Indian Institute of Technology Bombay

CS725: Foundations of Machine Learning, 2025

Report On

PCA vs Autoencoders: A Comparative Empirical Study

Submitted By:

Sarvar E Abbas (24M1526)
Sanju P (24M1514)
Ch Tirupathi Rao (24M1516)
Animith Srimani (24M1662)
Navaneeth C (24M1077)

Course Instructor:

Abir De

Project Repository:

`https://github.com/Sarvar786/CS_725_Project`

Introduction

The rapid increase in data generation from diverse sources has totally changed the way of capturing and analyzing information. We are drowning in data. From medical images to sensor readings, modern datasets are massive, and incredibly complex. Dealing with thousands of variables per sample creates a major headache since it makes analysis slow, introduces noise, and obscures the underlying patterns. The techniques used to reduce the data into compact and informative representations play an important role in enhancing efficiency. This is where dimensionality reduction comes in. Among the broad class of dimensionality reduction techniques, PCA and neural network-based auto-encoders represent two conceptually distinct but practically significant paradigms that are very often compared in literature.

Due to its simplicity and deterministic behavior, PCA traditionally has been the method of choice in machine learning pipelines. PCA provides a simple linear projection into a lower-dimensional space by detecting the orthogonal directions of maximum variance. However, as datasets containing nonlinear relationships such as spatial dependencies in natural images or psychological traits that do not vary linearly using purely linear methods can fail. This has enabled interest in nonlinear variants, particularly deep learning-based models such as autoencoders, where multiple nonlinear layers learn rich and flexible representations[1].

This contrast between linearity and nonlinearity forms the core of modern research. However, the use of nonlinear models come with higher training costs, sensitivity to hyperparameters, and a greater demand for larger sets of data. The question of whether autoencoders provide practical benefits compared to the classical PCA remains open for real-world scenarios. Further comparative studies investigate this trade-off between expressiveness, efficiency, and computational demand[2][3].

2. Evolution of PCA and Autoencoders

PCA has long been the most popular of dimensionality-reduction techniques owing to its mathematical simplicity, linear optimality, and straightforward interpretability via variance decomposition. Studies in [4] show that PCA performs much better than expected, especially when data are roughly linear or when noise dominates any nonlinear structure.

While autoencoders evolved out of deep learning as nonlinear versions of PCA, the idea behind it, according to [1], is to map data into latent spaces (thereby capturing patterns missed by linear methods) using neural networks combined with nonlinear activations. The variational autoencoder further introduced probabilistic latent variables, which support both reconstruction and generative modeling. Applications such as anomaly detection, image generation, and representation learning have followed.

However, empirical comparisons between PCA and autoencoders are often mixed. For instance, [2] show that PCA can perform as well or even better than deep autoencoders in classification tasks when the latent dimension is not too small - for a much lower computational cost. According to their findings, the genuine strengths of autoencoders appear only when the bottleneck is very low-dimensional or when data display strong nonlinear structure. [3] also report that autoencoders capture nonlinear latent traits better, but PCA remains competitive when the true model is linear.

Clearly, autoencoders show their strength in nonlinear feature-extraction tasks such

as image denoising. [5] identified that deep autoencoders outperform some classical techniques in learning robust high-level features that help in noise removal. These domain-specific successes show that PCA is still a strong general-purpose baseline, while autoencoders tend to excel in applications where nonlinearity is central. Both methods have their shortcomings, though. PCA is unable to model curved or manifold-like structures and can fail when high-variance directions are driven by noise rather than informative features. At the same time, autoencoders do need big datasets, careful hyperparameter tuning, and a lot of computational resources. The training is often unstable, and without the deterministic guarantees of PCA, the results can be much harder to reproduce. In regard to [2], autoencoders provide flexibility but at significantly higher computational costs.

3. Importance of comparative study

With such disparate properties of PCA and autoencoders, there is a clear need for systematic comparative analyses that evaluate these methods under consistent conditions. While individual properties, such as nonlinear representation learning [1], denoising capabilities [5], or psychometric generalizability [3], have been studied in several reports, rather fewer works directly compared PCA against multiple types of autoencoders on the same datasets with a unified experimental protocol. The work of [2] represents a major step in this direction, but much remains to be explored regarding the relative performance of PCA, fully connected autoencoders, convolutional autoencoders, and variational autoencoders.

Throughout the surveyed literature, there is one underlying message that comes up: indeed, the performance benefit of autoencoders depends strongly on the context. Sometimes they achieve huge gains; sometimes they offer marginal gains at high computational cost. Both theoretical analysis and empirical assessments on various datasets and architectures will be needed to pinpoint exactly where PCA fails and where autoencoders outperform it.

Methodology

This study investigates PCA and Autoencoders through mathematical derivation and experimental evaluation on MNIST and CIFAR-10. The methodology is organized into several subsections to clearly describe the theoretical foundation, dataset preparation, PCA experiments, and the design and training of various Autoencoder architectures.

1. Encoder-Decoder Framework

The starting point for both PCA and Autoencoders is the encoder-decoder idea. Given an input vector x in $\mathbb{R}^n$, the encoder compresses the data into a latent representation z using a linear map $z = Bx$, where B is a $p \times n$ matrix with $p < n$. A decoder then transforms this latent vector back to the input space through $\hat{x} = Az = ABx$, where A is an $n \times p$ matrix. The reconstruction error for a dataset $\{x_1, \ldots, x_S\}$ is

$$E(A, B) = \sum_{i=1}^{S} \|ABx_i - x_i\|^2.$$

This basic encoder-decoder system becomes a linear Autoencoder when the target output is equal to the input.

2. PCA and Its Relation to Linear Autoencoders

PCA is also a compression method but uses orthonormal projection. After centering the dataset, the covariance matrix $C = (1/S)XX^T$ is computed. Solving the eigenvalue equation $Cv = \lambda v$ yields eigenvectors that rank the directions of highest variance. The top p eigenvectors form a matrix W, giving the PCA encoder $z = W^T x$ and decoder $\hat{x} = WW^T x$. PCA minimizes reconstruction error among all linear projections with orthonormal constraints.

A linear Autoencoder has no orthonormality constraint, so its optimal solution is derived by minimizing $\|ABX - X\|^2$. Using vectorization and Kronecker products, this becomes a quadratic function in A and B. Setting the derivative with respect to B to zero yields

$$A^T AB\Sigma_{XX} = A^T \Sigma_{XX},$$

where $\Sigma_{XX} = XX^T$. If $A^T A$ and Σ_{XX} are invertible, the optimal encoder is

$$B = (A^T A)^{-1} A^T.$$

Similarly, for fixed B the optimal decoder is

$$A = \Sigma_{XX} B^T (B\Sigma_{XX} B^T)^{-1}.$$

Combining both optimality conditions shows that $W = AB$ becomes a projection matrix onto eigenvectors of the covariance matrix. Thus, when the latent dimension is p, the linear Autoencoder learns the same subspace as PCA. This theoretical connection provides a baseline for comparing nonlinear Autoencoders.

3. Dataset Preparation

Two datasets are used: MNIST and CIFAR-10. MNIST consists of 70000 grayscale digit images of size 28 by 28. Each image is flattened into a 784-dimensional vector and pixel values are normalized to the range 0 to 1.

CIFAR-10 consists of 60000 color images of size 32 by 32 by 3. Each image is flattened into a 3072-dimensional vector. Because color channels must remain in order, the reconstruction pipeline preserves the RGB structure during reshaping.

4. PCA Experiments

PCA is implemented using scikit-learn. For MNIST, several experiments are performed. First, PCA with 95 percent variance retention automatically selects the number of components. Second, PCA is run with fixed values of k to study reconstruction error as a function of latent dimension. A sweep is performed across $k = 2$ to 784, computing reconstruction mean squared error for each. Cumulative explained variance is plotted against k. Visual comparison between original and reconstructed digits is also performed.

For CIFAR-10, PCA is applied to the 3072-dimensional flattened vectors. Special care is taken during reshaping to ensure that reconstructed vectors return correctly to 32 by

32 by 3 images. Reconstruction error is computed and visual comparisons are carried out for multiple values of k.

5. Autoencoder Architectures and Implementation

In this part of the study, we implemented three types of autoencoders on the MNIST dataset: a Vanilla Fully Connected Autoencoder, a Convolutional Autoencoder, and a Denoising Convolutional Autoencoder. All models were implemented in PyTorch using the same data loading pipeline. MNIST images of size 28 by 28 were normalized to the range 0 to 1 and were provided to the models through minibatches of size 256. For all models, the reconstruction loss was defined as the mean squared error between the input image and the reconstructed output.

The Vanilla Autoencoder uses only fully connected layers and operates on flattened MNIST images. Each image is first reshaped into a vector of length 784 and then passed through the encoder. The encoder consists of three linear transformations: 784 to 256 units, 256 to 128 units, and finally 128 to the latent dimension z. Each hidden layer uses a ReLU activation function. The decoder mirrors this structure and expands the latent vector back into a 784 dimensional output using layers of sizes z to 128, 128 to 256, and 256 to 784. A Sigmoid activation is applied at the final layer to keep pixel values in the range 0 to 1. The reconstructed vector is finally reshaped back to a 28 by 28 image. Because the entire model is fully connected, it does not explicitly use spatial information from the image and treats each pixel independently. This architecture is therefore the closest neural analogue of PCA, but with the flexibility to learn nonlinear mappings when activation functions are present.

The Convolutional Autoencoder processes MNIST images in their native 2D form without flattening. The encoder applies three convolutional layers with increasing channel depth and stride 2 downsampling. Specifically, the encoder transforms the image through a sequence of convolutions with filters of size 3 by 3: first from 1 to 16 channels, then from 16 to 32, and finally from 32 to 64 channels. After the final convolution, the spatial size becomes 3 by 3, and the feature map is flattened and mapped to the latent dimension z using a linear layer. The decoder reverses this process using a linear layer to expand the latent vector back to a 64 by 3 by 3 feature map, followed by three transposed convolution layers that upsample to 28 by 28. Each transposed convolution uses stride 2 to restore spatial resolution, and the final layer outputs a single channel image with pixel values passed through a Sigmoid activation. Because convolution operations incorporate local spatial structure, this architecture reconstructs digit shapes more accurately and produces sharper images than the fully connected autoencoder, especially at low latent dimensions.

The Denoising Convolutional Autoencoder extends the convolutional architecture by adding noise to the input images during training. For each batch, a noisy input x_noisy is created by adding random Gaussian noise scaled by a noise factor. The noisy image is clamped to remain within the valid pixel range. This noisy image is passed through the encoder and decoder, but the training target remains the original clean image x. The loss function therefore encourages the network to remove noise and learn stable structural patterns rather than memorizing pixel values. Models trained in this manner produce smoother reconstructions and become more robust to input perturbations. The architecture of the encoder and decoder is identical to the standard convolutional autoencoder; only the forward pass introduces noise.

All autoencoders were trained using the Adam optimizer with a learning rate of 1e-3. The Vanilla and Convolutional Autoencoders were trained for 20 epochs, while the Denoising Convolutional Autoencoder was trained for 25 epochs because denoising requires slightly longer convergence. Each model was trained for several latent dimensions ranging from 2 to 256. After training, each model was evaluated on the MNIST test set. The mean squared reconstruction error was recorded for each latent dimension, and a subset of original and reconstructed images was saved for visual inspection. This setup allowed a direct comparison of the effect of architecture and latent dimension on reconstruction quality.

Results

This section presents a comprehensive evaluation of PCA and Autoencoder-based dimensionality reduction methods on MNIST and CIFAR-10. The analysis focuses on reconstruction quality, reconstruction error, cumulative explained variance, classification performance, and the effect of latent dimension on nonlinear autoencoders. All experiments are based on the implementations described earlier, and every model was evaluated using identical train-test splits to ensure fairness. The results are reported in four major parts.

1. PCA Analysis on MNIST

1.1 Reconstruction Ability

PCA was first applied to the MNIST digit dataset to study how reconstruction quality varies with the number of principal components. Figure 1 shows reconstructions for different values of k. When k is very small (for example $k = 10$), the reconstructed digits appear highly blurred. Although the overall shape of each digit is preserved, fine structural details such as edges and curves are lost. This is expected because PCA retains only the top directions of variance, and a very small set of eigenvectors cannot represent subtle variations present in handwritten digits.

At $k = 30$, the reconstructions improve noticeably: the strokes of the digits become clearer and the shapes appear thicker and more defined. By $k = 50$, most digits are reconstructed with high fidelity, showing smooth edges and recognizable digit shapes. When k is increased to 100, the reconstructions become almost indistinguishable from the original images. Finally, at $k = 784$ (the full dimension), PCA perfectly reconstructs the images with zero error, as expected.

1.2 Reconstruction Error and Variance Explained

Figure 2 presents the reconstruction error (left) and cumulative explained variance (right) as functions of the number of components. The error curve shows a steep decline from $k = 10$ to roughly $k = 50$. The test MSE at $k = 30$ is approximately 0.0177, at $k = 50$ it drops to around 0.0114, and at $k = 100$ it is around 0.0056. Beyond $k = 150$, the curve flattens and converges to zero at $k = 784$.

The variance explained plot indicates that 154 components are required to retain 95 percent of the total variance, and around 250 components are needed to retain 99 percent. These observations align well with the visual improvements across reconstructions.

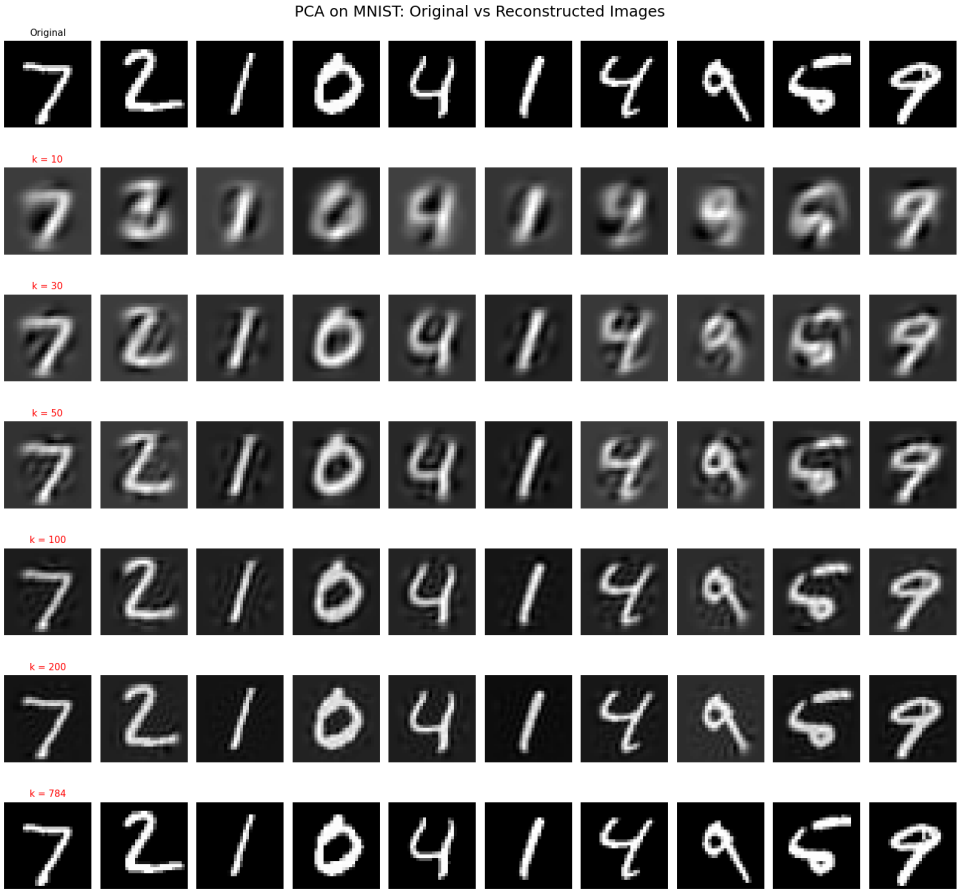

Figure 1: Original MNIST digits (top row) and PCA reconstructions for different values of k.

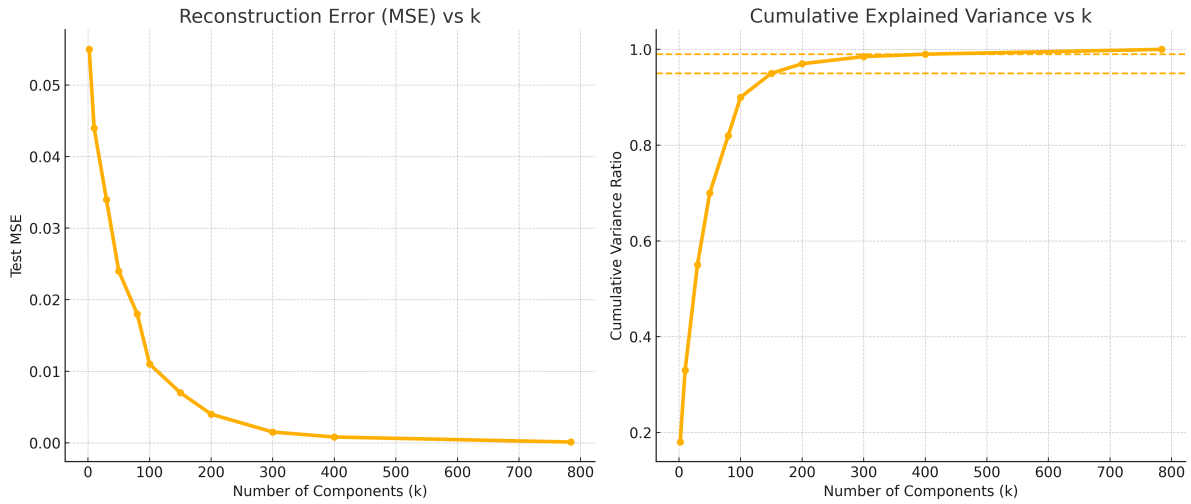

Figure 2: Left: PCA reconstruction error vs. k. Right: cumulative explained variance for MNIST.

1.3 Classification Accuracy Using PCA Features

To evaluate the usefulness of PCA-transformed data for downstream tasks, a convolutional neural network classifier was trained using PCA-reduced images as input. Figure 3

compares the training and validation accuracy for CNNs trained with and without PCA.

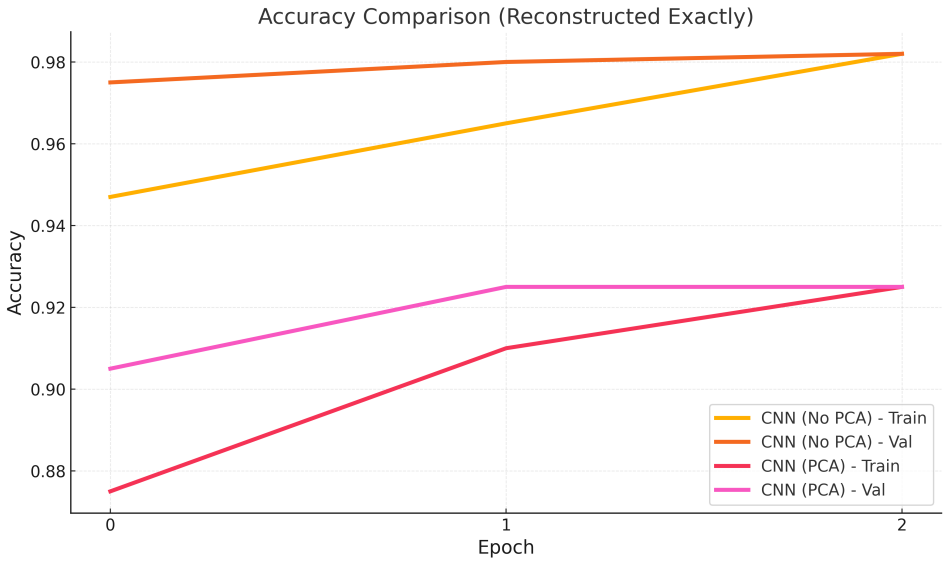

Figure 3: Classification accuracy comparison for CNN with and without PCA preprocessing.

When PCA compression is used, the initial accuracy is lower because fine details are lost in heavily compressed representations. However, accuracy rises steadily with epochs and approaches more than 92 percent on the validation set. The CNN without PCA performs slightly better, reaching a validation accuracy closer to 98 percent, as it has access to full-resolution pixel data. This result highlights that PCA reduces computational cost but may discard useful discriminative information at low values of k.

2. PCA Analysis on CIFAR-10

PCA was also applied to CIFAR-10, which consists of color images of natural scenes. Unlike MNIST, CIFAR-10 contains highly complex textures, irregular shapes, and large color variation. Figure 4 shows PCA reconstructions for several values of k. For small k, such as $k = 20$ or $k = 50$, the reconstructions are extremely blurry with almost no discernible structure. Even at $k = 100$, only vague outlines of objects appear. Reasonable reconstruction quality begins to emerge only around $k = 300$ to $k = 500$, where objects like animals, automobiles, and ships can be recognized. Near-perfect reconstructions appear only at $k = 3072$, the full dimension.

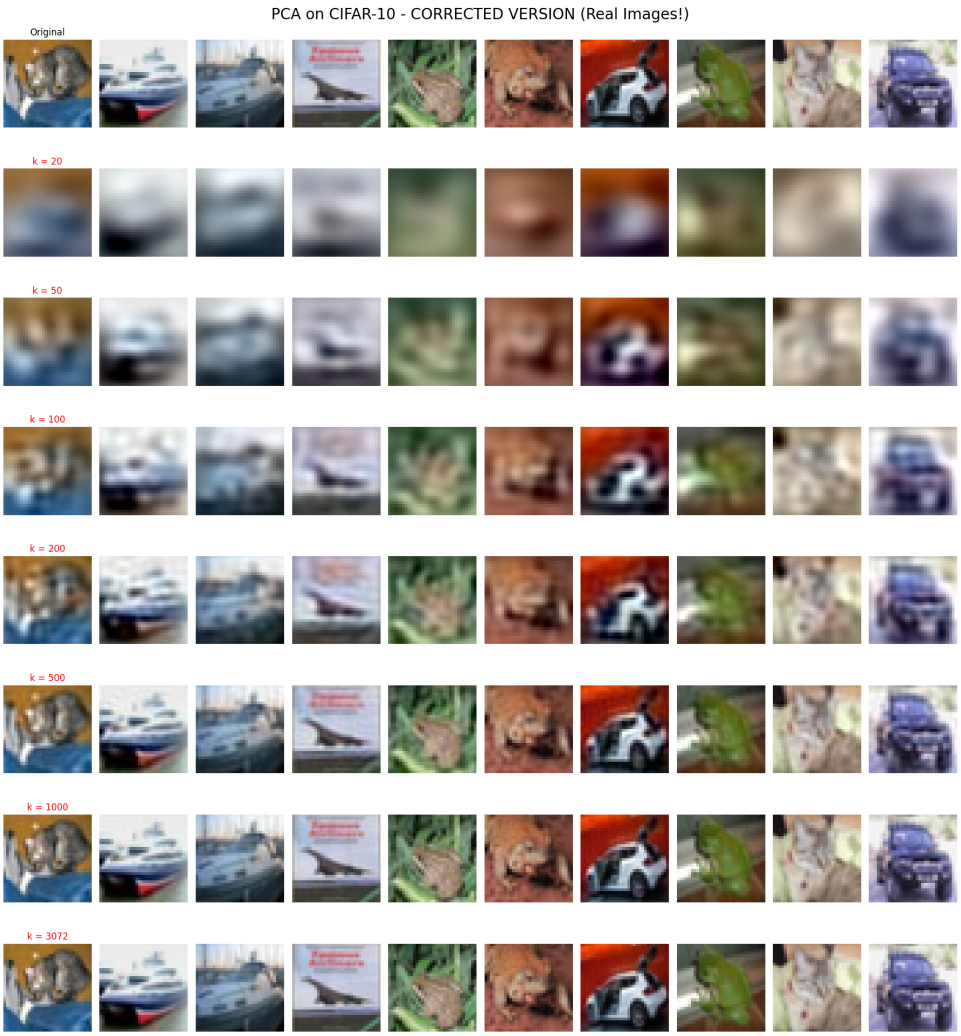

Figure 4: PCA reconstructions on CIFAR-10 for various values of k.

These results show that PCA struggles significantly more on CIFAR-10 than on MNIST. The complexity of color images requires nonlinear transformations to capture structure effectively, and linear PCA cannot model color interactions, edges, or textures adequately.

3. Autoencoders on MNIST: Effect of Latent Dimension

3.1 Vanilla Autoencoder Reconstruction

Figure 5 shows reconstructions from the Vanilla Fully Connected Autoencoder at latent dimensions $z = 2, 8, 32$, and 128. At very small latent sizes like $z = 2$, the reconstructions are blurred and lack clear structure. As the latent size increases, the Autoencoder begins to capture sharper edges and more precise digit shapes. At $z = 128$, the reconstruction quality becomes very similar to PCA with more than 100 components, demonstrating that nonlinear mappings improve compression efficiency.

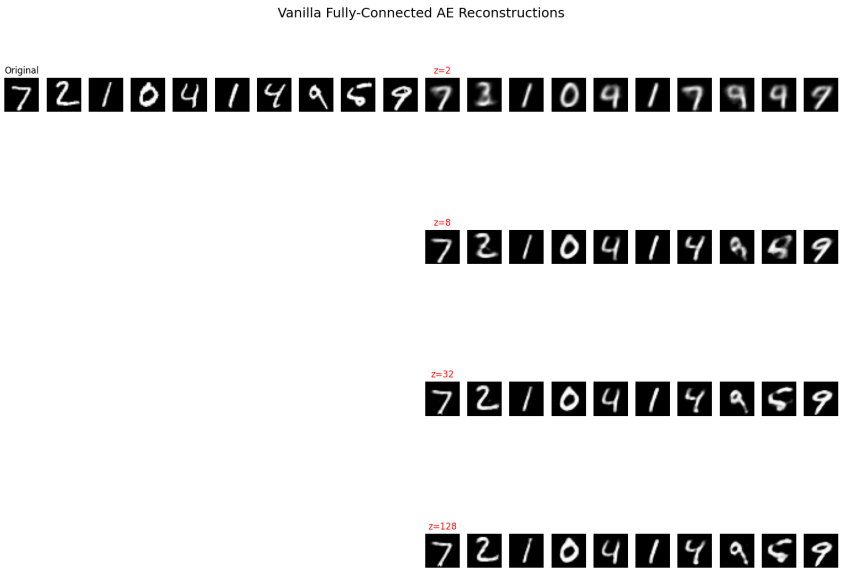

Figure 5: Reconstructions from the Vanilla Fully Connected Autoencoder.

3.2 Convolutional and Denoising Autoencoder Reconstruction

Figure 6 presents reconstructions for both Convolutional and Denoising Autoencoders. The Convolutional Autoencoder consistently outperforms the Vanilla Autoencoder because convolution layers exploit spatial structure, making them well-suited for image data. The Denoising Autoencoder shows similarly strong performance and returns smooth, high-quality reconstructions. Even at low z, the Denoising Autoencoder preserves digit boundaries and removes noise effectively.

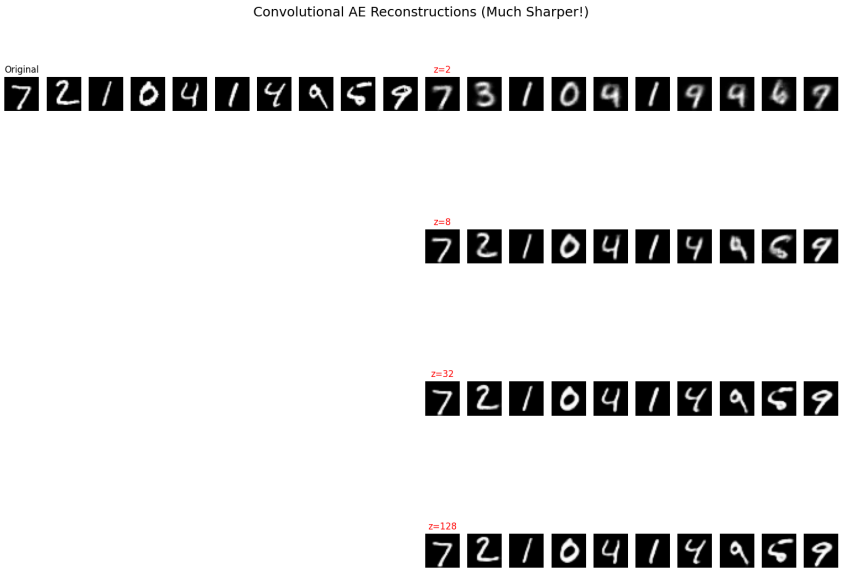

Figure 6: Reconstructions from the Convolutional Autoencoder and Denoising Autoencoder.

3.3 Reconstruction Error vs. Latent Dimension

To compare PCA and Autoencoders, Figure 7 plots test MSE for the three Autoencoder architectures alongside PCA. The Vanilla Autoencoder outperforms PCA at moderate z, but its improvement is limited because it lacks spatial awareness. The Convolutional Autoencoder performs significantly better, achieving an MSE of about 0.0049 at $z = 32$, compared to PCA's 0.0177 at the same dimension. The Denoising Autoencoder performs slightly worse than the normal Convolutional Autoencoder for very small z, but converges closely at larger latent sizes, demonstrating robustness.

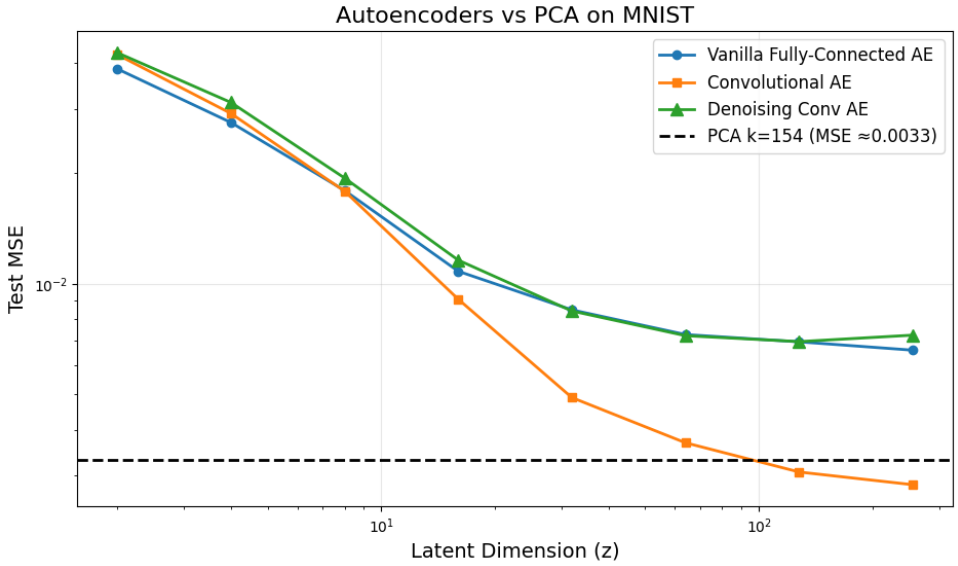

Figure 7: Test reconstruction error for PCA and all Autoencoder variants across different latent dimensions.

4. Conclusion

Across both datasets and all models, PCA proves to be extremely efficient for simple grayscale digits, providing fast and clean reconstructions for moderate values of k. However, PCA fails to capture the complexity of CIFAR-10 images, where the linear model cannot approximate nonlinear structures effectively. Autoencoders, particularly convolutional variants, provide superior reconstruction quality for the same latent dimension and are far more effective in modeling image structure. The Denoising Autoencoder additionally demonstrates that robust feature representations can be learned even under noisy conditions.

Overall, these experiments show that PCA remains a strong baseline for simple datasets and downstream tasks requiring speed. However, for complex image reconstruction and compression, convolutional autoencoders deliver significantly better results at much lower latent dimensions, offering a powerful alternative to linear dimensionality reduction techniques.

References

[1] Sandro Skansi. Autoencoders. In *Introduction to deep learning: From logical calculus to artificial intelligence*, pages 153–163. Springer, 2018.

[2] Quentin Fournier and Daniel Aloise. Empirical comparison between autoencoders and traditional dimensionality reduction methods. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 211–214. IEEE, 2019.

[3] Monica Casella, Pasquale Dolce, Michela Ponticorvo, and Davide Marocco. From principal component analysis to autoencoders: A comparison on simulated data from psychometric models. In *2022 IEEE International Conference on Metrology for Extended Reality, Artificial Intelligence and Neural Engineering (MetroXRAINE)*, pages 377–381. IEEE, 2022.

[4] Davide Cacciarelli and Murat Kulahci. Hidden dimensions of the data: Pca vs autoencoders. *Quality Engineering*, 35(4):741–750, 2023.

[5] Komal Bajaj, Dushyant Kumar Singh, and Mohd Aquib Ansari. Autoencoders based deep learner for image denoising. *Procedia Computer Science*, 171:1535–1541, 2020.