Reproducing 'Recorrupted-to-Recorrupted: Unsupervised Deep Learning for Image Denoising'

Ankith Kumar akumar643@gatech.edu
Ali Kazmi akazmi30@gatech.edu
Rohan Patwardhan rpatwardhan7@gatech.edu
Neal Bayya nbayya3@gatech.edu

Georgia Institute of Technology
North Ave NW, Atlanta, GA 30332

Abstract

Image denoising has been a large topic of interest, especially for the deep learning community, due to the importance of having clean data with minimal noise for uses such as training deep models. Although there have been strongly performing supervised denoisers, a lot of development has been made in the last couple of years to create effective unsupervised denoising models that do not require a ground-truth image for training purposes. Up to now, the performance of such models have not been comparable to their supervised counterparts. However, a group of researchers earlier this year proposed a data augmentation technique called recorrupted-to-recorrupted, or R2R, which was found to be competitive to many supervised denoisers, outperforming most unsupervised deep denoising approaches. In this investigation, we hoped to understand and recreate the experiments put forth by those researchers, due to the implications of such a development in unsupervised deep denoising approaches. By recreating the experiments used to test the R2R approach, we were able to get quite similar results to that of the original investigation, showing the strong potential that the proposed R2R method has, in the realm of unsupervised denoising.

1. Introduction

Low quality images are all around us. Faulty sensors, bad data, and loss of quality during data processing can all contribute to noisy data. Noisy data is data with an element of randomness to it, which reduces the image quality significantly. For Deep Learning models where we want to do things like Object Detection, Image Segmentation, or Classification, having clean training data is essential. We set out to build a model to automatically remove the noise from this training data, which would ideally improve the performance of any model trained on this denoised data. We built a model that can learn to remove the noise from images in an unsupervised manner, which is significant because unlabeled data is significantly less expensive to acquire than labeled data. We did not start this work from scratch, but instead set out to build upon a 2021 CVPR paper titled 'Recorrupted-to-Recorrupted: Unsupervised Deep Learning for Image Denoising' [2] .

1.1. Motivation

In the past, a majority of common effective models for image denoising have mainly utilized supervised learning. Examples include the feed-forward denoising convolutional neural network (DnCNN) [3] that is usually trained on noisy and clean pairs of images, as well as the Noise2Noise model which is trained on pairs of noisy images. There has been a significant amount of progress in the development of unsupervised learning for the purpose of denoising images in the past few years. Despite this, supervised learning methods still outperform unsupervised methods for this use case, in most circumstances. Now, one may wonder why having an unsupervised learning method for denoising images is necessary at all. It should be remembered that, when no ground-truth image is available, many supervised learning methods suddenly become obsolete, whereas unsupervised learning methods remain useful in these contexts.

For these reasons, Tongyao Pang, Huan Zheng, Yuhui Quan and Hui Ji aimed to develop a more powerful unsupervised denoiser earlier this year. They proposed a data augmentation method called recorrupted-to-recorrupted (R2R) [2], which was designed to account for the overfitting that was generally caused by the lack of ground-truth images in most circumstances that required unsupervised methods in the first place. This methodology was essentially reliant on the the authors' observation that the cost function defined on the pairs of noisy images that the R2R method constructs is equivalent to the cost function defined on pairs that consist of noisy images as well as their clean counterparts. According to the investigation, this novel method was a strong competitor to many supervised denoising models, strongly

outperforming existing unsupervised learning methods for denoising. In our investigation, we aim to be the first to analyze the methodologies used in that experiment in order to attempt to reproduce its results.

2. Background

The paper poses an unsupervised model which operates in the following manner:
We denote a noisy image as y = x + n, where n is some noise sampled from $N(0, \sum_x)$ and its noise free counterpart is x. The paper uses a recorruption scheme that takes each noisy image and generates a new pair of images $\hat{y}$ and $\tilde{y}$.
$\hat{y}$ = y + Az, $\tilde{y}$ = y - Bz
where A,B satisfy $AB^T =_x$ and z is sampled from $N(0, I)$. The CNN is then trained over this pair of images $(\hat{y}, \tilde{y})$
Since n, the noisy added to the noise free image, was sampled from a normal distribution, we can rewrite the equations for $\hat{y}$ and $\tilde{y}$ as follows:
$\hat{y} = y + \sqrt{\sum_x} D^T z$, $\tilde{y} = y - \sqrt{_x} D^{-1} z$
The point of making these pairs of recorrupted images is to test whether the model can take in an image $u + Az$ and output an image u^*, which is the model's attempt at denoising u. The paper claims that this process is equivalent to training a denoiser over the following pair: $(y + Az, x)$ in a supervised manner.

The goal is to create a trained model into which we can pass in arbitrary noisy images and generate a relatively accurate denoised image without ever utilizing the original clean image. This is because real world data is usually noisy and does not always have clean counterparts associated with it. A successful implementation of this model can compete with already existing supervised models and is important in the aforementioned case when ground truth data is not readily available.

3. Approach

We approached building an image denoiser by using a Convolutional Neural Network. It has input and output dimensions of batch size x H x W. That reflects the fact that our problem structure in Image Denoising maps from an input image to an output image. The Pytorch framework was used for implementation of our model, and we tried several architectures before settling on our final model. We used 5 convolution layers with ReLU activations, and the full model architecture can be seen in figure 1 below. The dimensions with -1 represent batch size, and are intentional. When training we used the Mean Squared Error loss, which we used from the pytorch nn module, and the Adam optimizer. In terms of hyperparameters, we had to choose the learning rate and number of epochs. We tuned these hyperparameters by hand until we got images that were denoised to a level satisfying to the human eye,

Figure 1. Shows our initial model architecture and number of trainable parameters

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 321, 481]             320
              ReLU-2         [-1, 32, 321, 481]               0
            Conv2d-3         [-1, 64, 321, 481]          18,496
              ReLU-4         [-1, 64, 321, 481]               0
            Conv2d-5        [-1, 128, 321, 481]          73,856
              ReLU-6        [-1, 128, 321, 481]               0
            Conv2d-7         [-1, 32, 321, 481]          36,896
              ReLU-8         [-1, 32, 321, 481]               0
            Conv2d-9          [-1, 1, 321, 481]             289
================================================================
Total params: 129,857
Trainable params: 129,857
Non-trainable params: 0
```

and the number of epochs was the most important factor in getting closer to that goal. We used learning rates between 1 and 0.0001 during our testing, as well as epochs between 5 and 50, when trying to hyper parameter tune.

We anticipated problems in determining a good model architecture, as there are many options and there is no guarantee the model used in the paper was the best. We also anticipated needing a large number of epochs, since in the paper they used 8000 epochs. We ended up not doing this due to computational limitations. To prepare for this, we purchased a colab pro subscription so we could have reliable access to faster GPU's.

We used our model architecture to initially test creation of clean images, but didn't use it for our experiments because the paper used an existing architecture and we wanted to reproduce that. We used a DNCNN implementation that the paper referenced for certain aspects of our experiments, specifically for sections 4.1 and 4.2 and 4.3.

We still ran into issues with batch size for experiment 4.1, due to the large size of our 180x180 images. With any batch sizes higher than 16, we ran out of memory on our GPU and could not train. This was specifically a problem for the DnCNN, which specified a higher batch size.

One unexpected problem we faced was with our initial implementation simply making an extremely noisy image. We recognized this problem by using opencv and displaying the original image vs the denoised image. To solve the issue, we had to dive into mathematics. When we were generating $\hat{Y} = y + Az$ and $\tilde{Y} = y - Bz$, we saw that the formulas for A and B were in Corollary 2 of the paper in section 3. $A = \sqrt{\sum_x} * D^T$, where * is a matrix multiplication, and $B = \sqrt{\sum_x} * D^{-1}$ where * is a matrix multiplication. We calculated z correctly as well as D and subsequently D^T and D^{-1}. However, for sigma we

initially did $\sqrt{noiselevel}$ * the identity matrix. We tried correcting this by believing that $\sqrt{\sum_x}$ was referring to square root of the covariance matrix of x, the original clean image. $\sqrt{\sum_x}$ was actually intended to be (noise level / 255.0) * identity matrix. This yielded good results, however after consultation with the original authors we found out that this was not the correct implementation. Also, we were initially creating a z matrix and our incorrect $\sqrt{\sum_x}$ matrix and using it for each image. After consultation, we changed this to have a new z matrix be created for each image in the tensor as well as a new $\sqrt{(\sum_x)}$ matrix created for each image. This in turn meant that each image had a different Az or Bz configuration that was added or subtracted, respectively, to said image to create $\hat{Y}$ and $\tilde{Y}$ respectively. Upon fixing this we were able to clearly see the noise being removed from the image, thus creating an image that resembles the original clean image to the human eye.

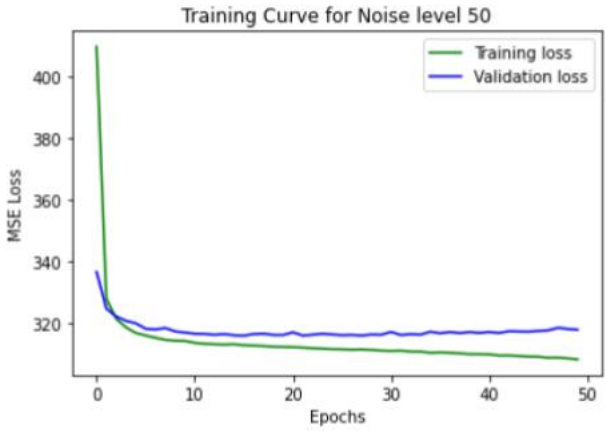
4. Experiments

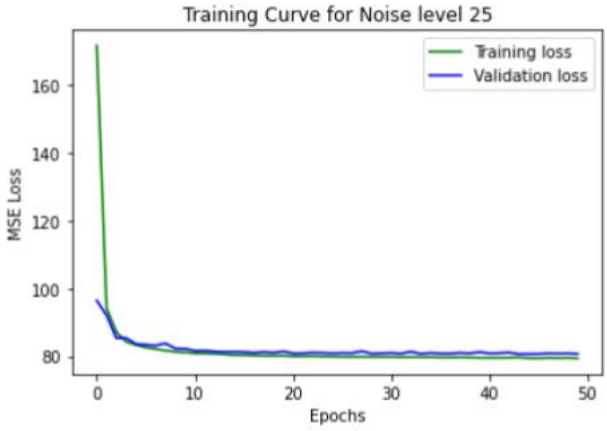
In the experiments section of the original paper, the authors evaluated their R2R process on the removal of additive white Gaussian Noise, as well as on real-world image denoising contexts. We effectively followed the same process in order to reproduce the results of the experiment. After performing the necessary data preprocessing steps as described in the following sections, we used the same network architecture as that of DnCNN, which is a commonly used denoising deep neural network. The models did slightly differ from section to section, so refer to the subsections below for more details on the training process for each experiment. After the preprocessing and training was completed for each experiment, the model was tested on the test set of data, and evaluated using metrics known as peak signal-to-noise ratio as well as structured similarity index in order to determine how effectively denoising had been performed by our models.

4.1. Reproducing Experiments on AWGN Removal

The first one involved an investigation regarding how well our models can remove additive white Gaussian Noise (AWGN). More specifically, we wished to test how well our model can denoise images from a modified version of the BSD68 dataset, which we synthetically corrupted using AWGN with variances = 25 and = 50. In order to start this experiment, we cropped and augmented the data from BSD400, which resulted in 40 x 40 images which would be used as our training data. We used a DnCNN network with 17 convolutional layers and 2 batch normalization layers, as well as a batch size of 128. This network was then trained on the train data that we had prepared in the previous step, with a learning rate of 10^{-3}, and 50 epochs. See the training curves in figure 2 below.

Figure 2. Our DnCNN training curve for multiple noise values





We also had a learning rate decay, with $\gamma = 0.2$ at the milestone of 30. Afterwards, we tested our model on the AWGN BSD68 dataset. For each image, we computed 50 random constructions of $\hat{y}$, and got the model output for each of the 50 constructions per image. The results were then averaged for each image, which was treated as our model output for each image. Afterwards, the average

Figure 3. Shows examples of an image being denoised with our model when $\sigma = 25$. The lowest and highest PSNR images are shown

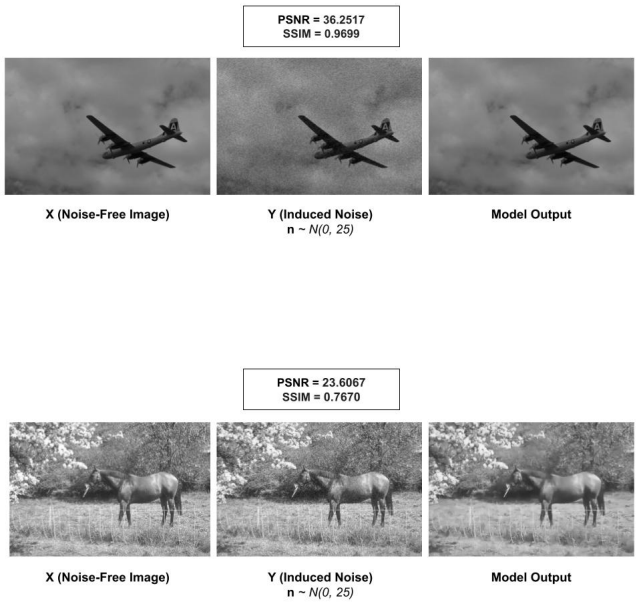
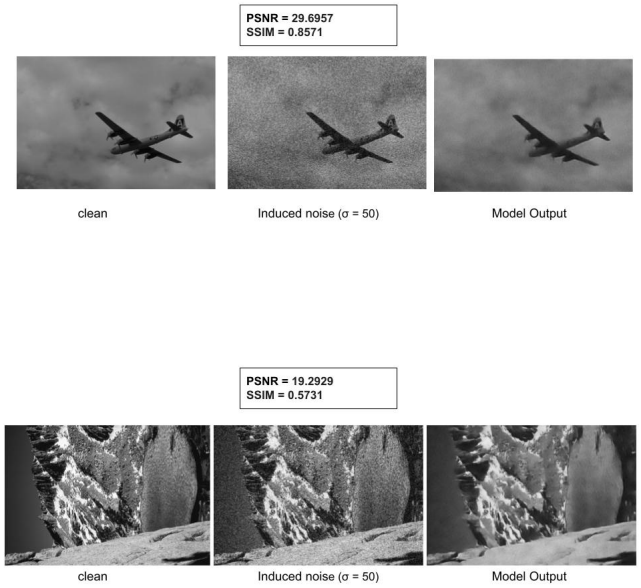


Figure 4. Shows examples of images being successfully denoised with our model when $\sigma = 50$. These are the highest and lowest PSNR respectively



PSNR (peak signal-to-noise ratio) and SSIM (structured similarity index) indices were calculated on the resulting images, as a metric to determine how well the denoising was performed. We were able to achieve an average PSNR index and average SSIM index extremely similar to that of the original investigation. See figure 7 in section 4.3 where T=50 below for the PSNR and SSIM distributions.

For AWGN of = 25, the original investigation had achieved an average PSNR and SSIM of 29.14 and 0.822, respectively. In our investigation, we got 28.1823, and 0.8673, respectively. For AWGN of = 50, the original investigation had achieved an average PSNR and SSIM of 26.13 and 0.709, respectively. In our investigation, we got 23.6822, and 0.6964, respectively. As seen here, we were able to effectively reproduce the results of the original investigation for this experiment. Figure 3 includes examples with sample images, which shows the clean image, the image with induced noise, as well as the output of the model. See figure 4 for similar details but for noise level 50.

4.2. Reproducing Experiments on Real-Word Image Denoising

The second category of experiments involved experiments on real-world image denoising, using three different representative datasets. Firstly, we used the PolyU training dataset, and used the specified preprocessing method in order to ready the data for training. In particular, the value Σ was determined from the data using a process similar to that outlined in Algorithm 1 in Section 2.4 of the research paper by Chen, Zhu Heng [1]. Afterwards, to model noise in each color channel, two matrices A and B were generated, where $A = 20 \times \sigma \times I$ and $B = \sigma \times I/20$. These were then used to determine $\hat{y}$ and $\tilde{y}$ just as we had done in the past, using the formulas $\hat{y} = y + Az$ and $\tilde{y} = y - Bz$, where y represents the images. We used a DnCNN network with 17 convolutional layers and 2 batch normalization layers, as well as a batch size of 4. This network was then trained on the train data that we had prepared in the previous step (with a 70/30 train-test-split), with a learning rate of 10^{-3}, and 30 epochs. We also had a learning rate decay, with $\gamma = 0.2$ at the milestone of 30. See the training curve in figure 5 below.

The model was then tested on our test set of images. Afterwards, the average PSNR and SSIM indices were calculated on the resulting images, as a metric to determine how well the denoising was performed. Yet again, we were able to achieve an average PSNR index and average SSIM index

Figure 5. Shows our PolyU training. Note there is only a training loss since the dataset was only 100 images and we decided to only do a train test split.

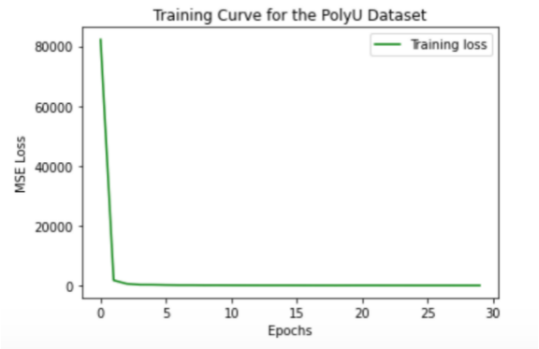
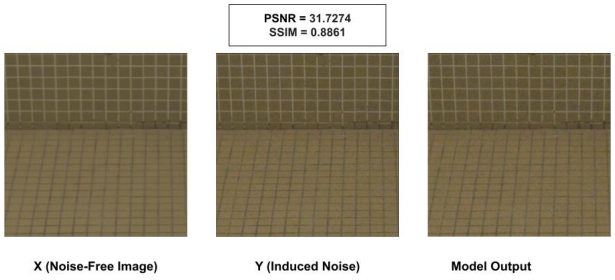


Figure 6. The PolyU dataset results were as expected, with the noise being removed by our model successfully

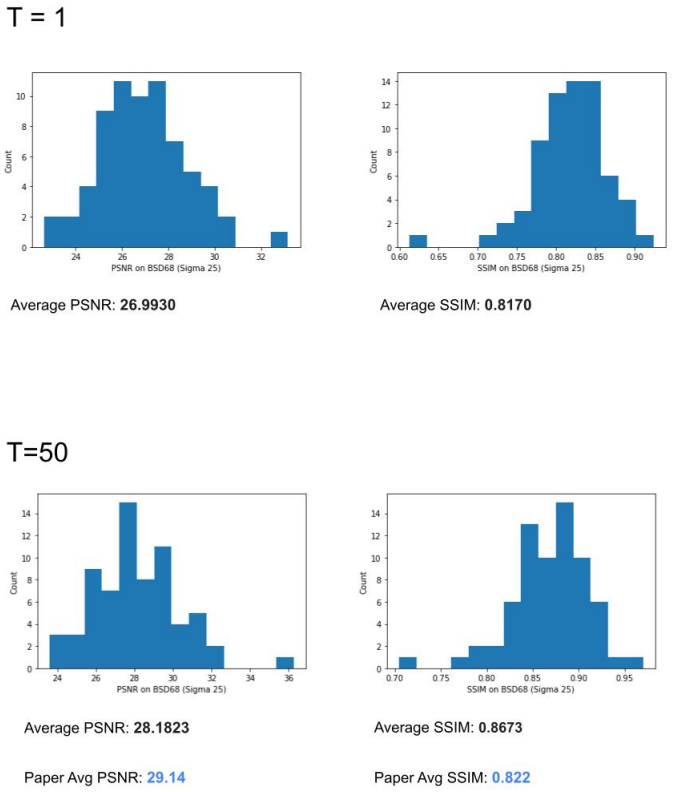


extremely similar to that of the original investigation. For the PolyU dataset, the original investigation had achieved an average PSNR and SSIM of 35.60 and 0.964, respectively. In our investigation, we got 32.3223, and 0.9085, respectively. As seen here, we were able to effectively reproduce the results of the original investigation for this experiment. Figure 6 also includes examples with sample images, which shows the clean image, the image with induced noise, as well as the output of the model for the median PSNR image.

4.3. Reproducing Ablation Study

The last set of experiments we performed was on ablation, and corresponds to section 4.3 in the original paper. Ablation refers to removal experiments. Here we are checking the robustness of our model with respect to the estimation error of noise level, in PSNR. We wanted to reproduce the results with the same predictions scheme as the original paper, and this experiment will be done on the BSD68 dataset. There were three experiments in this section, and each serves to justify decisions of our methodology in other sections.

Figure 7. The average PSNR and SSIM values for T=1 vs T=50. Both have $\sigma = 25$. Shows that T=50 is better, as expected



First, we will attempt to reproduce table 4 of the original paper. They test an image on the model, they generate 50 samples of $\hat{y}$ and pass them all through the model and then average the output. The first study shows that doing this averaging yields better performance than just one sample calculation of $\hat{y}$ being passed into the model. The T=50 results as well as the T=1 results can be seen in figure 7, and show that doing the averaging is indeed better. This is why for our section 4.1 experiments removing AWGN we kept T=50. Next, we will reproduce table 5 of the Recorrupted to Recorrupted paper, which checked the performance of different α values on the PSNR. The goal was to see how different values of the recorruption factor impact the final image. To perform the experiment we simply adjusted the α factor and ran our tests. We found similar values to the paper, which was a success in terms of reproducibility. Both us and the original paper found that α values did not have a significant effect on the PSNR, and correspondingly did not have much of an effect on the denoising visually. See our values in Figure 8. We also tested these results visually. Please see the examples below

Figure 8. Our reproduction of the table that shows recorruption levels does not drastically affect our ability to denoise images.

Cell Format: PSNR / SSIM

α	1	0.5	0.3	0.1
σ = 0.25	27.3876 / 0.8481	**28.1823 / 0.8673**	26.6439 / 0.8213	19.2013 / 0.4276
σ = 0.5	**23.8266 / 0.7195**	23.6822 / 0.6964	21.4280 / 0.5226	11.4554 / 0.0912

5. Conclusions

In this paper, we aimed to determine if the R2R method, proposed earlier this year, is indeed a feasible and effective unsupervised deep denoising approach. In order to do so, we understood and implemented the approach, testing it on its effectiveness in the removal of additive white Gaussian Noise, as well as on real-world image denoising contexts. Throughout the various experiments performed, we were able to get average PSNR and SSIM values that were extremely similar to that of the authors' original experiments, indeed supporting their claim that the R2R is an effective denoising approach for these contexts. With this great leap in the development of unsupervised deep denoising approaches, we hope that progress continues to be made in the field, making ground-truth images increasingly unnecessary for effective denoising.

6. Collaboration

We split up the work fairly evenly. Ali worked on creating a 5 layer model to learn to denoise on BSD68. Rohan worked on the PolyU experiments, and Neal worked on the AWGN removal section and introduced algorithm number 3 in the appendix (7.4.3). Ankith worked on the ablation study.

Everyone worked on at least 2 sections of the final paper as well.

References

[1] Guangyong Chen, Fengyuan Zhu, and Pheng Ann Heng. An efficient statistical method for image noise level estimation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 477–485, 2015.

[2] Tongyao Pang, Huan Zheng, Yuhui Quan, and Hui Ji. Recorrupted-to-recorrupted: Unsupervised deep learning for image denoising. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2043–2052, 2021.

[3] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, Jul 2017.

7. Appendix

7.1. Data

7.1.1 AWGN Removal Experiment

The Colored Berkeley Segmentation Dataset (CBSD) 400 is used to generate the training data for the experiment detailed in section 4.1. The dataset may be found at BSD400 Dataset

We pre-processed the images through gray-scaling and normalization within the range $[0, 1]$. We induced the additive white gaussian noise (AWGN) via the algorithm described in 7.4.1. As suggested in the R2R paper, the dataset was augmented by taking 40×40 crops of the original images (180×180). The algorithm for augmentation is described in 7.4.2.

Section 3 details the methodology of constructing the training data (specifically $\hat{y}$, $\tilde{y}$ from the tensor of crops). In our code, further described in section 7.3, the dataset class R2RDenoisingDataset contains the code for re-corruption.

The training curves shown in Section 4.1 were constructed with a train-test split of 80-20. Further augmentation may be done via rotation and flipping. In addition, we tested the dataset on the BSD 68 dataset, which is accessible via BSD-68. The algorithm for prediction is outlined in 7.4.2.

7.1.2 Real Image Noise Removal Experiment

The PolyU Dataset of 100 cropped images is used for the training and testing sections of the experiment detailed in section 4.2. This dataset is publicly accessible via: PolyU Dataset

The images were pre-processed via normalization within the range $[0, 1]$. As suggested by the R2R paper, the noise level was generated via the algorithm described in 7.4.4. A patch size of 32 was used to generate the noise level and the patch size affects the runtime of the computations. The paper this algorithm originates from chose a default patch size of 8.

Section 4.2 details the construction of $\hat{y}$ and $\tilde{y}$ which were used in training the data. The R2RDenoisingDataset in the Recorruption section of this experiment contains the code for this process. The training curves shown in section 4.2 were constructed with a train-test split of 70-30.

7.2. Experimental Details

7.2.1 AWGN Removal Parameters

The following hyperparameters may be adjusted to reproduce the results of Section 4.1 and Section 4.3: α, `noise level`, and T. The README file contains instructions for modifying these parameters and running the experiments. The recommended hyperparameters from the paper were used along with increasing and decreasing them to analyze model output quality.

7.2.2 Model Architecture

The model architecture and hyperparameters to produce the results in Section 4 was configured as suggested by [2]. Specifically, we use the DnCNN model architecture which is publicly accessible via DnCNN

7.2.3 Computing Infrastructure

The experiments in Section 4 were run on a NVIDIA Tesla K80 GPU. The runtime of the model training was around 15 minutes for the AWGN removal experiments and 30 minutes for the experiments on the PolyU dataset. The runtime for generating the patches and calculating $\hat{y}$ and $\tilde{y}$ for the PolyU dataset took 20 minutes with a path size of 32.

7.2.4 Performance Statistics

While the models were trained according to the squared L2 loss function, the following two performance statistics were used in analysis:

1. Peak Signal-to-Noise Ratio (PSNR): This index is measured in decibels and represents the quality of the image given the ground truth. A higher PSNR indicates higher image fidelity.

2. Structured Similarity Index (SSIM): The SSIM index measures the retention of structure in the image after processing. The SSIM index may range from -1 to 1, where 1 indicates that the images are identical.

7.3. Code

Our source code is publicly accessible at Code. The PSNR_Scores.pkl file contains the PSNR scores for all the images in the PolyU dataset and the SSIM_Scores.pkl file contains the SSIM scores for all the images in the same dataset. The checkpoint.pth file contains the most recent model state. The R2R-Reproducibility-PolyU-Experiment.ipynb file has the code for reproducing experiment 4.2 and the R2R-Reproducibility-AWGN-Removal.ipynb file has the code for reproducing experiments 4.1 and 4.3. The README file has information that pertains to running the code successfully, including descriptions of parameters and hyperparameters that are used.

7.4. Algorithms

7.4.1 Inducing AWGN Noise

A random noise tensor, n, is generated from the distribution $N(0, 1)$ and is multiplied by the noise level divided by 255. The noisy dataset is generated via $y = x + n$, where x is the normalized and gray-scaled clean image. Assuming the random sampling is O(1), the algorithm to generate noise is linear in runtime with respect to the size of the dataset.

7.4.2 AWGN Prediction Recorruption

The hyperparameter T controls the number of recorruptions and forward passes through the model needed to denoise a single image. We average the T outputs of the model to generate the final prediction.

7.4.3 AWGN Kernel-based Prediction

We originally hypothesized that testing images on variable sizes, which may be larger than the 40x40 crops used for training, would affect the reconstructed image's structure. As such, we constructed a method to denoise an image by evaluating 40x40 crops of the image in a kernel-based manner. The outputs of the crops are stitched together to generate the final model prediction. This method is computationally intensive, as the number of crops is $O(MN/c^2)$, where the image size is $M \times N$ and c is crop dimension (40). Furthermore, each crop would be passed into the model T times. The code for this prediction method is included in the code repository.

7.4.4 Patch Sigma Estimation

For the real world images in the dataset, the noise level is not applicable, so it is modeled via an existing algorithm described in this paper: Chen G , Zhu F , Heng P A . An Efficient Statistical Method for Image Noise Level Estimation[C]// 2015 IEEE International Conference on Computer Vision (ICCV). IEEE Computer Society, 2015. The code for implementing the algorithm was found on Estimation Algorithm. The observed image is first broken up into patches of size d and the eigenvalues of this decomposed dataset X is calculated with r eigenvalues, where $r = d^2$. Iterate from 1 to r and for each iteration, perform the calculation $\tau = \frac{1}{r-i+1}\sum_{j=i}^{r}(\lambda_j)$. If this value is the median of the subset of eigenvalues $(\lambda_j)_{j=i}^{r}$, the noise level is calculated as $\sqrt{\tau}$ and the algorithm stops. The runtime of this algorithm is $O(sr^2 + r^3)$. Calculating the mean vector and covariance matrix of the decomposed dataset X (which

was described in the paper), take $O(sr)$ and $O(sr^2)$ respectively. Calculating the eigenvalues takes $O(r^3)$. Sorting the eigenvalues from greatest to least and iterating from 1 to r to calculate τ both take $O(r^2)$. The variable s here is (M - d + 1)*(N - d + 1), where the dimensions of the input image I are MxNxc. When using the PolyU dataset, c turns out to be 3.