Variational Autoencoder for Digit Images Generation

Brandon Marquez Salazar

***Abstract*—This project presents the implementation and analysis of a Convolutional Variational Autoencoder (CVAE) for the generation and reconstruction of handwritten digit images from the MNIST dataset. The core architecture combines convolutional layers for feature extraction with the variational framework to learn a continuous, structured latent space. The model was trained to optimize the evidence lower bound (ELBO) loss, balancing reconstruction fidelity against the regularization of the latent distribution. Key aspects of the model's behavior were investigated, including the impact of the β hyperparameter on the reconstruction quality versus latent space disentanglement trade-off. The results successfully demonstrate the model's capability to reconstruct input images and generate novel, coherent digits by sampling from the learned latent distribution. Furthermore, visualization of the 2D latent space reveals clusters corresponding to digit classes, confirming the model's ability to learn meaningful representations without supervised labels.**

***Index Terms*—Variational Autoencoder, Deep Learning, Convolutional Neural Network, Generative Models, Unsupervised Learning, MNIST**

I. Introduction

THE rapid advancement of artificial intelligence and deep learning has cemented neural networks as transformative tools across research, industry, and daily life. The evolution from the simple perceptron to sophisticated architectures has unlocked new capabilities in machine perception and generation. Among these, autoencoders have emerged as a powerful family of models for unsupervised learning, characterized by a bottleneck structure that learns efficient data encodings.

Autoencoders function by first encoding input data into a compressed latent representation and then decoding this representation to reconstruct the original input. This process is not typically a "two-stage training" but a single process optimized to minimize reconstruction error. This framework is highly effective for tasks such as dimensionality reduction, denoising, and, crucially, learning meaningful data representations without labeled data. Prominent architectures include the denoising autoencoder and the U-Net, widely used in image segmentation.

This work focuses on a pivotal extension of the autoencoder: the Variational Autoencoder (VAE). Unlike deterministic autoencoders, the VAE introduces a probabilistic twist. It encodes inputs into a distribution over the latent space—specifically, a Gaussian distribution parameterized by a mean and a variance. Decoding then involves sampling from this distribution to generate new data. This fundamental shift from a deterministic encoding to a stochastic one is the key to the VAE's generative capability, allowing it to create new, coherent samples that resemble the training data.

Due to its powerful framework for generative modeling and representation learning, this project involves the implementation and thorough analysis of a Convolutional VAE. The experiment is designed to train the model on the MNIST dataset and explore its behavior through several lenses: the quality of its reconstructions, the structure of its learned latent space, and its ability to generate novel digit images.

II. Methodology

A. Dataset and Preprocessing

The MNIST database of handwritten digits was used for this study [3]. It consists of a training set of 60,000 examples and a test set of 10,000 examples. Each sample is a 28×28 pixel grayscale image. The following preprocessing steps were applied:

- **Normalization:** Pixel values were scaled to the range $[0, 1]$ by dividing by 255.
- **Reshaping:** Images were reshaped from $(28, 28)$ to $(28, 28, 1)$ to include a explicit channel dimension for compatibility with convolutional layers.

B. Model Architecture

A convolutional Variational Autoencoder (VAE) was implemented using the `TensorFlow` and `Keras` frameworks. The model comprises three core components:

1) Encoder: The encoder network $q_\phi(z|x)$ compresses an input image into a latent distribution parameters. Its architecture is as follows:

- **Layers:**
  - `Conv2D(filters=32, kernel_size=3, strides=2, activation='relu', padding='same')`
  - `Conv2D(filters=64, kernel_size=3, strides=2, activation='relu', padding='same')`
  - `Flatten()`
  - `Dense(16, activation='relu')`
- **Output:** Two parallel `Dense` layers output the mean μ and log variance $\log \sigma^2$ vectors of length `latent_dim = 2`.

2) Sampling Layer: A custom sampling layer utilizes the reparameterization trick to generate a latent vector z:

$$z = \mu + \sigma \odot \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, I)$$

3) Decoder: The decoder network $p_\theta(x|z)$ reconstructs an image from a latent vector z:

- **Layers:**
  - `Dense(units=7*7*64, activation='relu')`
  - `Reshape(target_shape=(7, 7, 64))`
  - `Conv2DTranspose(filters=64, kernel_size=3, strides=2, activation='relu', padding='same')`
  - `Conv2DTranspose(filters=32, kernel_size=3, strides=2, activation='relu', padding='same')`
  - `Conv2DTranspose(filters=1, kernel_size=3, activation='sigmoid', padding='same')`
- **Output:** A reconstructed image of shape $(28, 28, 1)$.

The model was trained by minimizing the β-Evidence Lower Bound (β-ELBO) loss:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta \cdot D_{KL}(q_\phi(z|x) \parallel p(z))$$

where $p(z)$ is a standard Gaussian prior $\mathcal{N}(0, I)$ and $\beta = 1$ for the standard VAE.

C. Experimental Setup and Training

All experiments were conducted in a Google Colaboratory environment. The model was trained using the Adam optimizer [1] with a learning rate of 1×10^{-3} and a batch size of 128 for 10 epochs.

The following experiments were designed to evaluate the model:

1. **Computational Efficiency:** The model was trained separately on a CPU and a GPU (NVIDIA Tesla T4). Total training time was recorded for both configurations to quantify computational acceleration.
2. **Ablation Study on Regularization:** The impact of different regularization techniques on generalization was evaluated by modifying the base model:
   - **L1/L2 Regularization:** Kernel regularizers were added to convolutional and dense layers ($\lambda_{L1} = 1 \times 10^{-5}$, $\lambda_{L2} = 1 \times 10^{-4}$).
   - **Dropout:** A `Dropout(rate=0.2)` layer was introduced after the encoder's `Flatten` layer.
   - **Batch Normalization:** `BatchNormalization()` layers were added after each convolutional layer.

D. Evaluation Metrics

Model performance was assessed through the following methods:

- **Qualitative Analysis:** Visual inspection of reconstructed and generated images for coherence and sharpness.
- **Quantitative Analysis:** Reconstruction loss (Binary Cross-Entropy) on the test set.
- **Robustness Validation:** K-Fold Cross-Validation ($k = 5$) was employed to ensure a robust performance estimate. The average reconstruction loss across all folds was reported.
- **Latent Space Analysis:** The 2D latent space was visualized by plotting the mean vectors μ of the test set, colored by their digit class, to assess unsupervised representation learning.

E. Implementation

The complete source code, implemented in Python using `TensorFlow` and `scikit-learn`, is available as an executable Google Colab notebook to ensure full reproducibility of all results.

III. Experiments

A. Computational Efficiency

First of all, the code was tested on a CPU and a copy on a GPU within Google Colab. The main objective was to measure the computational efficiency of the model and compare computational time usage between CPU and GPU.

B. Analysis of Reconstruction and Generation

The main goal here, is to evaluate the model's ability to reconstruct and generate digits from the MNIST dataset.

C. Latent Space Analysis

The objective is to investigate the structure and organization of the learned latent space and assess if meaningful disentangled representations were learned without supervision.

D. Validation through K-Fold Cross-Validation

Here the main objective is To obtain a robust and generalizable estimate of the model's reconstruction performance, mitigating the variance from a single random train-test split

IV. Results

In this section, the results of the experiments are presented and discussed.

A. Computational Efficiency

The results for computational efficiency are shown in Table I.

Table I: Training Metrics and Time Comparison per Epoch: CPU vs. GPU

Epoch	Device	Time (s)	Loss	Recon. Loss	Val Loss
1	CPU	93	271.71	270.31	185.32
	GPU	16	269.50	267.80	193.63
2	CPU	138	180.16	174.65	171.62
	GPU	3	192.16	188.83	187.40
3	CPU	86	171.17	165.08	167.95
	GPU	4	187.11	183.68	184.71
4	CPU	88	167.90	161.65	165.60
	GPU	5	184.73	181.23	182.44
5	CPU	140	165.42	159.01	163.75
	GPU	3	183.05	179.48	180.61
6	CPU	85	163.62	157.11	162.05
	GPU	5	181.35	177.70	179.02
7	CPU	143	162.07	155.55	161.18
	GPU	3	180.04	176.30	177.81
8	CPU	136	160.93	154.37	159.98
	GPU	5	178.88	175.12	176.98
9	CPU	84	160.00	153.46	159.39
	GPU	3	178.03	174.26	176.22
10	CPU	146	159.20	152.67	158.62
	GPU	3	177.17	173.37	175.61

B. Analysis of Reconstruction and Generation

C. Latent Space Analysis

D. Validation through K-Fold Cross-Validation

E. References

[1] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[2] Diederik P Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[3] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.