Table of Contents

Abstract

Image denoising is a complex inverse problem that involves removing noise artifacts to recover the true image. One of the reasons it is so complex is that it is not a one-one problem, but a one-many problem. Having some form of prior knowledge guiding the model can help reach solutions that are not only less noisy but are also perceptually valid. But the prior of the natural image distribution is difficult to estimate via normal means, and most denoising models only implicitly learn the distribution. In this work, autoregressive generative models are explored in their usage as priors for image denoising. They hold the special property of being able to calculate the pixel intensity distribution of an image, which can be used to easily calculate its likelihood. Specifically, this work aims to establish proof of concept that a PixelCNN model trained on the MNIST dataset can be used to denoise images of digits. A loss function was formulated and experimented with for this purpose. As a result, the model is able to achieve a PSNR value of 25.6 and SSIM value of 0.8.

Acknowledgements

I would like to express my gratitude to my supervisor Dr. Alexander Krull for his constant motivation and support throughout this project. From patiently listening to me to engaging in active discussions with me, his unparalleled support would never go unappreciated. I would also like to thank Prof. Abhirup Ghosh for his invaluable suggestions and critique that went into building this report. I would also like to thank my family for giving me the opportunity to pursue my dreams. And finally, my thanks to the University of Birmingham for providing me with a fruitful journey to be remembered.

Chapter 1: Introduction

Image denoising is the process of removing noise from an image and retrieve its original appearance and properties. While the description is simple, the actual problem is quite complex in nature. This is because of a few reasons:

i) Noise can come in various forms: It can be due to equipment or human errors, environmental disturbances, compression artifacts, and a plethora of other sources. Modelling noise is a research field in and of itself.

ii) Denoising is not a one-one problem: A noisy image may correspond to multiple denoised outputs, depending on our needs.

iii) Denoising might produce artifacts too: Distinguishing between noise and the ground truth can be difficult for the model, and it can end up adding or removing artifacts from the image.

iv) Data availability: Clean-noisy image pairs may not be always available, and synthetically created datasets are usually the only option.

This is by no means an exhaustive list but is indicative of how complicated image denoising can be. As described in [image denoising survey], denoising can be represented as:

$$y = x + n$$

Where y is the noisy image, x is the ground truth clean image, and n is the noise added to it by external causes. The job of a denoiser is to recover the clean image from the noisy image, by subtracting the noise from it.

A more practical definition to denoising would be that an optimal denoiser produces a clean output that:

i) Is perceptually valid.

ii) Maintains the original structure and properties of the image.

iii) Reduces the impact of noise.

White it is natural to train a denoising model using clean ground truth images as the target, the corresponding clean images may not be readily available. In these cases, it might be better to build models that do not explicitly require clean image pairs as input. One such workaround is to have some form of prior knowledge guiding the model as to what a clean image could be.

Autoregressive generative models are a special class of generative models that can explicitly provide the probability distribution of the input as the output. They are computationally more expensive to train than counterparts such as Generative Adversarial Networks but have their own attractive advantages too. GANs learn the input distribution implicitly, while Autoregressive models provide the distribution as output, making it easy and direct to calculate likelihood of images. The likelihood of a clean image should have higher likelihood than a noisy image, and this property can help direct the denoising model towards a plausible clean image for the given noisy image.

In this work, we explore the usage of a particular family of Autoregressive models, the PixelCNN family, as a prior for the denoiser.

1.1 Aim

This project aims to explore the usage of autoregressive models as priors for image denoising. It aims to study Gated PixelCNN model in specific and build a denoising module that can do the actual denoising of the images guided by the prior. At the end of the project, proof-of-concept results will be provided along with a proper study defining the conditions in which the results may be generated.

1.2 Assumptions

In order to adhere to the timeline of the project, certain assumptions and constraints have been made:

- We assume that the noise model is Gaussian noise, which is a popular benchmark for denoising algorithms. It is popular because it is simple to create and is a reasonable approximation for real-world noise.
- Only a single class of grayscale images are taken for ease of training of the prior, but this can be extended to multiple classes and dimensions. In this case, we chose the MNIST image dataset, and specifically only the digit 7.

These assumptions would help with keeping track of the core objective of the project.

1.3 Objectives

With these assumptions in place, the project deliverables are as follows:

- Train PixelCNN model on the training dataset.
- Evaluate the performance of the model and its viability as a prior.
- Develop the denoising module, and a loss function for denoising that includes the prior.
- Establish hyperparameters of interest and study their function and significance.
- Evaluate the denoising model using standard metrics.

1.4 Report Layout

The Project Report follows the following layout:

- Chapter 1 discussed the project aim and objectives.
- Chapter 2 goes into existing works in image denoising and the PixelCNN family.
- Chapter 3 gives the background regarding the project.
- Chapter 4 discusses the project setup.
- Chapter 5 lists out all the experiments done, and the results obtained.
- Chapter 6 discusses the results, outlines future work and concludes the report.

1.5 Summary

In this section, the aims, assumptions, and objectives of the project are laid out along with a gentle introduction into the project topic.

Chapter 2: Literature Review

Image denoising has been a growing field of research for decades. A lot of research went into developing methods to denoise images that were corrupted by various means. In this section, a brief recap of the exemplary works in image denoising and autoregressive models is laid out.

Denoising an image is a classical yet challenging problem that still sees new research despite being worked on for decades (McCann et al. 2017; Fan et al. 2019; Goyal et al. 2020; Tian et al. 2020). BM3D (Dabov et al. 2007) was a classical state of the art algorithm for image denoising that exploited similarity between patches in the image to identify and remove noise, a technique known as collaborative filtering. This was an extension of the Non-Local Means (Buades et al. 2011) approach to image denoising, which basically involves calculating the intensity of a pixel as the average of the intensities of the neighbouring pixels. But before Deep Learning arrived at the scene, it was thought that the denoising capabilities were exhausted and better denoisers may not be possible (Chatterjee and Milanfar 2010).

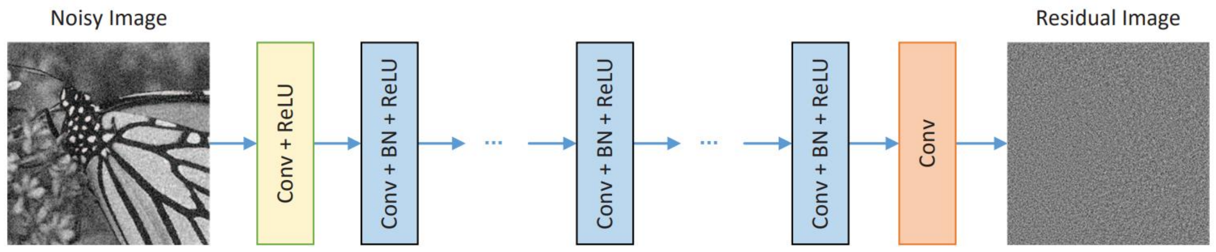


Figure 1: DnCNN (Zhang et al. 2017)

The field of image denoising has seen quite a few advances with the advent of deep learning and convolutional neural networks. With the advent of deep learning, DnCNN (Zhang et al. 2017) provided state of the art results, employing residual learning methods. Instead of finding the corresponding clean image for the input, DnCNN predicts the noise to be subtracted from the image. On the BSD68 dataset corrupted with gaussian noise of known standard deviations, it was able to achieve PSNR values of up to 26.23 dB (for σ=50). More impressively, compared to other methods at the time, DnCNN was one of the first algorithms that could perform blind Gaussian denoising where the noise level (i.e., Standard deviation) is unknown. In an effort to combine both previous state of the art approaches, Block-Matching Convolutional Neural Network (Ahn and Cho 2017) was born. This provided new state of the art results in image denoising. FFDNet (Zhang et al. 2018) was then proposed as an efficient means of denoising images with various noise levels and boasted speeds comparable to BM3D.

Ground truth pairs are not always available for noisy images, especially for real-world noise. And so, efforts were put into developing approaches that do not require clean images as input for training. One of the seminal works in this area is Noise2Noise (Lehtinen et al. 2018), which uses pairs of noisy images instead of noisy-clean image pairs to learn the pattern of noise in images and denoise them. Yet another way to approach denoising images without clean images available, is through the availability of a prior. One such work is the use of denoising autoencoders as priors for denoising (Bigdeli and Zwicker 2017). In this work, the authors use the output of a denoising autoencoder to construct the negative log likelihood of an image. In (Bigdeli et al. 2017), the authors follow a Bayesian approach to image denoising, but adopt a Gaussian kernel as the substitute for the natural image distribution.

Noise models can act as priors for training denoising models too, as demonstrated in (Tran et al. 2020) where a Generative Adversarial Network is used to learn the noise model in real-world noisy images to then train a denoising network. In (Ulyanov et al. 2020), the authors train a randomly initialized generator network on a corrupted image and use the network weights as a prior to various image tasks including denoising. In (Liu et al. 2021), the authors use an invertible neural network for denoising images. An invertible network can learn the distributions of the output and input and be able to transform them, which is the case when denoising an image. But in such cases where priors are used, there's no mention of explicitly estimating the density of the natural image distribution.

Autoregressive models are models which were used to predict variables that are time-varying in nature. It works on the simple principle that the present value of the output variable is dependent on the past values of the same variable linearly. As such, this class of predictive models were heavily used in time series forecasting and prediction scenarios (Harrison et al. 2003).

PixelRNN (Oord et al. 2016) was one of the first autoregressive models that employed the usage of LSTM layers to model the pixel intensity distribution of an image. It used LSTM layers to model the image along each row and diagonal. This is computed across each dimension for RGB images. In case of multi-dimensional images, the pixel intensity distribution generated for a pixel on a dimension depends on the pixel intensity distributions generated thus far in all the other dimensions as well. But using LSTMs to estimate density of the images is computationally expensive as an image can be very high dimensional and computing one pixel at a time sequentially takes longer in training and inference. Thus, PixelCNN was also introduced in the same research as a less computationally expensive version which employed convolutions to parallelize the process. PixelRNN achieved 3.00 bits/dim (lower is better) on CIFAR-10 dataset, while PixelCNN achieved 3.14 bits/dim.

(Van Den et al. 2016) then introduced Gated PixelCNN, an improved version of PixelCNN which mended the blind spot in its receptive field and introduced gated convolutional layers, similar to LSTMs. It achieved 3.03 bits/dim on CIFAR-10, a better score than PixelCNN and very close to PixelRNN while still using convolutions for speed.

PixelCNN++ was introduced by (Salimans et al. 2017) bringing a number of improvements to the original model. It introduced, among other modifications, discretized logistic mixture likelihood instead of Softmax, which they found to be more efficient. PixelCNN++ achieved 2.92 bits/dim on the CIFAR-10 dataset. The latest version in the PixelCNN family is the PixelSNAIL (Chen et al. 2017) which introduces self-attention mechanism into the mix, achieving 2.85 bits/dim on CIFAR-10.

Oftentimes with denoising neural networks, the distribution is only implicitly learnt in the network. It may be beneficial to leverage the explicit distribution outputs from autoregressive models. To our knowledge, the topic of Autoregressive Models, especially the PixelCNN family, as priors to image denoising has not had extensive research done on it.

Chapter 3: Background

3.1 Autoregressive Models

As mentioned, autoregressive models were extensively used for time series analysis due to their sequential nature. But it was soon found out that, ingeniously, the same principle can be used to predict pixel intensity values in images as well. Thus was born a class of generative models that predicted the probability distribution of the intensities of each pixel in an image, and generated images by sampling from these distributions.

Autoregressive models are generative models which try to learn the explicit distribution of the provided data. In terms of images, they can learn the distribution of intensities in each pixel as the output. Given an image, they are trained in a pixel-by-pixel orderly manner from the top-left pixel to the bottom-right pixel. The order of pixels that the model sees is of particular significance, as it assumes that the intensity of a given pixel only depends on the intensities of the pixels that came before it.

$$p(Image) = \prod_{i=1}^{n} p(x_i | x_1, x_2, \ldots, x_n),$$

where x_i denotes the intensity of the i^{th}pixel.

PixelCNN was also introduced which utilized convolutional layers, or a variation of them, to estimate density of images. Since the intensity distribution of a pixel under consideration is modelled to be dependent only on the pixels that have already been computed, we make used of Masked Convolutions to enforce this rule. Basically, all the pixels that are not “allowed to be seen” by a pixel under consideration are turned off or zeroed and only the predecessors are considered for computation.

This was much less computationally expensive as compared to PixelRNNs as convolutions are easier to parallelize. But they also had the inherent trade off, as the receptive field of a PixelCNN is based on the kernel size and number of convolutional layers in the model, while PixelRNN theoretically had access to the entire image as its receptive field. But this can be solved by including more parameters as required by the task at hand.

But, as shown by (Van Den et al. 2016), the original PixelCNN model made up of stacked masked convolutional layers showed a blind spot towards the top right side of the receptive field.

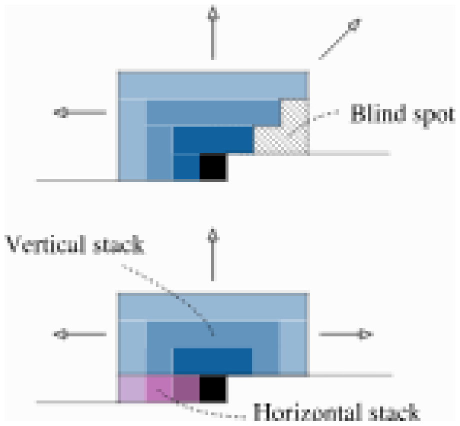


Figure 2 (Van Den et al. 2016)

As shown in Fig 2, the top image shows the stack of masked convolutions having a blind spot as mentioned. Instead, a gated form of PixelCNN was used where separate horizontal and vertical masked convolutions stacks were applied on the image. The horizontal stack covers the current row until the pixel in consideration, and the vertical stack covers all the rows that have been covered so far. This has been shown to mitigate the blind spot issue that occurred in PixelCNN. This study also employs the same mechanism for stated reasons.

It involves a stack of gated masked convolutional layers, which are a combination of stacks of vertical and horizontal masked layers as explained. Moreover, to bridge the gap between PixelRNN and PixelCNN, Gated PixelCNN uses the gated activation units similar to LSTMs which employ tanh and sigmoid activation functions instead of simple Rectified Linear Units (ReLU) as shown in Fig. 3. This allows for handling of more complex information than what could be possible with just ReLU connected layers.

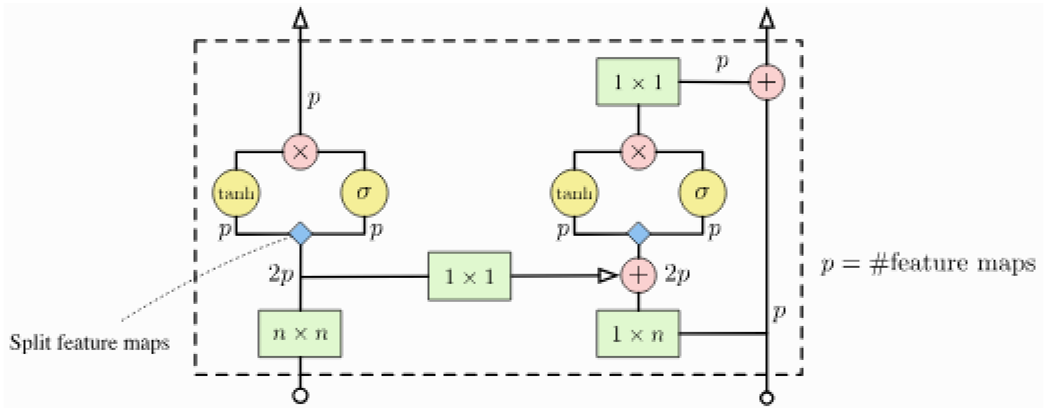


Figure 3 Gates in Gated PixelCNN (Van Den et al. 2016)

The output of a forward pass through the network are the probability distributions of pixel intensities for each pixel in the image. Given an $N \times N$ image is fed through the network, the output would be $N \times N \times 256$, with a 256-length vector for each pixel representing the probabilities for 256 intensity values.

3.2 Autoregressive Model Evaluation

The autoregressive model's success can be quantified by using the Bits Per Dimension loss metric (Oord et al. 2016). The Bits Per Dimension (BPD) loss, or Bits Per Pixel loss, is derived from the negative log likelihood we calculate from the generated output. It's primarily used for generative models and represents the average number of bits needed to represent each pixel in the generated image. It's a measure to quantify the difference between the predicted pixel distribution and the original pixel intensities of an image. It is calculated by converting the log likelihood from natural log to log of base 2, and then taking the mean across all pixels.

$$BPD \; = \frac{-1}{N}\sum_{i=1}^{N}\log_2 p(x_i)$$

Here, N represents the total number of pixels in the image, and $p(x_i)$ is the log likelihood.

3.3 Denoising Module

The denoising module involves a very simple mechanism of shifting pixel intensity values based on the inputs given. As mentioned earlier, this algorithm substitutes the need for ground truth pairs with the prior provided by the autoregressive model.

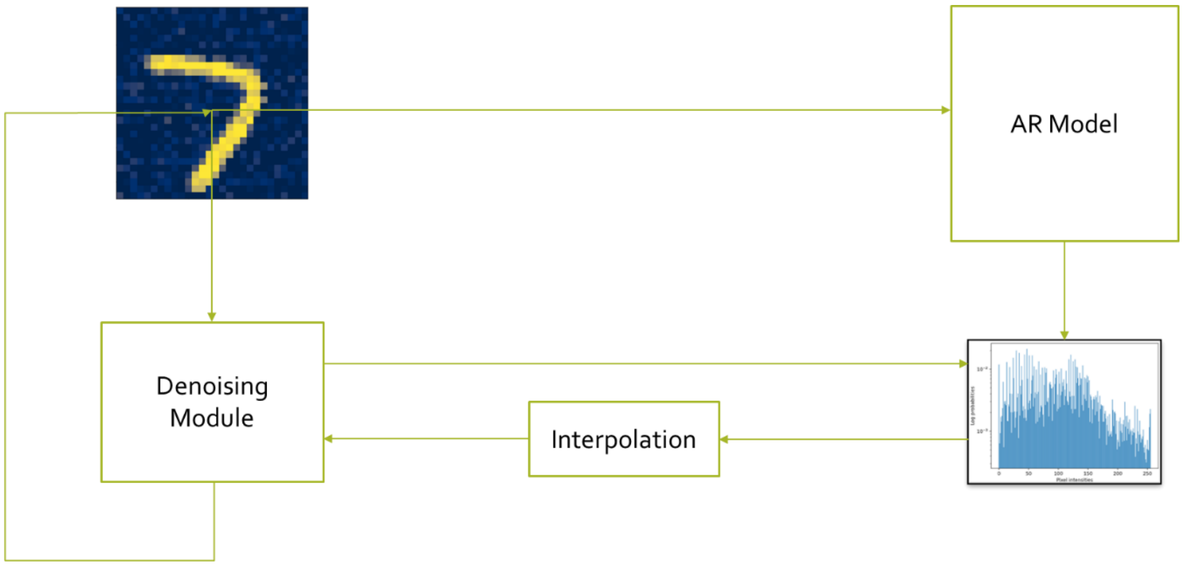


Figure 4: Model Overview

The denoising happens through the following steps:

- A clone of the image is taken and is labelled the "current estimate", and the pixel values in this current estimate will get updated with each iteration.

- In each iteration, the current estimate is fed through the autoregressive model, which gives the likelihood distributions for each pixel in the image.
- A loss function that makes use of both the prior and the noisy image inputs to drive the pixel values in the current estimate towards what a clean image could be.

3.4 The Loss Function

The goal of the model is to shift pixel values to maintain the structure of the noisy image but remove the noise by using the prior knowledge of what a clean image looks like.

If X represents the noisy image and Y represents a possible clean image, by Bayes' theorem:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Here,

$P(X|Y)$ – represents the noise model present in the noisy input.

$P(Y)$ – represents the probability distribution of possible clean images, which is exactly the prior knowledge obtained from the autoregressive model.

$P(X)$ – represents the probability distribution of the noisy image.

In this study, Gaussian noise is used and thus the noise model can be expressed through this equation:

$$P(X|Y) \ = \ \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

The conditional $P(Y|X)$ is desired to be maximized, and this results in the maximization of the terms on the right-hand side of the equation. During maximization, $P(X)$ does not change or affect anything and is thus disregarded. Since logarithm is a monotonically increasing function, the maximization operation does not change by taking natural log on both sides,

$$\ln P(Y|X) = \ln P(X|Y) + \ \ln P(Y)$$

Taking the log of $P(X|Y)$ gives us a term proportional to the residual error squared, while the $\ln P(Y)$ represents the log likelihood derived from the prior. With this in mind, the loss was the chosen to be a linear combination of Mean Squared Error (MSE) loss and Negative Log Likelihood (NLL) loss.

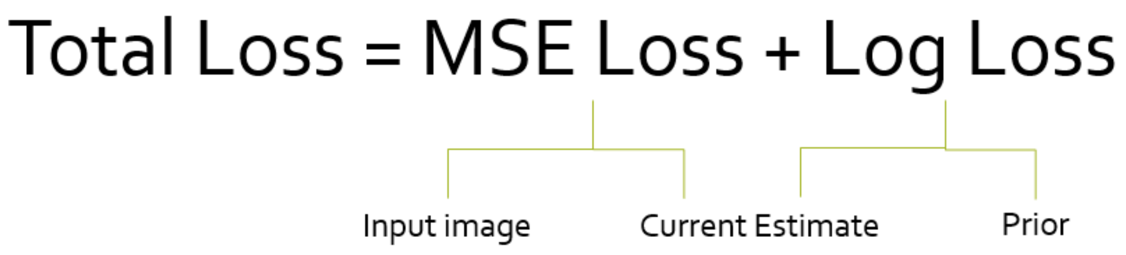


Figure 5: The Loss function

Here, the MSE loss between the input image and the current estimate makes sure that the output still takes after the input and is not over-influenced by the prior. The Log Loss component pushes the current estimate to move towards a cleaner image. Theoretically speaking, this loss should work fine as it is. But in practice, it was found that having the MSE Loss weighed down by a factor led to better and quicker results.

3.5 Summary

In this section, a detailed explanation about PixelCNN and the denoising module were given, along with the loss function to be used.

Chapter 4: Project Setup

All the experiments were run on Google Colab virtual machines. The training of PixelCNN was done on the T4 GPU virtual machine, while the denoising is fine to run on CPU.

4.1 Dataset

The dataset used was, by the initial assumptions, the MNIST dataset of digits. Specifically, only a single class was chosen amongst the ten classes of MNIST digits. These are grayscale images of the digit 7 written in various ways, all of size 28×28. The pixel intensity values range from 0 (black) to 255 (white).

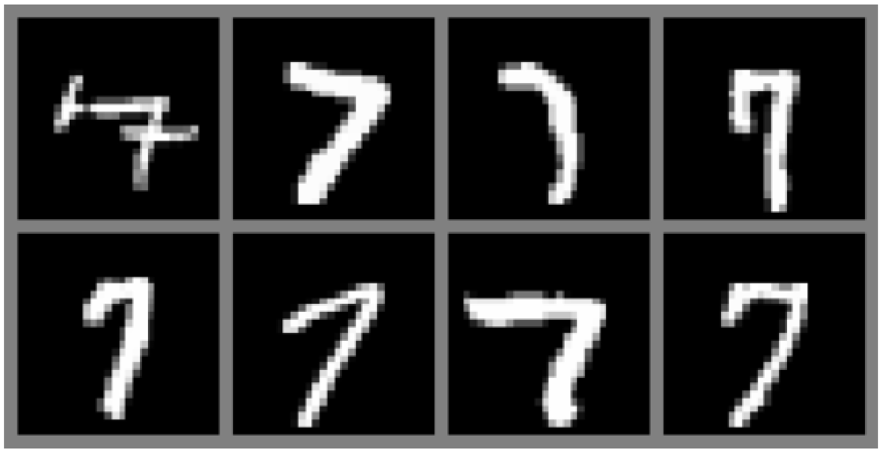


Figure 6: MNIST dataset samples

The initial dataset contains 6265 training images and 1028 test images. The training images were further split in an 80-20 fashion into the training set, containing 5012 images, and the test set, containing 1253 images. The only preprocessing step performed was discretization of the images, as the original dataset contained images of pixel values ranging 0-1, which was then converted to 0-255. Fig. 6 shows a sample of the data used to train the PixelCNN model.

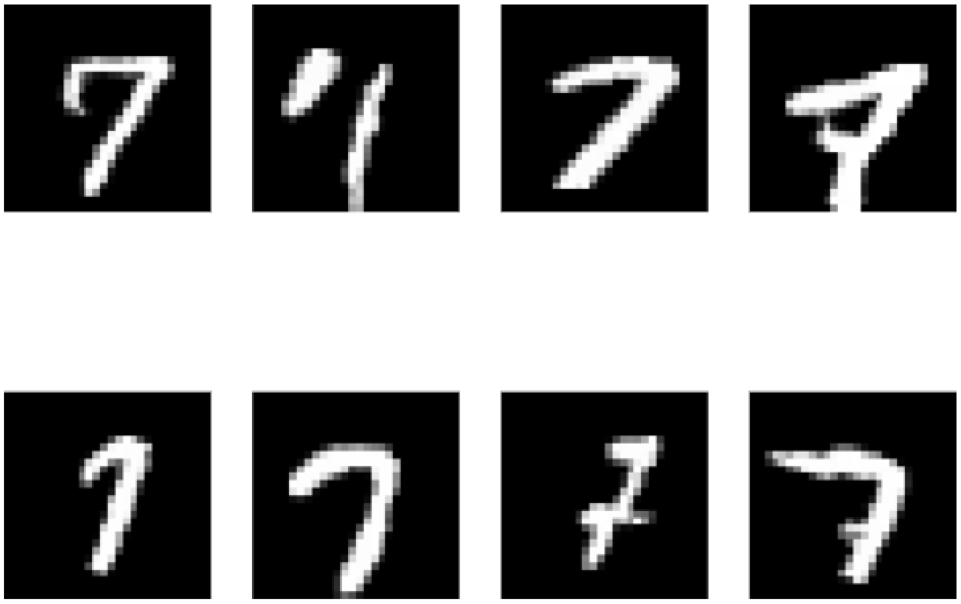


Figure 7: PixelCNN output

4.2 The PixelCNN Model

The model being used for this study (Lippe 2022) is a simple version of the Gated PixelCNN model, chosen for testing and experimentation convenience. With training, better models can be substituted easily as it does not change the actual denoising algorithm.

Fig. 7 showcases samples generated by the model after training for ~140 epochs with early stopping enabled to stop the training when a minimum threshold change of 0.005 in the validation loss isn't met within 15 continuous epochs. Adam optimizer was used, with a learning rate of 1e-3. Learning rate decay was also used, with the decay factor set as 0.99. It achieved a training loss of 0.857 BPD and a test loss of 0.855 BPD, consistent with each other. Stricter restrictions yielded slightly subpar models.

A random image was chosen from the training dataset, and a forward pass through the model was done using this image as input. Extracting the predicted pixel distribution for a random pixel in the centre of the image gives the following histogram:

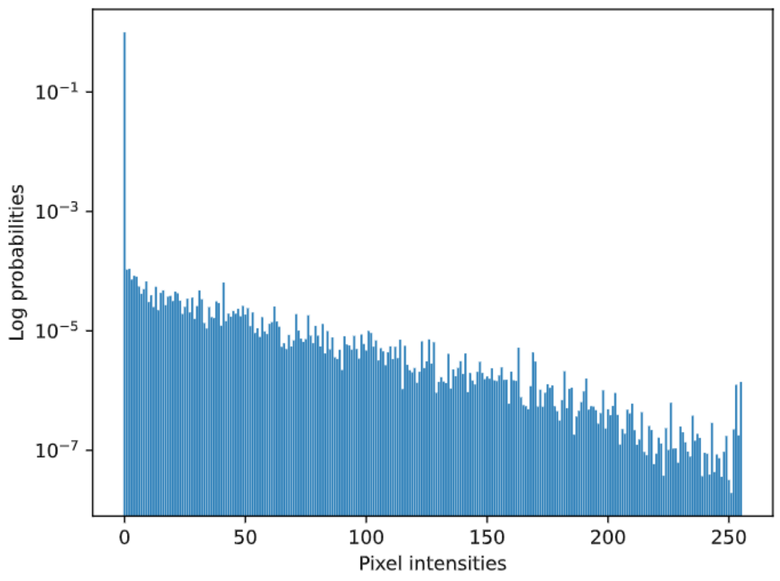


Figure 8: Intensity distribution of random pixel

Another interesting observation to be made is the pixel intensity distribution for the top left pixel, which is predominantly just background. This would imply that the pixel distribution generated by the model should be centred predominantly around an intensity value of 0. The following histogram agrees with this hypothesis.

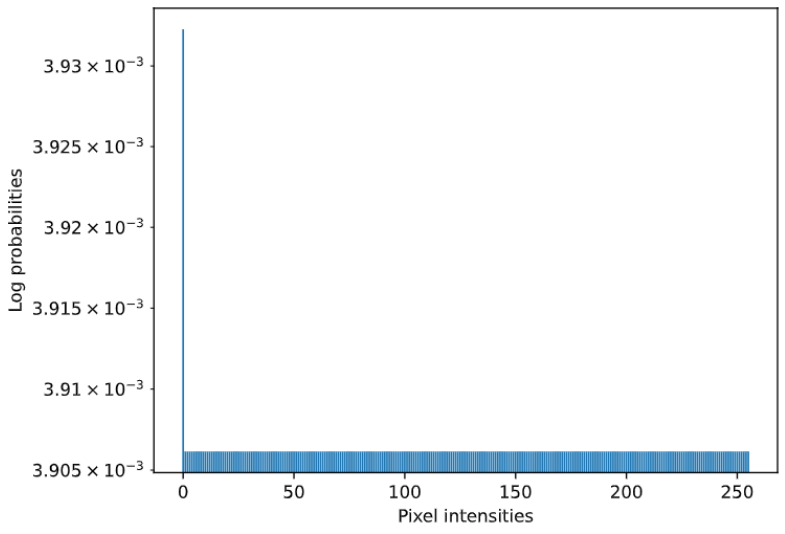


Figure 9: Intensity distribution of first pixel

4.3 Computing Log Likelihood

Likelihood here is a measure of confidence the model has on the image given to it, that the image is real and is known to the model. Considering that PixelCNN gives probability distributions as output, it becomes quite easy to calculate the log likelihood of an image. Given that the main principle of an Autoregressive model can be written as:

$$p(X) = \prod_{i=1}^{n} p(x_i|x_1, x_2, \ldots, x_n),$$

where x_i denotes the intensity of the i^{th}pixel, and p(X) denotes the joint distribution over all pixels.

$$-\log p(X) = -\sum_{i=1}^{n} \log\left(p(x_i|x_1, x_2, \ldots, x_n)\right)$$

The negative log likelihood is a well-known loss function that is quite popularly used.

The probability of the intensity of each pixel is taken and added up, and a mean may be taken to get the average or mean likelihood. This is of extreme importance in this scenario as, going by hypothesis, the model should have greater confidence on the images that it has been trained on, and lesser confidence on their noisier versions. This makes way for the autoregressive model to serve as a prior for the image denoising process.

As a general test, random images were chosen from the test dataset that the model hasn't seen before, and a noisier version of these images were taken by introducing zero mean gaussian noise with varied standard deviation values. Both pairs of images were fed to the model and the log loss was computed. The lesser the loss is, the more confidence the model has on the image.

Standard Deviation (σ)	Clean	Noisy
10	0.1625	1.0499
20	0.1694	0.9638
20	0.1446	1.0468
30	0.1934	1.0779

The above table shows the negative log likelihood values obtained, and it confirms our hypothesis. Thus, the Autoregressive model can serve as a good prior for our task.

But the priors cannot be used as is for optimization as the output distribution is discrete, with likelihoods for each of the 256 intensity values. But the denoising optimization process may require values in-between the 256 discrete values. To accommodate for this, linear interpolation is used. To make sure that the values obtained via interpolation does not exceed the intensity level bounds (0-255), gradient clipping is also used.

4.4 Evaluation Metrics

Evaluating the denoising performance is an important part of the pipeline. As mentioned earlier, we wish to evaluate that the denoiser:

i) Reduces noise in the resulting image.
ii) Retains the structure and properties of the original image.

Peak Signal to Noise Ratio (PSNR) and Structural Similarity Index (SSIM) are the chosen metrics for this study. They have both been extensively studied and are popularly used for image denoising(Horé and Ziou 2010; Fan et al. 2019). PSNR quantitatively measures the ratio of signal to noise in an image, and a higher PSNR indicates a cleaner image. It is derived from the Mean Square Error difference between the ground truth and the denoised image.

$$PSNR \; = \; 10.\log_{10}(\frac{I^2}{MSE}) \; dB$$

Where **I** denotes the maximum intensity value a pixel can take. In this case, it is 255. While PSNR by itself is a good measure for denoising, it does not capture perceptual quality in images.

SSIM (Wang et al. 2004) measures the structural similarity between two images. It takes into account similarity in texture, luminance, contrast and structure. It is a value between -1 and 1, and a higher value signifies better similarity.

$$\mathrm{SSIM}(\mathbf{x},\mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)}$$

Figure 10: (Wang et al. 2004)

For two images **x** and **y**, SSIM accounts for the means, variances and covariances of the two images. $C_1 \; and \; C_2$ are constants.

4.5 Summary

In this section, the project setup, and the loss metrics to be used for denoising were explained.

Chapter 5: Experiments and Results

5.1 Overview

The hyperparameters of interest to us are:

i) The standard deviation, sigma, of the noise applied to the images.
ii) The weight applied to the MSE loss, σ_w.
iii) The initial state of the current estimate.
iv) Step size of the denoising algorithm

This study focuses more on these factors and provides experimental results for each of them. It is important to note that the images used in these experiments are all from the test set that the autoregressive model has not seen yet, to make it fair.

5.2 Why do we require σ_w?

Let σ_w = 1.0, and we apply Gaussian noise of σ = 50 to images before attempting denoising. Iterating for 2000 epochs with a step size of 20,

Initial State	**PSNR**	**SSIM**
Random noise	15.56	0.54
Blank image	16.23	0.56
Given noisy image	15.71	0.54

Thus, without any weight applied to the MSE loss, the progress in denoising appears to be insignificantly slow. This can be reasoned out that the algorithm places too much preference on the initial state of the current estimate and not on what a clean image should look like.

5.3 Case 1: Varying σ_w with constant σ

Keeping the noise level at σ = 50.0, various values of σ_w were tested. The input noisy image was kept as the initial state, and the step size was kept at 20. Three different images from the test set were selected for study. The below figure shows one of the images and the corresponding corrupted version of it.

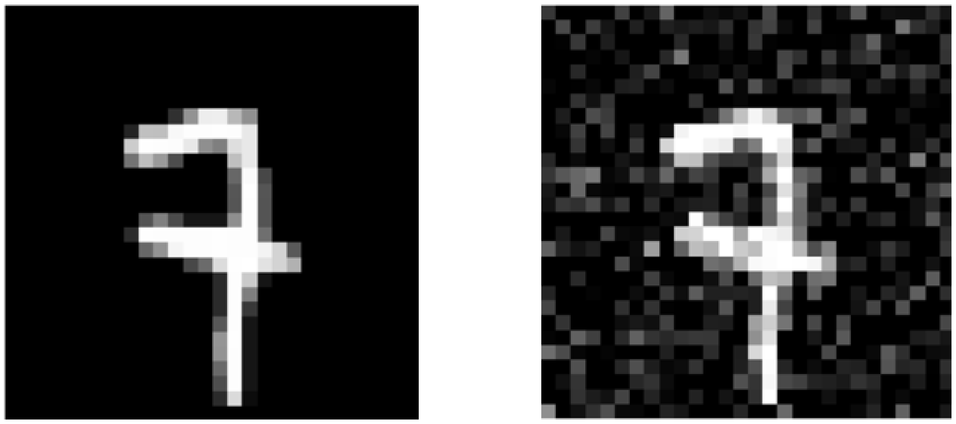


Figure 11: Case 1

Running the algorithm for 2000 iterations, the following results were observed.

σ_w	Image no.	PSNR	SSIM
	1	16.1	0.56
2.0	2	16.05	0.68
	3	16.4	0.48
	1	21.8	0.8
25.0	2	19.6	0.8
	3	20.22	0.7
	1	17.5	0.7
50.0	2	17.6	0.77
	3	19.5	0.67

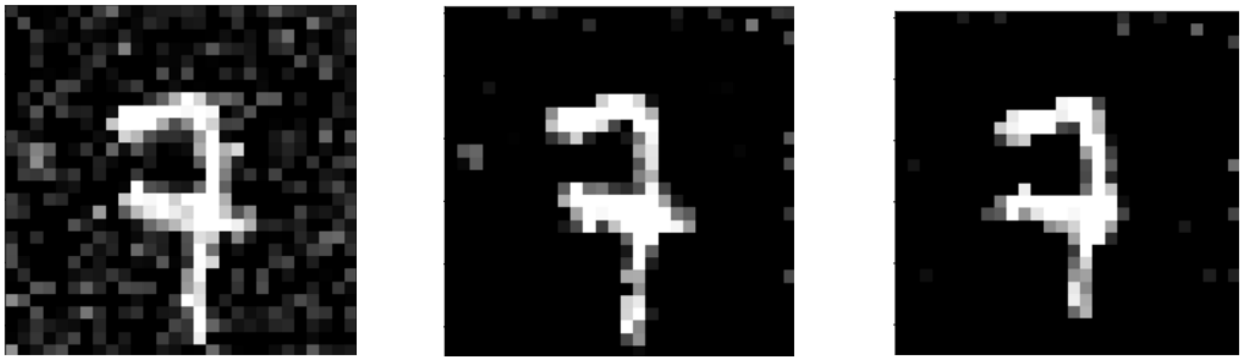


Figure 12: Case 1 results (Left to right: σ_w = 2, 25 & 50)

The above figure shows the results of the denoising for each σ_w value, respectively. A low value results in limited denoising capabilities, while a very high value results in the original image getting deformed. A value around 25.0-

30.0 is found to serve the best. But this, as can be seen in the following sections, also depends on the amount of noise present in the image.

5.4 Case 2: Varying σ with constant σ_w

Setting a value of 30.0 to σ_w, values of σ = 30, 50, and 75 are tested. The same conditions of input image as the initial state, step size of 20, and 2000 iterations are maintained. The same 3 images are tested upon, with a sample image and the corrupted versions shown below.

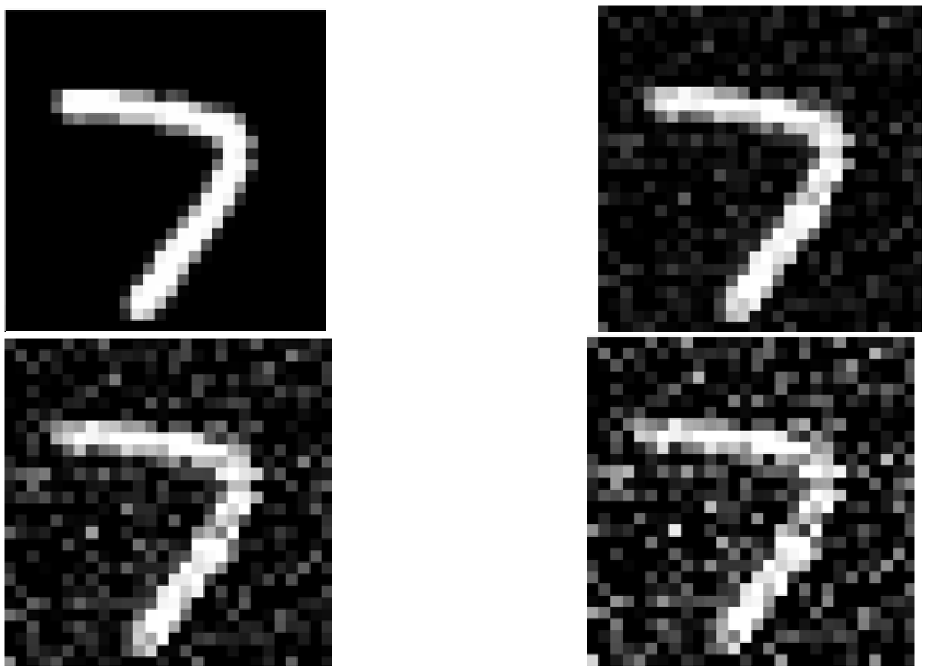


Figure 13: Case 2 (From top left to bottom right: σ = 0, 30, 50 & 75)

σ	Image no.	PSNR	SSIM
	1	23.4	0.79
30.0	2	24.38	0.86
	3	23.69	0.64
	1	20.27	0.8
50.0	2	19.55	0.83
	3	20.75	0.75
	1	17.11	0.74
75.0	2	15.8	0.73
	3	17.05	0.68

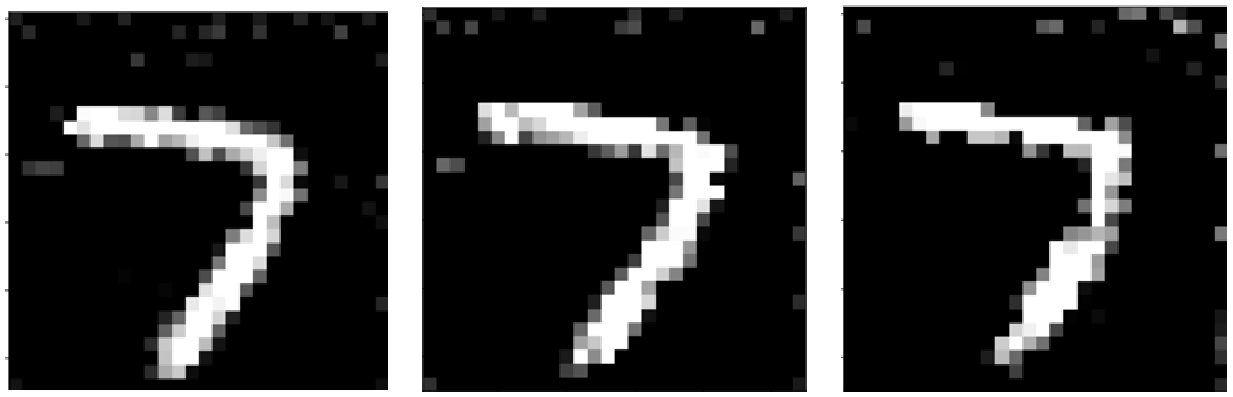


Figure 14: Case 2 results (Left to right: σ = 30, 50, 75)

We can see that as the noise level increases, the effective denoising decreases. But the model still manages to provide decent output till noise levels of 50.0 but falters afterwards. It should also be noted that the output looks slightly deformed at high noise levels.

5.5 Case 3: Different initial states

It is important to investigate if the initial state of the current estimate influences the denoising capabilities of the model.

In this case, σ_w is set at 30.0, and σ = 50 noise is used. Rest of the conditions remain the same, 2000 iterations with a step size of 20.

The following image shows one of the samples used, and the noisy version:

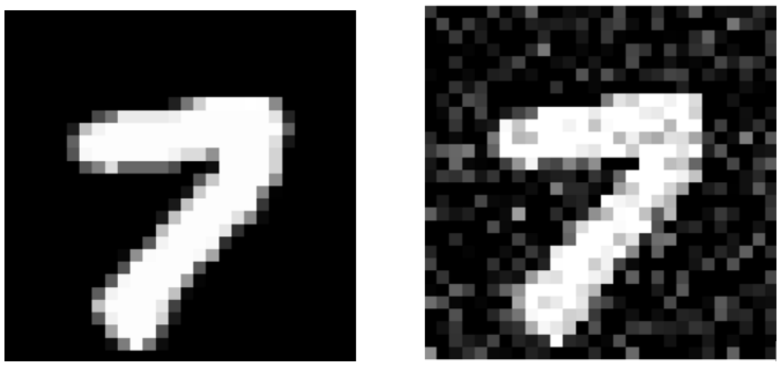


Figure 15: Case 3

Initial State	Image no.	PSNR	SSIM
	1	19.08	0.71
Random noise	2	19.26	0.84
	3	21.38	0.7
	1	19.6	0.8
Blank image	2	19.54	0.8
	3	20.76	0.66
	1	20.28	0.8
Noisy input	2	19.55	0.83
	3	20.75	0.75

From these initial tests, we can see that the initial state does not seem to have much of an impact on the effectiveness of denoising.

Below figure shows the results of this experiment on the sample image.

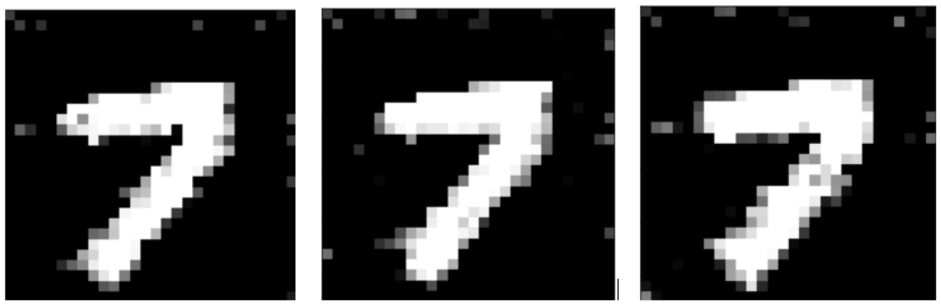


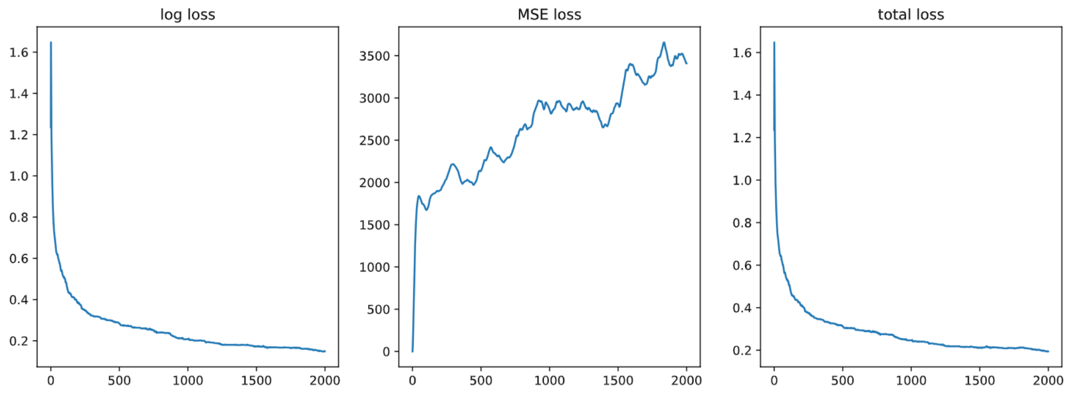
Figure 16: Case 3 output (Left to right for each state)

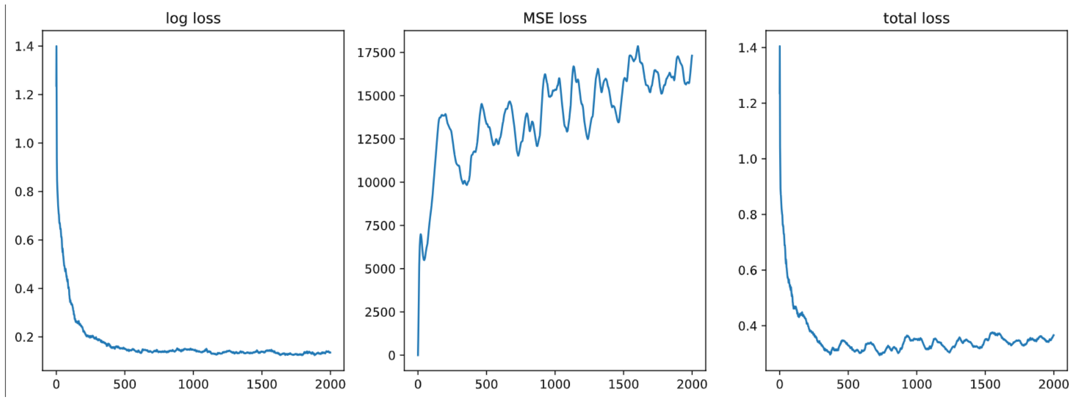
5.6 What about the step size?

So far, the step size was set at a default value of 20. It is necessary to compare the effect of different values of step sizes, against different levels of noise. For this study, a random image was selected from the test set and was corrupted with noise of different standard deviations. The input image was taken as the initial state, with a σ_w of 30 and the denoising algorithm was run for 2000 iterations consistently. The following table summarizes the results obtained from this experiment.

σ	Step Size	PSNR	SSIM
	5	25.67	0.82
30.0	20	24.4	0.68
	40	22.0	0.57
	5	20.4	0.66
50.0	20	20.06	0.63
	40	19.16	0.59
	5	16.97	0.5
75.0	20	16.82	0.61
	40	15.55	0.47

It is also worth looking into how smooth the convergence is in each case. Plots of the losses for each step size (for σ = 50.0) is given below.





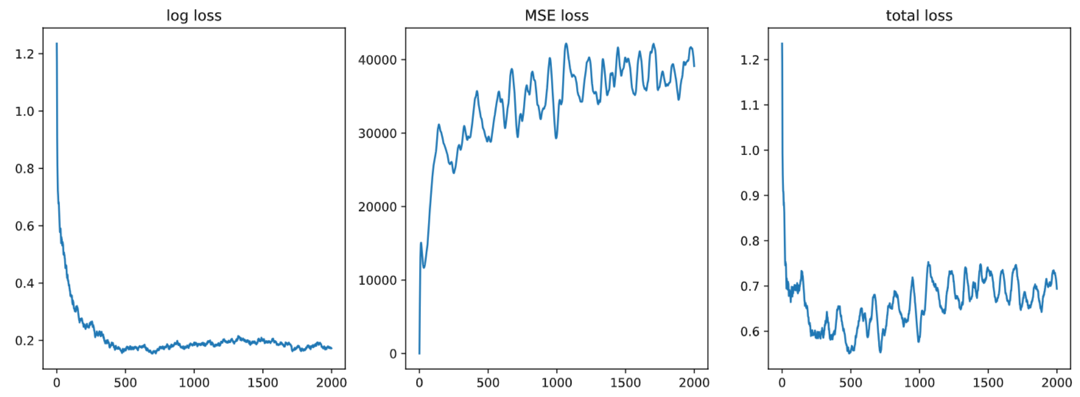


Figure 17: Convergence plots for step sizes 5, 20 and 40 (top to bottom)

Following the trend and setting a very low step size, however, does not yield good results. For σ = 50.0, and the same conditions as mentioned, setting the step size to 0.1 yielded the following results: PSNR of 17.52 and a SSIM of 0.5. The graphs in fig. below show the convergence plots. While the results begin to converge, they do so at a much slower pace that is practically not feasible and unnecessary.

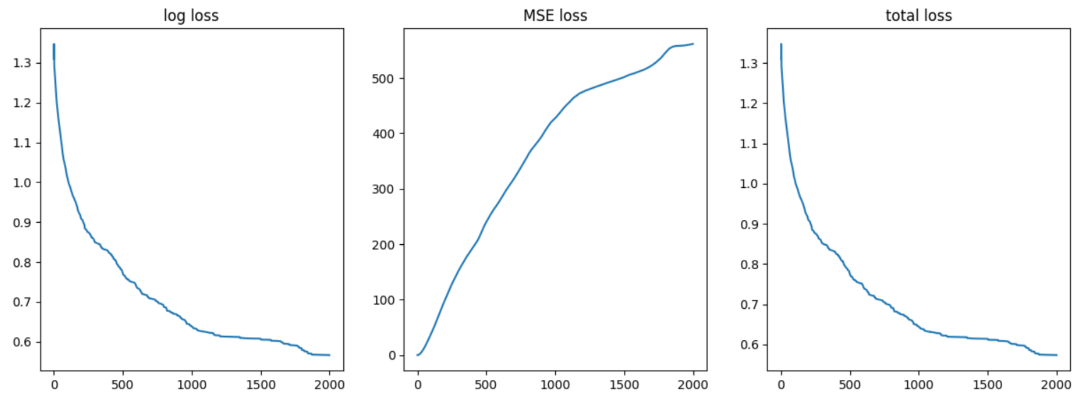


Figure 18: Convergence plot for step size of 0.1

5.7 What happens if we just use Log Loss?

Given how necessary weighing down MSE loss is, a reasonable curiosity could arise as to what would happen if just the Log Loss was used. It is to be established that the role of MSE loss is to make sure that the denoised output has similar structure to the input and supplements the prior as much as the prior supplements it.

At σ = 50.0, and 2000 iterations with just the log loss in the loss equation, we get the following output.

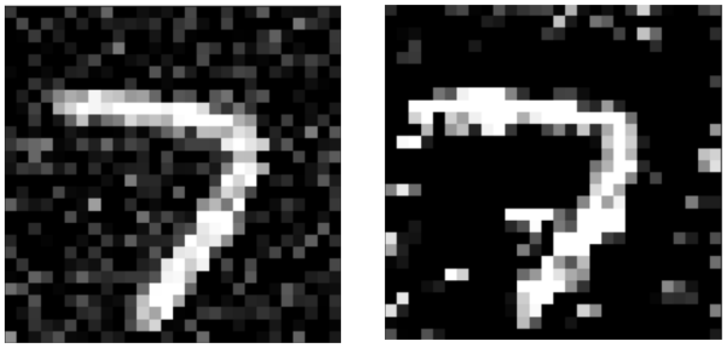


Figure 19: Input on left, output on right when discarding MSE Loss

Needless to say, this is not a successful result and shows the necessity of the MSE loss in the loss equation.

5.8 Summary

In this section, the hyperparameters of interest are laid out and multiple experiments were conducted to monitor the effects and significance of these hyperparameters on the denoising model and output.

Chapter 6: Discussion and Conclusion

6.1 Discussion

The algorithm manages to perform sufficiently well for initial testing and results, but areas of improvement are present and need to be discussed. From the above experiments, certain key points can be noted:

- Weighing down the MSE loss is necessary to achieve convergence faster. This gives more favour to the prior, pushing the algorithm to denoise the image.
- But too high of a weight on the MSE loss leads to loss of data during denoising. This is to be expected, as the model should also be able to retain original structure and other properties of the image.
- It was also established that denoising cannot happen without the MSE loss either. It acts as the guiding hand for denoising to make sure that the resulting image does not lose its original properties.
- The initial state of the current estimate does not seem to influence the results to a large extent. Since the majority of pixels in the MNIST images are black, having a blank image as the initial state could have been hypothesized to be a better choice. But the results are seen to be consistent throughout the cases. But while this is the case, it should also be noted that these are only initial tests, and that the MNIST images are quite simple in nature, popular as they may be. More complicated images could yield different results, but it should also be stated that it depends on how good the prior is.
- The step size matters a lot as to how smooth the convergence is and is directly related to how better the results are. Smoother optimization leads to better results in each case explored. It should also be noted that convergence can be reached much sooner than 2000 iterations provided the step size is appropriate. This is quite significant information, as each iteration takes 0.2-0.3s on average.

At noise levels of standard deviation $\sigma = 30$, the algorithm manages to perform quite well and achieve PSNR values of around 25.6 and SSIM values of 0.8. This is quite encouraging and proves that this could be a viable denoising method to be explored further.

In all cases, the denoising algorithm could not perform well in conditions of $\sigma \geq 50.0$. One improvement that can be made is to increase the kernel size of the convolutions to increase the receptive field of the network. A reasonable argument could also be made that this is because the autoregressive model is a simpler version of the original PixelCNN, and that better versions are available to choose from. With these models, standard image denoising datasets like SIDD (Smartphone Image Denoising Dataset) can be used for evaluation and comparison. Since the focus of this work is to obtain proof-of-concept results that test and show the viability of this method, this option remains to be explored in the future.

6.2 Future Work

While proof-of-concept results were achieved, it leaves a lot more experimentation to be desired that require more time and computational resources. The following points outline ideas that can be used to carry development of this algorithm forward in the future:

- Denoising for multiple classes is to be done. PixelCNN is a powerful model that can learn distributions of multiple classes at once, and training the prior well should lead to good denoising results.
- Testing on images of multiple dimensions. This again requires the autoregressive model to be trained on such images and is another natural extension to be tested.
- While Gaussian noise exists as a popular test bench for denoisers, real noise is much more difficult to remove from images and require more complex noise models in place. This is out of scope for this timeline.
- The autoregressive model chosen for this study was simple for convenience and experimentation purposes as it was not the main focus of this study, but the same could be extended to more powerful models that could provide better results. More layers could also be added to the same network to increase the number of parameters of the model.

6.3 Conclusion

The project successfully completed the main objectives it sought out to complete. The initial assumptions and constraints were consistently followed and helped make the project conceivable. An autoregressive model was trained on a single-class dataset of grayscale images and was successfully used as a prior for denoising said images. Moreover, extensive testing was done for various cases, and inferences were made that helped develop the algorithm further.

References

Ahn, B. and Cho, N.I. 2017. Block-Matching Convolutional Neural Network for Image Denoising. Available at: http://arxiv.org/abs/1704.00524 [Accessed: 7 June 2023].

Bigdeli, S.A., Jin, M., Favaro, P. and Zwicker, M. 2017. Deep Mean-Shift Priors for Image Restoration. Available at: https://github.com/siavashbigdeli/DMSP. [Accessed: 16 September 2023].

Bigdeli, S.A. and Zwicker, M. 2017. Image Restoration using Autoencoding Priors. *VISIGRAPP 2018 - Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications* 5, pp. 33–44. Available at: https://arxiv.org/abs/1703.09964v1 [Accessed: 18 June 2023].

Buades, A., Coll, B. and Morel, J.-M. 2011. Non-Local Means Denoising. *Image Processing On Line* 1, pp. 208–212. Available at: https://www.ipol.im/pub/art/2011/bcm_nlm/ [Accessed: 16 September 2023].

Chatterjee, P. and Milanfar, P. 2010. Is denoising dead? *IEEE Transactions on Image Processing* 19(4), pp. 895–911. doi: 10.1109/TIP.2009.2037087.

Chen, X., Mishra, N., Rohaninejad, M. and Abbeel, P. 2017. PixelSNAIL: An Improved Autoregressive Generative Model. *35th International Conference on Machine Learning, ICML 2018* 2, pp. 1364–1372. Available at: https://arxiv.org/abs/1712.09763v1 [Accessed: 23 June 2023].

Dabov, K., Foi, A., Katkovnik, V., Egiazarian, K. and Member, S. 2007. Image denoising by sparse 3D transform-domain collaborative eltering. *IEEE TRANSACTIONS ON IMAGE PROCESSING* 16(8).

Van Den, A. et al. 2016. Conditional Image Generation with PixelCNN Decoders. *Advances in Neural Information Processing Systems* 29.

Fan, L., Zhang, F., Fan, H. and Zhang, C. 2019. Brief review of image denoising techniques. *Visual Computing for Industry, Biomedicine, and Art* 2(1), pp. 1–12. Available at: https://vciba.springeropen.com/articles/10.1186/s42492-019-0016-7 [Accessed: 2 June 2023].

Goyal, B., Dogra, A., Agrawal, S., Sohi, B.S. and Sharma, A. 2020. Image denoising review: From classical to state-of-the-art approaches. *Information Fusion* 55, pp. 220–244. doi: 10.1016/J.INFFUS.2019.09.003.

Harrison, L., Penny, W.D. and Friston, K. 2003. Multivariate autoregressive modeling of fMRI time series. *NeuroImage* 19(4), pp. 1477–1491. doi: 10.1016/S1053-8119(03)00160-5.

Horé, A. and Ziou, D. 2010. Image quality metrics: PSNR vs. SSIM. *Proceedings - International Conference on Pattern Recognition*, pp. 2366–2369. doi: 10.1109/ICPR.2010.579.

Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M. and Aila, T. 2018. Noise2Noise: Learning Image Restoration without Clean Data. *35th International Conference on Machine Learning, ICML 2018* 7, pp. 4620–4631. Available at: https://arxiv.org/abs/1803.04189v3 [Accessed: 15 June 2023].

Lippe, P. 2022. UvA Deep Learning Tutorials.

Liu, Y., Qin, Z., Anwar, S., Ji, P., Kim, D., Caldwell, S. and Gedeon, T. 2021. Invertible Denoising Network: A Light Solution for Real Noise Removal. pp. 13365–13374. Available at: https://github.com/Yang-Liu1082/InvDN.git. [Accessed: 15 June 2023].

McCann, M.T., Jin, K.H. and Unser, M. 2017. Convolutional neural networks for inverse problems in imaging: A review. *IEEE Signal Processing Magazine* 34(6), pp. 85–95. doi: 10.1109/MSP.2017.2739299.

Oord, A. van den, Kalchbrenner, N. and Kavukcuoglu, K. 2016. Pixel Recurrent Neural Networks. Available at: https://arxiv.org/abs/1601.06759v3 [Accessed: 17 September 2023].

Salimans, T., Karpathy, A., Chen, X. and Kingma, D.P. 2017. PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. Available at: https://arxiv.org/abs/1701.05517v1 [Accessed: 23 June 2023].

Tian, C., Fei, L., Zheng, W., Xu, Y., Zuo, W. and Lin, C.W. 2020. Deep learning on image denoising: An overview. *Neural Networks* 131, pp. 251–275. doi: 10.1016/J.NEUNET.2020.07.025.

Tran, L.D., Nguyen, S.M. and Arai, M. 2020. GAN-based Noise Model for Denoising Real Images.

Ulyanov, D., Vedaldi, A. and Lempitsky, · Victor. 2020. Deep Image Prior. *International Journal of Computer Vision* 128, pp. 1867–1888. Available at: https://doi.org/10.1007/s11263-020-01303-4 [Accessed: 16 September 2023].

Wang, Z., Bovik, A.C., Sheikh, H.R. and Simoncelli, E.P. 2004. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing* 13(4), pp. 600–612. doi: 10.1109/TIP.2003.819861.

Zhang, K., Zuo, W., Chen, Y., Meng, D. and Zhang, L. 2017. Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Transactions on Image Processing* 26(7), pp. 3142–3155. doi: 10.1109/TIP.2017.2662206.

Zhang, K., Zuo, W. and Zhang, L. 2018. FFDNet: Toward a fast and flexible solution for CNN-Based image denoising. *IEEE Transactions on Image Processing* 27(9), pp. 4608–4622. doi: 10.1109/TIP.2018.2839891.

Appendix

The source code for this project can be found at Student Projects 2022-23 / sxm1806 · GitLab (bham.ac.uk). Please make sure that the 'saved_model.pt' file is present in the same directory as the notebook when it is run.

Credits for the PixelCNN code go to (Lippe 2022) - Github repo - uvadlc_notebooks