Evaluating the Impact of Network Depth and Bottleneck Capacity on MNIST Denoising

Alexander Eriksson Byström Jack Andersson Stridh

Abstract

This project investigates how changing the structure of a Denoising Autoencoder (DAE) affects its ability to clean up noisy images from the MNIST dataset. We tested nine different models by varying the network depth (from 1 to 3 layers) and the bottleneck size. To pick bottleneck sizes, we use Principal Component Analysis (PCA) to find the dimensions needed to keep 70%, 80% and 90% of the data's original variance. All models were trained to reconstruct clean digits after being fed versions corrupted by 50% Gaussian noise.

Our results show that bigger and deeper isn't always better for accuracy. The simplest model (one layer) with the largest bottleneck had the lowest mean squared error (MSE) as it produced a slightly blurred version of the digit that mathematically stays closer to the "average" clean image. On the other hand, the deeper three-layer models created much sharper images that look better to the human eye, but have higher error scores.

This reveals a perception-distortion trade-off: models that look "cleaner" to us often have higher numerical errors because they hallucinate sharp edges that aren't perfectly aligned with the original image.

1 Introduction

Autoencoders trained to reconstruct clean inputs from corrupted observations have become a standard approach for learning robust data representations in an unsupervised manner. By deliberately injecting noise during training, denoising autoencoders (DAEs) are prevented from learning trivial identity mappings and instead must capture the underlying structure of the data manifold. This framework has proven effective across various domains, from image processing to signal restoration, yet the relationship between network architecture and reconstruction quality remains incompletely understood.

Two fundamental architectural parameters govern the capacity and behavior of denoising autoencoders: the depth of the network (the number of successive nonlinear transformations) and the dimensionality of the bottleneck layer (the degree of information compression imposed). While increasing network depth allows for more complex functional mappings, it also introduces additional parameters and potential instabilities during optimization. Similarly, expanding the bottleneck preserves more information from the input but may reduce the regularizing effect of compression that enables generalization.

Prior work has established the theoretical foundations of denoising autoencoders, demonstrating that under certain conditions they learn to approximate the score function of the data distribution. However, empirical investigations into how architectural choices affect this approximation quality, particularly in the presence of varying noise levels, have been limited. Most existing studies focus on demonstrating that DAEs can denoise effectively, rather than systematically exploring the tradeoffs inherent in different architectural configurations.

This paper presents a controlled empirical study of how network depth and bottleneck dimensionality jointly influence denoising performance on the MNIST handwritten digit dataset. We construct a systematic grid of nine architectures, varying depth from one to three hidden layers while selecting bottleneck dimensions through Principal Component Analysis to capture 70%, 80%, and 90% of the variance in the training data. All models are trained on images corrupted with additive Gaussian noise and evaluated on their ability to reconstruct clean digits from similarly corrupted test images.

Our results reveal a non-monotonic relationship between architectural complexity and reconstruction error. The simplest architecture with the largest bottleneck achieves the lowest Mean Squared Error, producing conservative, smoothed reconstructions. In contrast, deeper networks generate sharper outputs with more pronounced edges and finer detail, yet incur higher numerical reconstruction error. Analysis of pixel-wise residuals indicates that this discrepancy arises from a fundamental tension between optimizing for point-wise accuracy and recovering perceptually salient image structure. These findings have implications for the choice of evaluation metrics in image reconstruction tasks and highlight the importance of aligning network architecture with the specific objectives of the application.

2 Background

Autoencoders are neural networks designed to learn compressed representations of data through an encoder-decoder architecture. A standard autoencoder learns to reconstruct its input by first compressing it into a lower-dimensional latent representation (the bottleneck) and then reconstructing the original input from this compressed form. The encoder maps $\mathbf{x} \to \mathbf{z}$ where $\mathbf{z} \in \mathbb{R}^k$ with $k \ll d$, and the decoder maps $\mathbf{z} \to \hat{\mathbf{x}}$, attempting to recover the original input.

A *Denoising Autoencoder* (DAE) extends this framework by training the network to reconstruct clean inputs from corrupted versions (Vincent 2008). Rather than learning the trivial identity function $f(\mathbf{x}) = \mathbf{x}$, the network must learn the underlying structure of the data to successfully remove noise. This forces the model to capture meaningful patterns rather than memorizing pixel-level details. To understand why this works, we turn to the manifold hypothesis.

2.1 The Manifold Hypothesis

To understand what an autoencoder does, we have to recognize that while each MNIST image is represented as a 784-dimensional vector, the actual space of meaningful digit images occupies a much smaller subspace. Given a picture of dimension (resolution) $n \times d$ with only grayscale values in $[0, 255] \subset \mathbb{Z}$, there exist $256^{(n \times d)}$ possible combinations.

However, the set of images which we humans interpret as digits is a very small subset of these $256^{(n \times d)}$ combinations and lies on a much lower-dimensional manifold than the $n \times d$-dimensional ambient space. This well-known hypothesis goes under the name of the *Manifold Hypothesis* (Bengio, Courville, and Vincent 2013). While this hypothesis cannot be easily visualized in high-dimensional space, we can illustrate the concept by projecting to a two-dimensional Euclidean space.

As shown in Figure 1, when we add Gaussian noise to a clean image, we effectively knock the data point off this manifold $\mathcal{M}$. Vincent 2008 characterizes the denoising process as learning a vector field that maps these corrupted points back onto the manifold, effectively capturing the local structure of the data distribution. To the human eye, being off the manifold makes the image resemble less of what we interpret as a digit.

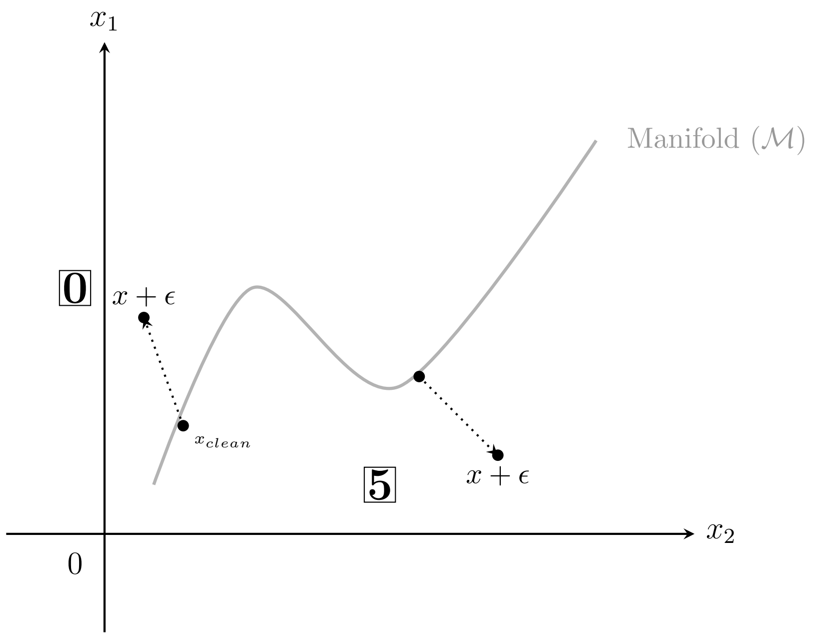

Figure 1: *(Our) Visualization of Gaussian noise in $\mathbb{R}^2$. The vector ϵ knocks the data point off the manifold $\mathcal{M}$ into a region of the space where the pixel combinations represent non-meaningful patterns.*

2.2 From Intuition to Bayesian Estimation

To wrap our heads around what the Denoising Autoencoder is actually learning, we can move from the geometric "manifold" view to a more probabilistic one. When we minimize the MSE as defined in Equation (6), we are essentially looking for the expected value of the clean data given the noisy version:

$$\mathcal{L}_{\text{DAE}} = \mathbb{E}_{\mathbf{x},\tilde{\mathbf{x}}}\left[\|\mathbf{x} - f_{\boldsymbol{\theta}}(\tilde{\mathbf{x}})\|^2\right] \tag{1}$$

From statistical estimation theory, the function $f_{\boldsymbol{\theta}}$ that best minimizes this error is the *conditional expectation* (the mean of the posterior distribution)[1]:

$$f_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}) = \mathbb{E}[\mathbf{X} \mid \tilde{\mathbf{X}} = \tilde{\mathbf{x}}] \tag{2}$$

where $f_{\boldsymbol{\theta}}$ still is defined as $f_{\boldsymbol{\theta}} = f_{dec} \circ f_{enc}$. This tells us that the bottleneck in Figure 2 isn't just a simple compression trick. It's forcing the network to figure out the most likely "true" signal $\mathbf{x}$ hiding under all that noise.

For the Gaussian noise $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$ we have introduced, we can use the magic formula known as *Tweedie's Formula* to map back from $\tilde{\mathbf{x}} \mapsto \hat{\mathbf{x}}$ by observing how dense the crowd of the data is nearby $(\nabla_{\tilde{\mathbf{x}}} \log p)$:

$$\mathbb{E}[\mathbf{X} \mid \tilde{\mathbf{X}} = \tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$$

If we plug this back into our DAE logic, we see that:

$$f_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}) \approx \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}) \tag{3}$$

Which is the most important result to here make sense of the DAE: the network is learning the score of the noisy data where the difference between the input $(\mathbf{x})$ and the output $\tilde{\mathbf{x}}$ is a vector that pushes the noisy point back toward the manifold $\mathcal{M}$ as we previously illustrated in Figure 1. In short, our neural network is approximating the score of the data distribution.

[1] Derivation of this can be found at Bishop 2006 and using notation in line with this paper can be found in Appendix A.

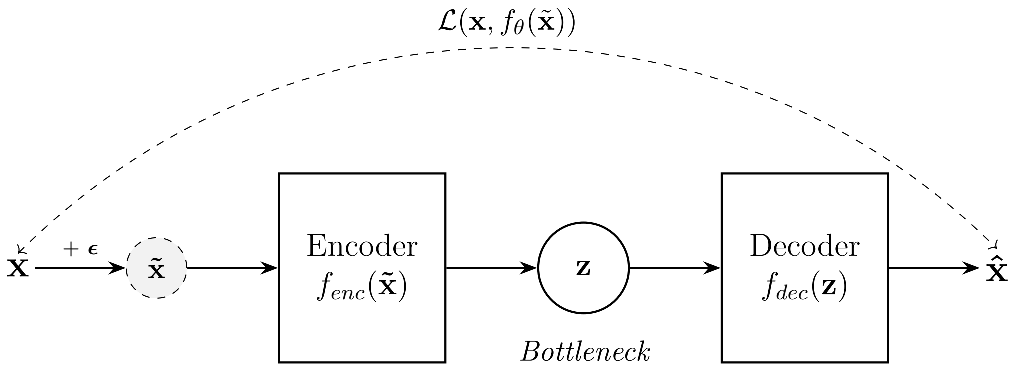
$\mathcal{L}(\mathbf{x}, f_\theta(\tilde{\mathbf{x}}))$
$\mathbf{x}$
$+\,\boldsymbol{\epsilon}$
$\tilde{\mathbf{x}}$
Encoder
$f_{enc}(\tilde{\mathbf{x}})$
$\mathbf{z}$
Bottleneck
Decoder
$f_{dec}(\mathbf{z})$
$\hat{\mathbf{x}}$

Figure 2: Overview of the architecture of a Denoising Autoencoder. The composite function $f_\theta = f_{dec} \circ f_{enc}$ maps the corrupted input $\tilde{\mathbf{x}}$ back to the original signal $\mathbf{x}$.

3 Data

Like the first project for this course we base our experiment on the *Modified National Institute of Standards and Technology* (MNIST) dataset. MNIST consists of 60,000 training ($\mathbf{X}_\mathcal{T}$) and 10,000 test ($\mathbf{X}_\text{test}$) images of 28×28 pixels, where each pixel is a grayscale value $x_{ij} \in [0, 255]$. These values are pre-processed by normalization so that $x_{ij} \in [0, 1]\ \forall i, j$.

After splitting the data into training and test, we will be left with the following two datasets to work with:

$$\mathbf{X}_\mathcal{T}^{10{,}000\times 784}, \qquad \mathbf{X}_\text{test}^{10{,}000\times 784}$$

The reason for the training dataset ($\mathbf{X}_\mathcal{T}$) being limited to 10,000 observations is once again due to computational constraints. The biggest contrast to our previous project in terms of data is that we will no longer be using any of the labels $\mathbf{y}_\text{train}, \mathbf{y}_\text{test} \in \{0, \ldots, 9\}^{10{,}000}$ as the methods we are now employing are all unsupervised.[2] See Figure 3 for a random draw of the ten digit classes.

[2]To be meticulous, we would argue for this method not being purely unsupervised, but rather what we call *self-supervised* where the autoencoder learns from unlabeled data by creating its own supervision. Since we have no labels y we do not have pairs of $(\mathbf{x}_i, \mathbf{y}_i)$, but rather pairs of $(\mathbf{x}_i, \hat{\mathbf{x}}_i)$.

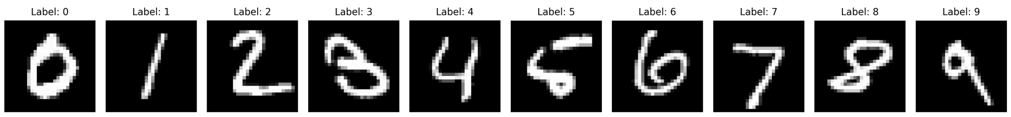

Figure 3: Representative samples from the MNIST test set showing the ten handwritten digit classes $\{0, \ldots, 9\}$.

4 Method

4.1 Injecting Additive Gaussian Noise

Previously, we utilized SVMs to compare how various kernels perform when Additive Gaussian Noise (AWGN) is applied to the test data. In that context, the corrupted test matrix $\tilde{\mathbf{X}}_{\text{test}}$ was formed by:

$$\tilde{\mathbf{X}}_{\text{test}} = \mathbf{X}_{\text{test}} + \mathbf{E} \tag{4}$$

where each element of the noise matrix is drawn from a normal distribution, $E_{ij} \sim \mathcal{N}(0, \sigma^2)$.

We now follow a similar logic for our Denoising Autoencoders, but we shift our focus from the matrix level to the individual sample level. For each image vector $\mathbf{x}_i$ in our design matrix $\mathbf{X}_{\mathcal{T}}$, we generate a corrupted version:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \boldsymbol{\epsilon}_i \tag{5}$$

where $\boldsymbol{\epsilon}_i \in \mathbb{R}^d$ is a noise vector with components $\epsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$. Our goal is to evaluate how effectively various DAE architectures can restore these corrupted vectors $\tilde{\mathbf{x}}_i$ to their original state $\mathbf{x}_i$ by minimizing the reconstruction error across the entire design matrix. A graphic example of how the image gradually gets more corrupted for higher noise levels can be seen in Figure 4.

In our previous paper (Byström and Stridh 2025b), we gradually applied increasing levels of noise, as illustrated in Figure 4. However, for the current DAE training, we have fixed the noise level at 50%. This specific intensity was selected to ensure a sufficiently high degree of corruption, forcing the network to move beyond simple local interpolation and instead capture the global structure of the data manifold.

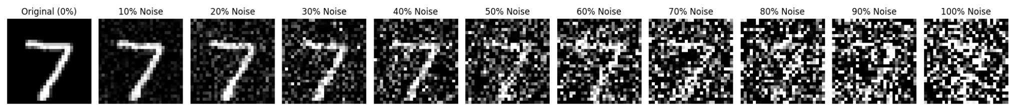

Figure 4: Progressive degradation of an MNIST digit sample under increasing levels of additive Gaussian noise $0 \le \sigma \le 1$.

4.2 Denoising Autoencoders

While a standard autoencoder is trained to reconstruct its input, a *Denoising Autoencoder* (DAE) is trained to reconstruct the original, clean input from a corrupted version of it. This architecture prevents the network from simply learning an identity function; instead, it forces the model to learn the underlying structure of the data manifold in order to "undo" the noise and recover the original signal.

As established in Section 2.2, the network learns to approximate the conditional expectation $\mathbb{E}[\mathbf{X} \mid \tilde{\mathbf{X}} = \tilde{\mathbf{x}}]$, which minimizes the Mean Squared Error (MSE) loss:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}\|^2 \tag{6}$$

Since the MNIST dataset consists of 784-dimensional vectors, the input and output layers are fixed by the data's dimensionality. However, the internal mapping is determined by the architect's choice of network depth and bottleneck capacity. By varying the number of layers and the quantity of neurons within each hidden layer, we control the network's ability to model complex manifolds while enforcing a strict information constraint.

The illustration in Figure 5 provides intuition for the term "bottleneck", as the network graphically converges to a constrained latent space (k) before expanding to reconstruct the original 28×28 pixel input.

4.2.1 Deciding Encoding-Decoding Architecture

To systematically evaluate these architectural trade-offs, we developed a grid of nine unique models. These are categorized by their depth (ranging from

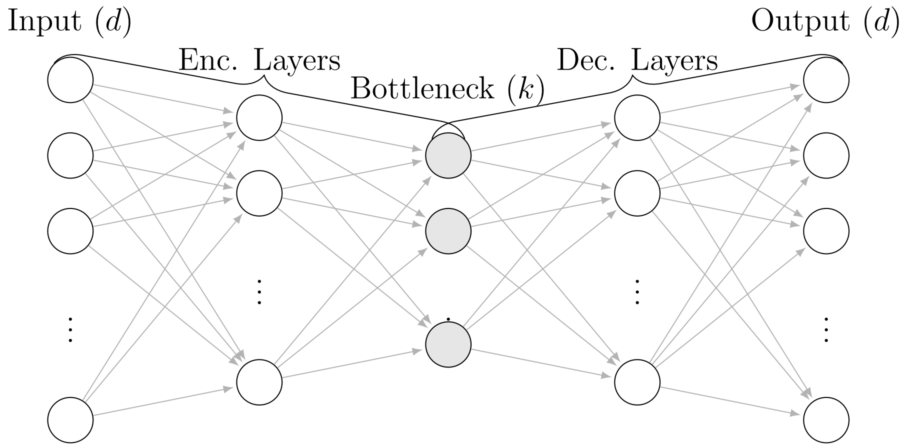

Figure 5: Neural network schematic of the Denoising Autoencoder. Circles represent neurons, and lines represent weight connections. The reduced number of neurons in the central layer (k) forces the network to learn a compressed manifold representation.

one to three hidden layers) and their latent capacity, which we determine using PCA to capture specific levels of variance in the training data.

4.2.2 Determining the Size of the Bottleneck using PCA

To make an informed choice regarding the bottleneck dimensionality (k), we first perform Principal Component Analysis (PCA), which serves as the optimal linear baseline for data reconstruction.[3] We define the sample covariance matrix (assuming zero-mean) as $\mathbf{\Sigma} = \frac{1}{n-1}\mathbf{X}^\top\mathbf{X}$. The objective is to find an orthogonal projection matrix $\mathbf{V} \in \mathbb{R}^{d\times k}$ that maximizes the captured variance, expressed through the trace of the projected covariance:[4]

$$\max_{\mathbf{V}} \operatorname{tr}\left(\mathbf{V}^\top\mathbf{\Sigma}\mathbf{V}\right), \quad \text{subject to } \mathbf{V}^\top\mathbf{V} = \mathbf{I}_k \tag{7}$$

The solution to Equation (7) is obtained via the eigenvalue decomposition $\mathbf{\Sigma}\mathbf{v}_i = \lambda_i\mathbf{v}_i$ where the eigenvectors $\mathbf{v}_i$ are arranged in descending order according to their corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$. To determine the specific size of the information bottleneck, we analyze the

[3]See module `PCA.py` at Byström and Stridh 2025a for implementation.

[4]To see a more comprehensive derivation see Appendix B.

cumulative explained variance ratio (CEVR) as a function of k:

$$\text{CEVR}(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{d} \lambda_j} \tag{8}$$

Using Equation (8), we select three specific bottleneck sizes that represent 70%, 80%, and 90% of the total variance. This results in a systematic experimental grid consisting of 9 unique DAE models, categorized by their depth (D) and latent capacity (k). The full experimental matrix is summarized in Table 1.

Depth (D)	**Network Structure**	**Bottleneck Sizes (k)**
1	$784 \to k \to 784$	{26, 43, 86}
2	$784 \to 256 \to k \to 256 \to 784$	{26, 43, 86}
3	$784 \to 256 \to 128 \to k \to 128 \to 256 \to 784$	{26, 43, 86}
PCA Target	Variance captured: 70%, 80%, and 90% respectively.	

Table 1: DAE Architectures categorized by depth and latent capacity k.

4.2.3 Training Procedure

In the training process,[5] we use 50 epochs with mini-batches of 128 images. For every batch, the process follows a standard cycle:

1. **Data Corruption:** Clean input vectors $\mathbf{x}^{(i)} \in [0,1]^{784}$ are corrupted with additive Gaussian noise following Equation (5):

$$\tilde{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} + \boldsymbol{\epsilon}^{(i)}, \quad \boldsymbol{\epsilon}^{(i)} \sim \mathcal{N}(0, 0.5^2 \mathbf{I}_{784})$$

   and then clipped to ensure pixel values remain within the valid $[0,1]$ range.

2. **Forward Propagation:** The corrupted input $\tilde{\mathbf{x}}^{(i)}$ is transformed through successive layers. For example, in a 3-layer encoder:

$$\begin{aligned} \mathbf{h}_1 &= \text{ReLU}(\mathbf{W}^{(1)}\tilde{\mathbf{x}}^{(i)} + \mathbf{b}^{(1)}), \\ \mathbf{h}_2 &= \text{ReLU}(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)}), \\ \mathbf{z} &= \text{ReLU}(\mathbf{W}^{(3)}\mathbf{h}_2 + \mathbf{b}^{(3)}) \end{aligned}$$

[5]The code for the training process can be found at Byström and Stridh 2025a, module: `DAE.py`.

where $\mathbf{z} \in \mathbb{R}^k$ is the bottleneck representation. Each weight matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$ with $n_{\text{out}} < n_{\text{in}}$ compresses information, forcing the network to retain only the most significant structural features. The decoder mirrors this architecture to reconstruct $\hat{\mathbf{x}}^{(i)}$, with internal layers using ReLU activation and a final sigmoid layer ensuring outputs stay in $[0, 1]$.

3. **Loss Computation:** The reconstruction error is measured using Mean Squared Error across the batch as defined in Equation (6):

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \|\hat{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)}\|^2$$

4. **Gradient Computation:** The gradients $\nabla_{\boldsymbol{\theta}} \mathcal{J}$ are computed via back-propagation by applying the chain rule:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{W}^{(l)}} = \frac{\partial \mathcal{J}}{\partial \mathbf{h}^{(l)}} \cdot \frac{\partial \mathbf{h}^{(l)}}{\partial \mathbf{W}^{(l)}}$$

5. **Parameter Update:** The Adam optimizer[6] updates parameters with learning rate $\eta = 0.001$:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}$$

   where $\hat{\mathbf{m}}_t$ and $\hat{\mathbf{v}}_t$ are bias-corrected running averages of gradients and squared gradients, enabling adaptive step sizes for stable convergence.

All nine models in our grid search use identical training configurations to ensure fair comparison across architectures.

[6]Adaptive Moment Estimation Optimizer.

5 Results & Discussion

We evaluated all nine DAE architectures on the test set corrupted with 50% Gaussian noise. Table 2 presents the Mean Squared Error (MSE) for each model configuration, which appears to reveal complex interactions between network depth and bottleneck capacity that could challenge conventional assumptions about architectural scaling.

	Bottleneck Dimension (k)		
Network Depth (D)	26	43	86
1	0.03755	0.02438	0.01816
2	0.02377	0.01917	0.01976
3	0.02318	0.02386	0.02576

Table 2: Mean Squared Error (MSE) Comparison Across Network Depth and Bottleneck Dimensionality. Lower values indicate better numerical reconstruction quality.

The lowest reconstruction error was achieved by the shallowest architecture with the largest bottleneck (D=1, k=86, MSE=0.01816). This result is initially counterintuitive, as deeper networks are generally expected to learn more sophisticated representations. However, three tentative patterns emerge from the data that may help explain this finding.

First, within the single-layer architecture, increasing bottleneck capacity seems to monotonically improve performance. The MSE decreases from 0.03755 (k=26) to 0.01816 (k=86), a reduction of approximately 52%. This behavior is broadly consistent with theoretical expectations: when the network has only one encoding-decoding transformation, a larger bottleneck likely preserves more information from the input, enabling better reconstruction without the same risk of learning spurious patterns.

Second, the two-layer models exhibit a non-monotonic relationship with bottleneck size. Performance improves from k=26 (MSE=0.02377) to k=43 (MSE=0.01917), but then degrades slightly at k=86 (MSE=0.01976). This suggests that mid-capacity bottlenecks might provide a reasonable balance for this depth, where the network has sufficient expressiveness to denoise effectively without overfitting to noise patterns in the training data.

Third, and most strikingly, the three-layer architecture shows a consistent degradation in MSE as bottleneck capacity increases. Moving from k=26 (MSE=0.02318) to k=86 (MSE=0.02576) represents an approximately 11% increase in reconstruction error. This pattern runs counter to the naive expectation that more capacity should enable better performance. One possible explanation is that the combination of increased depth and increased bottleneck size makes it easier for the network to memorize fine-grained noise patterns rather than learning the underlying clean manifold structure.

5.1 Depth-Capacity Interaction Effects

To better understand how depth and capacity interact, we can examine performance differences across depth for each fixed bottleneck size. At k=26, increasing depth from one to three layers reduces MSE by 38% (from 0.03755 to 0.02318), indicating that depth may provide substantial benefits when bottleneck capacity is severely constrained. The additional nonlinear transformations could allow the network to extract more meaningful features despite the limited information channel.

At k=43, the pattern shifts. The two-layer model (MSE=0.01917) outperforms both the one-layer (MSE=0.02438) and three-layer (MSE=0.02386) configurations. This suggests an optimal intermediate complexity for this bottleneck size, where sufficient depth appears to help the network learn useful representations without introducing the optimization challenges or overfitting tendencies associated with deeper architectures.

At k=86, we observe a complete reversal: the one-layer model achieves the best performance (MSE=0.01816), followed by the two-layer (MSE=0.01976) and three-layer (MSE=0.02576) models. The performance degradation with increasing depth at this bottleneck size suggests that when sufficient capacity exists in the bottleneck, additional depth can become detrimental rather than beneficial.

Overall, these patterns indicate that depth and capacity are unlikely to behave as independent architectural parameters. The optimal configuration appears to depend on finding the right balance: constrained bottlenecks seem to benefit from depth to compensate for limited capacity, while large bottlenecks perform best with shallow architectures that may avoid overfitting.

5.2 The Perception-Distortion Trade-off

While Table 2 provides quantitative evidence that shallow networks with large bottlenecks minimize reconstruction error, visual inspection of the outputs reveals a striking discrepancy between numerical performance and perceptual quality. Figure 6 displays reconstructions of a representative digit produced by each of the nine models.

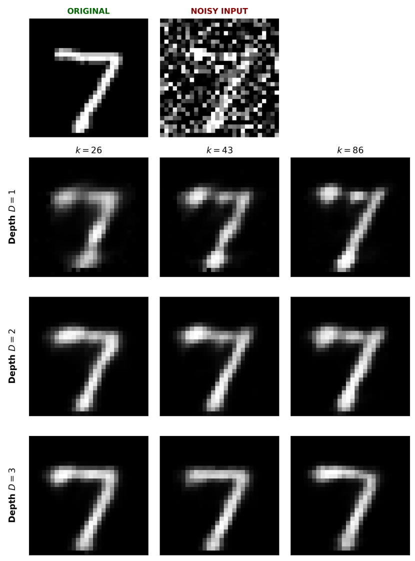

Figure 6: Reconstructed outputs for a single test digit across all nine DAE architectures.

The images in the bottom row (D=3) appear substantially sharper to human observers than those in the top row (D=1), with crisper edges and more clearly defined stroke boundaries. Yet paradoxically, these visually superior reconstructions correspond to higher MSE values. This phenomenon, known in the image processing literature as the perception-distortion trade-off (Blau and Michaeli 2018) reveals a fundamental tension between optimizing for human visual preference and optimizing for pixel-wise accuracy.

The source of this discrepancy becomes clear when we recall from Appendix A that the function minimizing MSE is the conditional expectation $E[X|\tilde{X} = \tilde{x}]$, which represents the posterior mean of all possible clean images that could have generated the observed noisy input. The shallow, high-capacity model approximates this posterior mean by producing a conservative blur that averages over plausible reconstructions. By avoiding commitment to any specific edge location, it minimizes the severe quadratic penalties that MSE assigns to spatially displaced features.

In contrast, deeper models produce reconstructions that project more decisively onto specific points on the data manifold. These networks generate sharp edges by committing to particular stroke locations and intensities. However, when these predicted edges are even slightly misaligned with the true digit structure, the resulting pixel-wise errors accumulate rapidly under the MSE metric. The network faces an unavoidable choice: produce a diffuse, low-error blur, or generate a sharp, visually appealing image with higher numerical error.

This counterintuitive relationship between visual quality and numerical accuracy is illustrated in Figure 7, which presents pixel-wise residual heatmaps for the two extreme architectures. The heatmaps reveal that the sharper, more visually appealing reconstruction from the deep model (right panel) actually exhibits substantially larger deviations from the original digit than the blurred output from the shallow model (left panel).

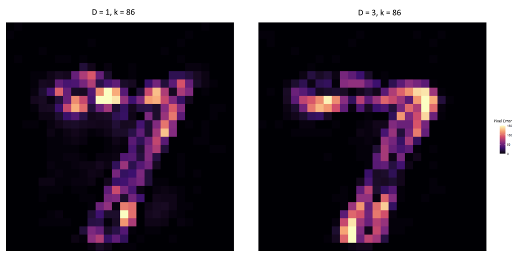

Figure 7: Pixel-wise absolute residual heatmaps showing reconstruction errors for the same test digit. Left panel shows errors from the shallow model (D=1, k=86, MSE=0.01816), right panel shows errors from the deep model (D=3, k=86, MSE=0.02576). Color intensity represents absolute error magnitude, with white indicating maximum deviation.

The shallow architecture's reconstruction (left panel) shows moderate errors distributed relatively uniformly along the digit's structure, with few pixels exhibiting extreme deviations. The deep model's reconstruction (right panel) reveals a strikingly different pattern: concentrated regions of high error (bright orange and white pixels) appear along the stroke boundaries and at key structural points. These hotspots indicate where the network has committed to sharp edge locations that are spatially offset from the ground truth.

The origin of these sharp but misaligned features may a form of learned "hallucination". During training, networks observe heavily corrupted inputs where 50% Gaussian noise nearly obliterates digit structure, yet must reconstruct clean targets with well-defined edges. Because fresh noise is sampled at each training step, the model learns to recognize degraded patterns and project them back onto the manifold of clean digits. Deeper architectures develop stronger structural priors about where edges should be located and how strokes should be shaped.

When confronted with ambiguous noisy inputs, these networks fill in missing details by imposing learned expectations, committing decisively to specific boundaries even when ground truth edges are positioned differently. The shallow model, lacking capacity for such detailed priors, hedges by producing a conservative blur that averages over plausible interpretations, resulting in moderate but diffuse errors rather than concentrated deviations.

This confirms the perception-distortion trade-off: reconstructions appearing sharpest to human observers are often most dissimilar to the original by pixel-wise comparison. Models minimizing MSE achieve low error through controlled blurring rather than accurate fine-scale recovery. For applications prioritizing perceptual quality over point-wise accuracy, evaluation metrics beyond MSE may be necessary.

6 Conclusion

This study systematically investigated how network depth and bottleneck capacity influence denoising performance in autoencoders trained on MNIST digits corrupted with 50% Gaussian noise. By testing nine architectures varying depth and bottleneck dimensions, we observed several patterns in the relationship between architecture and reconstruction quality.

Our primary finding is that the simplest architecture (D=1, k=86) achieved the lowest MSE (0.01816), outperforming all deeper variants. This result likely stems from how different architectures approximate the conditional expectation $E[X|\tilde{X} = \tilde{x}]$. Shallow networks produce conservative, smoothed reconstructions that minimize pixel-wise deviations, while deeper networks generate sharp, perceptually appealing images at the cost of higher numerical error when predicted edges are spatially misaligned.

The interaction between depth and capacity revealed three distinct patterns. At constrained bottlenecks (k=26), depth improved performance by 38%. At intermediate sizes (k=43), two-layer architectures achieved optimal balance. At large bottlenecks (k=86), performance degraded with increasing depth, suggesting overfitting to noise patterns rather than learning the underlying clean manifold.

Additionally, our residual analysis revealed a so-called perception-distortion trade-off in these models. The models producing the sharpest outputs systematically incurred higher MSE than those generating blurred but numerically accurate reconstructions. MSE penalizes sharp features that are slightly misaligned while rewarding diffuse predictions. This discrepancy suggests that for applications prioritizing perceptual quality, evaluation metrics beyond MSE may better align with human judgments.

These findings have practical implications for autoencoder design: shallow architectures with generous bottleneck capacity provide optimal numerical accuracy, while deeper networks may be preferable for perceptual quality despite higher MSE.

Future work could extend this analysis to Convolutional Neural Networks (CNNs) and non-Gaussian noise distributions, particularly on more complex, photorealistic datasets; such studies would determine if the perception-distortion trade-off is exacerbated when the underlying data manifold is harder to approximate than in the relatively simple MNIST domain

References

Bengio, Yoshua, Aaron Courville, and Pascal Vincent (2013). “Representation Learning: A Review and New Perspectives”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8, pp. 1798–1828. DOI: 10.1109/TPAMI.2013.50.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer. ISBN: 978-0387310732. URL: https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

Blau, Yochai and Tomer Michaeli (2018). “The perception-distortion trade-off”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6228–6237. DOI: 10.1109/CVPR.2018.00652.

Byström, Alexander Eriksson and Jack Andersson Stridh (2025a). *PCA Implementation for Denoising Autoencoder Bottleneck Selection*. https://github.com/Alexerby/stan52-project2-code.

– (Month of Submission 2025b). “The Robustness of SVM Kernels to Noise: A Comparative Analysis on MNIST”. Unpublished technical report, Project I for [STAN52].

Vincent, Pascal (2008). *Extracting and Composing Robust Features with Denoising Autoencoders*. Tech. rep. 1358. Université de Montréal. URL: https://www.cs.toronto.edu/~larocheh/publications/icml-2008-denoising-autoencoders.pdf.

A Appendix: Derivation of the Optimal MSE Estimator

To show why the Denoising Autoencoder targets the conditional mean, we look at the expected squared loss for our reconstruction function $f_\theta(\tilde{\mathbf{x}})$ relative to the clean target $\mathbf{x}$. Following the logic from Bishop 2006, the average loss is:

$$\mathbb{E}[L] = \iint \|f_\theta(\tilde{\mathbf{x}}) - \mathbf{x}\|^2 p(\mathbf{x}, \tilde{\mathbf{x}})\, d\mathbf{x} d\tilde{\mathbf{x}}$$

Our goal is to find the specific function $f_\theta(\tilde{\mathbf{x}})$ that minimizes this integral. We can rewrite the joint distribution using the product rule as $p(\mathbf{x}, \tilde{\mathbf{x}}) = p(\mathbf{x}|\tilde{\mathbf{x}})p(\tilde{\mathbf{x}})$. This allows us to express the loss as:

$$\mathbb{E}[L] = \int p(\tilde{\mathbf{x}}) \left[\int \|f_\theta(\tilde{\mathbf{x}}) - \mathbf{x}\|^2 p(\mathbf{x}|\tilde{\mathbf{x}})\, d\mathbf{x} \right] d\tilde{\mathbf{x}}$$

Since $p(\tilde{\mathbf{x}})$ is non-negative, minimizing the total loss is equivalent to minimizing the term inside the square brackets for every possible value of $\tilde{\mathbf{x}}$. If we take the derivative of that inner term with respect to $f_\theta(\tilde{\mathbf{x}})$ and set it to zero, we get:

$$\frac{\partial}{\partial f_\theta(\tilde{\mathbf{x}})} \int \|f_\theta(\tilde{\mathbf{x}}) - \mathbf{x}\|^2 p(\mathbf{x}|\tilde{\mathbf{x}})\, d\mathbf{x} = 2 \int (f_\theta(\tilde{\mathbf{x}}) - \mathbf{x}) p(\mathbf{x}|\tilde{\mathbf{x}})\, d\mathbf{x} = 0$$

By expanding the integral, we can solve for $f_\theta(\tilde{\mathbf{x}})$:

$$\int f_\theta(\tilde{\mathbf{x}}) p(\mathbf{x}|\tilde{\mathbf{x}})\, d\mathbf{x} = \int \mathbf{x} p(\mathbf{x}|\tilde{\mathbf{x}})\, d\mathbf{x}$$

Since $f_\theta(\tilde{\mathbf{x}})$ is a constant with respect to the integral over $\mathbf{x}$, and knowing that the conditional distribution $\int p(\mathbf{x}|\tilde{\mathbf{x}})\, d\mathbf{x}$ must sum to 1, the equation* simplifies to:

$$f_\theta(\tilde{\mathbf{x}}) = \int \mathbf{x} p(\mathbf{x}|\tilde{\mathbf{x}})\, d\mathbf{x} = \mathbb{E}[\mathbf{X}|\tilde{\mathbf{X}} = \tilde{\mathbf{x}}]$$

Which confirms that the optimal reconstruction for a noisy input is the mean of all possible clean images that could have produced it. In the context of our manifold hypothesis, the network is learning to find the "center of mass" of data points on the manifold $\mathcal{M}$ given the noisy observation $\tilde{\mathbf{x}}$.

B Appendix: Derivation of Principal Component Analysis

In Principal Component Analysis (PCA), we are seeking a linear projection of our data that captures as much of the original variance as possible. To derive this, let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be our design matrix, and we assume that the data has been pre-centered so that the sample mean is zero. We define our sample covariance matrix as $\mathbf{\Sigma} = \frac{1}{n-1}\mathbf{X}^\top \mathbf{X}$.

B.1 Our Objective: Maximizing Variance

We want to find a direction, defined by a vector $\mathbf{v}$, such that when we project our data onto it, the resulting variance is maximized. The variance of this projection can be expressed as:

$$\mathrm{Var}(\mathbf{X}\mathbf{v}) = \mathbf{v}^\top \mathbf{\Sigma} \mathbf{v} \tag{9}$$

B.2 Why we need a Constraint

If we were to naively attempt to maximize the expression $\mathbf{v}^\top \mathbf{\Sigma} \mathbf{v}$ without any constraint, the problem would be poorly specified. Because the expression is a quadratic form, we could simply increase the magnitude of $\mathbf{v}$ toward infinity and always find a "better" (larger) solution. To make the problem meaningful and find a unique direction, we must restrict $\mathbf{v}$ to be a unit vector, such that:

$$\|\mathbf{v}\|^2 = \mathbf{v}^\top \mathbf{v} = 1 \tag{10}$$

B.3 Solving the Optimization Problem

To solve this constrained maximization, we formulate a Lagrangian $\mathcal{L}$:

$$\mathcal{L}(\mathbf{v}, \lambda) = \mathbf{v}^\top \mathbf{\Sigma} \mathbf{v} - \lambda(\mathbf{v}^\top \mathbf{v} - 1) \tag{11}$$

where λ is our Lagrange multiplier. If we now take the gradient with respect to $\mathbf{v}$ and set it equal to zero, we get:

$$\nabla_{\mathbf{v}} \mathcal{L} = 2\mathbf{\Sigma}\mathbf{v} - 2\lambda\mathbf{v} = 0 \tag{12}$$

By rearranging this, we arrive at the following result:

$$\mathbf{\Sigma v} = \lambda \mathbf{v} \tag{13}$$

This tells us that the directions we are looking for are actually the eigenvectors of our covariance matrix $\mathbf{\Sigma}$. Furthermore, since the variance along the projection is $\mathbf{v}^\top \mathbf{\Sigma v} = \lambda$, we know that the eigenvalues quantify exactly how much variation is captured by each component. This is why we can use the cumulative sum of these eigenvalues to decide our bottleneck size k.