University Of Bretagne Occidentale
National Engineering School of Brest

UE Project: Report

Generative Autoencoding from Scratch

By

Yassine Boujelbene
Youssef Mahfoud

Academic Supervisor Pierre De Loor

Head of Master's Program Jérémy Rivière

Acknowledgements

We would like to extend our deepest gratitude to our jury members, **Mr.Jérémy Rivière** and **Mr.Pierre De Loor**, whose expertise, critical insights, and invaluable feedback have greatly enhanced the quality of this work. Their commitment to excellence in academic research has been a source of inspiration throughout this project.

We are especially grateful to our supervisor, **Mr.Pierre De Loor**, for his unwavering support, mentorship, and thoughtful guidance throughout the development of this work. His insightful advice and encouragement have been pivotal in navigating challenges and achieving our objectives.

We would also like to thank our families and friends for their patience, encouragement, and understanding as we dedicated our efforts to completing this project. Their unwavering support has been a constant source of strength and motivation.

Finally, we acknowledge the indirect contributions of countless others, whose assistance, discussions, and shared resources have enriched our learning experience. This work stands as a testament to the collaborative efforts and collective knowledge of the academic community. Thank you all.

Contents

List of Acronyms

MLP Multilayer Perceptron

CNN Convolutional Neural Network

VAE Variational Autoencoder

ReLU Rectified Linear Unit

Sigmoid Sigmoid Activation Function

Tanh Hyperbolic Tangent Activation Function

COCO Common Objects in Context

MNIST Modified National Institute of Standards and Technology

1 | Introduction

1.1 What are Autoencoders?

Autoencoders are a type of artificial neural network designed to learn efficient representations of data, primarily for unsupervised learning. The main objective of autoencoders is to compress the input data into a lower-dimensional latent space, after which the data is reconstructed back to its original form. This process encourages the network to learn important and meaningful features from the data, which can be utilized for tasks such as dimensionality reduction, anomaly detection, and generative modeling [1].

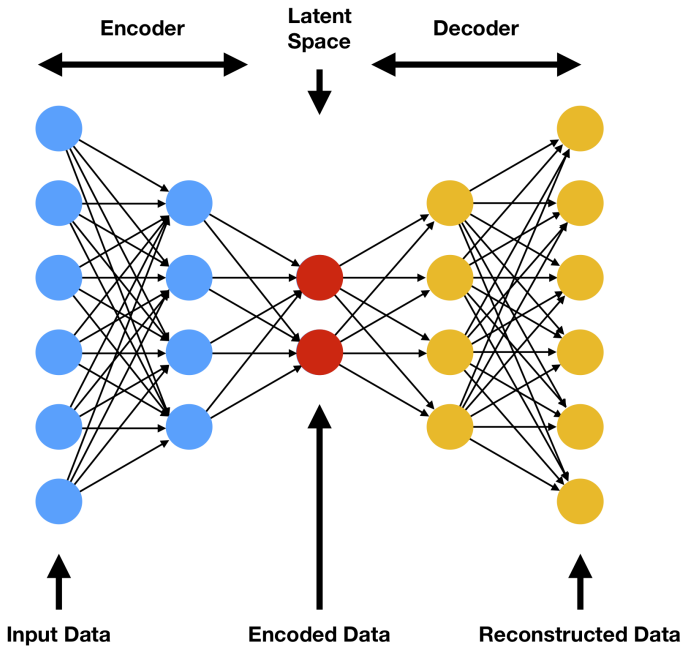

Figure 1.1: Autoencoder Architecture

1.2 Structure of Autoencoders

Autoencoders consist of two primary components:

- **Encoder:** The encoder compresses the input data into a latent representation, often referred to as the bottleneck or latent space. It learns a transformation function $h = f(x)$, where x represents the input data and h is the lower-dimensional latent vector. This step is generally achieved through a series of neural network layers.

- **Decoder:** The decoder takes the latent representation and reconstructs the original input using the transformation function $x' = g(h)$, where x' is the reconstructed input. The goal is for x' to closely resemble x, minimizing reconstruction error.

These components work together to optimize the reconstruction process, learning relevant features of the input data along the way [1].

1.3 Types of Autoencoders

Autoencoders are versatile neural networks that can be constructed in various ways to suit specific applications. Vanilla autoencoders, with fully connected encoder and decoder layers, focus on compact data representation for reconstruction [1]. Sparse autoencoders introduce sparsity constraints on the latent space, enhancing feature extraction and interpretability [2]. Denoising autoencoders train on corrupted inputs to reconstruct clean data, improving robustness against noise [3]. Convolutional autoencoders, leveraging convolutional layers, efficiently process spatial data such as images for tasks like compression and denoising. Variational autoencoders (VAEs) model latent representations probabilistically, enabling smooth latent space traversal and generating new data samples [4]. Contractive autoencoders add regularization to encourage invariance to small input changes [5]. Furthermore, stacked and deep autoencoders allow multi-layered architectures to learn hierarchical features for complex data, while conditional autoencoders incorporate auxiliary data to enable guided reconstruction and generation. Each design is built with specific configurations of layers, activation functions, and optimization strategies tailored to the underlying task. .

1.4 Challenges Addressed by Autoencoders

In this work, we focus on using autoencoders to address three main problems:

- **Reconstruction:** Autoencoders are used to compress input data into a latent representation and then reconstruct it back, maintaining the essential features.
- **Denoising:** Denoising autoencoders are employed to improve data robustness by training the model to clean noisy inputs and reconstruct them as noise-free outputs.
- **Conditional Generation:** Autoencoders are conditioned on additional labels or information to generate or modify data, offering fine-grained control over the generated outputs.

These three areas form the primary goals of our work, focusing on learning compact representations, improving data quality, and enabling more controlled data generation.

1.5 Working Methodology

In our collaborative approach, we structured the work effectively to ensure smooth task management and team synergy. Initially, we focused on developing a shared understanding of autoencoders, breaking down the core concepts to align our theoretical foundation. After that, we devised a detailed plan, where task distribution was based on model expertise and responsibilities. One team member worked on developing and optimizing the MLP (Multilayer Perceptron) architecture, while the other focused on the CNN (Convolutional Neural Network) for handling image data. Both members collaborated on the Variational Autoencoder (VAE), refining the probabilistic latent space for generative tasks.

We utilized tools such as Google Colab to work in real-time, committing code to a shared GitHub repository to maintain version control and track our progress. We also held regular coordination meetings on Discord to discuss updates, resolve challenges, and align on the next steps.

1.6 Report Structure and Outline

This report delves into the theory and application of autoencoders with a focus on three model types: MLPs, CNNs, and VAEs. The report covers their respective uses in reconstruction, denoising, and conditional generation tasks. Further, it details our methodology, dataset descriptions (MNIST and COCO), and compares model parameter variations for each task. We conclude with the insights, discussions, and potential advancements in autoencoder architectures.

1.7 Dataset Descriptions

We used two datasets: MNIST and COCO, chosen for their diversity and suitability for autoencoder testing:

- **MNIST:** A dataset of 70,000 28x28 pixel images of handwritten digits, ideal for testing basic models like MLPs [6].
- **COCO:** A collection of 200,000+ images across 80 categories with annotations, suitable for complex tasks like conditional generation using CNNs and VAEs [7].

  These datasets helped assess autoencoder performance on both simple and complex image data.

2 | MLP Implementation

2.1 Architectural Variations

The autoencoder models studied, based on Multilayer Perceptrons (MLPs), incorporate the principles we learned during the Master's course in Interactive Machine Learning, particularly in the design and training of neural networks. These models include the following components:

- **Encoder/Decoder Layers:** These layers define the network's capacity to learn and represent complex features. We experimented with shallow networks (two layers) and deeper networks (three layers).
- **Activation Functions:** We tested three activation functions:
  - ReLU
  - Sigmoid
  - Tanh

  These influence the network's ability to learn non-linear relationships in the data.
- **Latent Dimension:** The latent dimension controls the degree of compression. We explored dimensions from 2 to 64 to observe the trade-off between compression and image reconstruction quality.

2.2 Model Configurations

The following configurations were used to assess their impact on image reconstruction and denoising:

Model Name	Encoder Layers	Decoder Layers	Latent Dimension	Activation Function
Model 1	[500, 100]	[100, 500]	2	ReLU
Model 2	[500, 250, 100]	[100, 250, 500]	2	ReLU
Model 3	[500, 100]	[100, 500]	5	Sigmoid
Model 4	[500, 100]	[100, 500]	10	Tanh
Model 5	[500, 250, 100]	[100, 250, 500]	32	ReLU
Model 6	[500, 250, 100]	[100, 250, 500]	64	ReLU

Table 2.1: Autoencoder Model Configurations

2.3 Reconstruction

2.3.1 Encoder/Decoder Layers: Capacity, Generalization, and MNIST

Deeper networks, such as Model 2, exhibited better reconstruction performance than shallower networks like Model 1. Model 2, with its larger encoder/decoder layers, produced sharper, more detailed reconstructions, with fewer visible distortions. These improvements were corroborated by lower test loss, indicating better reconstruction accuracy. However, the difference in performance between the deeper Model 2 and the shallower Model 1 was minimal, showing that other factors, such as activation functions and latent dimension, had a greater influence on reconstruction quality. While deeper models may potentially introduce a risk of overfitting, we did not observe this in our experiments with the MNIST dataset. For more complex datasets, techniques like dropout or weight decay may be necessary to prevent overfitting.

2.3.2 Activation Function: Non-linearity and MNIST Characteristics

We compared ReLU, Sigmoid, and Tanh as activation functions. ReLU was effective for training, while Model 3, using Sigmoid activation, produced smoother and more visually accurate reconstructions. This result is likely due to Sigmoid's output range (0 to 1), which matches the normalized pixel values of MNIST images. While ReLU is a robust choice for hidden layers, Sigmoid might be a better option for the output layer in this scenario. Tanh, with a broader output range, did not show a significant advantage in this particular context.

2.3.3 Latent Dimension: Compression vs. Reconstruction Fidelity

Smaller latent dimensions, such as in Model 1 (dimension 2), created smoother reconstructions, but with some loss of detail. Larger latent dimensions, like those in Model 5 (dimension 32), preserved more intricate details while benefiting from a regularizing effect that discourages overfitting. Our results suggested that a latent dimension of 32 or 64 strikes a favorable balance between compression and reconstruction quality.

2.4 Denoising

2.4.1 Encoder/Decoder Layers and Activation Functions: Robust Feature Extraction for Denoising

We assessed how the architecture affects the autoencoder's ability to denoise images, specifically those corrupted with Gaussian, salt-and-pepper, and

speckle noise. Deeper models, such as Model 2, showed superior performance in denoising speckle noise, likely due to their capacity to learn more complex, abstract features of the data. These deeper models effectively separated signal from noise, particularly when handling the intricacies of speckle noise.

2.4.2 Activation Functions in Denoising

ReLU facilitated fast and effective denoising for speckle noise, likely because of its capacity to learn complex relationships between the signal and noise. Although other functions like Sigmoid and Tanh could theoretically perform better in certain noise environments, we found that ReLU yielded the best performance in this study. Our experiments showed that even with ReLU, Gaussian and salt-and-pepper noise remained more challenging to remove, suggesting that other approaches may be needed for those noise types.

2.4.3 Latent Dimension: Balancing Compression and Noise Removal

The latent dimension had a considerable effect on the autoencoder's ability to handle noise. For speckle noise, a smaller latent dimension (2 or 5) was effective for denoising by encouraging the model to discard noise while retaining important image features. Conversely, for Gaussian and salt-and-pepper noise, smaller latent dimensions, including 5, did not provide substantial benefits, and sometimes hindered denoising performance. This points to the importance of fine-tuning the latent dimension based on the type of noise being addressed. We found that a latent dimension of 2, and occasionally 5 (as seen in Model 3), was optimal for denoising speckle noise while maintaining important image details. For other noise types, alternative architectural adjustments might be necessary.

2.5 Incorporating One-Hot Labels in Autoencoders

2.5.1 Motivation

In this section, we introduce a **conditional element** into our existing autoencoder framework by concatenating a one-hot label vector $\mathbf{y}$ to the latent representation $\mathbf{z}$.

2.5.2 Changes to the Baseline Model

The principal modification to our previously discussed architecture is the label injection at the decoder input:

$$\tilde{\mathbf{z}} = [\mathbf{z}; \mathbf{y}] \tag{2.1}$$

Here, $\mathbf{z}$ is the latent vector learned by the encoder, and $\mathbf{y}$ is the one-hot encoding of the digit label. The concatenated vector $\tilde{\mathbf{z}}$ then serves as input to the decoder, enabling it to focus on label-appropriate characteristics during reconstruction.

2.5.3 Empirical Results

Convergence Dynamics

Shallow models, such as Models 1 and 2, converge rapidly and stabilize early; however, they result in a relatively high final reconstruction loss. This is primarily due to their lower latent dimensionality, which limits the encoder's capacity to store detailed information. In contrast, deeper models, including Models 3 through 6, require more epochs to converge but achieve a lower reconstruction loss. Their larger latent dimensions enable richer feature encoding, although this comes at the cost of increased training complexity.

Reconstruction Performance

Models with larger latent dimensions, such as Model 5 and Model 6, demonstrated superior reconstruction fidelity by leveraging their additional capacity to capture nuanced features of handwritten digits. For instance, Model 6, with a latent dimension of 64, excels at minimizing reconstruction error but requires more computational resources. In contrast, shallow, low-dimensional models like Model 1 and Model 2 exhibit higher reconstruction error. Their constrained latent space limits their ability to fully capture the variability in MNIST digits, even with label conditioning.

Generation Performance

Small latent spaces, as seen in shallow models, often produce coherent digits when sampling random latent vectors. The constrained latent space effectively regularizes itself, reducing the likelihood of generating "nonsense" vectors. However, large latent spaces in deeper models may result in less coherent samples unless additional regularization techniques, such as those from variational or adversarial frameworks, are applied. The increased degrees of freedom in these models can lead to "empty" regions in the latent space, causing random vectors to fall outside the learned manifold.

2.5.4 Interpretation and Conclusions

Despite adding a one-hot label to guide the decoder, the classic trade-off between reconstruction fidelity and generative coherence persists:

– High-dimensional, deeper models more effectively reconstruct detailed features, but their expansive latent spaces are more challenging to sample from reliably.
– Low-dimensional, shallow models achieve smoother sampling behavior but sacrifice reconstruction accuracy and detail.

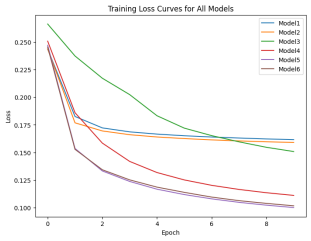

Figure 2.1: Loss function of models across epochs

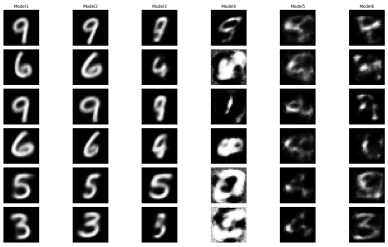

Figure 2.2: Generation performance across models

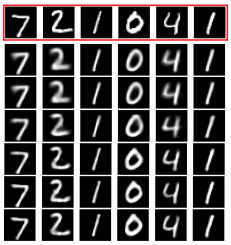

Figure 2.3: Reconstruction performance across models

2.5.5 Extension to the COCO Dataset

We also tested our six MLP models on the COCO dataset, down sampling images to 28×28 gray scale for compatibility. The results were unsatisfactory: MLPs struggled to encode the more complex spatial information, yielding severely distorted outputs. These findings confirm that pure MLPs are inadequate for spatially rich images, guiding us to shift toward CNN-based architectures for improved reconstruction fidelity.

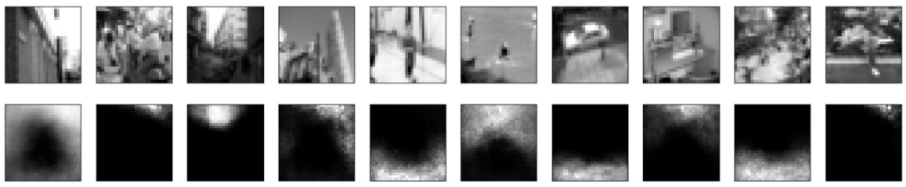

Figure 2.4: MLP on COCO

3 | CNN Implementation

3.1 Introduction

Given the limitations of MLP architectures in handling the spatial complexity of COCO images, we transitioned to convolutional neural networks (CNNs) to exploit their inherent ability to model spatial hierarchies. Unlike MLPs, which flatten inputs and lose spatial structure, CNNs retain the two-dimensional arrangement of pixels, making them well-suited for reconstructing spatially rich images. In this study, we focused exclusively on the denoising task as a simplified approach to evaluate the effectiveness of the autoencoder architectures.

3.1.1 Initial CNN Autoencoder Architecture

The initial CNN autoencoder was designed with an encoder-decoder structure. The encoder consisted of three convolutional layers with a kernel size of 3×3 and a stride of 2, progressively reducing spatial dimensions from 64×64 to 8×8. ReLU activations and Batch Normalization were applied to improve stability and convergence. A fully connected layer mapped the features to a compact latent representation, enabling the model to encode meaningful features of the input images.

The decoder mirrored the encoder's structure, starting with a fully connected layer reshaping the latent vector into a tensor of size $128 \times 8 \times 8$. This was followed by three transposed convolutional layers, reconstructing the spatial dimensions back to 64×64. The final layer employed a Tanh activation function to normalize the outputs to the range $[-1, 1]$. Initial experiments with this architecture showed that increasing the latent space size from 16 to 128 significantly improved reconstruction quality, reducing **the Mean Squared Error (MSE) from 0.0895 to 0.0420.**

3.1.2 Transition to a Fully Convolutional Design

To further refine the architecture, the flattening step in the encoder was removed, and decoding was performed directly from convolutional feature maps. The encoder utilized MaxPooling layers to downsample spatial dimensions, while the decoder employed MaxUnpooling layers with

stored pooling indices to upsample dimensions. This design eliminated the need for dense layers, preserving spatial structure throughout the network. Additionally, ReLU activations were replaced with Tanh to improve gradient flow.

This fully convolutional architecture reduced the **MSE to 0.0113** and increased the Peak Signal-to-Noise Ratio (PSNR) to 19.5258, indicating significant improvements in reconstruction quality. The preservation of spatial relationships in the feature maps proved crucial for denoising performance.

3.1.3 Enhancing the Bottleneck and Adding Skip Connections

In the next iteration, the number of features in the bottleneck layer was increased from 64 to 128, allowing the encoder to capture more detailed representations of the input images. Skip connections were also introduced between the encoder and decoder stages. These connections enabled the decoder to directly access feature maps from corresponding encoder layers, enhancing spatial information retention and reconstruction fidelity.

The encoder reduced spatial dimensions from 64×64 to 8×8 using three convolutional layers with a kernel size of 3×3, stride 2, and padding 1. The decoder upsampled the dimensions back to 64×64 using three transposed convolutional layers with a kernel size of 3×3, stride 2, and output padding 1. With these enhancements, the model achieved an **MSE of 0.0033**, marking a substantial improvement in reconstruction accuracy.

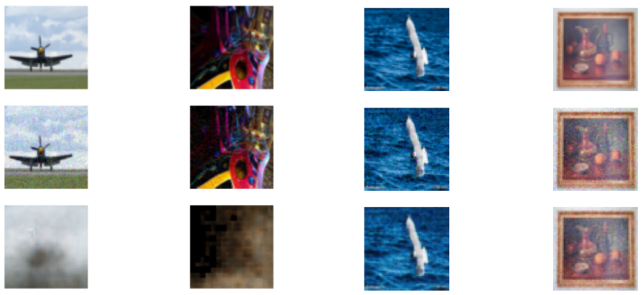

Figure 3.1: Initial architecture vs. last one Reconstructions

3.1.4 Challenges and Future Directions

Despite the low loss value of 0.0033, visual inspection of the reconstructed images revealed that the outputs were not entirely perfect, with some details still missing or distorted. This suggests that further work is required to achieve truly high-fidelity reconstructions.

Future directions may include refining the network architecture with advanced techniques such as attention mechanisms or residual blocks to

enhance feature extraction and reconstruction. Alternatively, other types of autoencoders, such as Variational Autoencoders (VAEs) or Generative Adversarial Networks (GANs), could be explored for further improvements in reconstruction quality. While significant progress has been made, achieving optimal performance remains an ongoing challenge. Importantly, expanding beyond denoising tasks to explore other image reconstruction applications, such as inpainting or super-resolution, could provide further insights into the robustness of the developed architectures.

Bibliography

[1] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[2] Andrew Ng. Sparse autoencoders. 2011. https://deeplearning.stanford.edu/tutorial/sparse-autoencoders/.

[3] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Alexandre Manzagol. Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103, 2008.

[4] D.P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2014.

[5] S. Rifai, P. Vincent, X. Muller, and B. Bengio. Contractive auto-encoders: explicit invariance during feature extraction. *Proceedings of the 28th International Conference on Machine Learning*, page 833–840, 2011.

[6] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

[7] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, PJ Dollar, C Lawrence Zitnick, and David Forsyth. Microsoft coco: Common objects in context. *Proceedings of the European Conference on Computer Vision*, page 740–755, 2014.