practical in this situation. Possible solutions to this problem include downloading scenes that others have created from the Internet, or automatically generating the scene descriptions. The latter approach will be used here as it is more efficient and ensures that the scenes are selected from a well-defined distribution.

In particular, I used the scene generating code from SBMC to generate the scenes for my training dataset [19]. This scene generator is capable of producing both indoor and outdoor scenes. For indoor scenes, it uses a dataset of room models called SunCG [51]. Unfortunately, this dataset is no longer available online due to legal issues [45]. I was therefore unable to generate indoor scenes, so my dataset consists of only outdoor scenes. This difference could affect the ability of my model to generalize to indoor scenes, and should be taken into account when comparing the performance of my model to SBMC.

For an outdoor scene, the scene generator starts by creating a flat ground plane. Then the generator adds up to 50 objects from the ShapeNetCore dataset [14]. This dataset contains around 51,000 3D models of common objects in 55 different categories [14] (see Figure 4.1 for some examples). These objects are placed in random positions on the ground plane and given random transformations of scale, rotation, and translation [19]. Next, random materials and textures are assigned to each object [19]. The materials are selected from the set offered by the PBRT v2 renderer (e.g. metal, glass, mirror, plastic) [43], and the textures are selected from the Describable Textures Dataset [15]. Describable Textures Dataset is a set of 5640 texture images collected from the Internet and organized into 47 categories [15] (see Figure 4.2 for some examples).

Lighting for the scenes comes from randomly selected high dynamic range (HDR) images

Figure 4.1: ShapeNet models in the chair, laptop, bench, and airplane categories [14]

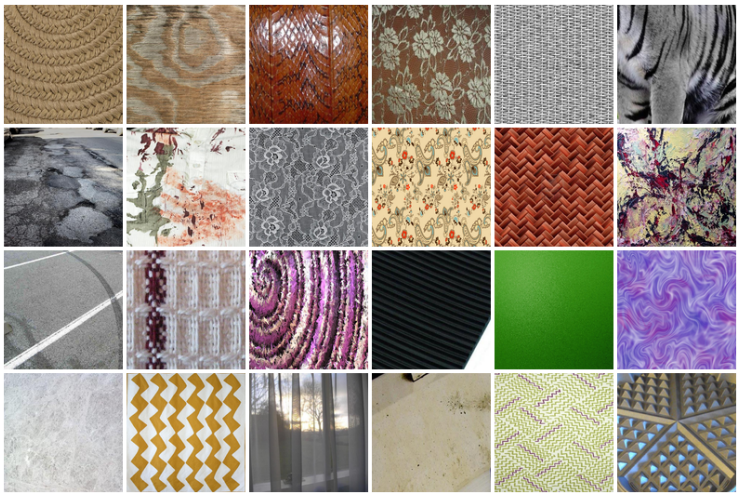


Figure 4.2: Texture images from the Describable Textures Dataset [15]

Figure 4.3: HDR environment maps from Poly Haven [60]

which are used as environment maps [19]. The HDR images are treated as emissive spherical light sources at infinite distance shining inwards toward the centre of the scene. The SBMC authors used 111 HDR images downloaded from the HDRI Haven (now called Poly Haven) website [60]. This website provides free-to-use HDR environment maps for the 3D rendering community [60] (see Figure 4.3 for some examples). For my dataset, I downloaded all of the 382 HDR images that were available on that website at the time. This larger set of environment maps could in theory allow the trained model to generalize better to different lighting scenarios. The scene generator also randomizes camera parameters including field of view, depth-of-field, and shutter speed [19]. Motion blur and depth-of-field effects are each enabled with 50% probability [19]. Finally, there is a rejection sampling process whereby scenes that are too simple or where the camera receives no light are removed [19]. The remaining scenes are then saved in the PBRT v2 scene format for rendering.

4.2 Rendering

For rendering the scenes I used the PBRT v2 renderer [43] with the patch provided by SBMC [19] and a few additional modifications. Although the newest version of PBRT is currently v3 [43], PBRT v2 is used to ensure compatibility with the other denoising methods that were used for comparisons in the SBMC paper [19]. These works ([25], [47], [48], [50]) were also implemented using PBRT v2. In the future, it would be useful to have a universal data format for sample and feature data so that denoisers would not be so closely tied to a particular renderer. However, such standardization is complicated as the required data is different for each denoising implementation. As with SBMC, I used the default perspective camera and path tracing integrator in PBRT v2 [19]. The rest of the rendering parameters were kept the same as well to allow for comparisons between my model and SBMC.

The SBMC patch to PBRT v2 adds several new features to the renderer. Most importantly, it allows the renderer to save the individual Monte Carlo samples separately as required by the SBMC implementation [19]. The patch also adds the ability to gather additional feature data during the rendering process and save that along with the samples [19]. These additional features are detailed in Appendix A of the SBMC paper [19]. Another change included in the patch allows the renderer to generate both a low-sample count noisy image and a high-sample count reference image of the same scene. For the noisy image the individual samples are saved separately, while for the reference image they are averaged together. All of this data is saved in a custom file format that allows for reading only the particular samples that are needed at runtime. The goal of this optimization is to improve read performance during the training process.

The most important of my changes to the rendering process is having the renderer generate two different low-sample count noisy images for each scene instead of one noisy image and one reference image. This change implements the Noise2Noise paradigm, where only noisy image pairs are used for training the denoiser [29]. Not generating the reference images also allows the rendering process to be significantly faster, as noisy images are thousands of times faster to render than clean images [29]. The two noisy images are rendered with different random seeds and the same number of samples. As with the original SBMC implementation, one of the noisy images is stored as separate samples, while the other is treated as the target and has its samples averaged together [19]. This treatment of the images corresponds to the minimal amount of data required for the training process, since the model requires the separate samples as inputs, but only needs the averaged samples to calculate the loss on the target image. Storing the images in this way saves space that would otherwise be used to store the samples of the second noisy image. This storage space can become significant because of the large number of renderings that make

up a training set.

Another minor change I made to the renderer is enabling PBRT v2 to read 32-bit floating point OpenEXR images. I used this image format to store the HDR environment maps instead of the PFM format used by SBMC as it has significantly smaller file sizes. PBRT previously supported reading 16-bit OpenEXR files, but some of the environment maps had values that were too large to fit in this format. Finally, I made some modifications to the lossless LZ4 compression that was used in the custom SBMC file format. Specifically, I enabled the High Compression mode and checksumming to further reduce the file sizes and to ensure data integrity between rendering and training [33]. Both of these changes were made for convenience and should not result in any changes to the rendered images.

4.3 Dataset

The initial goal in generating a training dataset was to match the specifications of the dataset used by SBMC [19]. Having a similar dataset allows for testing the Noise2Noise technique [29] on SBMC by applying its principles without changing any other variables. SBMC used a training dataset of around 300,000 example pairs at 128×128 resolution [19]. Only one example pair was rendered for each generated scene to maximize the diversity of the resulting renderings [19]. Each low-sample count noisy image was rendered at 8 samples per pixel, while the clean reference images were rendered at 4096 samples per pixel [19]. My goal was therefore to produce a similar training set, with the main difference being that each example pair should have two noisy images rather than a noisy and a clean image. Both of these noisy images should be rendered at 8 samples per pixel to match SBMC.

Even without having to render the clean reference images, generating such a dataset still requires a significant amount of computing power. I therefore made use of the Graham cluster at the University of Waterloo to complete this task. I used the standard compute nodes with 32 cores and 128 GB of memory. Generating the training set required 75 compute tasks, each running for 12 hours on one of these nodes, and each instructed to produce 4000 example pairs. The scene generation and rendering were both run as part of these tasks, fully parallelized to take advantage of the 32 cores on each node. The object, texture, and environment map datasets were downloaded to the node's local disk at the start of each job to avoid network latency. The resulting training set contains a total of 283,831 example pairs due to the rejection sampling, and takes up 3.6 TB of disk space. This entire process was also repeated again with a different starting seed to create a second training set. This second dataset contains 284,016 example pairs, and also takes up 3.6 TB of disk space. See Figure 4.4 for some example images from these datasets.

To monitor progress during the training process, a validation set was also required. Unlike the training set, the validation set must have clean references, so that it is apparent whether the training process is converging to the correct result. SBMC uses a validation set of around 1000 example pairs generated using the same process as their training set, with the noisy images rendered at 8 samples per pixel and the reference images rendered at 4096 samples per pixel [19]. Despite only requiring around 1000 validation pairs, rendering the validation set was an even greater computational undertaking than generating the training sets because of how much longer it takes to render the clean references. For the validation set I used 32 compute tasks, each running for three days, and each instructed to produce 32 example pairs. The starting seed for the validation set was different from any of the training sets. The resulting validation set contains 937 example pairs and takes up 13 GB of disk space. See Figure 4.5 for some example images from this dataset.

4.4 Model

The model used by SBMC is a deep CNN based on a U-net architecture [19, 46]. This model is specifically designed to accept the separate sample buffers produced by the renderer, and to consider their contributions to the final output individually [19]. The samples are treated as unordered and arbitrary in number, which requires special handling in the context of a CNN [19]. The model first makes use of a technique called embedding to construct a higher-dimensional representation of each sample [19]. These *sample embeddings* then undergo a repeated process of averaging together into *context features*, going through a U-net for context propagation, and then being converted into new embeddings [19]. The purpose of this process is for the embeddings, and therefore the samples, to learn about their roles in their local neighbourhoods and their contributions toward the final image [19]. Finally, a *splatting kernel* is produced for each sample that allows for the output image to be computed [19]. See Figure 3 in the SBMC paper for an overview of this architecture [19].

Although there are a large number of different hyper-parameters that can be tuned for this model, I assumed that the SBMC authors have already optimized most of them for their application. Therefore, I have focused on tuning the parameters that the Noise2Noise authors suggested were helpful in making their technique work effectively [29]. Tuning these parameters for the SBMC setting will determine whether the suggestions in the Noise2Noise paper hold true when Noise2Noise is applied to a different denoiser. The main hyper-parameters that the Noise2Noise authors discuss are the choice of tone mapping function, tone mapping placement, and loss function [29].

Tone mapping is the process of mapping HDR image values, which can range over

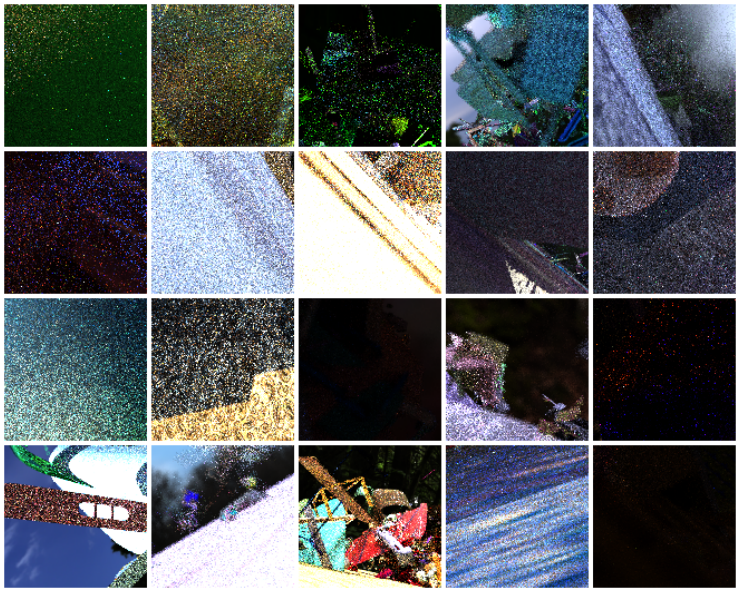


Figure 4.4: Example images from my training set at 8 samples per pixel

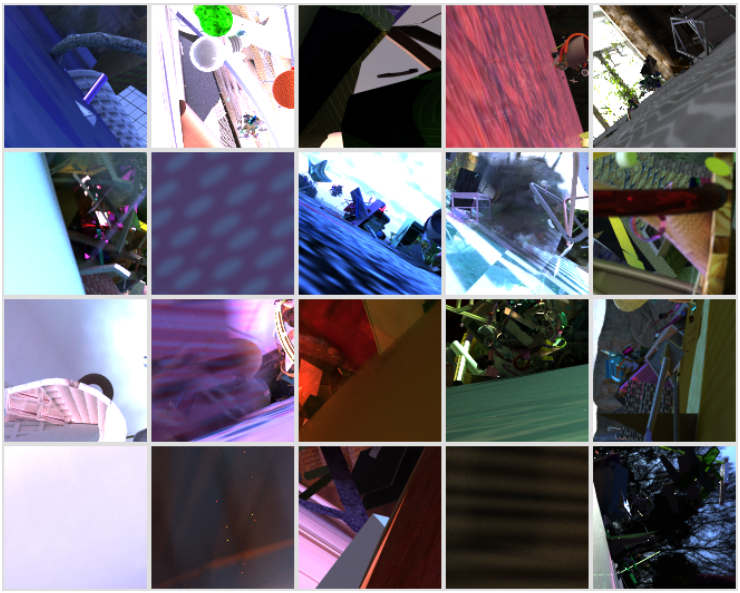


Figure 4.5: Example images from my validation set at 4096 samples per pixel

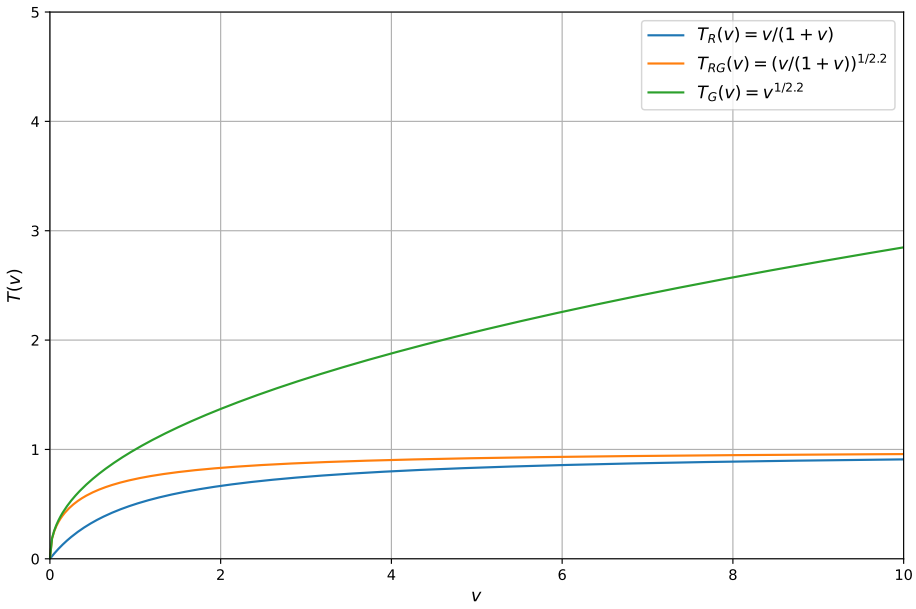


Figure 4.6: Tone mapping functions under consideration

many orders of magnitude, to a smaller range, often from 0 to 1. This process is used, for example, when displaying HDR images on a non-HDR monitor [44]. Tone mapping is also an important consideration when using HDR images with neural networks. Neural networks are generally configured to expect values in a small range, such as -1 to 1, and much larger values can cause problems known as exploding or vanishing gradients that can prevent the training process from converging [19]. Tone mapping can help with this problem, but questions remain as to which tone mapping operator to apply and where to apply it. SBMC uses Reinhard's tone mapping operator, $T_{\mathrm{R}}(v) = v/(1+v)$ [19], while Noise2Noise uses a variant of this operator that adds gamma compression, $T_{\mathrm{RG}}(v) = (v/(1+v))^{1/2.2}$ [29]. Gamma compression by itself, $T_{\mathrm{G}}(v) = Av^{\gamma}$, is also a common tone mapping operator, so I have included it in my evaluations. I use the standard values of $A = 1$ and $\gamma = \frac{1}{2.2}$ [44]. A plot of these tone mapping functions can be seen in Figure 4.6.

There are several different places where tone mapping can be applied, including the model inputs, the model outputs, and the reference images. SBMC and Noise2Noise both find that tone mapping the model inputs is beneficial [19, 29], so I follow that practice for my evaluations. Tone mapping the model inputs helps avoid the scenario where the

neural network is operating directly on HDR image values. SBMC uses a separate tone mapping function for the model inputs, $T_i(v) = \frac{1}{10}\ln(1+v)$, which I keep in place for all of my experiments. Tone mapping for the model outputs and the reference images occurs immediately before they are compared by the loss function, which can help avoid the influence of overly bright outliers in the training data [19]. SBMC applies tone mapping to both the network outputs and to the reference images [19]. Noise2Noise tries several options and recommends applying tone mapping to the network inputs only, and not to the model outputs or to the reference images [29]. In my experiments I apply tone mapping to the model inputs in all cases, and evaluate several options for tone mapping the model outputs and the reference images (see Section 5.2).

Loss functions are used to compare the network output with the reference image to produce a gradient that is used to update the network parameters through backpropagation. The choice of loss function can have a large impact on how the training process converges. The SBMC authors evaluate several different loss functions, including the standard mean squared error (MSE) loss, $L_2 = (f_\theta(\hat{x}) - \hat{y})^2$, and the relative MSE loss, $L_{\text{rMSE}} = (f_\theta(\hat{x}) - \hat{y})^2/(\hat{y} + \epsilon)^2$ [19]. The Noise2Noise authors consider the MSE loss as well as a modified version of relative MSE, $L_{\text{HDR}} = (f_\theta(\hat{x}) - \hat{y})^2/(f_\theta(\hat{x}) + \epsilon)^2$ [29]. The relative MSE and L_{HDR} losses are normalized to reduce the impact of outliers, and the Noise2Noise authors claim that their L_{HDR} loss is better suited for training with noisy references [29]. The evaluation of the different tone mapping and loss functions can be found in Section 5.2.

4.5 Training

Training was conducted in mostly the same way as SBMC, with some modifications for performance and for ease of use. As with SBMC, the number of input samples per pixel was selected randomly during the training process as a form of data augmentation [19]. The objective is to train a network that can handle different numbers of samples per pixel [19]. The chosen number of samples is between 2 and 8, which is the number of samples in each rendering in the training set. With SBMC this choice is made for each training example, while in my implementation the choice is made for each training batch. This modification allows for greater parallelism, and therefore improved training performance. SBMC has a fixed batch size of one because of this limitation, whereas with my implementation the batch size can be increased and split over multiple GPUs. Convergence should not be meaningfully affected since the network is still exposed to a variety of sample counts and the training data is shuffled before each epoch. In particular, I use a batch size of 16 split over four Nvidia Tesla T4 GPUs with 16 GB of memory. These GPUs are part of the

Graham cluster at the University of Waterloo.

The SBMC authors found that storage and I/O bandwidth was the bottleneck in training their models [19]. The data size is certainly an issue, with the large dataset and individual record sizes (around 13 MB per training example) posing difficulties. However, I found that I was able to increase the batch sizes and parallelism until I reached the limit of GPU memory without running into I/O limitations. The differences in performance could be due to the storage systems that were used. My datasets were stored on a Lustre-based distributed file system and read in by eight worker threads in parallel over an InfiniBand network connection. I also experimented with storing the training data on locally-attached solid-state drives (SSDs). This configuration resulted in similar performance, except in rare instances of increased network congestion where the SSD would out-perform the network-based storage system.

The SBMC authors trained their networks on a single Nvidia Titan X (Pascal) GPU until the loss on the validation set stopped improving, which they reported as generally taking around three to four days [19]. I found it somewhat difficult to follow this procedure, since in my experiments the loss followed an exponential curve, falling relatively quickly at first and then slowly afterwards. The exponential decay made it hard to determine exactly when the validation loss had converged, as letting the training run longer would continue to improve the loss at a progressively slower rate. Getting to the point where the curve appeared nearly linear generally took around one to two days in my experiments. Assuming that the SBMC authors used this same criteria as their stopping point, my results represent an improvement on their reported training times. This improvement is especially interesting because Noise2Noise training can be expected to take longer than training with clean references, with the Noise2Noise authors reporting that their training took about twice as long to converge [29]. This speedup can be attributed to my performance enhancements, including parallelization, since the GPUs I used are individually slower than the one used for SBMC (8.1 vs. 11 single-precision TFLOPS [4, 5]).

Chapter 5

Results

5.1 Baseline Model

The goal of this project is to apply the Noise2Noise technique of training on noisy image pairs to the SBMC denoising method [29, 19]. With this goal in mind, a natural first step is to simply attempt to train an SBMC model with the default parameters on a dataset of noisy image pairs. This configuration will be the baseline model. The SBMC model has many tunable parameters, but the relevant ones have to do with the tone mapping and loss functions. The default SBMC model uses Reinhard's tone mapping operator, $T_{\mathrm{R}}(v) = v/(1+v)$, and applies it to both the network output and to the target image before they are fed into the loss function [19]. For the loss function, the SBMC text indicates that the MSE loss is used [19], while the code uses relative MSE by default. I will be following the SBMC code in this instance and using relative MSE for the baseline model.

To monitor the training process, I plotted the loss on the current training batch over time as the training proceeded. I also plotted the loss averaged over the entire validation set, which was computed at regular intervals during the training process. These plots can be seen in Figure 5.1. Separately, I plotted the training and validation losses computed using the relative MSE loss function on the linear network outputs and target images without any tone mapping. These plots can be seen in Figure 5.2. This loss function will be computed for every training session and will be used as a standard for comparisons between different configurations of tone mapping and loss functions. As an additional way of monitoring the training process, I stored the input, network output, and target images

for sample training and validation batches at regular intervals. Examples of these images can be seen in Figures 5.3 and 5.4.

There are several interesting observations that can be made of Figures 5.1 through 5.4. The first and most important is that the network does seem to be successfully learning. This can be determined in several ways. First, the validation losses are generally decreasing over time. This means that the network is learning to denoise images that are not part of the training set. Another way to see that the network is learning is by looking at the images in Figures 5.3 and 5.4. For the training images in Figure 5.3, it appears that, despite the input and target images both being noisy, the network is outputting something that looks like a denoised version of the same scene. This is an indication that the Noise2Noise training process is successfully directing the network toward the correct output, which is the mean of the underlying noisy image distribution for each scene. For the validation images in Figure 5.4, it appears that in most cases the network, when given a noisy input image, is generating an image that looks similar to the corresponding reference image. The learning process can be further observed by looking at how the network's output on a particular validation image changes over time, as shown in Figure 5.5 for the first image in the validation set. The images in this figure show qualitatively that the network's denoising ability improves as the training progresses.

Another observation that can be made of the graphs in Figures 5.1 and 5.2 is that both the training and validation losses are decreasing quickly at the start of the training process, and then much more slowly or not at all after that. For the training loss, this would normally indicate a problem with the learning in most cases, but it is the expected behaviour in the Noise2Noise setting [29]. The model is effectively being asked to learn how to convert from one noisy realization of a scene to another one. This task is impossible in general since the two noisy images are independently drawn from the same distribution [29]. The training loss will therefore make some improvement, as the model starts by outputting effectively random images and eventually learns to output images from within the scene's distribution. The model will then continue to converge towards the mean of the distribution, but the training loss will no longer improve because the target images are drawn from the distribution at random [29]. Another property of the training loss to note is the amount of variation or noise over time. The noise appears to be relatively low, which indicates that the batch size, 16 in this case, is large enough to ensure that the gradient updates are not too noisy [26].

For the validation loss, the shape of the curve is not ideal. A quick drop followed by slow or no improvement can indicate that the optimization process is being overly aggressive and moving around chaotically without finding a local minimum [26]. The ideal optimization process would cause the loss to decrease exponentially, which will generally

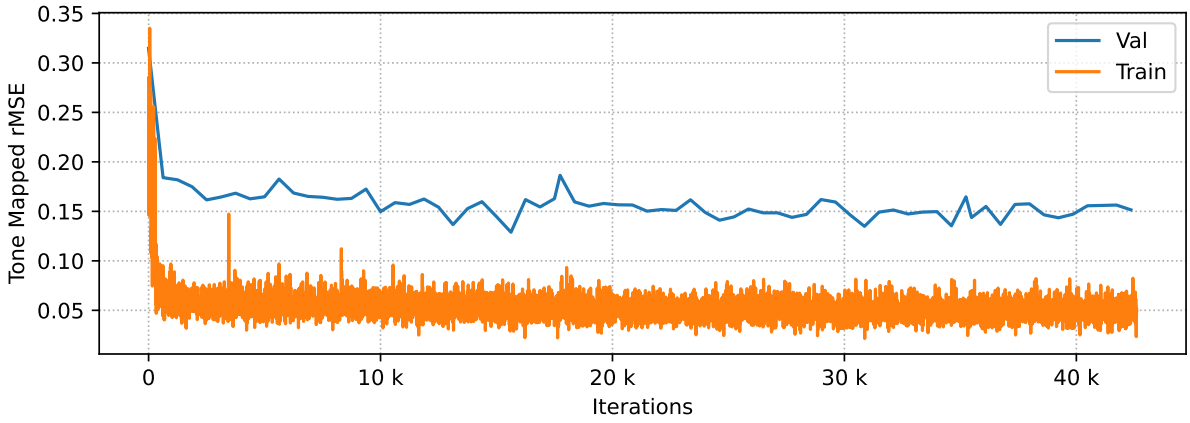


Figure 5.1: Training and validation losses for the baseline model

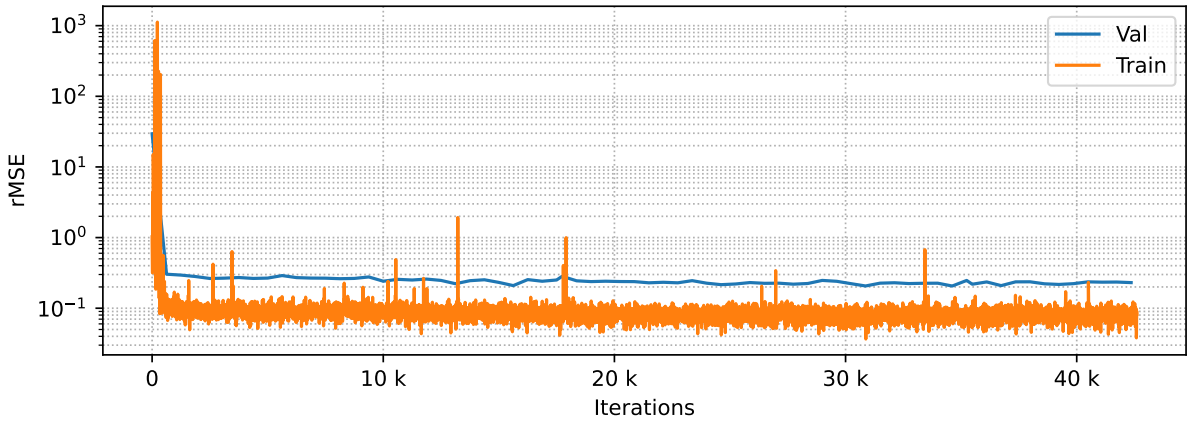


Figure 5.2: Training and validation relative MSE for the baseline model

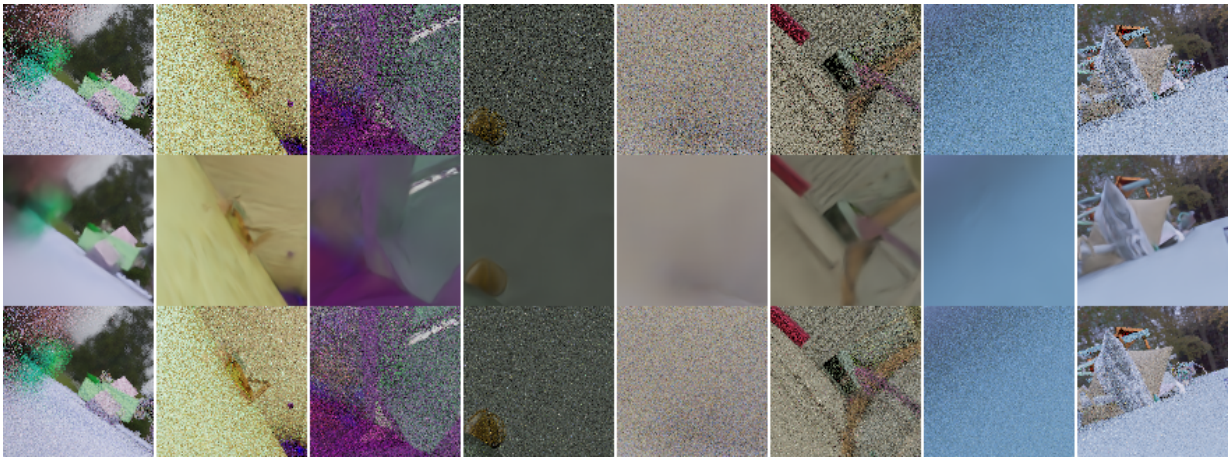


Figure 5.3: From top to bottom: input, network output, and target images from a training batch from the baseline model

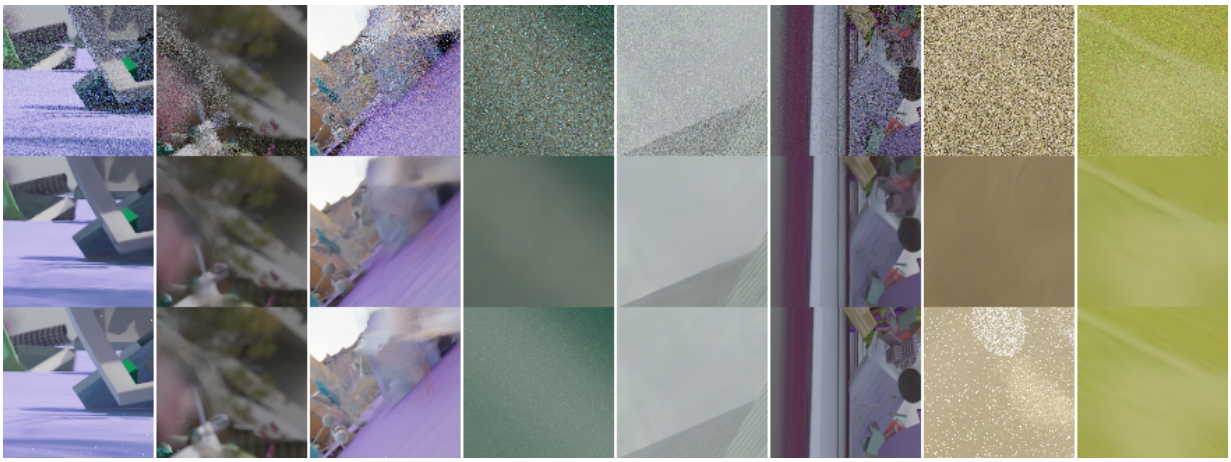


Figure 5.4: From top to bottom: input, network output, and reference images from a validation batch from the baseline model

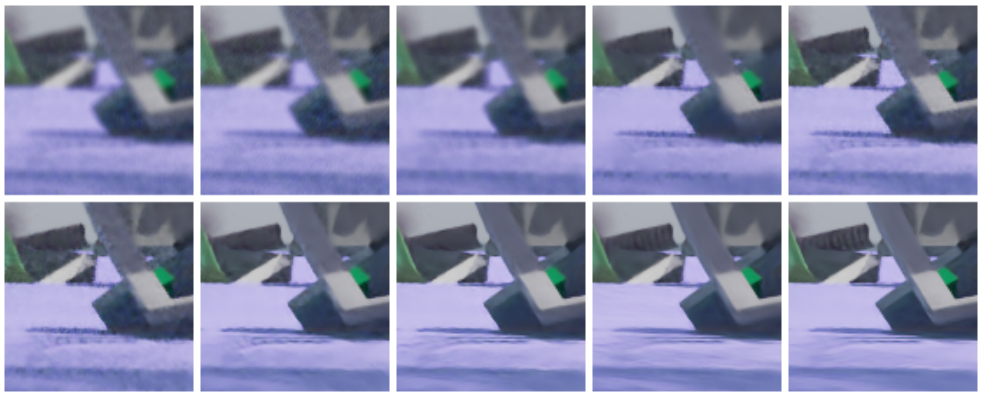


Figure 5.5: Baseline model outputs for a single validation image during the training process (time increases from left to right, first then second row)

lead to the best results [26]. This situation can often be improved by lowering the learning rate [26], however in my experiments this did not improve the relative MSE loss and led to worse qualitative image quality. Because of this lack of improvement, I left the learning rate at the default SBMC value of 10^{-4} for all subsequent trials. Another point to note about the validation loss is that it remains higher than the training loss throughout the training process in both Figures 5.1 and 5.2. This means that the average loss on the validation set is higher than the average loss on most of the training batches. For a highly effective denoiser, the validation loss should eventually drop below the training loss, since the validation targets have significantly lower noise than the training targets. This effect is observed in later experiments with better performance.

5.2 Tone Mapping and Loss Functions

The next experiment will involve trying different combinations of tone mapping and loss functions to see which configuration results in the best performance. I focus on these parameters in particular because they are areas in which the SBMC and Noise2Noise papers differ in their recommendations. The SBMC authors try several different loss functions and state that they all give qualitatively similar results [19], while the Noise2Noise authors indicate that the choice of tone mapping and loss function is essential to achieving the best performance in their scenario [29]. As mentioned in Sections 4.4 and 5.1, SBMC uses Reinhard's tone mapping operator, $T_{\mathrm{R}}(v) = v/(1+v)$, which is applied to both the network output and to the target image before they are fed into the loss function [19].

The loss functions considered by the SBMC authors include the standard MSE loss, $L_2 = (f_\theta(\hat{x}) - \hat{y})^2$, and the relative MSE loss, $L_{\text{rMSE}} = (f_\theta(\hat{x}) - \hat{y})^2/(\hat{y} + \epsilon)^2$ [19]. Noise2Noise uses a modified version of Reinhard's tone mapping operator that adds gamma compression: $T_{\text{RG}}(v) = (v/(1+v))^{1/2.2}$ [29]. The authors evaluate several configurations of how to apply this operator, including not applying it at all, and applying it to the target images but not to the network outputs [29]. They do not consider the option of applying tone mapping to both the network outputs and to the target images. The Noise2Noise authors report that the best configuration is to use their own loss function, $L_{\text{HDR}} = (f_\theta(\hat{x}) - \hat{y})^2/(f_\theta(\hat{x}) + \epsilon)^2$, without any tone mapping of the network outputs or the targets [29].

There are many possible choices of tone mapping functions, loss functions, and how to apply them. Previous works have not thoroughly explored the full range of possible combinations of these parameters to determine which achieve the best performance. A further question is whether the particular configuration recommended by the Noise2Noise authors is always the best one to use when applying their method. To answer these questions, I evaluate the full range of combinations of the above loss functions, tone mapping functions, and where to apply them, on the particular case of using Noise2Noise training with the SBMC denoising method. I consider three tone mapping operators: Reinhard's, Reinhard's with gamma compression, and plain gamma compression. I also consider three options of where to apply each tone mapping operator: on both the network output and the target image, on just the target image and not on the network output, or on neither. I do not consider the option of applying the tone mapping operator to only the network output and not to the target image, as this is nonsensical. This scenario would require the network to generate huge values that would then be tone mapped down to reach the already large HDR values in the target images. In contrast, tone mapping just the target images makes sense as this would be asking the network to generate values in the tone mapped scale. Finally, I consider three loss functions: MSE, relative MSE, and L_{HDR}.

Evaluating all of the possible combinations of three loss functions, three tone mapping functions, and three options of where to apply them, will require a grid search with a total of 27 entries. However, some of the entries are redundant: there is no need to try the different tone mapping functions when tone mapping is not used at all. This reduces the total number of required trials from 27 to 21. I trained a network using each of these configurations, with all other parameters remaining the same as the baseline model from Section 5.1. Each network was trained for 24 hours on four Nvidia Tesla T4 GPUs, and the same data was recorded as for the baseline model. The baseline model corresponds to one of these configurations, specifically the one with Reinhard's tone mapping operator applied to both the network outputs and the targets, and using the relative MSE loss function. Since there are too many training runs to plot each one individually, I have plotted only

the non-tone mapped relative MSE loss, which was calculated for each run regardless of which tone mapping and loss function was used for training. These losses all have the same scale, and can therefore be plotted on the same axes. I have shown only the validation curves and omitted the training curves, for two reasons. The first is that the training curves are too noisy to view all together, and the second is that training curves do not convey much information in the case of Noise2Noise training. The resulting plot with all of the validation curves can be seen in Figure 5.6. The output of each trained model on a single validation image can be seen in Figure 5.7. Discussion of these results can be found in Chapter 6.

5.3 Additional Training Data

This experiment will involve taking the best performing combination of parameters from Section 5.2, and training another model using those same parameters on a larger dataset. The goal of this experiment is to determine whether the performance of the model can be improved by training on more data. The results will also contribute to testing the Noise2Noise authors' hypothesis that better performance can be achieved by spending one's compute budget on generating and training on noisy targets than by generating and training on clean reference images [29]. However, even my larger dataset required a much smaller compute budget than the one generated by the SBMC authors with clean reference images. Therefore, obtaining better performance than SBMC with my larger dataset would be a strong confirmation of the hypothesis, but obtaining similar or worse performance would not disprove the idea. It could still be possible that spending the same compute budget as the SBMC authors to generate and train on noisy image pairs would result in better performance, although that is not tested here. Comparisons of the denoising performance between my models and SBMC can be found in Section 5.4.

The model from Section 5.2 that achieved the lowest non-tone mapped relative MSE on the validation set used the L_{HDR} loss with the T_{RG} tone mapping operator applied to both the network output and to the target image. I therefore trained this model again on a dataset composed of my original set of 283,831 training examples and my additional set of 284,016 examples, for a total of 567,847 example pairs. The two datasets were shuffled together so that the network was exposed to examples from both datasets at random. This model was trained for seven days on four Nvidia Tesla T4 GPUs to allow for performance improvements that may occur over longer training times. The same model was also trained with the original dataset for seven days to act as a comparison. In both cases the same data was recorded as for the baseline model. See Figure 5.8 for a plot of the training and

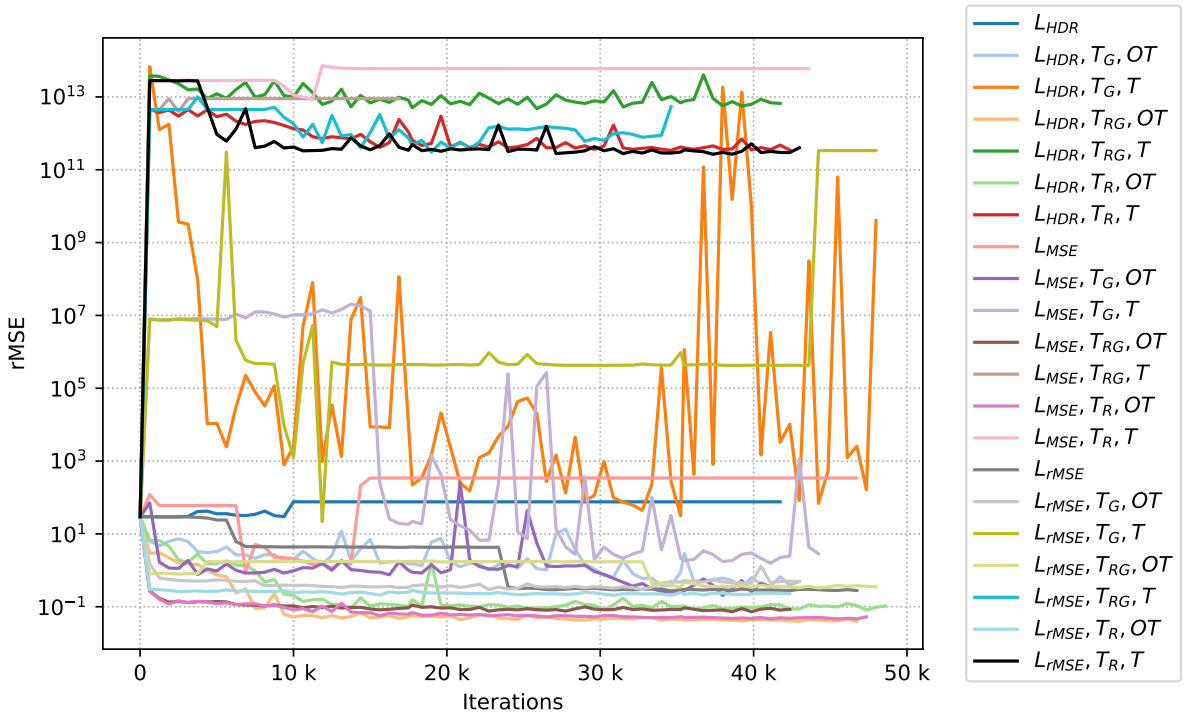
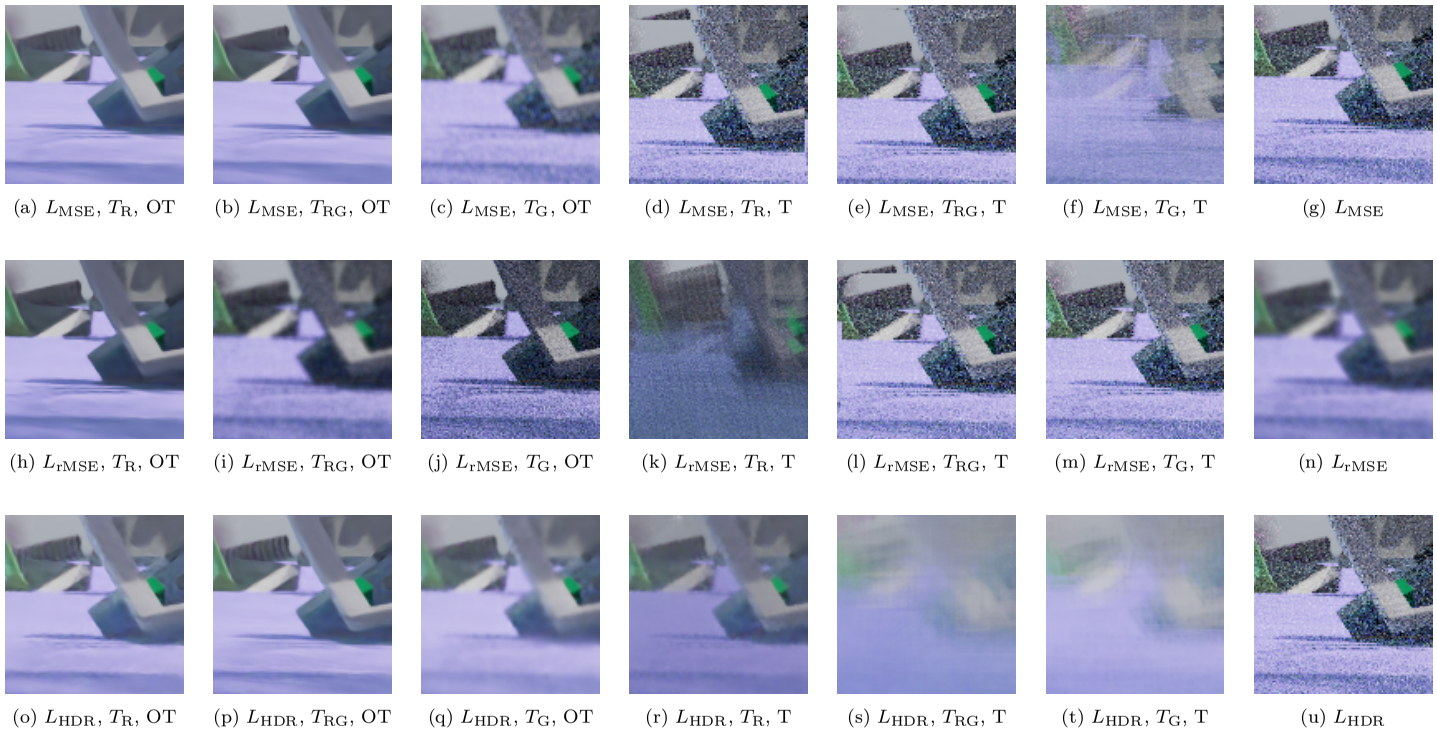


Figure 5.6: Validation losses for all of the combinations of loss functions, tone mapping functions, and tone mapping placement (OT means that tone mapping is applied to both the network output and to the target, T means that tone mapping is applied to the target only, and a loss function alone means that no tone mapping is applied)



(a) L_{MSE}, T_{R}, OT (b) L_{MSE}, T_{RG}, OT (c) L_{MSE}, T_{G}, OT (d) L_{MSE}, T_{R}, T (e) L_{MSE}, T_{RG}, T (f) L_{MSE}, T_{G}, T (g) L_{MSE}

(h) L_{rMSE}, T_{R}, OT (i) L_{rMSE}, T_{RG}, OT (j) L_{rMSE}, T_{G}, OT (k) L_{rMSE}, T_{R}, T (l) L_{rMSE}, T_{RG}, T (m) L_{rMSE}, T_{G}, T (n) L_{rMSE}

(o) L_{HDR}, T_{R}, OT (p) L_{HDR}, T_{RG}, OT (q) L_{HDR}, T_{G}, OT (r) L_{HDR}, T_{R}, T (s) L_{HDR}, T_{RG}, T (t) L_{HDR}, T_{G}, T (u) L_{HDR}

Figure 5.7: Model outputs for a single validation image for all of the combinations of loss functions, tone mapping functions, and tone mapping placement (OT means that tone mapping is applied to both the network output and to the target, T means that tone mapping is applied to the target only, and a loss function alone means that no tone mapping is applied)

validation losses for both the model trained on the original dataset and the one trained on the larger dataset. Discussion of these results can be found in Chapter 6.

5.4 Comparison to Previous Work

The next step will involve comparing the best model achieved through my experiments to previous models from the literature. The model that achieved the lowest non-tone mapped relative MSE in my tests used the L_{HDR} loss with the T_{RG} tone mapping operator applied to both the network output and to the target image. Both the model trained on the original dataset and the one trained on the larger dataset will be included in the comparisons. The models will be compared against SBMC and several other previous works that were used for comparisons in the SBMC paper [19]. Comparisons are performed using a test set provided by the SBMC authors, which consists of 55 scenes collected from publicly available sources and converted to the PBRT v2 format (see Figure 5.9) [19]. The SBMC authors provide reference images for each scene rendered at 1024×1024 resolution with 8192 samples per pixel [19]. They do not, however, provide the low-sample count input images with the additional feature data, so I rendered these using the provided scene files. One of the provided test scenes ("living-room-2") had an error during the rendering, so I was unable to use it and my test set therefore consisted of the remaining 54 scenes.

An advantage of the SBMC test set being in the standard PBRT v2 format is that it is compatible with a wide range of existing denoising implementations. This choice of scene format enabled the SBMC authors to compare against five other denoisers, with the results shown in Table 1 of their paper [19]. Another benefit of providing a publicly available test set and error numbers is that future works can make comparisons without having to recompute the error numbers for the existing denoisers. I make use of this property by computing the error of my model on the SBMC test set, and then comparing to the other denoisers using the numbers from the SBMC paper [19]. Two error metrics are used for these comparisons: relative MSE, and the structural dissimilarity index $\text{DSSIM} = 1 - \text{SSIM}$. Structural similarity index (SSIM) is an error metric that takes into account the local structure of the images being compared and attempts to produce error numbers that correspond to how humans rate the image similarity [53]. Both of these metrics are used without any tone mapping, and are computed so that lower numbers are better. The results of running my model on the test set as compared to the other denoisers are shown in Table 5.1 and Figure 5.10. Some examples of the outputs of my denoiser and those of the SBMC denoiser on various test images are shown in Figure 5.11. Discussion of these results can be found in Chapter 6.

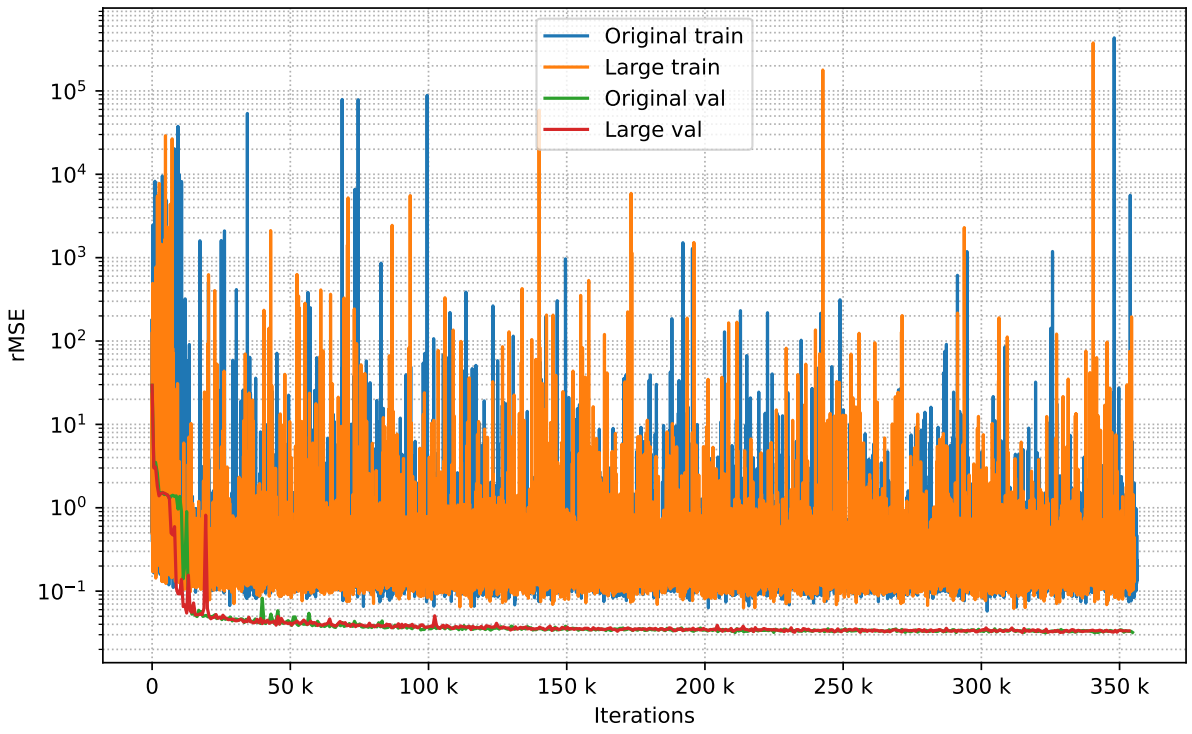


Figure 5.8: Training and validation losses for the L_{HDR}, T_{RG}, OT model trained on the original dataset and on the large dataset for seven days

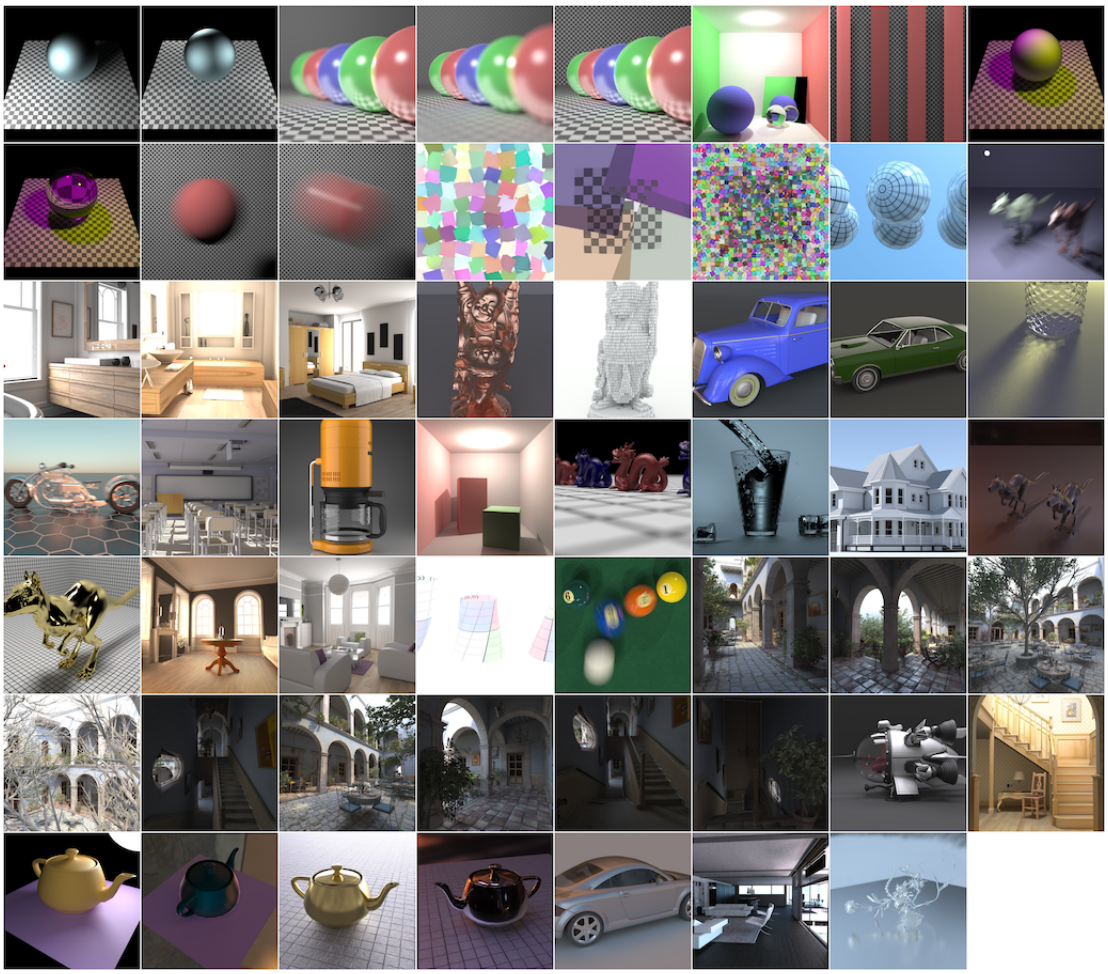


Figure 5.9: The 55 scenes comprising the SBMC test set [19]

		Input	Mine (Orig.)	Mine (Large)	SBMC [19]	Bako [7]	Bako (f.t.) [7]	Bitterli [10]	Kalantari [25]	Rousselle [47]	Sen [50]
4 spp	rMSE	17.3054	0.8775	0.6240	**0.0482**	1.0867	*0.4932*	1.0847	1.5814	2.0416	1.0352
	DSSIM	0.4648	*0.0809*	0.0814	**0.0685**	0.1294	0.1173	0.1014	0.0869	0.1575	0.1009
8 spp	rMSE	7.4732	0.3249	*0.2992*	**0.0382**	8.1238	0.8471	0.9149	1.6684	2.0721	0.5670
	DSSIM	0.3995	0.0701	*0.0695*	**0.0599**	0.1190	0.0940	0.0818	0.0708	0.1175	0.0987
16 spp	rMSE	11.0416	0.1415	*0.1069*	**0.0315**	21.3297	0.2934	0.9488	1.8151	2.0481	0.3348
	DSSIM	0.3373	0.0617	0.0601	**0.0542**	0.1128	0.0762	0.0700	*0.0600*	0.0898	0.0986
32 spp	rMSE	16.5478	0.0853	*0.0597*	**0.0274**	31.4400	0.1427	1.1344	1.7398	1.6447	0.2731
	DSSIM	0.2775	0.0567	0.0541	**0.0510**	0.1106	0.0619	0.0630	*0.0516*	0.0685	0.1005
64 spp	rMSE	12.0608	0.0609	*0.0443*	**0.0261**	0.1359	0.1553	0.8747	1.6228	1.6974	–
	DSSIM	0.2223	0.0542	0.0511	0.0494	*0.0407*	0.0536	0.0566	**0.0393**	0.0547	–
128 spp	rMSE	1.7717	0.0503	*0.0382*	**0.0254**	0.0757	0.1229	0.9039	1.7394	1.8176	–
	DSSIM	0.0488	0.0535	0.0502	0.0488	**0.0353**	0.0432	0.0565	*0.0414*	0.0466	–

Table 5.1: Relative MSE and DSSIM error metrics for each denoiser at various input samples per pixel (spp), averaged over the SBMC test set (**bold** indicates the best result, *italics* indicate the second best result)

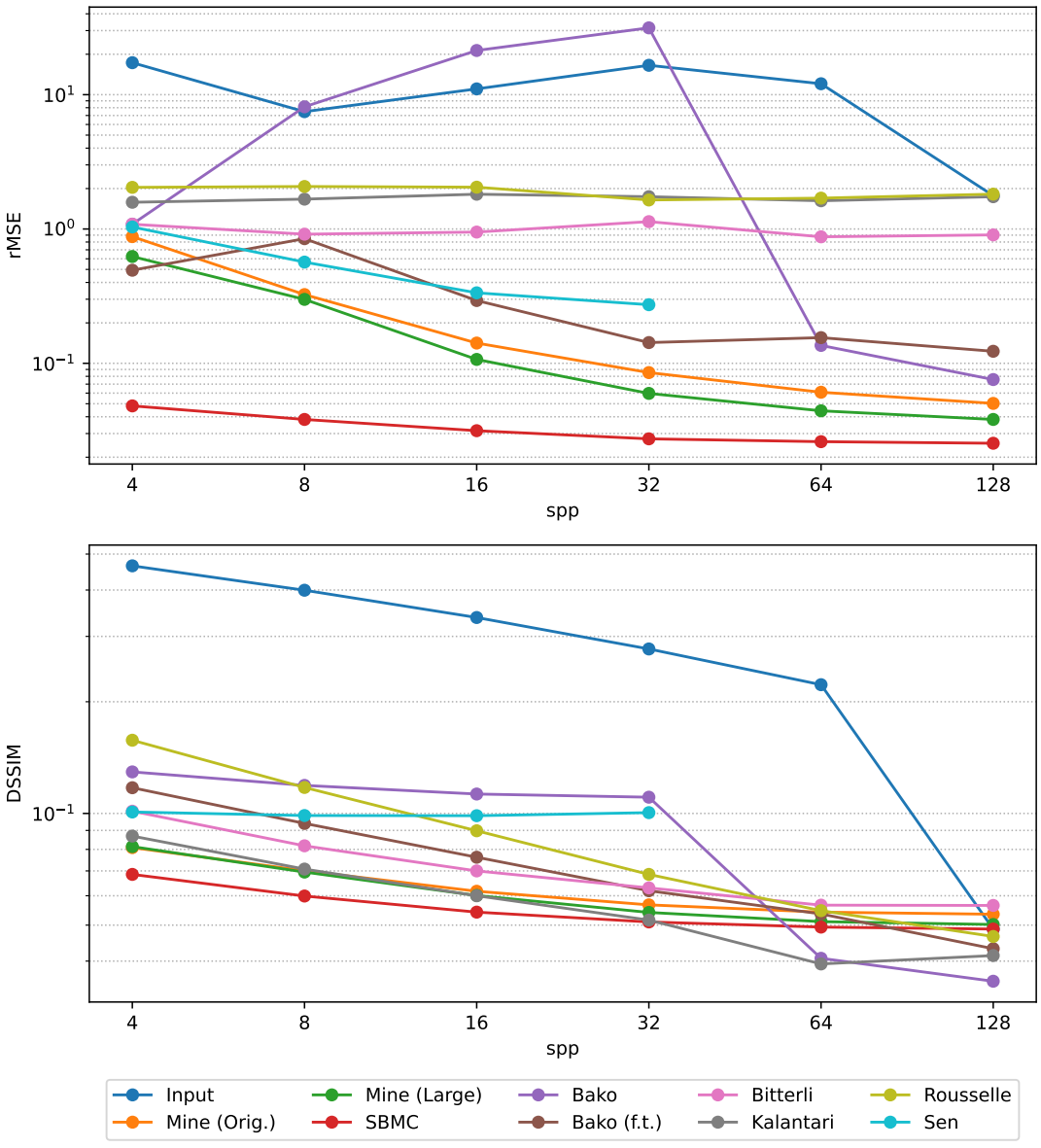


Figure 5.10: Relative MSE and DSSIM error metrics for each denoiser at various input samples per pixel (spp), averaged over the SBMC test set

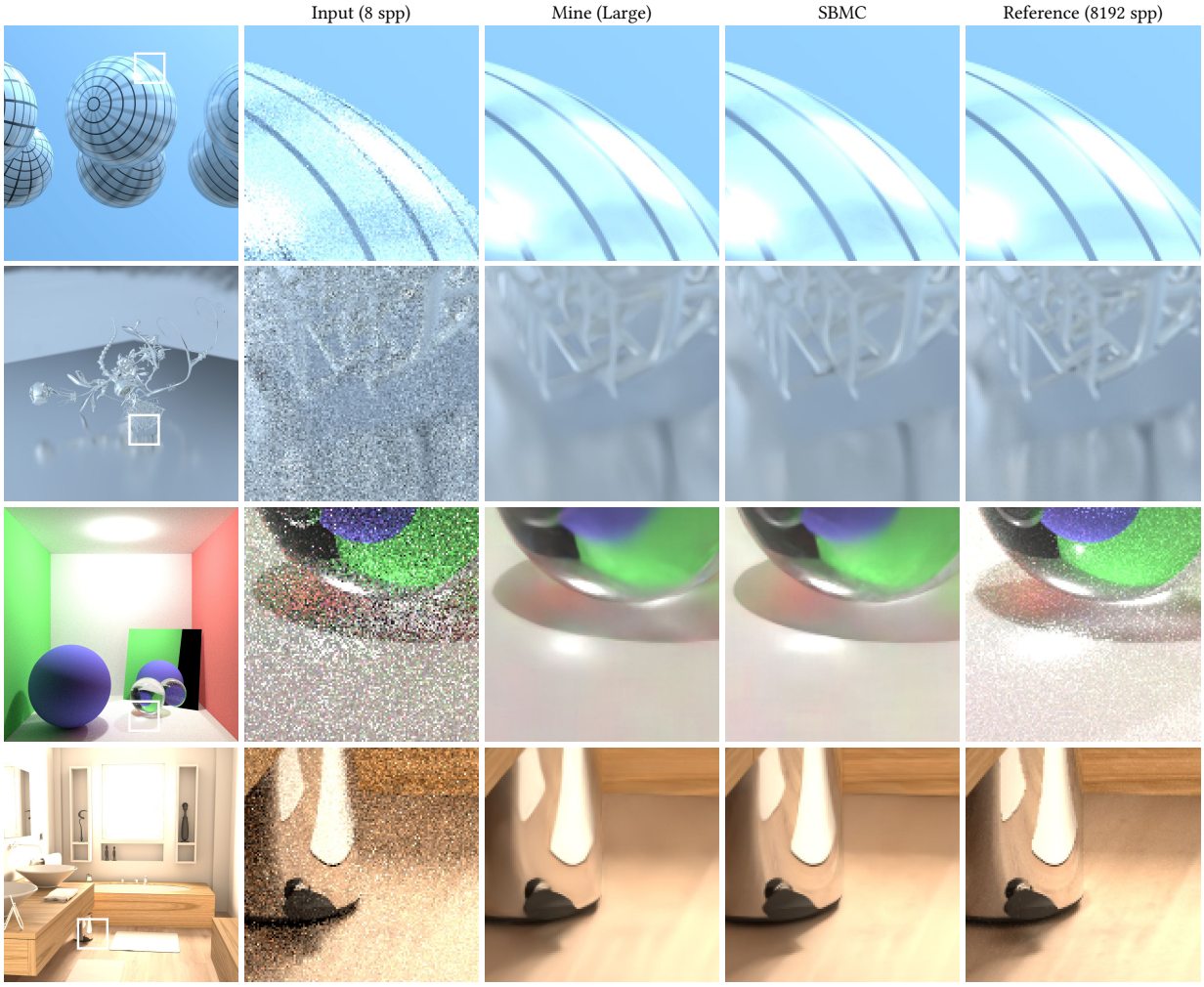


Figure 5.11: Comparison between the output of my denoiser and that of the SBMC denoiser on several test images

Chapter 6

Discussion

Section 5.2 discusses an experiment where different combinations of tone mapping functions, tone mapping placement, and loss functions are evaluated to see which is optimal. The results of this experiment are shown in Figures 5.6 and 5.7. The first thing to note about Figure 5.6 is that the validation curves cover different numbers of iterations. This is mainly caused by fluctuations in the network bandwidth that was available for transferring training data to the servers that were performing the training, causing the training to proceed at slightly different speeds during each run. Since each training run was limited to 24 hours, they were able to complete slightly differing numbers of iterations during that time. However, it appears that each of the training runs has either successfully converged, or is unlikely to converge if left to run for a larger number of iterations. Two of the training runs also stopped early before reaching the allowed time limit. These runs, "L_{MSE}, T_{RG}, T" and "L_{rMSE}, T_{RG}, T", were both stopped because the loss value became infinite during the training process.

In Figure 5.6, there are several distinct groups of loss curves. The first are the curves where the validation loss immediately rises to a high value in the range of 10^{11} to 10^{14}, and stays in that range for the rest of the training process. These losses are failing to converge, and consist entirely of runs that use tone mapping only on the target images. The two runs that were stopped early due to infinite losses are also both members of this group. The next group of loss curves are those that bounce around between vastly different loss values for the duration of the training process. This group also fails to converge, and consists of the remaining three training runs that use tone mapping only on the target images and T_{G} for the tone mapping function. With all of the training runs that use this strategy failing to converge, it seems clear that this is not an effective tone mapping configuration. The lack of effective denoising can be confirmed by looking at the validation images in Figure 5.7,

where all of the models trained with this tone mapping configuration exhibit issues from not removing the noise to over-blurring. One configuration, "L_{MSE}, T_{G}, T", does achieve a relatively low loss value of around 1.6, but its output is qualitatively poor. The final group of loss curves consists of the remaining curves that do successfully converge to a relative MSE value below around 500. The performance of these configurations can be evaluated further by examining the validation images in Figure 5.7.

Turning to Figure 5.7, the last column represents models that were trained without tone mapping on either the network output or the target images. Although these models do converge, they have widely differing performance. The "L_{HDR}" and "L_{MSE}" models converge to higher relative MSE values on the validation set of 76 and 343, respectively, and produce overly noisy validation images. The "L_{rMSE}" model converges to a lower relative MSE value around 0.3, and produces an overly blurry validation image that still contains low frequency noise. These results indicate that not using tone mapping, while an improvement over tone mapping only the target images, still does not produce images of acceptable quality with any of the tested loss functions. The final group of models, forming the first three columns in Figure 5.7, consists of models that were trained using tone mapping on both the output and the target images. These models all performed relatively well, converging to relative MSE loss values below around 1.5. The shape of the loss curves is also interesting, with the losses for some configurations dropping quickly at first and others falling more slowly. This distinction did not seem to have a large impact on the learning success, however, as the most successful models come from both categories.

Within the group of configurations where tone mapping is applied to both the output images and to the target images, one clear trend is that T_{G} is the least effective tone mapping operator. The three training runs that use this operator have among the worst loss values in this group, and the validation images are either over-blurred or still noisy, and overly dark in the case of "L_{rMSE}, T_{G}, OT". The remaining configurations using the T_{R} and T_{RG} tone mapping operators all perform well, with the exception of "L_{rMSE}, T_{RG}, OT", where the validation image is significantly more blurry and darker than the others, and a worse relative MSE loss value is achieved. The other five configurations have the best loss values of all of the trials. Their validation images are generally of passable quality, with more subtle differences present in some of the details. The validation image for the "L_{rMSE}, T_{R}, OT" configuration appears to be overly dark, which seems to complete a pattern of the L_{rMSE} configurations in this group producing overly dark outputs. The "L_{rMSE}, T_{R}, OT" model also has the worst loss value of the remaining five models under consideration. The other four models with the best loss values come in two groups with similar loss values: "L_{HDR}, T_{R}, OT" and "L_{MSE}, T_{RG}, OT", followed by "L_{MSE}, T_{R}, OT" and "L_{HDR}, T_{RG}, OT". These models all use the L_{MSE} and L_{HDR} loss functions, and the

T_{R} and T_{RG} tone mapping functions. Interestingly, however, the loss functions and tone mapping functions are both are swapped between the two groups. This means that one loss function or tone mapping function is not necessarily better than the other, but specific combinations do perform better than others. The configuration that achieved the best loss value was "L_{HDR}, T_{RG}, OT", with a relative MSE loss value on the validation set of 0.0397. This configuration also presents the qualitatively best validation image, doing an especially good job at preserving texture details without over-blurring. The second best performing configuration was "L_{MSE}, T_{R}, OT", which achieved a relative MSE loss value of 0.0465.

Comparing to SBMC and Noise2Noise reveals several differences between my findings and theirs. Starting with tone mapping placement, the Noise2Noise authors recommend not tone mapping the network output or the target image [29]. They also consider tone mapping the target images only, but find that this approach produces worse results [29]. My findings support the conclusion that tone mapping the target images only is ineffective, although my results appear qualitatively to be significantly worse than theirs. However, unlike the Noise2Noise authors, I obtained poor results when using no tone mapping, with their recommended L_{HDR} loss function producing worse results than L_{rMSE} in this case. This is surprising, since the Noise2Noise authors provide both a theoretical justification and experimental results showing that their configuration is optimal [29]. Instead, I obtained the best performance when tone mapping both the network outputs and the targets, as is done by SBMC [19]. This result contradicts with the findings of the Noise2Noise authors, who claim that non-linearly tone mapping the target images in the Noise2Noise setting will skew the distribution of the noise and lead to incorrect results [29].

For the choice of tone mapping function, Noise2Noise uses T_{RG} while SBMC uses T_{R} [29, 19]. I found that these two tone mapping functions have similar performance in some cases, but in other cases switching between the two can produce significantly different results in ways that are hard to predict. T_{G} was included for completeness and performed the worst out of all of the tone mapping functions. As for the loss function, Noise2Noise recommends L_{HDR} while SBMC uses L_{MSE} and indicates that L_{rMSE} has similar performance [29, 19]. L_{HDR} did appear in the most effective configuration, but in an unexpected setting where it was used with tone mapped network outputs and targets, instead of without tone mapping as recommended by the Noise2Noise authors [29]. The second most effective configuration used L_{MSE}, and corresponded to the exact configuration used by SBMC [19], while L_{rMSE} did not appear in the top four configurations. This is promising for the idea that Noise2Noise-style training could be used with existing denoisers without having to adapt their parameters for the Noise2Noise setting.

The next experiment, described in Section 5.3, involves training the most effective model found in the parameter search, "L_{HDR}, T_{RG}, OT", with twice as much training data

to see if that results in any improvement. The results of this experiment are shown in Figure 5.8. As can be seen in this figure, training with the larger dataset did not result in a significant improvement in relative MSE loss on the validation set compared with training on the original dataset. In fact, the model trained on the original dataset reached a slightly lower validation loss value. However, as described below, the model trained on the larger dataset did achieve lower loss values on the test set in most cases. Extending the training time from one day to seven days resulted in slightly lower validation loss values for both models. Figure 5.8 also shows that, for these more effective models, the validation loss will fall below the training loss during the training process. At this point, the average loss on the validation set becomes lower than the average loss on the training batches. This behaviour is expected because the validation targets have significantly lower noise than the training targets, and the network outputs will eventually become closer to the validation targets than they are to the training targets.

The last experiment, described in Section 5.4, involves evaluating the models from the previous experiment on the SBMC test set and comparing the results to SBMC and to the other models that were evaluated by the SBMC authors [19]. The results of this comparison are shown in Table 5.1 and Figure 5.10. SBMC achieved the best results for all of the different input sample counts with the relative MSE error metric. SBMC also achieved the best results for the lower sample counts with the DSSIM error metric, with some of the other denoising methods taking the lead at higher sample counts. Both of my models produced results that are close to the SBMC results for all sample counts in terms of DSSIM. The results are close for relative MSE as well at higher sample counts, and about an order of magnitude larger at lower sample counts. My models had similar results to each other in all cases, with the model trained on the larger dataset performing slightly better in all cases but one. Compared to the other denoising methods, my models came in second after SBMC in the majority of cases where SBMC had the best performance. This performance was achieved despite my model being trained only on outdoor scenes (see Section 4.1) while being evaluated on a test set that included many indoor scenes. If my model were trained on an equivalent dataset to SBMC, only with noisy targets instead of clean ones, I would expect it to achieve closer results.

Figure 5.11 shows some examples of the outputs of my model and the SBMC model on several images from the SBMC test set. The first scene involves rotating spheres with motion blur hovering over a mirror. The zoomed-in portion shows one of the spheres with specular highlights and blurred textures, where the output from my model is similar to that of SBMC and to the reference image. The second scene contains a complicated metallic structure that is suspended over a reflective surface. The zoomed-in portion shows that my model again produces output that is similar to that of SBMC, capturing all of the

intricate details of the structure and only missing some small areas of shadow. Both my model and SBMC over-blur some of the details of the reflection of the structure. The third scene contains a Cornell box with matte, reflective, and refractive objects. The zoomed-in portion shows that the reference image is still quite noisy. Both models produce outputs that are visually better than the reference image, with my model producing a slightly smaller caustic than SBMC. Both my model and SBMC also over-blur the reflection of the purple sphere and miss a small highlight on the glass sphere. The final scene is a complete model of a bathroom lit by a diffuse light source on the far wall. The zoomed-in portion shows that my model produces a similar denoised result to SBMC, although my model misses some detail in the wood texture in the background and in the reflection of the floor in the trash can. Both my model and SBMC miss some details of the textured floor in the portion of the image where it is in shadow.

Some of the images in the SBMC test set still have visible noise, which illustrates the difficulty of producing clean reference images. This issue raises some concerns about how evaluation is performed for Monte Carlo denoising. As with most computer graphics applications, the goal is generally to produce images that look believable to a human observer. If the images in a test (or validation) set contain visible noise, then denoising models will be evaluated on how well they can reproduce this noise, which does not look believable to a human. The goal should then be to render reference images that do not contain visible noise. However, this objective is hard to define because noise can be more or less apparent for different people and under different viewing conditions [53]. It is therefore difficult to determine what constitutes an acceptable level of noise for reference images. In practice, reference images produced by Monte Carlo rendering will always have some amount of noise, and error metrics may be sensitive to this noise in ways that differ from human perception. Overall, the evaluation of Monte Carlo denoisers is a complicated problem without an easy solution. The best available approach is to use an error metric such as SSIM that attempts to mimic human perception [53], and to spend as much computer time as is available to make the references as clean as possible.

There are several other issues involving the test set that merit discussion. The first is that, in this scenario, the validation and test sets come from different distributions. The validation set is produced in the same way as the training data, with randomly generated scenes. The test set, on the other hand, is curated from manually composed scenes that are designed to test specific lighting effects. This experiment design was inherited from SBMC [19], and is common among machine learning-based Monte Carlo denoisers. The design is used in these settings because random scenes can be easily generated for training and validation, whereas manually composed scenes are harder to obtain but represent the desired target distribution, and so are used for the test set. In supervised machine learning

more generally, however, the standard practice is for the training, validation, and test sets to be drawn from the same distribution. With different distributions, the trained model will have to learn to generalize not just from the training set to the validation set, but also from the validation set to the test set. The later generalization cannot be measured during the training process without an additional component. It would be helpful to have a second validation set with images from the same distribution as the test set, to monitor this generalization while tuning hyperparameter values. Unfortunately, this second validation set was not provided by the SBMC authors. This leads to another concern, namely overfitting the test set. It is important to use only the validation set for hyperparameter tuning, and to evaluate the resulting model on the test set only once at the end of the process, as was done here. This is hard to enforce, however, with a public test set, and the field as a whole can eventually produce solutions that are specific to that particular test set and do not generalize well. A potential solution to this issue is to run regular competitions where submitted models are evaluated on a hidden test set that is only revealed once the competition has ended.

Chapter 7

Conclusion

The first goal of this thesis was to answer the question of whether Noise2Noise training can be applied to a state-of-the-art denoising technique to achieve similar results on a lower budget. To answer this question, I modified the SBMC scene generator to generate a dataset of noisy image pairs. I then used this dataset to train a number of SBMC models to determine which parameters yield the best results. I have concluded that it is possible to achieve similar results to the original SBMC paper by using Noise2Noise training with an SBMC model. The results using the best combination of parameters are qualitatively and quantitatively similar to SBMC, often taking second place after SBMC and ahead of the other denoising techniques. My training dataset of noisy image pairs can be generated for a computing cost of around $900, rather than the millions of dollars required to generate a traditional dataset with clean reference images.

The second goal of this thesis was to establish whether using Noise2Noise training could allow for even better performance to be achieved by generating and training on more example pairs. To answer this question, I used the SBMC scene generator to generate a second dataset of around the same size as the initial one. Training an SBMC model with the best parameters on the two datasets combined resulted in an improvement in performance over training with the initial dataset. However, this improved performance, while close in many cases, still did not match that of SBMC in the quantitative results. This difference could be due to other factors besides the use of Noise2Noise training, such as the lack of indoor scenes in my training dataset. It is also possible that the performance could be further enhanced by using even more training data. These results support the hypothesis that better performance can be achieved with additional training data, but do not answer the question of whether this improved performance could exceed that of training with clean references. Further work is needed to answer this question, perhaps by

using even more training data and ensuring that its composition is identical apart from using noisy instead of clean reference images. Future works could also examine whether smaller datasets could be sufficient to obtain similar results, thereby saving even more computation and storage costs.

The third and final goal of this thesis was to determine whether the tone mapping and loss parameters recommended by the Noise2Noise authors apply more generally to the use of Noise2Noise training with other denoisers. To answer this question, I trained different SBMC models on my training set using every combination of tone mapping function, tone mapping placement, and loss function from the SBMC and Noise2Noise papers. Surprisingly, I found that the parameters recommended by Noise2Noise were not effective in this situation, whereas the default SBMC parameters were quite effective. This suggests that the optimal combination of parameters has more to do with the problem domain than with the training method. I also discovered a combination of parameters that was slightly more effective in this scenario than the default SBMC configuration, namely tone mapping the network outputs and the target images using the Reinhard with Gamma tone mapping function along with the L_{HDR} loss function. In general, since the optimal parameters appear to be problem-specific, conducting a search for the best parameters would be advisable for new denoising models. Future works in this area could investigate how widely these recommendations are applicable, and could evaluate additional tone mapping and loss functions.

The ultimate goal of this project was to reduce the difficulty and cost of training state-of-the-art Monte Carlo denoising models. The main cost in training these models comes from generating datasets containing tens to hundreds of thousands of clean reference images that must be rendered using huge amounts of computing power. I have shown that generating these clean reference images is not strictly necessary, and that even state-of-the-art denoising models can be trained using the Noise2Noise technique with noisy image pairs. These datasets are still too large to be shared easily, but researchers can generate their own without incurring much expense. This technique reduces the cost of generating training datasets by thousands of times, putting them within reach of smaller research groups and individual hobbyists. It is my hope that this work will make it easier to reproduce results in Monte Carlo denoising, and will enable more researchers to contribute to this field.

References

[1] Amazon S3 Simple Storage Service Pricing - Amazon Web Services. URL: https://aws.amazon.com/s3/pricing/.

[2] EC2 On-Demand Instance Pricing - Amazon Web Services. URL: https://aws.amazon.com/ec2/pricing/on-demand/.

[3] Intel® Open Image Denoise. URL: https://www.openimagedenoise.org/.

[4] NVIDIA T4 Tensor Core GPUs for Accelerating Inference. URL: https://www.nvidia.com/en-us/data-center/tesla-t4/.

[5] TITAN X Graphics Card for VR Gaming from NVIDIA GeForce. URL: https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/.

[6] Jonghee Back, Binh-Son Hua, Toshiya Hachisuka, and Bochang Moon. Deep combiner for independent and correlated pixel estimates. *ACM Transactions on Graphics*, 39(6):242:1–242:12, November 2020. doi:10.1145/3414685.3417847.

[7] Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony DeRose, and Fabrice Rousselle. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)*, 36(4), July 2017. doi:10.1145/3072959.3073708.

[8] Joshua D. Batson and Loic A. Royer. Noise2Self: Blind Denoising by Self-Supervision. *ICML*, 2019. URL: http://proceedings.mlr.press/v97/batson19a.html.

[9] Pablo Bauszat, Martin Eisemann, and Marcus Magnor. Guided image filtering for interactive high-quality global illumination. In *Proceedings of the Twenty-second Eurographics conference on Rendering*, EGSR '11, pages 1361–1368. Eurographics Association, June 2011. doi:10.1111/j.1467-8659.2011.01996.x.

[10] Benedikt Bitterli, Fabrice Rousselle, Bochang Moon, José A. Iglesias-Guitián, David Adler, Kenny Mitchell, Wojciech Jarosz, and Jan Novák. Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings. *Computer Graphics Forum*, 35(4):107–117, July 2016. doi:10.1111/cgf.12954.

[11] Ashish Bora, Eric Price, and Alexandros G. Dimakis. AmbientGAN: Generative models from lossy measurements. In *ICLR*, 2018. URL: https://openreview.net/forum?id=Hy7fDog0b.

[12] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 60–65. IEEE, 2005. doi:10.1109/CVPR.2005.38.

[13] Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36(4):98, 2017. doi:10.1145/3072959.3073601.

[14] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. *arXiv:1512.03012 [cs.GR]*, December 2015. URL: http://arxiv.org/abs/1512.03012.

[15] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing Textures in the Wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3606–3613, June 2014. URL: https://openaccess.thecvf.com/content_cvpr_2014/html/Cimpoi_Describing_Textures_in_2014_CVPR_paper.html, doi:10.1109/CVPR.2014.461.

[16] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, August 2007. doi:10.1109/TIP.2007.901238.

[17] Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik P. A. Lensch. Edge-avoiding À-Trous wavelet transform for fast global illumination filtering. In *Proceedings of the Conference on High Performance Graphics*, HPG '10, pages 67–75. Eurographics Association, June 2010. doi:10.2312/EGGH/HPG10/067-075.

[18] Elmar Eisemann and Frédo Durand. Flash photography enhancement via intrinsic relighting. *ACM Transactions on Graphics*, 23(3):673–678, August 2004. doi:10.1145/1015706.1015778.

[19] Michaël Gharbi, Tzu-Mao Li, Miika Aittala, Jaakko Lehtinen, and Frédo Durand. Sample-based Monte Carlo Denoising Using a Kernel-splatting Network. *ACM Trans. Graph.*, 38(4):125:1–125:12, July 2019. doi:10.1145/3306346.3322954.

[20] Jie Guo, Mengtian Li, Quewei Li, Yuting Qiang, Bingyang Hu, Yanwen Guo, and Ling-Qi Yan. GradNet: unsupervised deep screened poisson reconstruction for gradient-domain rendering. *ACM Transactions on Graphics*, 38(6):223:1–223:13, November 2019. doi:10.1145/3355089.3356538.

[21] Jon Hasselgren, Jacob Munkberg, Marco Salvi, Anjul Patney, and Aaron Lefohn. Neural temporal adaptive sampling and denoising. In *Computer Graphics Forum*, volume 39, pages 147–155, July 2020. doi:10.1111/cgf.13919.

[22] Yuchi Huo and Sung-eui Yoon. A survey on deep learning-based Monte Carlo denoising. *Computational Visual Media*, 7(2):169–185, June 2021. doi:10.1007/s41095-021-0209-9.

[23] Mustafa Işık, Krishna Mullia, Matthew Fisher, Jonathan Eisenmann, and Michaël Gharbi. Interactive Monte Carlo denoising using affinity of neural features. *ACM Transactions on Graphics*, 40(4):37:1–37:13, July 2021. doi:10.1145/3450626.3459793.

[24] James T. Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 143–150. Association for Computing Machinery, August 1986. doi:10.1145/15922.15902.

[25] Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. A Machine Learning Approach for Filtering Monte Carlo Noise. *ACM Trans. Graph.*, 34(4):122:1–122:12, July 2015. doi:10.1145/2766977.

[26] Andrej Karpathy. CS231n Convolutional Neural Networks for Visual Recognition. URL: https://cs231n.github.io/.

[27] Alexander Krull, Tim-Oliver Buchholz, and Florian Jug. Noise2Void - Learning Denoising From Single Noisy Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2129–2137, June 2019. URL: https://openaccess.thecvf.com/content_CVPR_2019/html/Krull_

Noise2Void_-_Learning_Denoising_From_Single_Noisy_Images_CVPR_2019_paper.html, doi:10.1109/CVPR.2019.00223.

[28] Samuli Laine, Tero Karras, Jaakko Lehtinen, and Timo Aila. High-Quality Self-Supervised Deep Image Denoising. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, volume 32, pages 6970–6980, December 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/2119b8d43eafcf353e07d7cb5554170b-Abstract.html.

[29] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2Noise: Learning Image Restoration without Clean Data. In *International Conference on Machine Learning*, pages 2965–2974. PMLR, July 2018. URL: http://proceedings.mlr.press/v80/lehtinen18a.html.

[30] Weiheng Lin, Beibei Wang, Lu Wang, and Nicolas Holzschuch. A Detail Preserving Neural Network Model for Monte Carlo Denoising. *Computational Visual Media*, 6:157–168, April 2020. doi:10.1007/s41095-020-0167-7.

[31] Weiheng Lin, Beibei Wang, Jian Yang, Lu Wang, and Ling-Qi Yan. Path-based Monte Carlo Denoising Using a Three-Scale Neural Network. *Computer Graphics Forum*, 40(1):369–381, February 2021. doi:10.1111/cgf.14194.

[32] YiFan Lu, Ning Xie, and Heng Tao Shen. DMCR-GAN: Adversarial Denoising for Monte Carlo Renderings with Residual Attention Networks and Hierarchical Features Modulation of Auxiliary Buffers. In *SIGGRAPH Asia 2020 Technical Communications*, pages 1–4. Association for Computing Machinery, December 2020. doi:10.1145/3410700.3425426.

[33] Takayuki Matsuoka. LZ4 - Extremely fast compression. URL: https://lz4.github.io/lz4/.

[34] Michael D. McCool. Anisotropic diffusion for Monte Carlo noise reduction. *ACM Transactions on Graphics*, 18(2):171–194, April 1999. doi:10.1145/318009.318015.

[35] Xiaoxu Meng, Quan Zheng, Amitabh Varshney, Gurprit Singh, and Matthias Zwicker. Real-time Monte Carlo Denoising with the Neural Bilateral Grid. In *Eurographics Symposium on Rendering (EGSR)*. The Eurographics Association, 2020. doi:10.2312/sr.20201133.

[36] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949. doi:10.1080/01621459.1949.10483310.

[37] Bochang Moon, Nathan Carr, and Sung-Eui Yoon. Adaptive Rendering Based on Weighted Local Regression. *ACM Transactions on Graphics*, 33(5):170:1–170:14, September 2014. doi:10.1145/2641762.

[38] Bochang Moon, Jong Yun Jun, JongHyeob Lee, Kunho Kim, Toshiya Hachisuka, and Sung-Eui Yoon. Robust Image Denoising Using a Virtual Flash Image for Monte Carlo Ray Tracing. *Computer Graphics Forum*, 32(1):139–151, 2013. doi:10.1111/cgf.12004.

[39] Bochang Moon, Steven McDonagh, Kenny Mitchell, and Markus Gross. Adaptive polynomial rendering. *ACM Transactions on Graphics*, 35(4):40:1–40:10, July 2016. doi:10.1145/2897824.2925936.

[40] Jacob Munkberg and Jon Hasselgren. Neural Denoising with Layer Embeddings. In *Computer Graphics Forum*, volume 39, pages 1–12, July 2020. doi:10.1111/cgf.14049.

[41] Ryan S. Overbeck, Craig Donner, and Ravi Ramamoorthi. Adaptive wavelet rendering. *ACM Transactions on Graphics*, 28(5):1–12, December 2009. doi:10.1145/1618452.1618486.

[42] Georg Petschnigg, Richard Szeliski, Maneesh Agrawala, Michael Cohen, Hugues Hoppe, and Kentaro Toyama. Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics*, 23(3):664–672, August 2004. doi:10.1145/1015706.1015777.

[43] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016. URL: https://pbrt.org/.

[44] Erik Reinhard, Wolfgang Heidrich, Paul Debevec, Sumanta Pattanaik, Greg Ward, and Karol Myszkowski. *High dynamic range imaging: acquisition, display, and image-based lighting*. Morgan Kaufmann, 2010.

[45] Dan Robitzski. A startup is suing Facebook, Princeton for stealing its AI data, June 2019. URL: https://futurism.com/tech-suing-facebook-princeton-data.

[46] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, November 2015. doi:10.1007/978-3-319-24574-4_28.

[47] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. Adaptive rendering with non-local means filtering. *ACM Transactions on Graphics*, 31(6):195:1–195:11, November 2012. doi:10.1145/2366145.2366214.

[48] Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. Robust Denoising using Feature and Color Information. *Computer Graphics Forum*, 32(7):121–130, 2013. doi:10.1111/cgf.12219.

[49] Holly E. Rushmeier and Gregory J. Ward. Energy preserving non-linear filters. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 131–138. Association for Computing Machinery, July 1994. doi:10.1145/192161.192189.

[50] Pradeep Sen and Soheil Darabi. On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Transactions on Graphics*, 31(3):1–15, May 2012. doi:10.1145/2167076.2167083.

[51] Shuran Song, Fisher Yu, Andy Zeng, Angel X. Chang, Manolis Savva, and Thomas Funkhouser. Semantic Scene Completion from a Single Depth Image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1746–1754, July 2017. URL: https://openaccess.thecvf.com/content_cvpr_2017/html/Song_Semantic_Scene_Completion_CVPR_2017_paper.html, doi:10.1109/CVPR.2017.28.

[52] Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. Denoising with Kernel Prediction and Asymmetric Loss Functions. *ACM Transactions on Graphics*, 37(4):124:1–124:15, July 2018. doi:10.1145/3197517.3201388.

[53] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004. doi:10.1109/TIP.2003.819861.

[54] Kin-Ming Wong and Tien-Tsin Wong. Deep residual learning for denoising Monte Carlo renderings. *Computational Visual Media*, 5(3):239–255, September 2019. doi:10.1007/s41095-019-0142-3.

[55] Bing Xu, Junfei Zhang, Rui Wang, Kun Xu, Yong-Liang Yang, Chuan Li, and Rui Tang. Adversarial Monte Carlo denoising with conditioned auxiliary feature modulation. *ACM Transactions on Graphics*, 38(6):224:1–224:12, November 2019. doi:10.1145/3355089.3356547.

[56] Ruifeng Xu and Sumanta N. Pattanaik. A novel Monte Carlo noise reduction operator. *IEEE Computer Graphics and Applications*, 25(2):31–35, March 2005. doi:10.1109/MCG.2005.31.

[57] Xin Yang, Dawei Wang, Wenbo Hu, Li-Jing Zhao, Bao-Cai Yin, Qiang Zhang, Xiao-Peng Wei, and Hongbo Fu. DEMC: A Deep Dual-Encoder Network for Denoising Monte Carlo Rendering. *Journal of Computer Science and Technology*, 34:1123–1135, September 2019. doi:10.1007/s11390-019-1964-2.

[58] Xin Yang, Dawei Wang, Wenbo Hu, Lijing Zhao, Xinglin Piao, Dongsheng Zhou, Qiang Zhang, Baocai Yin, Qiang Cai, and Xiaopeng Wei. Fast Reconstruction for Monte Carlo Rendering Using Deep Convolutional Networks. *IEEE Access*, 7:21177–21187, December 2018. doi:10.1109/ACCESS.2018.2886005.

[59] Jiaqi Yu, Yongwei Nie, Chengjiang Long, Wenju Xu, Qing Zhang, and Guiqing Li. Monte Carlo denoising via auxiliary feature guided self-attention. *ACM Transactions on Graphics (TOG)*, 40(6):1–13, December 2021. doi:10.1145/3478513.3480565.

[60] Greg Zaal. HDRIs • Poly Haven. URL: https://polyhaven.com/hdris/.

[61] Shaokun Zheng, Fengshi Zheng, Kun Xu, and Ling-Qi Yan. Ensemble denoising for Monte Carlo renderings. *ACM Transactions on Graphics (TOG)*, 40(6):1–17, December 2021. doi:10.1145/3478513.3480510.

[62] Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and Sung-Eui Yoon. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)*, 34(2):667–681, May 2015. doi:10.1111/cgf.12592.