

# AN OVERVIEW OF AUTOENCODER ARCHITECTURES WITH A COMPARATIVE STUDY OF VANILLA AND CONVOLUTIONAL VARIANTS

Gökhan Karabıyık <sup>1\*</sup> and Zsolt Csaba Johanyák <sup>2,3</sup>

<sup>1</sup> Department of Management of Information Systems, Institute of Social Sciences, Aksaray University, Türkiye, <https://orcid.org/0000-0002-4678-394X>

<sup>2</sup> Department of Information Technology, GAMF Faculty of Engineering and Computer Science, John von Neumann University, Hungary, <https://orcid.org/0000-0001-9285-9178>

<sup>3</sup> Institute of Mechatronics and Vehicle Engineering, Bánki Donát Faculty of Mechanical and Safety Engineering, Óbuda University, Hungary  
<https://doi.org/10.47833/2025.2.CSC.001>

---

## Keywords:

autoencoder  
vanilla autoencoder  
convolutional autoencoder  
denoising autoencoder

## Article history:

Received 28 June 2025  
Revised 10 July 2025  
Accepted 9 July 2025

---

## Abstract

*Autoencoders have become a fundamental tool in unsupervised learning, addressing various challenges such as dimensionality reduction, denoising, anomaly detection, and generative modeling. At their core, autoencoders consist of an encoder that compresses input data into a lower-dimensional representation and a decoder that reconstructs the original input. While standard autoencoders are effective for feature extraction, they suffer from generalization issues, leading to the development of specialized variants.*

*This paper provides an overview of several autoencoder types, including Denoising Autoencoders (DAEs) that enhance robustness against noise, Variational Autoencoders (VAEs) that introduce probabilistic modeling, Sparse Autoencoders that enforce feature selectivity, Contractive Autoencoders (CAEs) that ensure stability against small input changes, Adversarial Autoencoders (AAEs) that integrate generative adversarial training, Convolutional Autoencoders (CAEs) optimized for image processing, and Sequence-to-Sequence Autoencoders designed for sequential data. Each variant offers unique advantages for specific machine learning tasks. Additionally, compression was implemented using vanilla and convolutional autoencoders, and the results were evaluated. These autoencoder types were chosen because they are widely used in compression.*

---

## 1 Introduction

In recent years, autoencoders have emerged as a foundational component in unsupervised and self-supervised learning, playing a pivotal role in tasks such as dimensionality reduction, denoising, anomaly detection, and generative modeling [1]. Originally introduced as a simple neural network architecture that learns to reconstruct its input through a compressed latent representation, autoencoders have evolved into a diverse family of models tailored to handle a variety of data types, constraints, and learning objectives.

---

\* Corresponding author.  
E-mail address: [gokhan.karabiyik@asu.edu.tr](mailto:gokhan.karabiyik@asu.edu.tr)

At the heart of every autoencoder lies a dual-stage architecture: an encoder that compresses input data into a low-dimensional latent space, and a decoder that reconstructs the input from this compact representation. The reconstruction objective forces the model to capture the most salient features of the data, making autoencoders effective tools for feature learning. However, as the complexity of data and tasks has increased, the need for more specialized autoencoder variants also appeared.

To address limitations of the basic autoencoder such as poor generalization, overfitting, or inability to model probabilistic distributions researchers have developed a wide range of autoencoder types. These include denoising autoencoders (DAEs), which improve robustness by learning to reconstruct clean inputs from noisy versions; variational autoencoders (VAEs), which incorporate probabilistic inference and have become central to generative modeling; sparse and contractive autoencoders, which enforce constraints to encourage interpretability or robustness; and adversarial autoencoders (AAEs), which integrate generative adversarial networks (GANs) to regularize latent representations. Each variant is designed to fulfill specific learning objectives or to enhance generalization in different ways.

This paper presents a comprehensive overview of the different types of autoencoders, categorizing and comparing these models. We aim to provide a clearer understanding of how autoencoders have adapted to modern machine learning challenges and where their future potential lies.

## 2 Autoencoders and types

Autoencoders are neural network architecture designed to learn efficient representations (encoding) of input data in an unsupervised manner. Their core functionality involves compressing data into a lower-dimensional latent space and subsequently reconstructing the original input from this representation (decoding) [1]. Formally, the encoder function  $f_\theta(x) = z$  maps an input  $x$  to a latent code  $z$ , and the decoder  $g_\phi(z) = \hat{x}$  attempts to reconstruct the input. The training objective is to minimize the reconstruction loss  $L(x, \hat{x})$ , typically measured using mean squared error (MSE) or cross-entropy loss, depending on the nature of the input data.

Over time, multiple variants of autoencoders have been introduced to address limitations such as poor generalization, lack of structure in latent spaces, and sensitivity to noise. These variants include sparse, denoising, variational, contractive, adversarial, convolutional, and sequence-to-sequence autoencoders, each with unique architectures and use cases. Clustering algorithms such as K-means are frequently used to evaluate the quality of latent representations learned by autoencoders, especially in unsupervised settings where class labels are not available [2].

### 2.1 Basic autoencoder (Vanilla)

The standard autoencoder is composed of a symmetric feedforward neural network with an encoder and decoder. It is trained to minimize the reconstruction loss between the input and output. While simple, this model forms the basis for more advanced variants. It has some limitations, i.e. it tends to memorize the data if the latent space is too large, it has poor generalization capability to unseen data and it does not model the underlying data distribution.

Additionally, the reliance on pixel-wise reconstruction loss can cause the model to overlook high-level semantic features, resulting in visually accurate but semantically meaningless outputs. Recent studies have explored alternative loss functions and latent space regularizations to address these issues and improve generalization performance [3].

### 2.2 Denoising autoencoder (DAE)

Denoising autoencoders are a variation of standard autoencoders designed to clean up noisy data typically found in real-world datasets [4]. A DAE improves robustness by learning to reconstruct clean inputs from noisy versions. This allows the model to focus on essential features and ignore noise, which makes it helpful for tasks like the image and speech denoising. DAEs are a form of neural network that aim to learn compact representations of data without supervision. Their main goal is to find meaningful encodings of the input data, often for tasks like dimensionality reduction, by enforcing that the input can be reconstructed from its encoded form. What sets denoising autoencoders apart

from traditional autoencoders is their ability to recover the original input from a noisy or altered version, thereby effectively learning how to eliminate noise from the data.

Denoising involves training the autoencoder to focus on the essential characteristics of the data while disregarding irrelevant noise. This is done by compelling the model to emphasize the most critical aspects during reconstruction. Consequently, the network gains resilience against noisy inputs. A denoising autoencoder shares a similar architecture with a conventional autoencoder. It includes an input layer, a set of hidden layers that make up the encoder, a bottleneck layer that holds the compressed representation, followed by hidden layers that form the decoder, and finally an output layer. The main distinction lies in the training phase, where the input is intentionally altered with noise before being processed by the network.

The type and amount of noise introduced can significantly impact the model's effectiveness and often needs to be carefully adjusted. Similar to other neural network models, training denoising autoencoders can demand substantial computational resources, particularly when dealing with large-scale datasets or intricate model structures. Although they are capable of filtering out noise, improper regularization may lead to the loss of important details or useful information within the data.

### **2.3 Sparse autoencoder (SAE)**

Sparse autoencoders enforce sparsity on the latent representation, meaning only a small number of neurons are activated for a given input [5]. This is achieved by adding a sparsity penalty term to the loss function, such as Kullback-Leibler divergence between the average activation and a small target value. Key features include encouraging the learning of independent and interpretable features, mimicking the behavior of biological neurons, and being useful in high-dimensional, low-sample-size scenarios. Applications include feature extraction for text and images, and unsupervised pre-training for deep networks.

### **2.4 Contractive autoencoder (ContAE)**

The contractive autoencoder introduces a penalty on the Jacobian matrix of the encoder activations with respect to the input. This encourages the model to learn representations that are robust to small variations in input. Key features include emphasizing local invariance, learning stable and robust embeddings, and being useful for manifold learning. Applications include representation learning in noisy environments, anomaly detection, where sensitivity to noise in input data must be minimized [6].

### **2.5 Variational autoencoder (VAE)**

The variational autoencoder is a probabilistic extension that models the latent variables as a distribution rather than a point estimate. Instead of encoding an input to a single latent point, VAEs learn the parameters of a probability distribution (typically Gaussian), and sample from this distribution to reconstruct the input [7]. Key features include incorporating Bayesian inference into autoencoders, learning a generative model of data, and having a latent space with a continuous, smooth structure. The loss consists of a reconstruction term and a Kullback-Leibler divergence term that regularizes the latent distribution toward a prior. Applications include generative modeling (e.g., image synthesis), anomaly detection, and semi-supervised learning.

### **2.6 Adversarial autoencoder (AAE)**

The adversarial autoencoder combines the architecture of an autoencoder with the adversarial training principle from GANs [1]. In AAEs, a discriminator network tries to distinguish between the latent vectors and samples from a target distribution, while the encoder attempts to fool it, thereby aligning the encoded distribution with a desired prior. Key features include enforcing desired distributions in the latent space, providing a flexible and modular framework, and combining the benefits of VAEs and GANs. Applications include domain adaptation, data generation with controlled semantics, and privacy-preserving learning.

## 2.7 Convolutional autoencoder (CAE)

Convolutional autoencoders are specialized for image data [8]. They replace fully connected layers with convolutional and deconvolutional layers to better capture spatial hierarchies and reduce parameter count. Key features include preserving spatial structure in images, being effective for image denoising and inpainting, and working directly with raw image pixels. Applications include medical imaging, satellite imagery compression, and image segmentation pre-training.

## 2.8 Sequence to sequence autoencoder

Sequence-to-sequence autoencoders are a type of autoencoder architecture designed to handle sequential data, where both the input and output are sequences (e.g., text, time series, or audio) [1]. Sequence-to-sequence autoencoders use encoder-decoder architectures to capture temporal dependencies, handle variable-length sequences, and learn compact representations of sequential data. Applications include machine translation, speech recognition, time-series forecasting, and text summarization.

## 2.9 Sinkhorn autoencoder (SAE)

The sinkhorn autoencoder (SAE) is a generative model that combines the structure of a traditional autoencoder with the principles of optimal transport to learn meaningful latent representations [9]. Unlike variational autoencoders, which impose a probabilistic prior through KL divergence, SAE enforces alignment between the encoded latent distribution and a target prior using the Sinkhorn divergence—a differentiable approximation of the Wasserstein distance. This approach enables more precise control over the geometry of the latent space, facilitating better distribution matching without requiring reparameterization tricks. SAE has demonstrated strong performance in tasks such as representation learning and generative modeling, particularly excelling in maintaining the structure and diversity of complex datasets.

Patrini et al. [9] evaluated the performance of the Sinkhorn AutoEncoder (SAE) model across various tasks and datasets. The model's effectiveness was assessed in representation learning, data reconstruction, and generative modeling. The study highlighted SAE's advantages over other autoencoder variants by comparing it with Variational Autoencoders (VAE), Wasserstein Autoencoders (WAE), Sliced Wasserstein Autoencoders, Hungarian Autoencoders, and Hyperspherical VAEs. Experiments were conducted on large-scale datasets, including handwritten digits and human face images. Reconstruction Error (RE) and Fréchet Inception Distance (FID) were used as evaluation metrics. The results demonstrated that SAE offers flexible and powerful architecture based on optimal transport principles. In particular, it outperformed other methods in aligning distributions in the latent space, contributing to more robust and realistic generative performance.

## 3 Comparative study of Vanilla and Convolutional autoencoders

In our investigation, image compression was performed using two types of autoencoder architectures: the vanilla autoencoder (AE) and the convolutional autoencoder (CAE). The application was developed in Python, and experiments were conducted on the MNIST dataset—a widely used benchmark for handwritten digit recognition introduced by LeCun et al. [10]. The dataset consists of grayscale images of handwritten digits, serving as a standard for evaluating the performance of image processing models.

A comparative analysis was carried out between the AE- and CAE-based implementations. The Python source codes for both models are presented in Table 1. The original handwritten images and their compressed reconstructions using both methods are shown below, enabling visual inspection of reconstruction quality.

During the experiments, the AE application completed execution in approximately one minute, whereas the CAE implementation required around 43 minutes. This significant difference in running time suggests that AE is preferable when speed and computational efficiency are prioritized. However, when higher reconstruction fidelity and visual quality are required, CAE proves to be the

more effective choice. Upon visual inspection, the CAE-reconstructed images appear more similar to the original inputs.

*Table 1. Codes Used in Applications*

Vanilla Autoencoder	Convolutional Autoencoder
<pre> import numpy as np import matplotlib.pyplot as plt from tensorflow.keras.datasets import mnist from tensorflow.keras.models import Model from tensorflow.keras.layers import Input, Dense from tensorflow.keras.utils import plot_model (x_train, _), (x_test, _) = mnist.load_data() x_train = x_train.astype('float32') / 255. x_test = x_test.astype('float32') / 255. x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:]))) x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:]))) input_img = Input(shape=(784,)) encoded = Dense(32, activation='relu')(input_img) decoded = Dense(784, activation='sigmoid')(encoded) autoencoder = Model(input_img, decoded) encoder = Model(input_img, encoded) autoencoder.compile(optimizer='adam', loss='binary_crossentropy') autoencoder.fit(x_train, x_train,                 epochs=20,                 batch_size=256,                 shuffle=True,                 validation_data=(x_test, x_test)) encoded_imgs = encoder.predict(x_test, verbose=0) decoded_imgs = autoencoder.predict(x_test, verbose=0) n = 10 plt.figure(figsize=(20, 4)) for i in range(n):     ax = plt.subplot(2, n, i + 1)     plt.imshow(x_test[i].reshape(28, 28), cmap='gray')     plt.title("Original")     plt.axis('off')     ax = plt.subplot(2, n, i + 1 + n)     plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')     plt.title("Reconstruction")     plt.axis('off') plt.show() </pre>	<pre> import numpy as np import matplotlib.pyplot as plt from tensorflow.keras.datasets import mnist from tensorflow.keras.models import Model from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D import tensorflow.keras.backend as K (x_train, _), (x_test, _) = mnist.load_data() x_train = x_train.astype('float32') / 255. x_test = x_test.astype('float32') / 255. x_train = np.reshape(x_train, (len(x_train), 28, 28, 1)) x_test = np.reshape(x_test, (len(x_test), 28, 28, 1)) input_img = Input(shape=(28, 28, 1)) x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img) x = MaxPooling2D((2, 2), padding='same')(x) x = Conv2D(16, (3, 3), activation='relu', padding='same')(x) x = MaxPooling2D((2, 2), padding='same')(x) x = Conv2D(16, (3, 3), activation='relu', padding='same')(x) x = UpSampling2D((2, 2))(x) x = Conv2D(32, (3, 3), activation='relu', padding='same')(x) x = UpSampling2D((2, 2))(x) decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x) autoencoder = Model(input_img, decoded) autoencoder.compile(optimizer='adam', loss='binary_crossentropy') autoencoder.fit(x_train, x_train, epochs=20, batch_size=256, shuffle=True, validation_data=(x_test, x_test)) decoded_imgs = autoencoder.predict(x_test) n = 10 plt.figure(figsize=(20, 4)) for i in range(n):     ax = plt.subplot(2, n, i + 1)     plt.imshow(x_test[i].reshape(28, 28), cmap='gray')     plt.title("Original")     plt.axis('off')     ax = plt.subplot(2, n, i + 1 + n)     plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')     plt.title("Reconstructed")     plt.axis('off') plt.show() </pre>

The reason why the two autoencoders under study have different runtimes despite being used for the same purpose is due to the different layer structures of the models. AE consists of only fully

connected (dense) layers and processes the input data as one-dimensional vectors. This architecture is quite simple and fast. In contrast, the CAE model includes more complex layers such as Conv2D, MaxPooling2D, and UpSampling2D. These layers require significantly more computation, especially for processing the spatial portions. Secondly, the data format and computational intensity are factors that affect the processing time. While the vanilla autoencoder processes the data by converting them into 784-dimensional vectors, the convolutional autoencoder keeps the data (28x28x1) organized, and this structure is maintained across layers. This increases the number of applications it performs on an input sample, thus extending the training time. Finally, although CAE has fewer parameters, this does not make it faster. The total computational loss is not only the number of parameters but also the amount of computation performed by the layers. Convolutional layers perform more processing due to the floating filter values on the image. All this results in a large difference between the training conditions of these two models, which are used for similar data.

While AE and CAE are used for a similar purpose—reconstructing data by compressing it—they differ significantly in their architecture and resource usage. These differences are particularly notable in terms of the number of parameters, training time, and the efficient use of computational resources. AE consists of fully connected (dense) layers and processes the input image by flattening it (a 784-dimensional vector). This structure ignores spatial information, particularly for image data. However, thanks to its simplicity, this model requires less computation and can run faster on the CPU. However, the vanilla autoencoder has a significantly higher parameter count; even with only two layers, it contains approximately 50,000 parameters.

On the other hand, CAE processes the image in 2D and preserves spatial relationships thanks to convolutional layers. While this model's architecture may appear more complex, its total parameter count is approximately 12,000. This means it has approximately four times fewer parameters than the vanilla model. This makes the model lighter and more memory-efficient. The computational load of CAE is higher than that of the vanilla model due to convolutional operations. However, this difference becomes an advantage when running on a GPU, as convolutional operations are more suitable for parallel computing.

When the structure of the AE code fragment written above is examined, it is aimed to compress the handwritten digits in the MNIST dataset and then reproduce the original image from this compressed representation. The dataset used consists of grayscale images, each 28x28 pixels in size. In order for the model to work, these two-dimensional images were first flattened and converted into one-dimensional vectors. Each image thus became a 784-dimensional vector. In addition, pixel values were normalized between 0 and 255 and pulled between 0 and 1. In this way, more efficient operation of the functions was ensured.

First, an input layer was defined in the architecture of the model. After the input, a compression layer called the "encoder" section was created. This layer consists of a Dense (fully connected) layer and contains only 32 neurons. The input data was compressed into this 32-dimensional space using the ReLU activation function. This 32-dimensional vector is the "encoded" version of each input image; in other words, it is a smaller but meaningful representation of the input data. From this encoded data, the original 784-dimensional image is attempted to be recreated via the "decoder" section of the model. The decoder layer also consists of a Dense layer and uses a sigmoid activation function at the output. Since the sigmoid function limits the output between 0 and 1, it is a suitable choice to reproduce normalized pixel values. The model is trained so that the input and output data are the same. This is the typical learning method of autoencoders: the model is given an input and the same input is expected as the target (output). In other words, the model tries to reproduce its own input. During training, the model measures how similar the output is to the original using the binary cross entropy loss function. The Adam algorithm was preferred for optimization, which is a powerful method that is widely used today.

After the model was trained, the encoder model was defined as a separate structure in the code. In this way, it was possible to obtain only the encoded representations of the test data. In addition, the test data was first compressed by passing it through the encoder, and then these representations were reconstructed with the autoencoder model. Thus, it was possible to observe how well the representations learned by the model could represent the original.

Finally, both the original and the reconstructed images were plotted side by side with the matplotlib library. Thanks to this visualization, it is possible to concretely evaluate how successful the model is. If there is a loss of clarity or lack of detail between the images, this indicates that the model needs deeper structures or different architectures. In short, although this autoencoder offers a very simple structure, it is quite functional for understanding the basic principles and for introducing low-dimensional representation learning. Depending on the applications to be made, this structure can be deepened with more layers if desired.

This type of data means that each pixel can be interpreted as a probability (the closer to 0, the blacker, and the closer to 1, the whiter). Therefore, it is common to use a sigmoid activation function in the output layer, and a binary cross entropy loss function is preferred accordingly. Binary cross entropy measures how closely the model's outputs can approximate the target values by treating each pixel as an independent binary classification problem. This loss function allows for more precise and detailed optimization, especially when per-pixel accuracy is required. Furthermore, a loss function such as the MSE is preferred when continuous values are clearer and the distribution is wider, while binary\_crossentropy performs better on low-intensity images normalized between 0 and 1. This loss function is more suitable for datasets that are simple and close to binary color distribution, such as MNIST.

### 3.1 Performance evaluation

The Structural Similarity Index (SSIM) is a metric that quantitatively measures the structural similarity between two images. SSIM takes into account luminance, contrast, and structure. In this respect, it provides an assessment closer to human visual perception than metrics like MSE and PSNR, which rely solely on pixel-by-pixel differences. In the AE model, the data is flattened to (num\_samples, 784). For the SSIM calculation, these vectors are reshaped to 28x28 pixels. In the CAE model, the data is (num\_samples, 28, 28, 1). In this case, the last dimension is removed to make it suitable for SSIM calculations. SSIM calculations in Python are performed using the structural\_similarity() function in the scikit-image library. This function works with the following parameters:

```
from skimage.metrics import structural_similarity as ssim  
score = ssim(original_img, reconstructed_img, data_range=1.0)
```

To measure the overall performance of the model, the SSIM value was calculated for each image in the test set and these values were averaged. The results show that CAE's SSIM scores are higher because it learns spatial relationships through convolutional layers, producing outputs that are structurally closer to the original image. However, AE, by flattening the input data, fails to fully preserve the overall structure, resulting in a lower SSIM score compared to CAE.

The output was also evaluated using the MSE (Mean Squared Error) metric, one of the most common reconstruction error metrics. AE's failure to consider the input data in its entirety and to consider the overall structure, particularly in image data, leads to the model's inability to learn the spatial organization of the input data. This generally results in higher MSE values. In contrast, CAE uses convolutional and pooling layers to learn local features of the data. This structure is particularly sensitive to the relationships between pixels and provides more successful reconstructions by preserving the overall structure of the image. Therefore, in the application, CAE was observed to produce lower MSE than AE, indicating that the model reconstructed the input data more accurately. While the average MSE value of AE on the test data was around 0.0145, CAE trained on the same data reduced this error to 0.0078. These results demonstrate that CAE can learn both more efficient and more accurate representations in visual data.

To better understand the study results, a comparison was made using the PSNR (Peak signal-to-noise ratio) technique. PSNR measures the quality between the original and reconstructed data, and a higher PSNR value indicates better image quality. Significant performance differences were found between these two applications based on the PSNR metric. The AE application's PSNR values ranged from 19 to 22 dB, while the CAE application measured between 25 and 30 dB. This difference demonstrates that the convolutional autoencoder is more effective in improving visual quality.

## 4 Discussion

The comparative evaluation of vanilla and convolutional autoencoders presented in this study reveals notable trade-offs between model simplicity, training efficiency, and reconstruction fidelity. Vanilla autoencoders, owing to their fully connected architecture, offer computational efficiency and ease of implementation. Their ability to perform dimensionality reduction in a straightforward manner makes them suitable for rapid prototyping and applications where low latency is a priority. However, the inherent limitation of flattening input images before feeding them into dense layers results in a loss of spatial information, which significantly impacts reconstruction quality—particularly for data types like images, where spatial structure is crucial.

In contrast, the convolutional autoencoder demonstrated superior performance in preserving visual characteristics and fine-grained spatial details. This improvement can be attributed to the localized nature of convolutional filters, which allow the network to learn hierarchical feature representations directly from raw pixel grids. The use of max-pooling and upsampling layers further enables efficient compression and reconstruction without discarding critical spatial relationships. The higher SSIM scores obtained by the convolutional model validate its effectiveness in maintaining visual fidelity. These findings align with the broader consensus in the literature, where convolutional architectures are considered the preferred approach for image-centric tasks. Nevertheless, the trade-off in computational demand must be considered. The convolutional autoencoder required significantly longer training times compared to its vanilla counterpart. This increased complexity may not be justified in scenarios where real-time processing or limited hardware resources constrain system performance. Therefore, the choice between vanilla and convolutional autoencoders should be context-dependent—guided by the specific demands of the task, the nature of the data, and available computational resources.

Additionally, while both models were evaluated using the MNIST dataset, which is relatively clean and low in complexity, further studies on more challenging datasets—such as CIFAR-10 or real-world noisy images—could better highlight the robustness advantages of convolutional structures. Moreover, integrating noise during training, as in denoising autoencoders, or enforcing latent space regularization, as in variational or adversarial autoencoders, may further improve generalization and enhance performance in downstream tasks.

Ultimately, the experimental outcomes emphasize the importance of architectural choices in representation learning. Future work could explore hybrid architectures that combine the strengths of both dense and convolutional layers, or investigate the incorporation of attention mechanisms to dynamically focus on salient regions in the input. As autoencoders continue to evolve, their adaptability and modularity will remain key factors in addressing increasingly complex and domain-specific machine learning challenges.

## 5 Conclusion

Autoencoders have emerged as a vital component in modern machine learning, offering robust solutions for tasks such as dimensionality reduction, denoising, anomaly detection, and generative modeling. As data and task complexity have grown, various autoencoder architectures have been developed to address challenges in feature extraction and representation learning. This study provided an overview of several autoencoder variants, each with distinct advantages suited to specific applications, highlighting the flexibility of autoencoders in handling diverse data types and problem domains.

Beyond static applications, autoencoders also show promise in dynamic and evolving environments. Their ability to learn compact and adaptive representations of high-dimensional data can complement cloud-based identification strategies in real-time systems, as demonstrated by Blazic et al. [11]. Similarly, in IoT-based robotic systems, convolutional autoencoders are well-suited for efficiently processing sensory data, potentially enhancing reactive navigation systems that rely on fuzzy cognitive maps for decision-making, such as those described by Vaščák et al. [12].

The comparative analysis between vanilla and convolutional autoencoders in the context of image compression further underscores the importance of aligning model choice with application needs. While vanilla autoencoders excel in computational efficiency and quick training times, convolutional autoencoders achieve greater reconstruction fidelity by capturing intricate spatial



patterns, as reflected in higher Structural Similarity Index (SSIM) scores. These findings suggest that selecting the appropriate autoencoder architecture should depend on whether the primary objective is speed or precision.































In addition to their effectiveness in image compression and feature learning, autoencoders have the potential to support system-level analysis in fields such as diagnostics and reliability engineering. For instance, simulation-based approaches like the Monte Carlo method have been successfully applied to evaluate the reliability of systems with complex interconnections [13]. Integrating such simulation frameworks with data-driven representation learning models, including autoencoders, could offer promising avenues for predictive maintenance and robust system modeling.

Fundamental structures currently in use, such as the vanilla autoencoder and convolutional autoencoder (CAE), lay significant groundwork for future advancements in artificial networking [14] [15]. These models have become powerful tools not only for data compression and repartitioning, but also for providing deep learning solutions in numerous application domains. More advanced and specialized versions of such architectures are expected to be widely used in the future, particularly in areas such as cybersecurity, medical diagnosis, autonomous driving, and digital art creation. In cybersecurity, autoencoders offer an effective method for identifying anomalies. A model that learns from normal system or network changes can identify anomalous behavior, enabling early detection of malware, spoofing, or logging attacks.

In the medical field, convolutional autoencoders can be used to minimize medical noise, produce high-quality images, and detect abnormal textures. This could play a significant role in generating early diagnostic results and increasing the reliability of AI-assisted diagnostic applications. In autonomous driving technologies, CAEs can improve geometric perception by more accurately processing sensor isolation. Reducing noise in LIDAR and camera recordings allows for safer real-time decision-making. These structures can also be used to detect unexpected driving behaviors or sensor malfunctions.

In the field of digital art and creative content, autoencoder architectures enable the development of a new visual language. These structures offer highly effective solutions for tasks such as style transfer, image completion, and the adjustment of low-quality media content. Especially when integrated with generative models, it is possible to manipulate the features of original artworks or personalized content.

Table 2. Original and Derived Images

Original Image	AE Predictions	CAE Predictions
		
		
		
		
		
		
		
		
		
		

## References

- [1] W. H. Lopez Pinaya, S. Vieira, R. Garcia-Dias, and A. Mechelli, "Autoencoders," in *Machine Learning*, Elsevier, 2020, pp. 193–208. doi: 10.1016/B978-0-12-815739-8.00011-0.
- [2] I.-D. Borlea, R.-E. Precup, F. Dragan, and A.-B. Borlea, "Centroid Update Approach to K-Means Clustering," *Adv. Electr. Comp. Eng.*, vol. 17, no. 4, pp. 3–10, 2017, doi: 10.4316/AECE.2017.04001.
- [3] S. Dias Da Cruz, B. Taetz, T. Stifter, and D. Stricker, "Autoencoder and Partially Impossible Reconstruction Losses," *Sensors*, vol. 22, no. 13, p. 4862, Jun. 2022, doi: 10.3390/s22134862.
- [4] P. Li, Y. Pei, and J. Li, "A comprehensive survey on design and application of autoencoder in deep learning," *Applied Soft Computing*, vol. 138, p. 110176, May 2023, doi: 10.1016/j.asoc.2023.110176.
- [5] A. Makhzani and B. Frey, "k-Sparse Autoencoders," 2013, *arXiv*. doi: 10.48550/ARXIV.1312.5663.
- [6] S. Aktar and A. Y. Nur, "Robust Anomaly Detection in IoT Networks using Deep SVDD and Contractive Autoencoder," in *2024 IEEE International Systems Conference (SysCon)*, Montreal, QC, Canada: IEEE, Apr. 2024, pp. 1–8. doi: 10.1109/SysCon61195.2024.10553592.
- [7] D. P. Kingma and M. Welling, "An Introduction to Variational Autoencoders," *FNT in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019, doi: 10.1561/22000000056.
- [8] X. Guo, X. Liu, E. Zhu, and J. Yin, "Deep Clustering with Convolutional Autoencoders," in *Neural Information Processing*, vol. 10635, D. Liu, S. Xie, Y. Li, D. Zhao, and E.-S. M. El-Alfy, Eds., in Lecture Notes in Computer Science, vol. 10635, Cham: Springer International Publishing, 2017, pp. 373–382. doi: 10.1007/978-3-319-70096-0\_39.
- [9] G. Patrini *et al.*, "Sinkhorn Autoencoders," in *Proceedings of the 35th Uncertainty in Artificial Intelligence*, Tel Aviv, Israel: PMLR, Jul. 2019, pp. 733–743.
- [10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.
- [11] S. Blazic, D. Dovzan, and I. Skrjanc, "Cloud-based identification of an evolving system with supervisory mechanisms," in *2014 IEEE International Symposium on Intelligent Control (ISIC)*, Juan Les Pins, France: IEEE, Oct. 2014, pp. 1906–1911. doi: 10.1109/ISIC.2014.6967642.
- [12] J. Vaščák, L. Pomšár, P. Papcun, E. Kajáti, and I. Zolotová, "Means of IoT and Fuzzy Cognitive Maps in Reactive Navigation of Ubiquitous Robots," *Electronics*, vol. 10, no. 7, p. 809, Mar. 2021, doi: 10.3390/electronics10070809.
- [13] L. Pokorádi, "Monte-Carlo Simulation of Reliability of System with Complex Interconnections," *Vehicles*, vol. 6, no. 4, pp. 1801–1811, Oct. 2024, doi: 10.3390/vehicles6040087.
- [14] Y. Liu, C. Ponce, S. L. Brunton, and J. N. Kutz, "Multiresolution convolutional autoencoders," *Journal of Computational Physics*, vol. 474, p. 111801, Feb. 2023, doi: 10.1016/j.jcp.2022.111801.
- [15] P. Bedi and P. Gole, "Plant disease detection using hybrid model based on convolutional autoencoder and convolutional neural network," *Artificial Intelligence in Agriculture*, vol. 5, pp. 90–101, 2021, doi: 10.1016/j.aiia.2021.05.002.