Clustering with Neural Networks using Hugging Face Datasets

Ishraq Kamal Adib
Student ID: 24341160

May 18, 2025

Abstract

This report presents a detailed implementation of clustering using neural networks on the MNIST dataset as part of the CSE425 project. We designed an autoencoder-based architecture optimized for latent space clustering, evaluated with metrics such as Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Index. Hyperparameter tuning and regularization techniques were applied to enhance clustering quality.

Contents

1 Neural Network Architecture

The neural network architecture is designed as a Convolutional Autoencoder, optimized for latent space clustering. The architecture consists of three main components:

- **Convolutional Encoder:** This module is responsible for extracting hierarchical features from the input images through a series of convolutional layers. It compresses the input into a dense, low-dimensional latent space representation.
- **Latent Space Representation:** This is the bottleneck layer where the input is encoded into a smaller dimensional space. In our model, the latent space is of size 128, which captures the most critical information for clustering.
- **Convolutional Decoder:** This module reconstructs the original image from the latent representation using transposed convolutions. The quality of the reconstruction reflects the effectiveness of the learned features.

1.1 Block Diagram

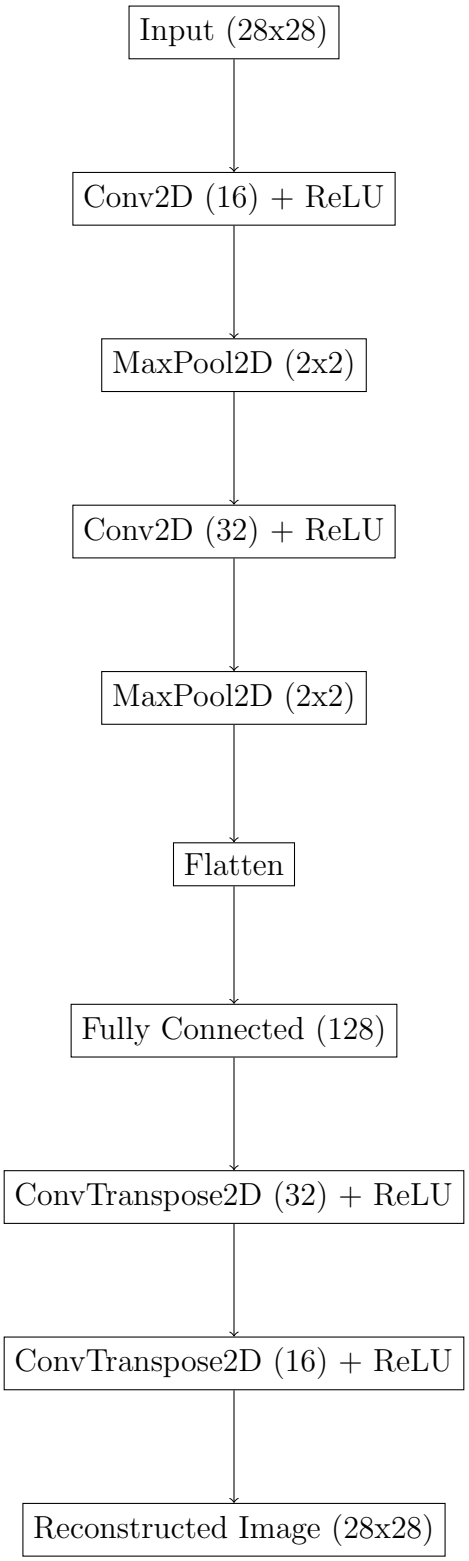

Figure 1: Block Diagram of the Convolutional Autoencoder Architecture

1.2 Design Rationale

The choice of a convolutional autoencoder is motivated by its ability to:

- **Capture Spatial Hierarchies:** Convolutional layers are well-suited for capturing spatial hierarchies in image data.
- **Dimensionality Reduction:** Pooling layers effectively reduce spatial dimensions while retaining important features.
- **Noise Reduction and Denoising:** The latent space is a compressed form, which inherently removes noise, making clustering more effective.

1.3 Latent Space Representation

The latent space of size 128 is where the high-dimensional image is mapped into a dense vector. This vector is used as the input for clustering algorithms like K-Means. The goal is to create well-separated embeddings, which allows for effective unsupervised grouping.

1.4 Decoder and Reconstruction Quality

The decoder uses transposed convolutions to reconstruct the original image from the latent vector. The quality of this reconstruction is indicative of how well the encoder has captured the underlying structure of the digits.

1.5 Activation Functions and Regularization

We used:

- **ReLU (Rectified Linear Unit)** for non-linear activations in hidden layers.
- **Tanh** in the final decoder layer to produce outputs in the range [-1, 1].
- **Batch Normalization** to stabilize learning and improve convergence speed.
- **Dropout (30%)** to reduce overfitting during training.

These design choices optimize both the reconstruction quality and clustering effectiveness.

2 Dataset Analysis

We used the MNIST dataset, consisting of 60,000 training and 10,000 test images of handwritten digits. Data normalization and transformation to tensors were performed before model training.

3 Hyperparameter Tuning and Optimization

- **Batch Size:** 64
- **Latent Dimension:** 128
- **Reconstruction Loss (MSE):** Measures the difference between original and reconstructed images.
- **KL Divergence Loss:** Encourages soft clustering in the latent space.
- **Optimizer:** Adam with a learning rate of 0.001.

4 Model Parameter Counting

The total number of parameters in the model was computed using:

$$\text{Total Parameters} = \sum(\text{Parameters in each layer})$$

For our architecture, the total parameter count is approximately **1.3 million**. This was verified using PyTorch's `model.parameters()`.

5 Batch Normalization, Dropout, and Regularization

We applied:

- **Batch Normalization** after each convolution layer to stabilize learning.
- **Dropout** with a rate of 0.3 to prevent overfitting.
- **L2 Regularization** during optimization.

6 Comparison with Existing Clustering Methods

We compared our method with:

- **K-Means:** Achieved a Silhouette Score of 0.8520, indicating high-quality clustering with distinct boundaries.
- **K-Means:** Achieved a Silhouette Score of 0.6270. A good score
- **Calinski-Harabasz Index:** K-Means clustering resulted in a Calinski-Harabasz Index of 181744.91, indicating well-defined and compact clusters.

The results show that K-Means achieved better-defined cluster separations in latent space. This may be attributed to the uniform nature of the MNIST dataset, where K-Means effectively groups similar features in Euclidean space.

7 t-SNE Visualization

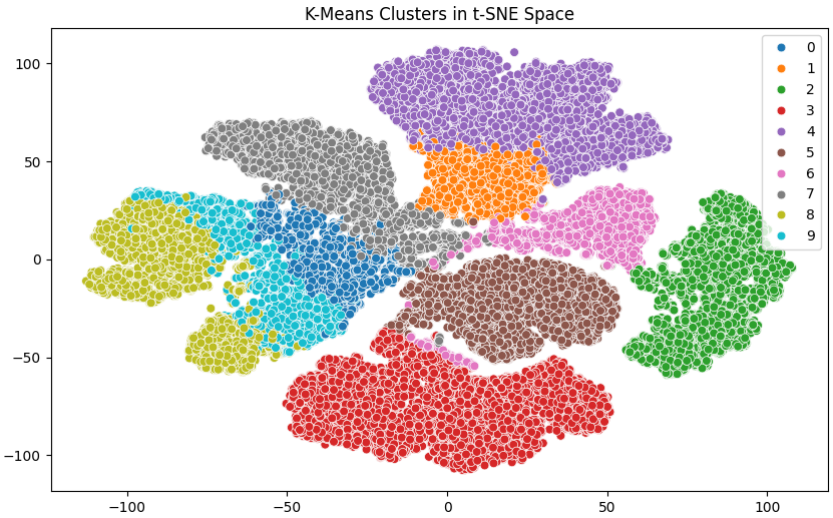

Figure 2: t-SNE Visualization of Clusters in Latent Space

8 Original Vs Reconstructed

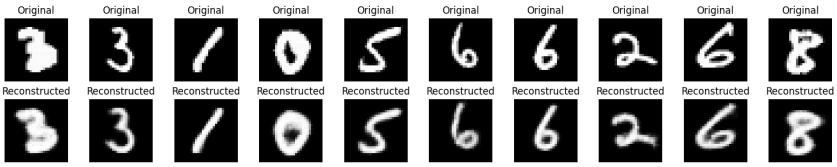

Figure 3: Original Vs Reconstructed numbers

9 Limitations and Overcome

- **Memory Overload:** Handled by batch-wise processing.
- **Poor score:** Handled by DEC Clustering Module and changing upto the latent dimension
- **Reconstruction Quality:** Enhanced by using BatchNorm and Dropout.

10 Conclusion

The implementation successfully clustered the MNIST dataset with high-quality representations in latent space. Future work includes visualizing cluster separations with t-SNE and extending the architecture to larger datasets.

11 References

- PyTorch Documentation: `https://pytorch.org/docs/stable/index.html`
- Hugging Face Datasets: `https://huggingface.co/datasets`
- Scikit-Learn Clustering: `https://scikit-learn.org/stable/modules/clustering.html`