

---

# Image Restoration without Clean Data

---

Ji Tin Justin Li - A 20423037

Subowen Yan - A 20430537

## Abstract

We apply the different statistical distribution to image reconstructed by deep learning, to be specific, convolution neural networks(CNN). The model will try to map the corrupted image to a clean image. The result will be extraordinary. It is absolutely able to restore images by applying CNN to corrupted images, without explicit image priors as input to the networks. In fact, we show that a single model can do Bernoulli, Gaussian, and Poisson distribution removal, as well as, text removal. It is all based on corrupted data input to the model.

## 1. Problem statement

Signal remaking from debased or deficient estimations is a significant subfield of statistical analysis. The deep neural networks have started significant enthusiasm for staying away from the statistical modeling of signal corruptions. Replaced by mapping corrupted image data to the unobserved clean data through CNN. It will use a corrupted image only as training input and a clean image as a target.

$$\operatorname{argmin}_{\theta} \sum_i L(f_{\theta}(\hat{x}_i), y_i)$$

where  $f_{\theta}$  is a set of a parametric, under the loss function  $L$ . The  $\hat{x}$  is a random variable based on statistical distribution given by a clean image. For instance, Training data may include images with Gaussian noise in it. It may have text in the image. Or it could lose some pixels in the image, and so on.

## 2. Paper overview

The Noise2Noise paper [1] explores signal reconstruction by using machine learning methods, more specifically convolutional neural networks. It is suggested that *it is possible to learn to restore images by only using images corrupted with noise*. A convolutional neural model architecture is proposed, trained and tested, using synthetic noisy images, Monte Carlo rendered images, as well as practical real life data of undersampled MRI scan images.

From a theoretical standpoint, the paper suggests a key idea: *corrupting the training target with zero-mean noise does not change what the learned weights of the network.*

This has several implications for data collection in real life:

1. Noisy training targets perform just as well as clean targets when training regression models (neural networks).
2. In many real life scenarios, lower quality data can be collected with lower capture budget i.e. higher volume of data (of lower quality) can be collected with the same capture cost.

Intuitively, this is particularly helpful to data scientists since data constraints are the bottlenecks of many machine learning problems, and more available data usually leads to better generalization of models.

Below is a list of practical experiments that were run by the authors.

### *Training with synthetic data with zero-mean noise filters*

Various noise generation processes were used to produce synthetic noisy targets, such as additive gaussian noise, poisson noise, multiplicative bernoulli noise, and text overlay filters. The results were promising as the trained models (with noisy targets) do show similar performance to models trained with clean targets. Comparison with other restoration benchmarks were also included, which showed the proposed network produced similar results to RED30 [2], another noise restoration conv. network with auto-encoders.

### *Training with Monte Carlo rendered images*

Images from virtual scenes are rendered using Monte Carlo path tracing methods. This rendering method obtains the intensity of each pixel by calculating the expectation value from the sampled “light rays”, which inherently introduces noise to the rendered image. The noise introduced by the random path sampling process is zero-mean, such that the theory mentioned above applies. The practical results agree with the theory

background. I.e. The model trained with noisy targets performs similarly to the model trained with clean targets.

### Real life data application with undersampled MRI images

Modern MRI techniques rely on compress sensing to reduce the capture cost of each MRI scan. The signal is undersampled, and non-linear reconstruction is performed to produce a scan. A sampling parameter  $\lambda=10$  is selected which controls the percentage of samples that are retained relative to a full sampling. The results show that the model trained with noisy targets shows similar testing loss to the model trained with clean targets as well.

## **3. Proposed work**

### **3.1 Main Objective**

We are to construct convolutional neural networks that will be trained to do image restoration. The main objective of our work is to validate whether zero-mean noise on target images would affect the learned weights and performance of the final trained model. Our work will closely follow what the Noise2Noise paper suggests.

We aim to reproduce the results produced by the Noise2Noise paper, from the section of training the models using synthetic data produced by gaussian, poisson, bernoulli and text filter. The Monte Carlo rendering experiments and MRI will not be replicated by us due to the lack of expertise and time constraints.

### **3.2 Practical experiments and implementations**

Although the source code of their experiment is accessible on github [3], it is written using tensorflow. We are to implement and train our own model using keras with python. The model architecture follows the given architecture provided in the paper, which contains a combination of convolution layers, maxpooling layers, upsampling layers and inception blocks which concatenate multiple convolution operations. The image data will be obtained from publicly available sources, which includes Kodak and BSDS300. We use two different datasets to feed the neural network. Kodak contains 24 pictures [4]. BSDS300 has 200 train images and 100 test images [5]. The implemented network shall be trained on the xsede cluster gpus. In the end, 4 separate models will be trained using the 4 synthetic datasets (gaussian, poisson, bernoulli and text filter), and we shall evaluate their performances on noise restoration.

## 4. Implementation details

### 4.1 Data preparation and preprocessing

As mentioned above, we use images from Kodak and BSDS300 for training. Two scripts that assist in downloading the images are included: “download\_kodak.py” and “download\_BSDS300.py”.

“download\_kodak.py” script is provided by the Noise2Noise repository, and “download\_BSDS300.py” is implemented by ourselves.

As for preprocessing, we applied 4 different kinds of filters (additive gaussian, poisson, multiplicative bernoulli, and text) to the clean images to create noisy input targets and noisy output targets. This preprocessing is done by the script “data.py”, which is based on the data.py script provided in this github repository [6]. This script contains a NoisyDataset() class which reads the given directory and returns noisy image pairs according to the given parameters. We modified the code so that it no longer uses the pytorch backend for image processing operations, in order to reduce unnecessary package dependencies.

### 4.2 Model implementation and architecture

For the Keras model implementation, the loaded NumPy [7] data arrays are normalized by dividing by 255. Next, we shuffle the indices and feed the data into the neural network that we build through Keras. We use a functional API [8] to build CNN. The optimizer that we use is Adam [9]. The learning rate, beta1, and beta2 are, respectively,  $1e-4$ , 0.9, and 0.99. The loss function is L2 loss [10], mean squared error. The matrix that is used during compile is L2, as well.

NAME	Nout	FUNCTION
INPUT	n	
ENC_CONV0	48	Convolution $3 \times 3$
ENC_CONV1	48	Convolution $3 \times 3$
POOL1	48	Maxpool $2 \times 2$
ENC_CONV2	48	Convolution $3 \times 3$
POOL2	48	Maxpool $2 \times 2$
ENC_CONV3	48	Convolution $3 \times 3$
POOL3	48	Maxpool $2 \times 2$
ENC_CONV4	48	Convolution $3 \times 3$

POOL4	48	Maxpool $2 \times 2$
ENC_CONV5	48	Convolution $3 \times 3$
POOL5	48	Maxpool $2 \times 2$
ENC_CONV6	48	Convolution $3 \times 3$
UPSAMPLE5	48	Upsample $2 \times 2$
CONCAT5	96	Concatenate output of POOL4
DEC_CONV5A	96	Convolution $3 \times 3$
DEC_CONV5B	96	Convolution $3 \times 3$
UPSAMPLE4	96	Upsample $2 \times 2$
CONCAT4	144	Concatenate output of POOL3
DEC_CONV4A	96	Convolution $3 \times 3$
DEC_CONV4B	96	Convolution $3 \times 3$
UPSAMPLE3	96	Upsample $2 \times 2$
CONCAT3	144	Concatenate output of POOL2
DEC_CONV3A	96	Convolution $3 \times 3$
DEC_CONV3B	96	Convolution $3 \times 3$
UPSAMPLE2	96	Upsample $2 \times 2$
CONCAT2	144	Concatenate output of POOL1
DEC_CONV2A	96	Convolution $3 \times 3$
DEC_CONV2B	96	Convolution $3 \times 3$
UPSAMPLE1	96	Upsample $2 \times 2$
CONCAT1	$96 + n$	Concatenate INPUT
DEC_CONV1A	64	Convolution $3 \times 3$
DEC_CONV1B	32	Convolution $3 \times 3$
DEC_CONV1C	$m$	Convolution $3 \times 3$ , linear act.








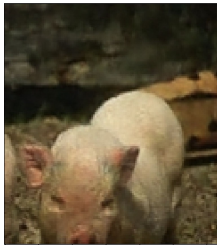


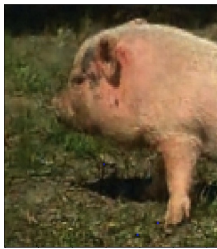

*Table 1, Network architecture used in our experiments. Nout denotes the number of output feature maps for each layer. The number of network input channel  $n$  and output channel  $m$  depends on the experiment. All convolutions use padding mode “Same”, and except for the last layer are followed by a leaky ReLU [11] activation function.*

### 4.3 Training the model on xsede

To ease the training process and job submission on xsede, the entire data preparation process is bundled into one single .py file, named “train\_model.py”, which takes 2 input arguments: filter type and dataset type. An sbatch job description file is also included, named “prj\_test.sb”.

### 4.4 Model evaluation and inference

For inferencing the result, we load an image from the BSD 300 dataset's test folder. And pass the image into the predicted model, which we get from the Keras API. We use Matplotlib to show the predicted image.

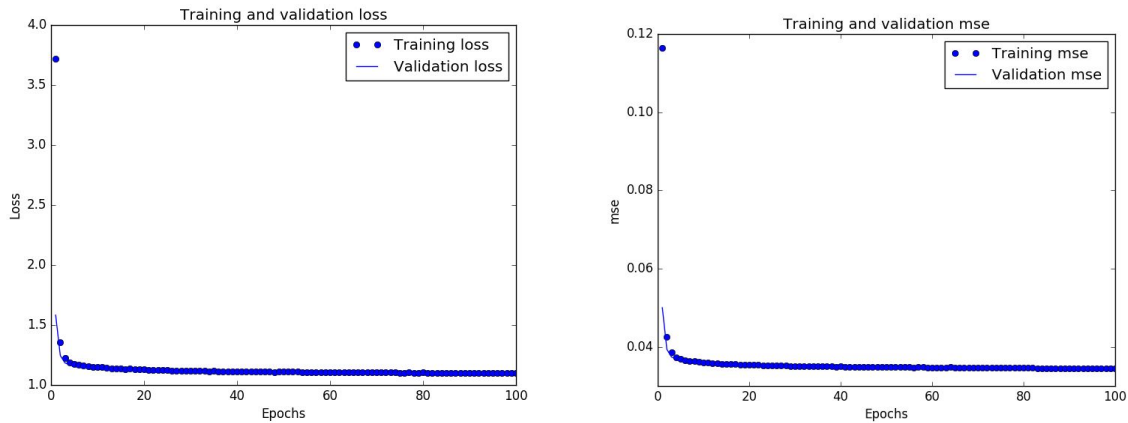
Dataset	Image filter	Input image	Output image	True image
Kodak	Bernoulli			
BSD 300	Bernoulli			
Kodak	Gaussian			
BSD 300	Gaussian			

Kodak	Poisson			
BSD 300	Poisson			
Kodak	Text			
BSD 300	Text			

*Table 2, for the dataset column, we use Kodak or BSD 300 to feed the network. For the image filter column, we use multiplicative Bernoulli distribution, Gaussian distribution, Poisson distribution, or text as filter applying to the image. For the input image column, it shows the images that are being applied by a filter to feed the network. For the output image column, it presents the result of the image that the filtered image passes into the network. For the true image column, it delivers a clean image that is the target image.*

From Table 2, we can see that the model works well with zero-mean distribution, such as multiplicative Bernoulli, Gaussian, or Poisson. The less difference of noise compared to the image, the better our model predicts, just like Gaussian noise. The multiplicative Bernoulli or Poisson will predict an image darker compared to the true image. It might

be because the noise tends to be more extreme. For the text noise, we can tell that it works but not so great. It is probably because the text is not zero-mean distribution. The “randomness” of the text noise is hard to capture.



*Chart 1, both charts are from training images with Poisson distribution using Kodak datasets.*

From chart 1, we know both the training and validation loss chart shows how loss moves with respect to epochs. The training and validation MSE shows the same. Both of the charts have the same trend. As the number of epochs increases, the result of loss or mean squared error decreases. The reason that we only show two charts here is what other charts which we get from training different datasets and/or distributions are all having the same trend.

## 5. Conclusion

In conclusion, we have presented the zero-mean distribution applied to deep neural networks to recover image signals. It is absolutely possible or even better to turn a corrupted image into a clean image without observing a clean image. We have shown that the statistical distribution noise, such as multiplicative Bernoulli, Gaussian, or Poisson can be removed with general-purpose deep convolutional models. This will significantly benefit many areas in the future.



## Reference

- [1] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila. Noise2Noise: Learning image restoration without clean data.
- [2] Mao, Xiao-Jiao, Shen, Chunhua, and Yang, Yu-Bin. Image restoration using convolutional auto-encoders with symmetric skip connections. In Proc. NIPS, 2016.
- [3] <https://github.com/NVlabs/noise2noise/tree/master/dnnlib>
- [4] <http://r0k.us/graphics/kodak/kodak/>
- [5] <https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/BSDS300-images.tgz>
- [6] <https://github.com/shivamsaboo17/Deep-Restore-PyTorch/blob/master/data.py>
- [7] <https://docs.scipy.org/doc/numpy/reference/generated/numpy.load.html>
- [8] <https://keras.io/models/model/>
- [9] <https://keras.io/optimizers/#adam>
- [10] <https://keras.io/losses/>
- [11] <https://keras.io/layers/advanced-activations/>