Deep Learning Based Denoiser for Images Rendered by Monte Carlo Sampling

Hossein Souri
Center for Automation Research, UMIACS
University of Maryland, College Park
Email: hsouri@umiacs.umd.edu

Mohsen Zakeri
Center for Automation Research, UMIACS
University of Maryland, College Park
Email: mzakeri@cs.umd.edu

I. Abstract

***Abstract*—In this work, we design a denoiser for images rendered by Monte Carlo sampling, based on the approach introduced in "Deep Image Prior"[1]. They have demonstrated that the structure of generator network is enough for learning a lot of parameters for any images without any prior learning step. We build on top of their design a new single image denoiser which works best for denoising the images rendered by Monte Carlo sampling. We try to benefit from additional features provided by Monte Carlo sampling such as normals, depth and etc. to improve the performance of denoiser. Our result show that using the features provided by the MC renderer in addition to a random matrix as the input of the neural networks often improve the quality of the denoised images.**

The Monte Carlo denoiser code and results of this work can be found at https://github.com/hsouri/deep-image-prior.

II. Introduction

Deep learning has experienced explosive growth in the recent years, achieving sufficient maturity to provide effective tools for computer science and engineering fields. In computer graphics, many traditional problems are now better handled by deep-learning based data-driven methods. There are a lot of applications for the deep learning in this areas such as denoising renderings, image decomposition, directional decomposition, mesh labeling, and 3D reconstruction. On the other hand, Monte Carlo denoising and has been the focus of extensive research [2]. Deep leanig can be used in denoising the rendered image from the general distributed Monte Carlo effects such as depth of field, motion blur, glossy reflections, and global illumination). Most deep learning works are using extensive labled input data to train the network. often we don't have access to a large rendered images and the ground truth data. Therefore, using a single image deep learning method seems to be very precious for the field. Recently, a single image denoising method has been introduced in [1]. They show show that a randomly initialized neural network can be used as a handcrafted prior with excellent results in standard inverse problems such as denoising. This method can be modified by using the features generated by the Monte Carlos denoising. In this work we show that using this idea we can denoise the Monte Carlso rendered images using an single image deep learning frame.

III. Method

The main goal is benefiting from the additional features provided by a Monte Carlo (MC) sampler for designing the images rendered by MC integrator. According to [1] a convolutional neural network can be trained with only a single image starting from a random input. They have demonstrated that the network structure is enough for capturing the main features of an image which can be used for a lot of applications such as denoising.

We propose to use the generated Monte Carlo features as the input of the netweok to improve the performance of the denoiser, specially for denoising the images rendered by MC sampling. To do so, we try to train the neural network with different kinds of input and show the performance of the denoising task on different scenes.

We have two different designs for using the additional features in the training process. First one is using the additional features for directing the network in the loss evaluation step. In this approach, we increase the depth of output by the number of additional feature channels, while still having the first three channels account for the color features of the denoised image. The network is estimating not only the noisy image main feauters, but also it is estimating the additional features like normal channels. Therefore, the loss function is calculated for all these channels rather than just for the main color channels. So, in this approach the input to the channel is still a random input, as is in the main paper, but the output is modified to estimate the additional features as well, and the hope is this will direct the training process to learn more features of the image. We call this approach IOC (increase output channels).

The second approach is to use the additional features as inputs of the network. This way, instead of providing a random three channel matrix as the input of the network, we add the features from the MC renderer as well. So, the new input will consists of the random vectors and features from the MC renderer. Since the depth of the channel inputs are increased considerably in this case, we also increase the size of inner channels of the network as well to account for the higher complexity of the input channels and better training. We call this approach IIC (increase input channels)

For our training purpose we use two different networks, Skip and Resnet. Skip is the network used in the denoising

(a) Noisy input image **(b)** 1000 iterations **(c)** 10000 iterations

Fig. 1: The effect of number of iterations on the denoising

experiment (and also many other experiments) in the Deep Prior Image paper which is basically an encoder-decoder network. For the Skip network, we try the default size for the number of intermediate channels in encoder and the decoder which is [8, 16, 32, 64, 128], and also we double the size of each channel by 2 for the input with higher dimensions. In all the experiements, for the Resnet network, we use 10 blocks each of 32 number of intermediate channels.

We run all the experiments for 3000 iterations since running more iterations will lead to overfitting and getting noisy outputs. The reason that the network ties to minimize the loss function which is a metric for computing the difference between the noisy input and the output. When we have a very low loss value, we are generating the output very similar to the noisy input. Therefore, we should stop training before getting to close to the noisy input. figure 1 shows the effect of iterations on the denoising task.

We also set the learning rate for the networks to 0.001. However, to avoid large changes when the loss becomes relatively small, we decrease the learning rate by a factor of 0.1 when loss falls bellow 0.0005. This way, when the network finds an answer which is able to produce a high quality image (having a small loss function), we don't let the network to take large steps and gets far from the optimum point. Consequently, the network will then look for the optimum in a more local space and will converge better to a good answer.

IV. Results

In this section we explain and show the results of our modified model. We have used the images rendered by the MC sampler, Tungstan[1][2], which provides all these additional features including 4 vectors for normal, 4 vector for depth and also 4 vectors for albedos. We use the examples provided by them for the testing the denoisig tasks in this work. More specifically we show the detailed results in different mods on 3 images of their examples naming "cornell-box", "car2" and "lamp".

We evaluate the result of the network in four main different modes: Only using the noisy input as the input, using only the MC features as the input, using the combination of noisy input and the MC features as the input, and last one is using individual MC features (i.e., only normals or only albedos) as the input for the network. We tested this on three different noisy images, and here we provide the value of the loss function and the PSNR of the network output to both the noisy image and also the image without the noise (which is the goal of the denoiser) in tables I, II and III. Furthermore, we also provide the visual output of the networks to compare the result of denoising process in figures 2, 3 and 4.

We have tested both different approaches for using the additional MC features in training the network and the bottom two rows in each table correspond to the result of the network using the IOC approach. As it's illustrated by these results, this approach did not generate results as good as the IIC approach, when the additional features are used in addition to the random vectors as an input for the network.

The result also indicate that the Skip network is the appropriate choice for this method which only uses a single image for training. As shown in the tabless, the loss value of the ResNet networks are about x10 greater than the same values from the Skip network. One interesting observations in denoising with the ResNet network is that using the additional MC features for training the network plays a significant role in improving the output of the denoiser. More specifically, the PSNR of the network output with the truth image for the car scene is increased from 20.78 to 27.69 after adding the additional MC features to the network input.

As our analysis shows, the Skip network produces more promising results for denoisng the images rendered by Monte Carlo sampling, we tried to investigate the results of this network more carefully. So, we tried different types of inputs for this network including, using a random matrix, only the normal features (normal, normala, normalb, and normal_variance), only the depth features, only the albedo features, all the MC features together, and using both the random matrix and all the MC features at the same time. One interesting observation is that using the random input generally leads to better results compare to using only one of the MC faeatures individually. This observation is only true for the box and the lamp scene. Our thought about this is that using a single MC feature mostly leads to a biased output as the input is already biased to some specific feature, while using a random matrix does not have any bias and it will be easier for the network to be trained

[1]https://github.com/tunabrain/tungsten

[2]https://github.com/tunabrain/tungsten/tree/master/src/denoiser

with that random input compare to a biased input.

Furthermore, the result for denoising with using all the MC features without any random input shows that it's not always as good as using only the random matrix for the input. Since for all the scenes the results of these two different modes is comparable, the network is able to produce better results by using the random matrix for the box scene, while using the MC features will lead to better denoised images for the car and the lamp scene.

The best denoiser is always achieved when both the random matrix and the MC features are used as the input of the network. While the difference is not very significant, but our experiments shows that without any exception using the random matrix in addition to the MC features leads to a better network in terms of all the metrics.

To investigate the impact of the network architecture we used different sizes for the inner channels of the encoder and the decoder in the Skip network. This is important specifically when the number of input channels are increased significantly and that is when we use all the MC features in the input. Therefore, for these two cases we also provide the result of the network with double channel sizes. To do so, we use the intermediate channel sizes [16, 32, 64, 128, 256] instead of [8, 16, 32, 64, 128] for both the encoder and the decoder of the Skip network. The result shows that using more intermediate channels in the network will lead to a better denoiser, more specifically for the car scene the PSNR of the denoised image improve after changing the inner channels of the network.

For more examples we provide the improvement achieved by using additional MC features compared to using only random input for training the network on a number of scenes in table IV. The visual output for comparing the result of denoising is also provided in figures 5 to 14.

V. Conclusion

Building a single image deep learning denoiser in important as it doesn't need a large dataset to train on. Our This work shows that a denoiser for the images rendered by Monte Carlo sampling can be built by training a deep neural network with only a single image. Our implementation is built on top of [1], where the authors have shown the structure of a generator network is enough to capture a great deal of low-level image statistics. The input to the network in their work is only a random matrix, while we show in this work that using the additional features provided by Monte Carlo renderer on top the random input can improve the performance of the denoiser. The use of the MC features for a denoiser are mostly crucial for learning the structure of images with sharp edges and lots of details as it has been indicated in the results.

References

[1] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Deep image prior," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9446–9454.

[2] S. Bako, T. Vogels, B. McWilliams, M. Meyer, J. Novák, A. Harvill, P. Sen, T. Derose, and F. Rousselle, "Kernel-predicting convolutional networks for denoising monte carlo renderings." *ACM Trans. Graph.*, vol. 36, no. 4, pp. 97–1, 2017.

TABLE I: Denoising the cornell box scene with different inputs and different types of networks

Network Type	inner chanels	input	loss	PSNR1 (output to noisy image)	PSNR2(output to ground truth)
Skip	[8, 16, 32, 64, 128]	random	0.00024	36.206701	26.084307
Skip	[8, 16, 32, 64, 128]	only normals	0.001518	28.188236	24.375119
Skip	[8, 16, 32, 64, 128]	only depths	0.000958	30.188466	24.532702
Skip	[8, 16, 32, 64, 128]	only albedos	0.005776	22.383485	19.506452
Skip	[8, 16, 32, 64, 128]	all features	0.000294	35.320125	25.572898
Skip	[16, 32, 64, 128, 256]	all features	0.000266	35.753139	25.912861
Skip	[8, 16, 32, 64, 128]	all features + random	0.000233	36.320069	**26.101154**
Skip	[16, 32, 64, 128, 256]	all features + random	**0.000221**	**36.549776**	26.090269
ResNet	10 blocks, 32 channels	random	0.005769	22.389221	19.541218
ResNet	10 blocks, 32 channels	all features	0.003606	24.429487	21.091101
ResNet	10 blocks, 32 channels	all features + random	0.002939	25.318088	21.733304
Skip (loss with MC features)	[8, 16, 32, 64, 128]	random	0.000268	34.33857	25.550674
ResNet (loss with MC features)	10 blocks, 32 channels	random	0.022163	19.348973	17.250534

(a) noisy image **(b)** random input **(c)** random and all MC features as the input

Fig. 2: The output of deonoising for the box scene

TABLE II: Denoising the car scene with different inputs and different types of networks

Network Type	inner chanels	input	loss	PSNR1 (output to noisy image)	PSNR2 (output to ground truth)
Skip	[8, 16, 32, 64, 128]	random	0.000544	32.64113	29.085062
Skip	[8, 16, 32, 64, 128]	only normals	0.000434	33.625937	29.472658
Skip	[8, 16, 32, 64, 128]	only depths	0.00044	33.564252	29.493578
Skip	[8, 16, 32, 64, 128]	only albedos	0.00043	33.668865	29.554421
Skip	[8, 16, 32, 64, 128]	all features	0.000455	33.417124	29.396223
Skip	[16, 32, 64, 128, 256]	all features	0.000375	34.264971	29.604018
Skip	[8, 16, 32, 64, 128]	all features + random	0.000433	33.639577	29.528747
Skip	[16, 32, 64, 128, 256]	all features + random	**0.000355**	**34.495242**	**29.754112**
ResNet	10 blocks, 32 channels	random	0.006187	22.085333	20.780359
ResNet	10 blocks, 32 channels	all features	0.000807	30.928864	27.128531
ResNet	10 blocks, 32 channels	all features + random	0.000589	32.302177	27.690971
Skip (loss with MC features)	[8, 16, 32, 64, 128]	random	0.002006	26.158636	24.976312
ResNet (loss with MC features)	10 blocks, 32 channels	random	0.015964	18.145663	17.179351

(a) noisy image **(b)** random input **(c)** random and all MC features as the input

Fig. 3: The output of deonoising for the car scene

TABLE III: Denoising the lamp scene with different inputs and different types of networks

Network Type	inner chanels	input	loss	PSNR1 (output to noisy image)	PSNR2(output to ground truth)
Skip	[8, 16, 32, 64, 128]	random	0.002573	25.895526	26.328462
Skip	[8, 16, 32, 64, 128]	only normals	0.002431	26.142035	26.602357
Skip	[8, 16, 32, 64, 128]	only depths	0.002769	25.577502	25.809062
Skip	[8, 16, 32, 64, 128]	only albedos	0.002626	25.807467	26.141597
Skip	[8, 16, 32, 64, 128]	all features	0.002426	26.151798	26.68463
Skip	[16, 32, 64, 128, 256]	all features	0.002398	26.202242	26.635606
Skip	[8, 16, 32, 64, 128]	all features + random	0.002406	**26.18623**	**26.848126**
Skip	[16, 32, 64, 128, 256]	all features + random	**0.002391**	26.213971	26.432002
ResNet	10 blocks, 32 channels	random	0.011976	19.216943	18.805076
ResNet	10 blocks, 32 channels	all features	0.00353	24.522443	25.440231
ResNet	10 blocks, 32 channels	all features + random	0.003553	24.494133	25.518517
Skip (loss with MC features)	[8, 16, 32, 64, 128]	random	0.001479	24.928114	25.048508
ResNet (loss with MC features)	10 blocks, 32 channels	random	0.022306	17.065236	16.380919

(a) noisy image **(b)** random input **(c)** random and all MC features as the input

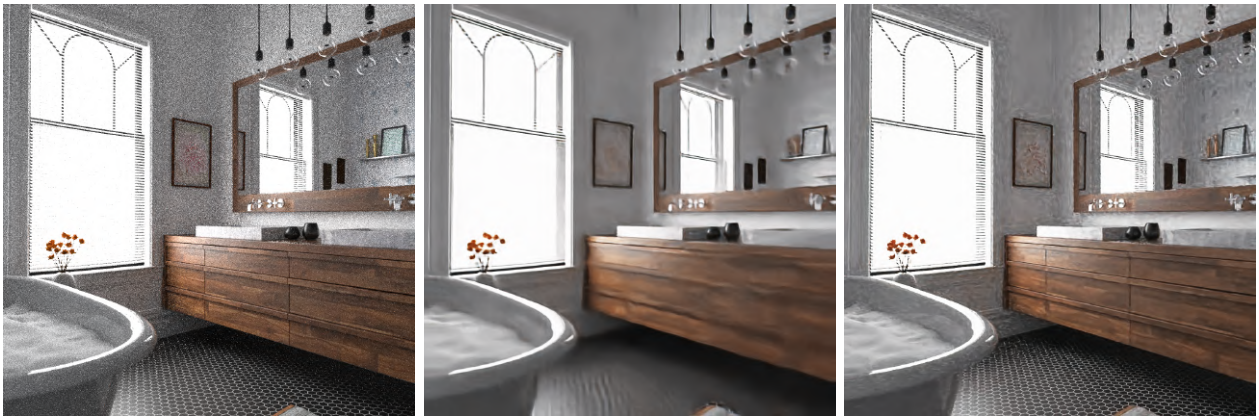
Fig. 4: The output of deonoising for the lamp scene

TABLE IV: Impact of using additional features in denoising different images with the Skip network for different scenes

Image	input	loss	PSNR1 (output to noisy image)	PSNR2(output to ground truth)
bathroom	random	0.014977	18.24563	21.463301
bathroom	all + random	0.011612	19.350903	23.557407
bathroom2	random	0.003222	24.918322	21.969846
bathroom2	all + random	0.002863	25.431873	22.002809
house	random	0.001736	27.603325	22.520975
house	all + random	0.001336	28.741478	22.606693
curly-hair	random	0.002615	25.825129	26.946244
curly-hair	all + random	0.001908	27.194585	28.309165
car	random	0.001301	28.85584	27.285519
car	all + random	0.001002	29.990364	28.240949
furball	random	0.001468	28.333437	29.362395
furball	all + random	0.00131	28.828612	29.540828
hair-culr	random	0.000579	32.370467	31.235705
hair-culr	all + random	0.00046	33.369388	32.488125
living-room-2	random	0.004688	23.290463	25.085683
living-room-2	all + random	0.004233	23.73316	25.456587
living-room-3	random	0.002876	25.4124	23.675514
living-room-3	all + random	0.002781	25.557493	23.539517
material-testball	random	0.001205	29.191484	26.517421
material-testball	all + random	0.001181	29.276202	26.629929
staircase	random	0.00267	25.734287	23.738901
staircase	all + random	0.001306	28.839495	24.558577
staircase2	random	0.003489	24.573149	24.326205
staircase2	all + random	0.003075	25.121635	24.625213
straight-hair	random	0.001681	27.743962	27.473526
straight-hair	all + random	0.001444	28.405044	28.043986
dragon	random	0.000884	30.535705	28.036115
dragon	all + random	0.000473	33.25376	28.710375

(a) noisy image **(b)** random input **(c)** random and all MC features as the input

Fig. 5: The output of deonoising for the furball scene



(a) noisy image **(b)** random input **(c)** random and all MC features as the input

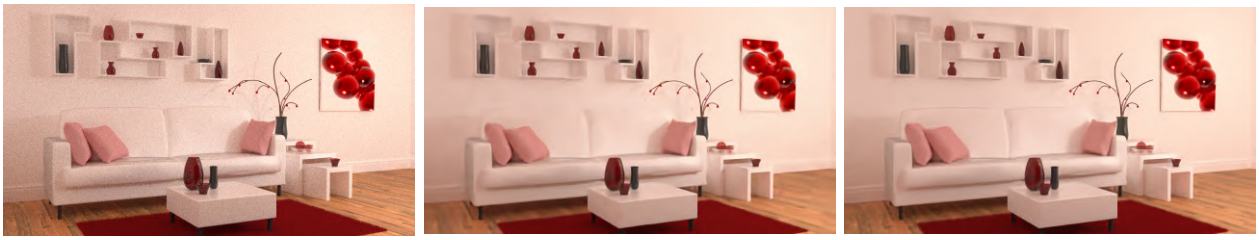
Fig. 6: The output of deonoising for the bathroom scene

(a) noisy image **(b)** random input **(c)** random and all MC features as the input

Fig. 7: The output of deonoising for the bathroom2 scene

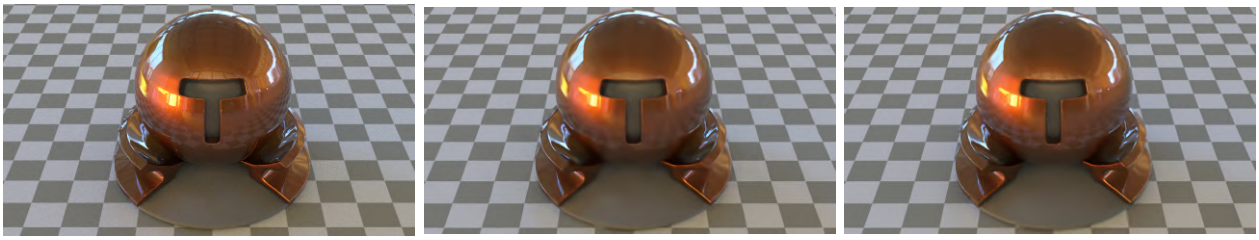
(a) noisy image **(b)** random input **(c)** random and all MC features as the input

Fig. 8: The output of deonoising for the living-room2



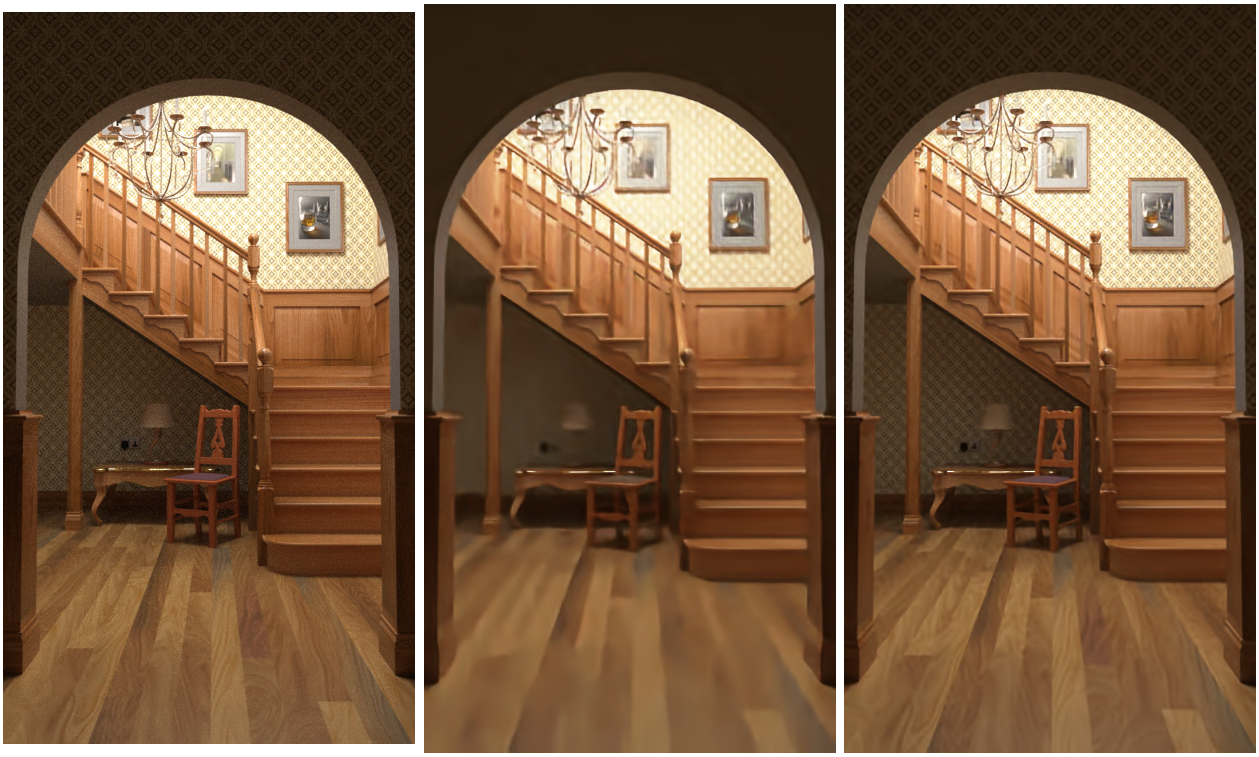
(a) noisy image **(b)** random input **(c)** random and all MC features as the input

Fig. 9: The output of deonoising for the living-room3 scene



(a) noisy image **(b)** random input **(c)** random and all MC features as the input

Fig. 10: The output of deonoising for the material-testball scene



(a) noisy image **(b)** random input **(c)** random and all MC features as the input

Fig. 11: The output of deonoising for the staircase scene

(a) noisy image **(b)** random input **(c)** random and all MC features as the input

Fig. 12: The output of deonoising for the staircase2 scene

(a) noisy image **(b)** random input **(c)** random and all MC features as the input

Fig. 13: The output of deonoising for the straight-hair scene

(a) noisy image **(b)** random input **(c)** random and all MC features as the input

Fig. 14: The output of deonoising for the dragon scene