The 11th IEEE International Conference on Dependable Systems, Services and Technologies, DESSERT'2020
14-18 May, 2020, Kyiv, Ukraine

Image noise reduction by denoising autoencoder

Lev Yasenko
Department of System Programming and Specialized Computer Systems National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
l.stydy.kpi@gmail.com

Yaroslav Klyatchenko
Department of System Programming and Specialized Computer Systems National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
k_yaroslav@scs.kpi.ua

Oksana Tarasenko-Klyatchenko
Department of System Programming and Specialized Computer Systems National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
oxana@scs.kpi.ua

***Abstract*—Neural networks are used in many tasks today. One of them is the images processing. Autoencoder is very popular neural networks for such problems. Denoising autoencoder is an important autoencoder because some tasks we need a preprocessed image to get less noisy result. This research describes ways to analyze noisy images produced by a physically-based render engine and how to reduce that noise. The results showed that the algorithms are logarithmic.**

Keywords—render, noise, autoencoder, denoise, neural network

I. INTRODUCTION

In computer graphics physical renders are used to create photo-realistic images. There are several algorithms for such renders. In this research we will use Cycles render because it is simple and free to use. It is an open-source physically-based render engine. After rendering with Cycles, there is noise in the image. To reduce it, the number of samples should be increased a lot. That require additional resources in time and hardware. We can use multi-core central processor units and additional video-cards or render-farms to reduce time. However such methods are expensive both in purchase and in use. To solve this problem, we can use a denoising autoencoder (DAE). It is a type of autoencoder (AE) that is a type of neural network (NN). We are using installed in our computer video-card Nvidia GTX 1050. The 3D-scene for creating dataset contains these 3D-objects: pale with diffuse material (roughness = 0.1), glossy cube (roughness = 0.283), glass sphere (roughness = 0.1, IOR = 1.45), monkey with subsurface scattering (scale = 1, radius = 1, 0.2, 0.1), light source is 'sun'. The average rendering speeds: 0.84 seconds per frame (1 sample images set), 1.58 seconds/frame (32 samples images set), 51.09 seconds/frame (2048 samples images set.

II. RESEARCH NOISY IMAGES

A render engine is a program that creates images based on 3d models.

The integrator is the rendering algorithm used to compute the lighting. Cycles currently supports a path tracing integrator with direct light sampling. It works well for various lighting setups, but is not as suitable for caustics and some other complex lighting situations [1].

Images quality depends on the number of samples per pixel. In the real world we have the same problem, for example to create image with less noise we can use a set of noisy images. So let's look at image averaging.

Consider a noisy image g(x, y) formed by the addition of noise η(x, y) to an original image f(x, y); that is,

$$g(x,y)=f(x,y)+\eta(x,y) \tag{1}$$

where the assumption is that at every pair of coordinates (x, y) the noise is uncorrelated and has zero average value. The objective of the following procedure is to reduce the noise content by adding a set of noisy images { gi(x, y) }.

If the noise satisfies the constructions just started, it can be shown that if an image g(x, y) is formed by averaging K different noisy images.

$$\bar{g}(x,y)=\frac{1}{K}\sum_{i=1}^{K} g_i(x,y) \tag{2}$$

then it follows that

$$E\{\bar{g}(x,y)\}=f(x,y) \tag{3}$$

where E{g(x, y)} is expected value of g [2].

To estimate images noise against reference image we should use dispersion. The reference 2048 samples image that has much less noise. Descriptive statistics which measure dispersion attempt to give a numerical indication of the with, or spread, of the observed values forming the frequency distribution. Dispersion can be constructed using the squares of individual deviations which eliminates the negative deviation problem [3]. Let's look at the following program text to evaluate the estimation of noise.

```
def read_images(pth):
    # 'im_0' is the reference image
    im_0 = np.array(cv2.imread(pth + 'Image0000.png'))
    # 'v' is an array of dispersions
    v = []
    for i in range(1, 257):
        im_tmp = np.array(cv2.imread(pth + 'Image' +
str(i).zfill(4) + '.png'))
        im_tmp = (im_tmp – im_0)**2
        # we need to divide sum of 'im_tmp' by its shape
(512*512)
        v.append(np.sum(im_tmp/512/512))
    return v
if __name__ == '__main__':
    v = read_images('./noisy/')
    plt.figure()
    plt.plot(range(1, 257), v, linewidth=2)
    plt.scatter(range(1, 257, 16),
        [(-np.log2(x/256)/9. + 1./9.)*v[0] for x in range(1, 257, 16)],
        color='red')
    # (-np.log2(x/256)/9. + 1./9.)*v[0] – logarithmic function
    plt.savefig("image.png")
```

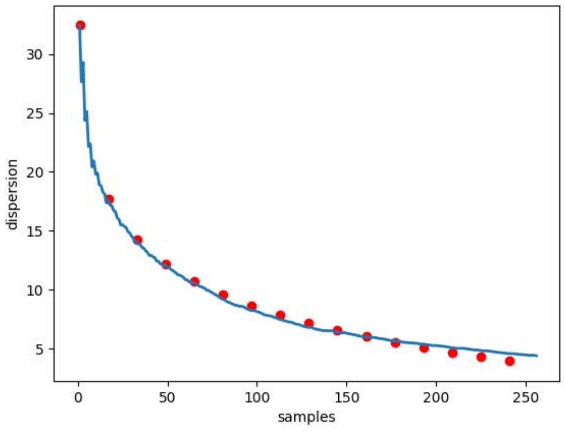

Fig. 1. 'line' - estimated data, 'dots' - logarithmic function.

From the "Fig.1" we can see that the noise logarithmically depends on the samples. And the rate of noise reduction decreases with increasing the number samples.

Our set of images is an animation, where camera is rotating over the 3d objects. The view shows that some noise remains in the same place.

III. Denoising autoencoder

Machine learning (ML) is a field of artificial intelligence that uses statistical techniques to give computer system the ability to "learn" from data, whithout being [4].

We will use DAE to solve the problem of slow noise reduction. DAE is type of autoencoders that adds noise to inputs during a training process. Autoencoders are one of the unsupervised deep learning models. The aim of an autoencoder is dimensionality reduction and feature discovery. An autoencoder is trained to predict its own input, but to prevent the model from learning the identity mapping, some constraints are applied to the hidden units. The simplest form of an autoencoder is a feedforward neural network where the input x is fed to the hidden layer of h(x) and h(x) is then fed to calculate the output x . The following equation can describe an autoencoder:

$$\bar{x} = O(a(h)) = Sigmoid(c + w * h(x)), \quad (4)$$

$$h(x) = g(a(x)) = Sigmoid(b + wx), \quad (5)$$

where x is outputs, a is a linear transformation, g are activation functions, Sigmoid is the logistic function.

In a denoising autoencoder the goal is to create a more robust model to noise. The motivation is that the hidden layer should be able to capture high level representations and be robust to small changes in the input. The input of a DAE is noisy data but the target is the original data without noise [5].

In our research noisy input is created before training by rendering different quantities of samples. And we are using convolution neural network (CNN). The input parameters of such a NN model correspond to individual pixel [6].

We created several model to compare which is better. Models 1, 2 and 3 contain 3 layers each. Input and output layers shape is '3*512*512' because our images shape is the same. The difference is in the middle layer (1 - '1*1024*1024', 2 - '3*512*512', 3 - '16*128*128'). Model 4 has the same input and output but in the middle it contains 3 layers ('32*128*128', '128*64*64', '32*128*128').

The fourth is DAE and it has the next structure showed on "Fig. 2".

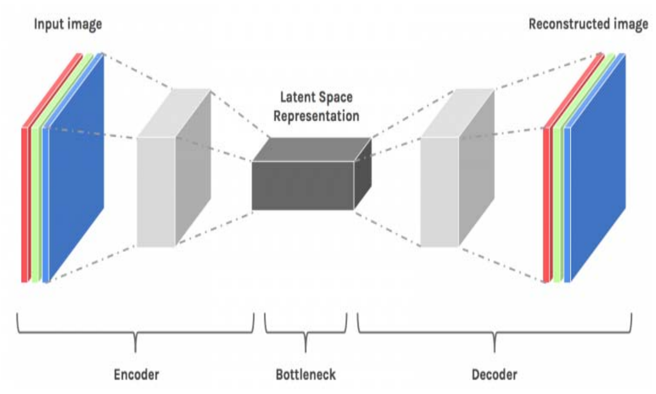

Fig. 2. Structure of denoising autoencoder [7].

The denoising of images corrupted with Gaussian noise is an active research topic in image processing and neural networks. Currently the best published results come from a deep convolutional architecture that uses hierarchical skip connections. The same architecture also yields state-of-the-art results for super-resolution, JPEG deblocking, and text removal. The network is trained using a large number of natural images, and it learns to complete or correct the images to look locally like the training image patches. We build on this approach, but in our application the noise has very different characteristics compared to the additive Gaussian noise. Some samples typically have a very high energy (well outside the dynamic range of an image), while most areas appear black. The input pixel values are therefore not even approximately correct, but we do know that they are correct on the average [8].

Moreover, there is another type of the DAE. This is recurrent denoising autoencoder.

Recurrent neural networks (RNN) are used for processing arbitrarily long input sequences. An RNN has feedback loops that connect the output of the previous hidden states to the current ones, thus retaining important information between inputs. This makes it a good fit to our application for two reasons. First, in order to denoise a continuous stream of images, we need to achieve temporally stable results. Second, because our inputs are very sparse, the recurrent connections also gather more information about the illumination over time [9].

IV. Results

We are using different train-test splits to see how it changes the results. The sum of training and test renges is 2048. Also we have 4 ranges for each split. The first two are for training and testing, and the last two are for validation of the first two.

Let's test our model with 5% of 2048 32 samples images for training and 95% for testing. The result is in "Fig. 3".

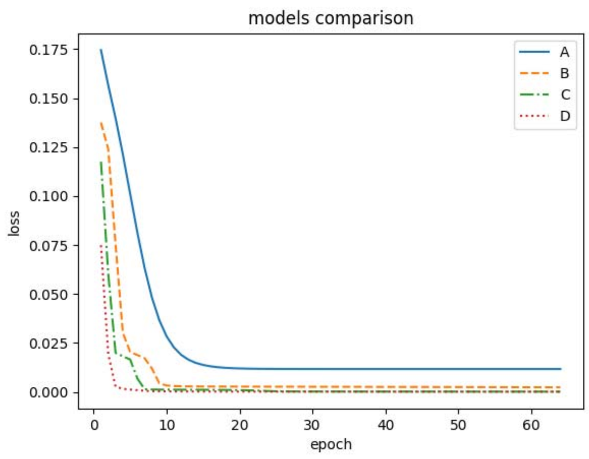

Fig. 3. A - model 1, B - model 2, C - model 3, D - model 4.

As we can see from "Fig. 2" that model 4 is the best choice when we are searching for the best loss per epoch. To estimate losses we used MSEloss that is dispersion. Also images are normalized while training (from RGB channels 0-255 to 0-1). Now let's look at the time of training. For the first model the time is 305.83 seconds, for the second – 228.60, for the third – 276.62 and for the fourth – 557.00.

Then we should look at output images to decide which model to take.

Let's look at "Fig. 4-7".

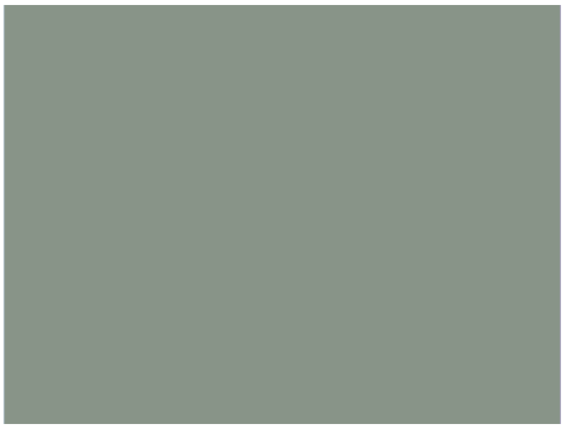

Fig. 4. Model 1 output.

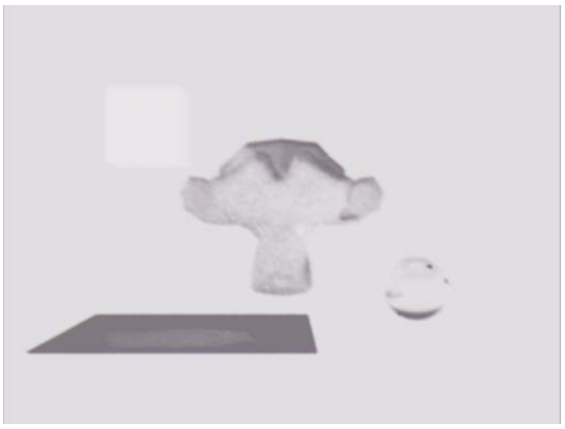

Fig. 5. Model 2 output.

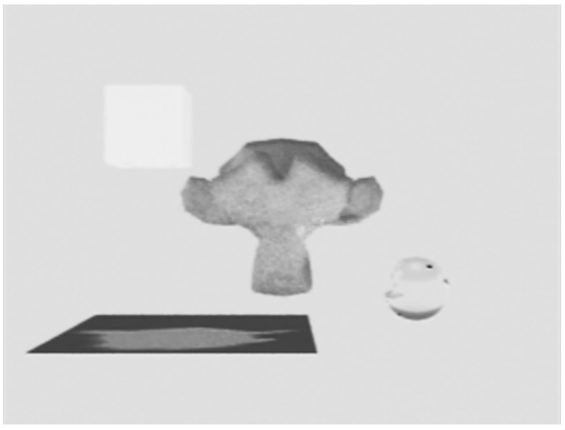

Fig. 6. Model 3 output.

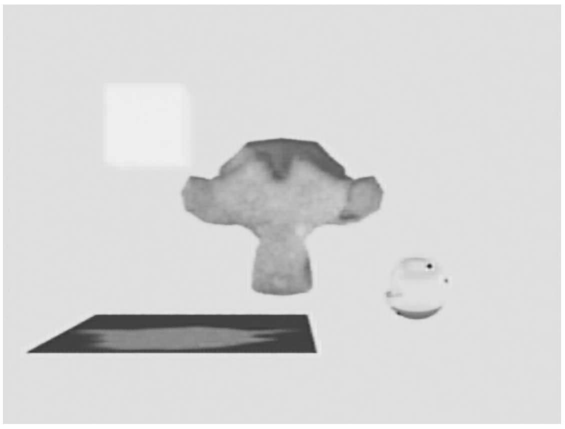

Fig. 7. Model 4 output.

So the first autoencoder is completely unworkable. After 64 epochs of training the loss is about 10e-2 and the output image is gray. We can't use the second model for denoising too, because it returns black and white images from color inputs. Model 3 is much better. It returns color images with losses about 10e-4. And model 4 returns color images with losses about 10e-5. For furthermore research we will use model 4 because it has better results in losses per epoch. However, model 3 is 2.01 times faster than model 4.

Then we checked how NN trains with another train-test split and with the same images as before. We have such splits as 5/95, 25/75, 50/50, 75/25. Let's look at "Fig. 8".

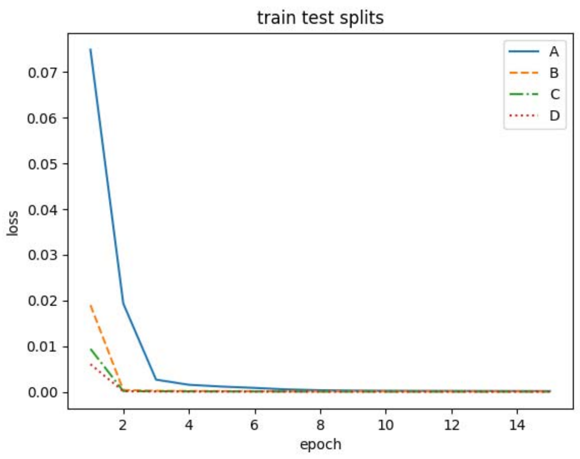

Fig. 8. A – 5/95, B – 25/75, C – 50/50, D – 75/50.

Every time the model was trained with 64 epochs, but "Fig. 8" has only 15 epochs because after training losses it differ by less then 10e-5. As a result, there is no reason to use more images for training if you want to get much better results. We can see that after 8 epochs losses are very similar and the speed of decreasing losses has become much slower.

After that we checked different inputs for this DAE such as 1 sample noise images, 32 samples images, rendered Normals and Albedo. Cycles can render not only physically-correct noisy images but also normals and albedo. We tried different inputs and the results are shown in "Fig. 9".

We can see the result after 32 samples image denoising in "Fig. 7". Other results are shown in "Fig. 10-12".

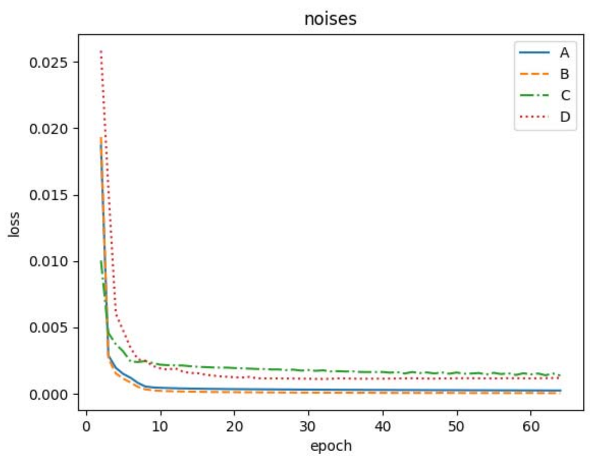

Fig. 9. A – 1 samples, B – 32 samples, C – Normals, D – Albedo.

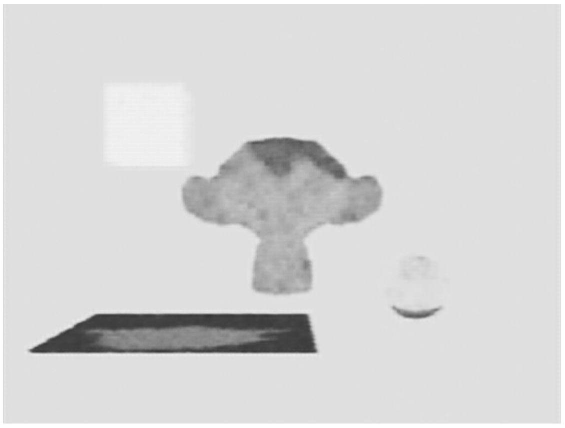

Fig. 10. Denoised 1 samples image.

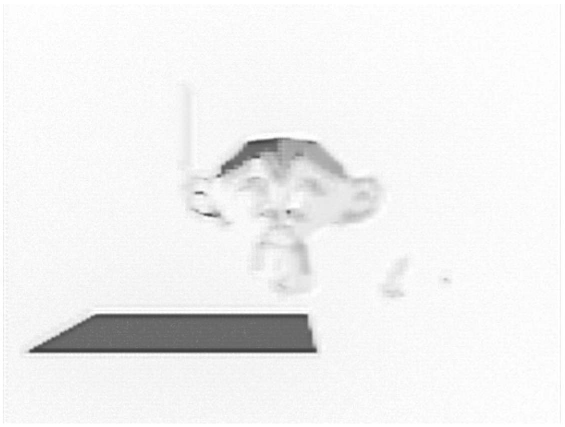

Fig. 11. Rebuilded normals image.

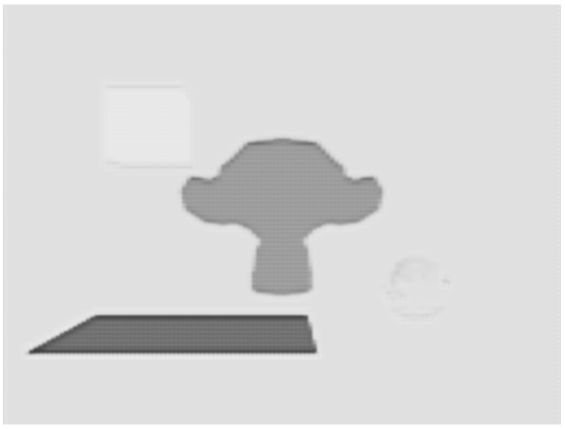

Fig. 12. Rebuilded albedo image.

In conclusion to this section, images with less level of the noise can be easily denoised and without a doubt this NN can be used not only for denoising but also as a transformer for other images. However, we can get fine results only if we use noisy images. Some DAEs can use everything as input and then return better results, e.t. "Intel Open Image Denoiser".

Ray tracing (RT) algorithm is similar to path tracing. The ray tracing filter is a generic ray tracing denoising filter which is suitable for denoising images rendered with Monte Carlo ray tracing methods like unidirectional and bidirectional path tracing. It supports depth of field and motion blur as well, but it is not temporally stable. The filter is based on a CNN, and it aims to provide a good balance between denoising performance and quality. The filter comes with a set of pre-trained CNN models that work well with a wide range of ray tracing based renderers and noise levels.

It accepts either a low dynamic range (LDR) or high dynamic range (HDR) color image as input. Optionally, it also accepts auxiliary feature images, e.g. albedo and normal, which improve the denoising quality, preserving more details in the image [9].

And the last part of this research includes testing for 5/95 dataset of 32 samples images but with 256 epochs ("Fig. 13-14").

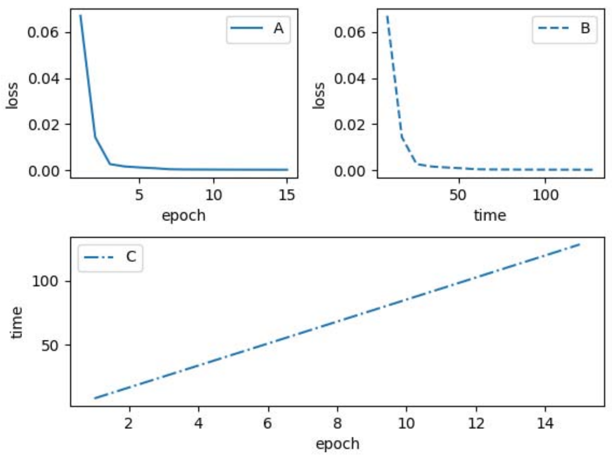

Fig. 13. Graphs of losses, time and epochs.

This NN was trained for about 2185.99 seconds. To show dependency graphs it is not necessary to take the whole diapason of epochs. In "Fig. 13" we can see that time of training depends on epochs linearly (graph 'C'), which is why the first two graphs look similar.

Keeping in mind the graph from "Fig. 1" we will try to evaluate how much better the results were after image processing by the DAE.

We need to find the level of loss that is equal to the level of samples. We know that the more samples, the less noise. Therefore, after training, the losses between input and output should be less than before. If the output image is not better than the input, then our model is not correct and we need to create a new NN. Then we need to find approximate time of rendering this result without DAE. After finding time and number of samples we can estimate how much faster or slower is using DAE with rendering than just rendering.

In "Fig. 14" we see time dependency, and how loss depends on samples.

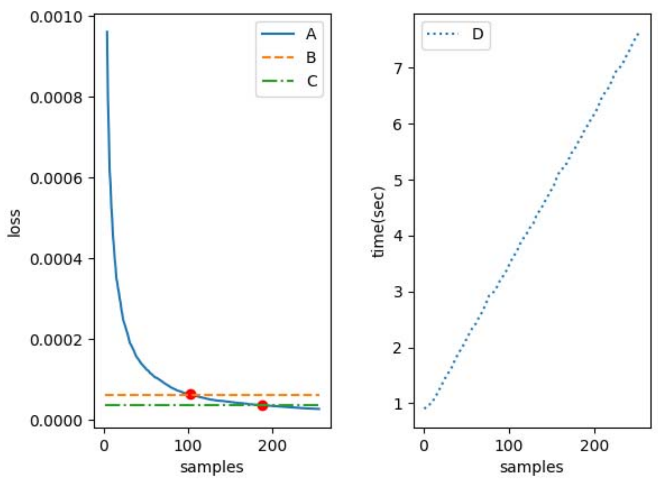

Fig. 14. Losses per sample and time per sample.

The left graph ('A') in "Fig. 14" is the same as the graph in "Fig. 1", but the estimated images were normalized. 'B' is the mean result of 5/95 32 samples images set trained by 64 epochs loss and it crosses 'A' in 108 samples. 'C' is the mean result of 5/95 32 samples images set trained by 256 epochs loss and it crosses 'A' in 186 samples.

The right graph ('D') in "Fig. 14" shows that time depends on samples linearly. However, by using average speed of rendering we will find average speed for rendering with 186 samples.

Let's make some equations to find the approximate time ('k', 'b' and 'x' are variables).

$$32 = k * 1.58 + b, \tag{6}$$

$$2048 = k * 51.09 + b, \tag{7}$$

$$186 = k * x + b, \tag{8}$$

where $k = 40.72$ and $x = 5.36$.

Approximately we need 3.049 hours for rendering 2048 frames with 186 samples.

V. CONCLUSION

After training our model we still have loss in outputs. The main thing is that we don't need to train DAE every time we want to render a new set. We have to train it once. The length of the training range may not be large. Even if the speed of decreasing losses become lower with decreasing epochs, it is important that the more epochs, the less losses. Time depends on the epochs linearly, so for better results you need more time. The best result we have got after training is 186 samples and this process has taken 2185.99 seconds and 95% of 2048 frames have been denoised in 84.26 seconds. The rendering time of all frames is about 1.58*2048 = 3235.84 seconds. All the operations took 1.52 hours. This is a little bit faster than approximately 3.049 hours for rendering, but if we can train it once, it means we can reduce 2185.99 seconds or 0.6 hours. As a result, it will be 3.3 times faster.

REFERENCES

[1] Sampling, Blender manual, https://docs.blender.org/manual/en/latest/render/cycles/render_settings/sampling.html.

[2] Gonzalez Rafael C., Woods Rechard E., "Digital image processing", 2002, pp 112-113.

[3] Thomas Reginald William, Huggett Richard J., Modelling in Geography: A Mathematical Approach, 1980, p 17.

[4] Hu, Z., Advances in Computer Science for Engineering and Education II, 2019, p 626.

[5] Askari R., Auto Encoders, https://reyhaneaskari.github.io/AE.htm.

[6] Hu, Z., Petoukhov, S., Dychka, I., He, M., Advances in Computer Science for Engineering and Education, 2019, p 570.

[7] Kopczyk D., Denoising Autoencoder: Part I – Introduction to Autoencoders, https://dkopczyk.quantee.co.uk/dae-part1 .

[8] Chaitanya C.R.A., Kaplany A.S., Schied C., Salvi M., Lefohn A., Nowrouzezahrai D., Aila T., Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder, 2017, p 98:4.

[9] Intel® open image denoise (high-performance denoising library for ray tracing), https://openimagedenoise.github.io/documentation.html.