Exploring Performance of Autoencoder Neural Networks in Image Denoising

Denis Lalaj (94683588)
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC, Canada
`ldeni@student.ubc.ca`

Abstract

A major challenge relating to data used for building machine and deep learning models is that the data almost always comes with missing values, outlier points or corrupted features. In particular when working with images of the simplest representation built on one channel pixels, it is common to see strong variation in the intensity which in the context of imaging is perceived as noise. We explore the ability of autoencoder neural networks in de-corrupting images under different architectures and evaluate their performance based on the latent representation shrinkage ratio and the depth of the network.

1 Introduction

1.1 Motivation: Noisy Data

Data collection is the primary step in approaching any Machine or Deep Learning oriented task. This can be achieved in different ways based on the type of data that is being collected. For imaging oriented tasks it is often performed via sensors or other signal collecting physical devices such are cameras, drones, or telescopes.

A major challenge with data collection is that it is susceptible to noise originating from instrumentation failure, lighting conditions, radiation, reflection and plenty of other phenomena [1]. These can easily cause unusual pixel measurements that appear randomly and disrupt the expected image. Throughout this report, we will refer to the term **noise** not in the context of outliers but to indicate stochastic variations of unusual pixel intensities present in image-like data.

1.2 Traditional Image Denoising

Traditional Signal Processing methods to eliminate noise and more generally signal noise have been around for a while now. As the type of noise present in images can often be categorized in some large families including but not restricted to salt-pepper noise, blur, and Gaussian noise, there exist common noise filtering techniques to revert such effects. Some of the most popular are mean filters, median filters, and Weiner filters [2]. The major downside of such denoising techniques is that they are preferred for their corresponding type of noise and might not perform as well otherwise.

1.3 Autoencoders

We shift out attention to the **autoencoder** architecture. An autoencoder in the context of artificial intelligence is a type of neural network whose goal is to learn how to reconstruct observation data [3]. It consists of an **encoder layer** which downsamples or compresses input data $\mathcal{I}$ to what is typically called the **latent space** $\mathcal{Z}$. Further the **decoder layer** upsamples or decompresses the latent representation $\mathcal{Z}$ to a reconstructed output $\mathcal{I}'$.

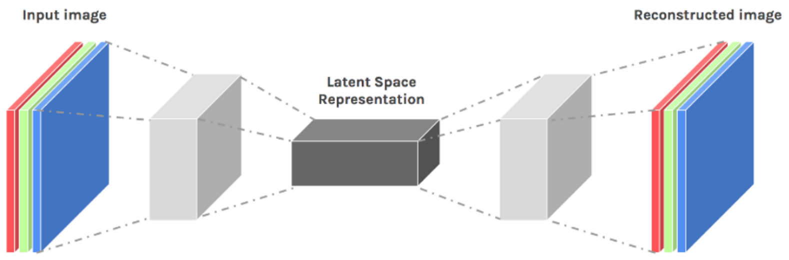


Figure 1: Simple Autoencoder Architecture

We let $\mathcal{E}$ and $\mathcal{D}$ be the encoder "function" and decoder "function" respectively. Then we establish the following three equations for input image $\mathcal{I}$, latent representation $\mathcal{Z}$, and output image $\mathcal{I}'$.

$$\mathcal{Z} = \mathcal{E}(\mathcal{I})$$
$$\mathcal{I}' = \mathcal{D}(\mathcal{Z})$$
$$\mathcal{I}' = \mathcal{D}(\mathcal{E}(\mathcal{I}))$$

A good autoencoder is one which minimizes a chosen **loss function** $\mathcal{L}$ measured as $\mathcal{L}(\mathcal{I}, \mathcal{I}')$ We discuss the choice of the loss function in later parts of this report.

There are two very useful observations that we highlight for autoencoders:

- They can provide strong **compression** capabilities via the latent space which is a compact high information multidimensional subspace $\mathcal{Z}$ [4].
- They can be adapted to be used for **denoising** purposes when they are equipped with a powerful loss metric and fed noisy input data [5].

1.4 Problem Definition: Autoencoders for Noise Removal

A **denoising autoencoder** can be considered as an extension to the abovementioned autoencoder where the input image has been **corrupted** under some noise function $\mathcal{C}$ to noisy input $\mathcal{N} = \mathcal{C}(\mathcal{I})$. The goal of a denoising autoencoder is then to reconstruct an output that approximates the original uncorrputed input $\mathcal{I}$ so that it **minimizes** the loss $\mathcal{L}(\mathcal{I}, \mathcal{D}(\mathcal{E}(\mathcal{N})))$.

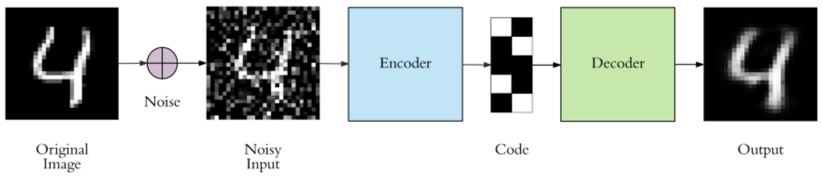


Figure 2: Denoising Autoencoder Pipeline

In this report we experiment with two aspects of building a denoising autoencoder:

- We analyze the effect of the **dimensionality** of the latent space $\mathcal{Z}$ as a factor of the compression of the image. 3 varying sizes of latent representation are tested to demonstrate the performance of the denoising capabilities of neural autoencoders.
- We investigate the **depth** of the encoder and decoder layer. Since the task of this type of denoiser is slightly more challenging that that of a regular autoencoder due to lossy reconstruction, we vary the count of hidden layers to analyze the performance of the network.

2 Dataset Choice and Preprocessing

2.1 MNIST Dataset

The choice of the dataset was restricted to the **MNIST** Handwritten Digits one for performance evaluation reasons. As building a strong denoiser for complex data would require strong background in neural network layers, the focus here instead was to consider the performance of a simpler denoiser for varying depths and compression levels of the input image. The dataset was split into training, validation, and testing data where training and validation data was used to generate "strong" denoisers and testing data was used to check the **performance** of these denoisers against unseen images.

2.2 Noisy MNIST

The MNIST Dataset by default is not susceptible to the noise we are interested in for this report. Therefore data was curated manually by implanting **artificial noise** via the standard normal distribution modelled as a noise function $\mathcal{C} = \mathcal{N}(0, 1)$. The main steps related to the data preprocesing stage are listed below.

- **Max-min** normalization in the range [0, 1]. This corresponds to $\frac{X_i - min(X)}{max(X) - min(X)}$.
- Noise was added according to the noise function discussed above. To achieve a reasonable amount of image **corruption** on the dataset, the added noise is scaled by a multiplicative factor that ensures it does not entirely dominate the original image: $\mathcal{X} + \alpha \cdot \mathcal{C}$, for uncorrupted input $\mathcal{X}$, noise $\mathcal{C}$ and multiplicative factor $\alpha = 0.5$.
- **Clipping** was further applied via Pytorch to squeeze all data in the [0, 1] range. This mitigates the presence of noise from the earlier step which can lead pixels in our images to cross the boundaries of our initial [0, 1] normalization range.

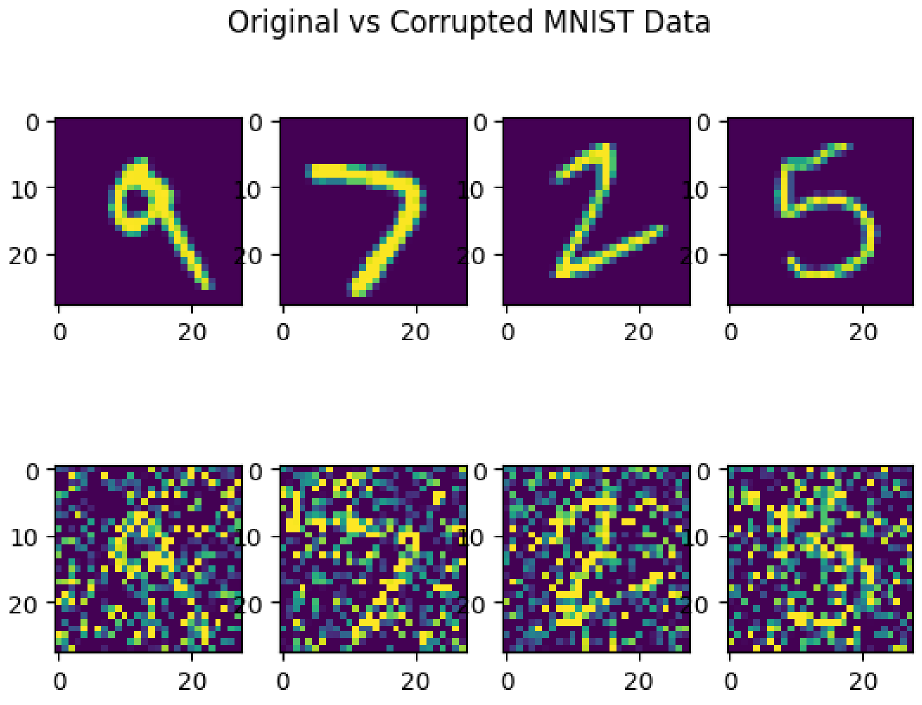


Figure 3: Data Corruption via Gaussian Noise

Figure 3 displays a few randomly sampled images before and after applying noise to the MNIST Dataset. The human eye can likely guess the original of these noisy images with a little bit of effort. The goal however is to allow a model to learn how to perform **lossy reconstruction** to approximate the ground truth images as accurately as possible by removing most of the noise.

3 Methodology

3.1 Loss Function

An important decision in building a neural network architecture is the choice of the loss function. Certain loss functions seem to be well suited for different types of learning problems and they mostly get categorized based on regression or classification like tasks [6]. We emphasize that the aim of a denoising autoencoder is to reconstruct the original image as accurately as possible by removing noise. Intuitively for input image $\mathcal{I}$ and reconstructed output $\mathcal{I}'$ we aim to minimize the $\mathcal{L}_2 = ||\mathcal{I}' - \mathcal{I}||_2$ norm.

As this is similar to the usual goal of regression tasks, the **Mean Squared Error** (MSE) loss function is chosen as shown in the case of a single image here. The MSE is averaged over each batch that runs through the autoencoder.

$$MSE = \sum_{i=1}^{D} \sum_{j=1}^{D} (\mathcal{I}_{ij} - \mathcal{I}'_{ij})^2, \text{ where } \mathcal{I}, \mathcal{I}' \in \mathcal{R}^{DxD}$$

Figure 4 below provides the baseline over which the MSE loss is applied. The original image $\mathcal{I}$ is considered a ground truth example whereas the reconstructed image $\mathcal{I}'$ is the output of the autoencoder which is produced from a corrupted version $\mathcal{N}$ of the original image. As such, the MSE loss function is not concerned whatsoever with the noisy input, it merely penalizes the autoencoder to incentivize learning how to remove noise optimally.

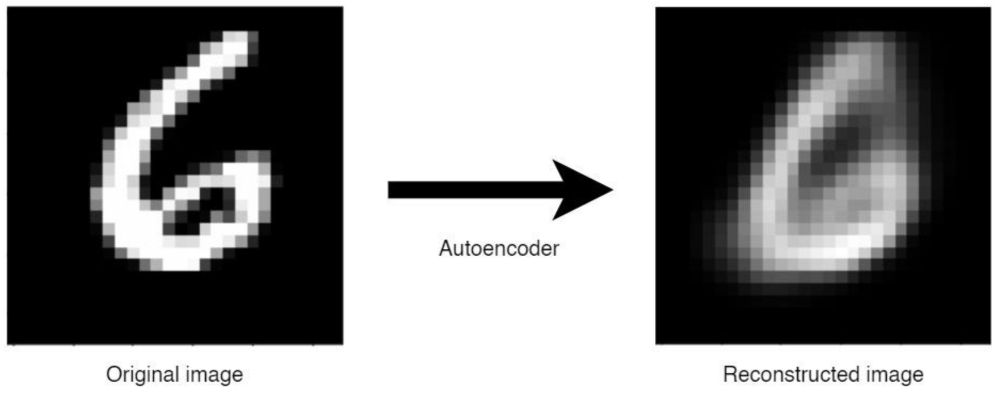


Figure 4: Ground Truth vs Reconstructed Image Example

3.2 Network Hyperparameters

We investigate denoising performance based a varying architecture of the autoencoder where the two **hyperparameters** of interest are the shrinkage ratio which results from the **dimensionality** of the latent space, and the network **depth** that is decided by the layers the network is built on.

- We consider varying the depth of the neural network at the heart of this denoiser. The proposed autoencoders are **symmetrical** with respect to the latent space.

  The encoder applies a sequence of **Convolutional** layers followed by ReLU activation layers to downsample the input to the latent space. The decoder applies **inverse Convolutional** layers (transpose convolutions) followed by ReLU functions to upsample The last ReLU layer is substituted for a Sigmoid one to squeeze the output into the (0, 1) range just as the input range to the autoencoder.

  The number of stacked layers for the encoder and decoder was varied between 1, 3 and 5 leading to architectures of 2, 6, and 10 hidden layers in total.
- The latent space maintains the lower dimensional high information representation of the network so it is a **bottleneck** of the autoencoder. The MNIST dataset consists of single-channel images of shape 28x28 which we denote in this report as (N_{chan}, H, W). In the case of MNIST this is $(1, 28, 28)$.

We explored the compression factor with respect to latent representation $\mathcal{Z}$ of the autoencoder architecture to evaluate its performance for small and large latent spaces. The **latent space shape** was varied between $(4, 13, 13), (4, 8, 8), (4, 5, 5)$.

3.3 Evaluation Metrics

The proposed baseline for evaluating our autoencoders was centered around the loss function mentioned on all of the training, validation, and testing data.

- **Training and validation** loss was collected over all training epochs for each proposed autoencoder architecture.
- **Testing loss** was reported as a single scalar for each proposed autoencoder architecture.

In Table 1 shown below, we present an overview of the architectures that were considered for this analysis.

We highlight the **compression rate** as an important factor which shows the ratio in total pixel count of the latent space compared to the original data. For this study we vary this to investigate how much compression an autoencoder can endure given that the compression is lossy - it does not retain all the original data information.

We also consider the **number of layers** present in our autoencoder as another evaluation metric of an autoencoder. We anticipate to see better performance for deeper networks compared to the more shallow ones.

Table 1: Proposed Autoencoder Architectures and Characteristics

Model Name	Latent Space Dim	Compression Rate	Convolutional Layers
DAE_5_13	(4, 13, 13)	86.22%	5 Conv2D + 5 ConvTranspose2d
DAE_5_8	(4, 8, 8)	32.65%	5 Conv2D + 5 ConvTranspose2d
DAE_5_5	(4, 5, 5)	12.76%	5 Conv2D + 5 ConvTranspose2d
DAE_3_13	(4, 13, 13)	86.22%	3 Conv2D + 3 ConvTranspose2d
DAE_3_8	(4, 8, 8)	32.65%	3 Conv2D + 3 ConvTranspose2d
DAE_3_5	(4, 5, 5)	12.76%	3 Conv2D + 3 ConvTranspose2d
DAE_1_13	(4, 13, 13)	86.22%	1 Conv2D + 1 ConvTranspose2d
DAE_1_8	(4, 8, 8)	32.65%	1 Conv2D + 1 ConvTranspose2d
DAE_1_5	(4, 5, 5)	12.76%	1 Conv2D + 1 ConvTranspose2d

4 Experiments: Training and Tuning the Models

4.1 Software Dependencies

All models were built and trained via the Python programming language with support from the NumPy and Matplotlib library, as well as the PyTorch framework. The implementation environment is listed below. Models were trained using the T4 Google Colab GPU to allow for CUDA-enabled performance speedup.

- Python version: 3.7.13
- PyTorch version: 2.1.0+cu121
- NumPy version: 1.23.5
- Matplotlib version: 3.7.1

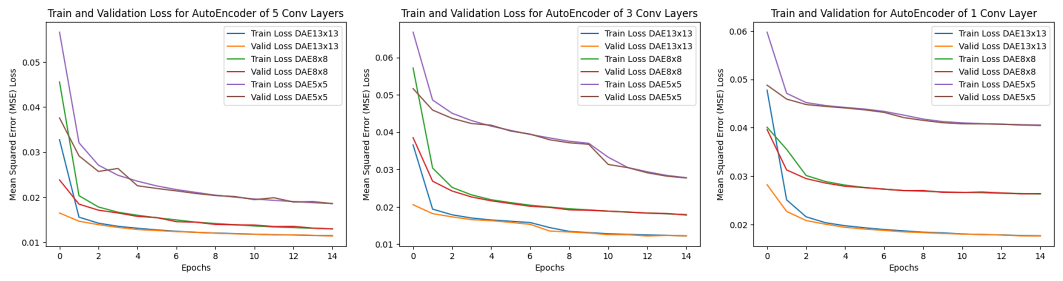
4.2 Training and Validation Pipeline

Each model as presented in Table 1 in the previous section was trained and tuned using the pipeline described in the following algorithm. The algorithm provides a high level description and correspoding code can be found in the attached Jupyter Notebook.

Algorithm 1 Train-Validate Denoising Autoencoder

```
Require: $\mathcal{X} \leftarrow \mathcal{MNIST}$                                  ▷ Load all MNIST Data
Require: $\mathcal{D} \leftarrow \mathcal{MODEL}$                                  ▷ Autoencoder Model Instance
Require: $\mathcal{C} \leftarrow \mathcal{GAUSSIAN}$                               ▷ Noise Function
Require: $\mathcal{L} \leftarrow \mathcal{MSE}_{LOSS}$                             ▷ MSE Loss Function
Require: $\mathcal{OPT} \leftarrow \mathcal{ADAM}_{OPTIM}$                         ▷ Adam Optimizer Function
  $\mathcal{X}_{train}, \mathcal{X}_{valid} \leftarrow 0.8 * \mathcal{X}, 0.2 * \mathcal{X}$   ▷ Split train-validation data in 80-20 ratio
  for $i = 1$ to $\mathcal{N}_{epochs}$ do
      for $\mathcal{B}_{train}$ in $\mathcal{X}_{train}$ do                     ▷ Train Loop
          $\mathcal{NB}_{train} \leftarrow \mathcal{C}(\mathcal{B}_{train})$           ▷ Apply Noise to Sample Batch
          $\mathcal{O}_{train} \leftarrow \mathcal{D}(\mathcal{NB}_{train})$           ▷ Run Batch through Autoencoder
          $Train_{loss} \leftarrow \mathcal{L}(\mathcal{O}_{train}, \mathcal{B}_{train})$   ▷ Update Train Loss
          $\mathcal{L}.backward()$                                               ▷ Backpropagate Loss
          $\mathcal{OPT}.step()$                                                 ▷ Update Weights
      end for
      for $\mathcal{B}_{valid}$ in $\mathcal{X}_{valid}$ do                     ▷ Valid Loop
          $\mathcal{NB}_{valid} \leftarrow \mathcal{C}(\mathcal{B}_{valid})$           ▷ Apply Noise to Sample Batch
          $\mathcal{O}_{valid} \leftarrow \mathcal{D}(\mathcal{NB}_{valid})$           ▷ Run Batch through Autoencoder
          $Valid_{loss} \leftarrow \mathcal{L}(\mathcal{O}_{valid}, \mathcal{B}_{valid})$   ▷ Update Valid Loss
      end for
  end for
```

Each Autoencoder model was trained for 15 epochs as that proved to be sufficient on average to allow model convergence for each setup. Training and validation loss was tracked on a per epoch basis. Plots were generated on a architectural basis so that for a given number of layers of the denoising autoencoder we consider each possible latent space dimensionality. The purpose of this plotting choice was to demonstrate via a side by side comparison the training performance for the variation of the compression rate of the autoencoder.



(a) Deepest Model (b) Medium Depth Model (c) Shallowest Model

Figure 5: Training and Validation Loss for Different Models

The plots from Figure 5 gives us a few interesting points to discuss:

- The validation loss follows the training loss as they both eventually **converge** while they decrease. There is no significant gap between the two which gives us comfort that the model should neither be highly underfitting nor overfitting.
- We establish the first **claim** that for a given number of Conv. layers, reduction in the latent space somewhat harms the model's performance as the training and validation loss are higher for smaller latent spaces. This visibly seems supported by each subplot in Figure 5 wherein the loss curves for smaller latent space models are higher than those for higher latent space models. We call this claim $(*)$.
- We establish the second **claim** that on average a shallower model seems to yield a higher loss than a deeper model. This can be noticed by the side by side comparison of the three plots in Figure 5 as we see the converged loss for both validation and training. We call this claim $(**)$.

We now further analyze the models using the test data on the next section of this report to verify our claims on completely unseen data.

5 Results and Conclusions

5.1 Test Data Performance

Now we analyze the models on the unseen test data and consider the loss response for the different architectures as displayed in the bar chart in Figure 6.

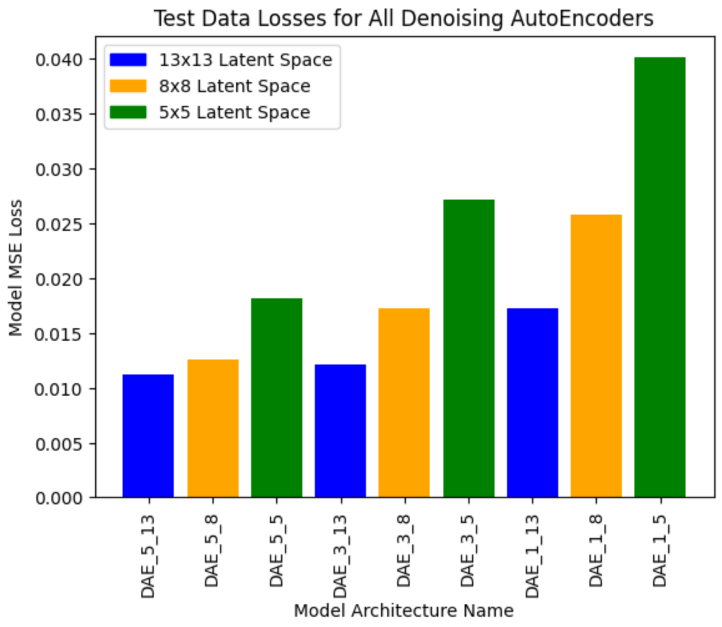


Figure 6: Test Loss Report for all Model Architectures

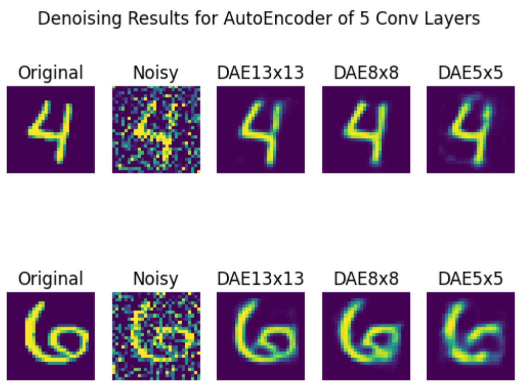
There is evidence from the unseen test data to support the claims we established in our earlier section. We notice that the deepest architecture gives the lowest loss for all latent space sizes both during training and validation, as well as during testing. Further these are the observations we can draw from the plot:

- The models relying on a 13x13 latent space size reasonably perform the best. This is not surprising considering the compression ratio for these models is around 86% as seen in Table 1 from the previous section. Since the model can retain a lot of details after applying the encoder layer, it makes it easier to recognize noise in the original input.
- As the latent space size decreases to 8x8 with a compression ratio of 32.65%, the test loss reported increases a bit compared to the models with a latent space size 13x13. The significant detail is that the increase in loss for this scenario is larger for the shallower models which is explained by the fact these models struggle to encode the input data well enough and lose more information that the model with more hidden layers. Overall these models are still not performing that poorly especially considering their compression ratio.
- The 5x5 latent space models suffer from high test loss regardless. We attribute this to the fact the compression ratio in this case is as low as 12.76% which makes is extremely challenging for the model to differentiate the noise present in the input batches and allow the model to learn a meaningful representation of the input to the latent space.

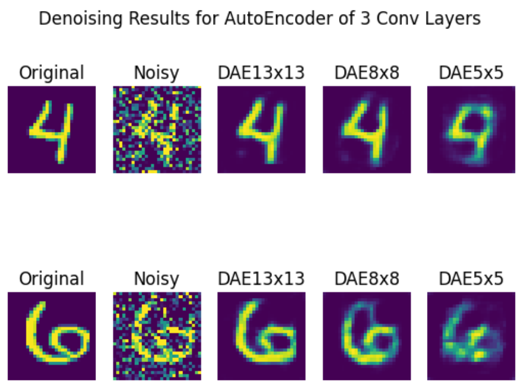
5.2 Denoising Benchmarks

Now we show a few samples of test data as they undergo the steps of denoising from the original uncorrupted input. We can certainly tell apart the model labeled as DAE13x13, which uses the largest latent space, yields strong denoising results. In all three cases it does a reasonably good job of removing the implanted noise; it produces images which are rather similar to the original data with some amount of blur. As expected, between the three different DAE13x13 models the reconstruction quality slowly drops as we use less hidden layers. The model with 5 convolutional layers in the encoder is generating a slightly better output than the one with 1 convolutional layer. This scales the same for the smaller models DAE8x8 and DAE5x5. Towards this end we can see our test results support claim $(**)$ made in section 4.2.

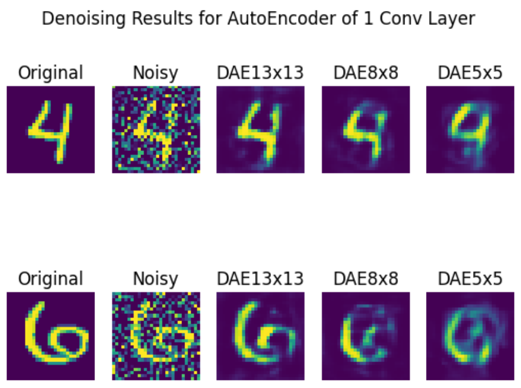
Similarly we argue that when we analyze the output generated by each model with decreasing latent space sizes for a fixed number of convolutional layers, reconstruction quality drops too. Each of the subplots of Figure 7 is a testament to this. It seems that a model with a small latent space barely manages to remove the noise properly. In fact it looks as if it is just learning an average over the entire MNIST dataset and this becomes more clear when we combine the small latent space size with fewer hidden layers. These results, help up support claim $(*)$ from section 4.2.



(a) 5 Layer Model



(b) 3 Layer Model



(c) 1 Layer Model

Figure 7: Training and Validation Loss for Different Models

5.3 Closing Remarks

In conclusion we have constructed a baseline to show that autoencoders are a powerful technique that can be used to perform image denoising. Their performance surely varies based on their architectural choice but we see that despite a drop in the denoising capability as we decrease the latent space dimensions of the network, their compression capabilities are rather strong (with about 30% compression ratio we get relatively strong denoising capabilities). Just like all neural networks their performance is also dependent on their hidden layers and we notice that utilizing more hidden layers improves the denoising ability of this network. Nonetheless with some tradeoffs between the latent space size and network depth we can get a strong performing model that is neither too deep nor uses a large bottleneck.

5.4 Future Work Thoughts

For future work, it would be interesting to explore the bounds of the depth of the network before it starts to run into overfitting territory. In addition, as there is no direct accuracy measure for denoising tasks in the context of neural network accuracy, one interesting pursuit would be to append a 10-class classifier (it could be a neural network itself or perhaps a multiclass logistic regression model) at the end of the pipeline and attempt label prediction on the denoised data. This can give a baseline on measuring prediction accuracy which is dependent on how strong our denoiser performs as well.

References

[1] A.K. Boyat and B.K. Joshi, "A Review Paper: Noise Models in Digital Image Processing," arXiv:1505.03489 [cs.CV], 2015.

[2] N. R. Soora, S. Vodithala and J. S. H. Badam, "Filtering Techniques to remove Noises from an Image," 2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), Chennai, India, 2022, pp. 1-9, doi: 10.1109/ACCAI53970.2022.9752476.

[3] U. Michelucci, "An Introduction to Autoencoders," arXiv:2201.03898 [cs.LG], 2022.

[4] L. Theis, et al., "Lossy image compression with compressive autoencoders," arXiv:1703.00395 [cs.CV], 2017.

[5] K. Bajaj, D. Singh, and M. Ansari, "Autoencoders Based Deep Learner for Image Denoising," Procedia Computer Science, vol. 171, pp. 1535-1541, 2020. doi: 10.1016/j.procs.2020.04.164.

[6] L. Ciampiconi, A. Elwood, M. Leonardi, A. Mohamed, and A. Rozza, "A survey and taxonomy of loss functions in machine learning," arXiv:2301.05579 [cs.LG], 2023.