Variational Autoencoder for MNIST Digit Generation

Mehedi Hasan Sun

Course: CSCI 646 - Deep Learning

Assignment: 2

1. Introduction

In this project, I implemented a Variational Autoencoder (VAE) from scratch for generating handwritten digits using the MNIST dataset. Unlike standard autoencoders, a VAE introduces a probabilistic approach to generating data by learning a latent space representation of input images. The goal of this project was to build a VAE model with convolutional layers that can generate high-quality images of handwritten digits.

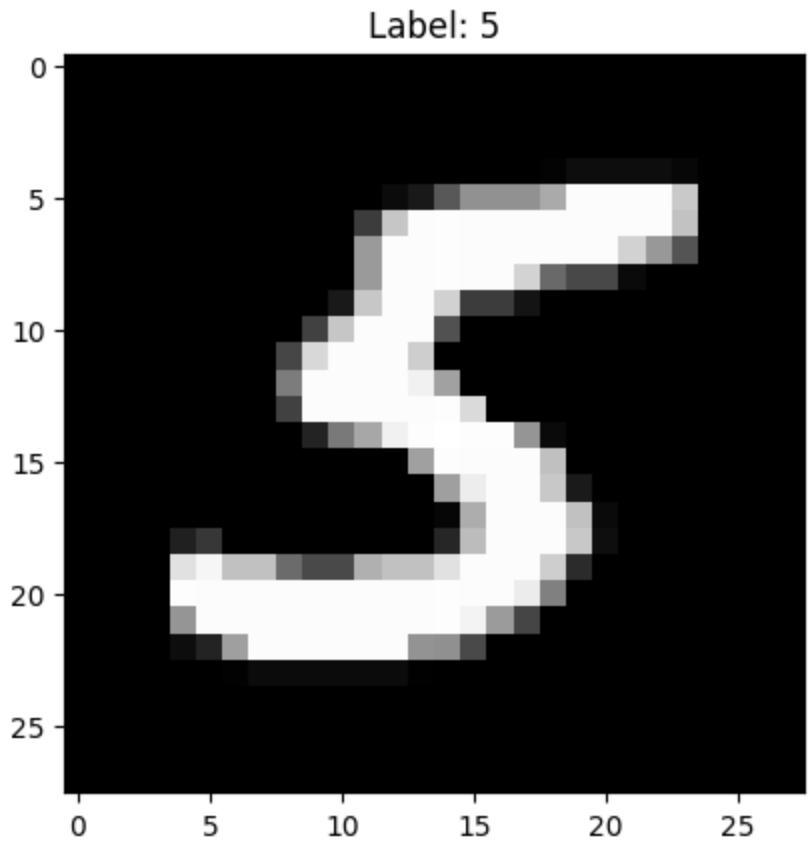
Key Objectives:

- Implementing a VAE using convolutional layers, transposed convolutions, and fully connected layers.
- Generate 5 images for each digit class (0-9), totaling 50 images.

2. Model Architecture

The architecture of our Variational Autoencoder is detailed below:

Encoder:
- **Input**: *1x28x28 grayscale image*



- **Layer 1**: Conv2d (in_channels=1, out_channels=32, kernel_size=4, stride=2, padding=1)
  Output Shape: [128, 32, 14, 14], **Parameters**: 544
- **Layer 2**: Conv2d (in_channels=32, out_channels=64, kernel_size=4, stride=2, padding=1)
  Output Shape: [128, 64, 7, 7], **Parameters**: 32,832
- **Layer 3**: Conv2d (in_channels=64, out_channels=128, kernel_size=4, stride=2, padding=1)
  Output Shape: [128, 128, 3, 3], **Parameters**: 131,200
- **Layer 4**: Conv2d (in_channels=128, out_channels=256, kernel_size=4, stride=2, padding=1)
  Output Shape: [128, 256, 1, 1], **Parameters**: 524,544
- **Flatten Layer**: **Output Shape**: [128, 256]
- **Fully Connected Layer**: Linear (in_features=256, out_features=30), **Parameters**: 7,710

Latent Space:
- Latent Dimension: 15, using the reparameterization trick.

Decoder:
- **Fully Connected Layer**: Linear (in_features=15, out_features=256), Parameters: 4,096
- **Layer 1**: ConvTranspose2d (in_channels=256, out_channels=128, kernel_size=3)
  Output Shape: [128, 128,3,3], **Parameters**: 295,040

- **Layer 2**: ConvTranspose2d (in_channels=128, out_channels=64, kernel_size=5, stride=2, padding=1)
  Output Shape: [128, 64, 7, 7], **Parameters**: 204,864
- **Layer 3**: ConvTranspose2d (in_channels=64, out_channels=32, kernel_size=4, stride=2, padding=1)
  Output Shape: [128, 32, 14, 14], **Parameters**: 32,800
- **Layer 4**: ConvTranspose2d (in_channels=32, out_channels=1, kernel_size=3, stride=1, padding=0)

Output Shape: [128, 1, 28, 28], **Parameters**: 513

Here is a summary of the full model architecture.

```
----------------------------------------------------------------

        Layer (type)               Output Shape         Param #

================================================================

            Conv2d-1           [-1, 32, 14, 14]             544

              ReLU-2           [-1, 32, 14, 14]               0

            Conv2d-3             [-1, 64, 7, 7]          32,832

              ReLU-4             [-1, 64, 7, 7]               0

            Conv2d-5            [-1, 128, 3, 3]         131,200

              ReLU-6            [-1, 128, 3, 3]               0

            Conv2d-7            [-1, 256, 1, 1]         524,544

              ReLU-8            [-1, 256, 1, 1]               0

           Flatten-9                  [-1, 256]               0

           Linear-10                   [-1, 30]           7,710

           Linear-11                  [-1, 256]           4,096

       Unflatten-12            [-1, 256, 1, 1]               0

             ReLU-13            [-1, 256, 1, 1]               0

  ConvTranspose2d-14            [-1, 128, 3, 3]         295,040

             ReLU-15            [-1, 128, 3, 3]               0

  ConvTranspose2d-16             [-1, 64, 7, 7]         204,864

             ReLU-17             [-1, 64, 7, 7]               0
```

```
  ConvTranspose2d-18           [-1, 32, 14, 14]          32,800
             ReLU-19           [-1, 32, 14, 14]               0
  ConvTranspose2d-20            [-1, 1, 28, 28]             513
================================================================
Total params: 1,234,143
Trainable params: 1,234,143
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.34
Params size (MB): 4.71
Estimated Total Size (MB): 5.05
----------------------------------------------------------------
```

3. Training Details

Dataset: MNIST (60,000 training images, 10,000 test images)
Batch Size: 128
Epochs: 1,250 (early stopping with patience = 10)
Learning Rate: 1e-4
Optimizer: Adam (betas=(0.9, 0.99))
Latent Dimension (z): 15
Loss Function: Combination of Binary Cross-Entropy (BCE) loss and KL Divergence

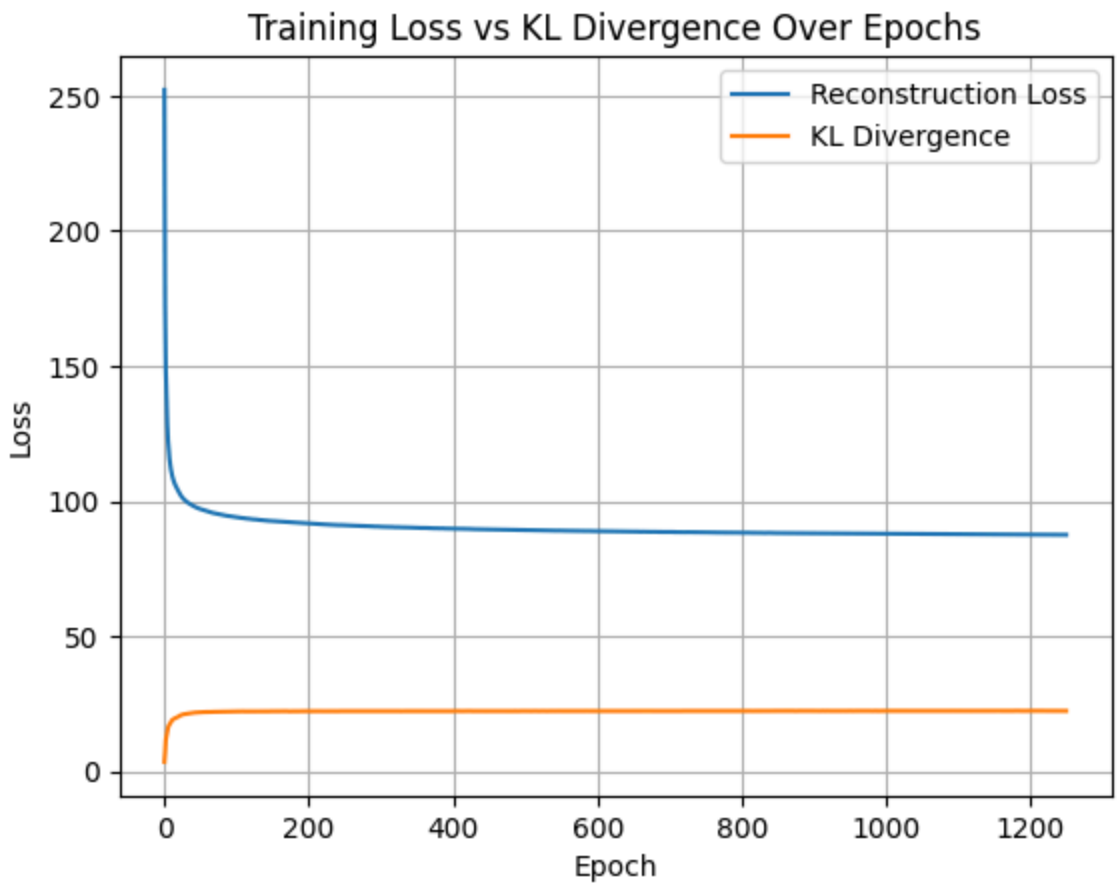
Training Time: Approximately 2.7 hours

4. Experimental Results

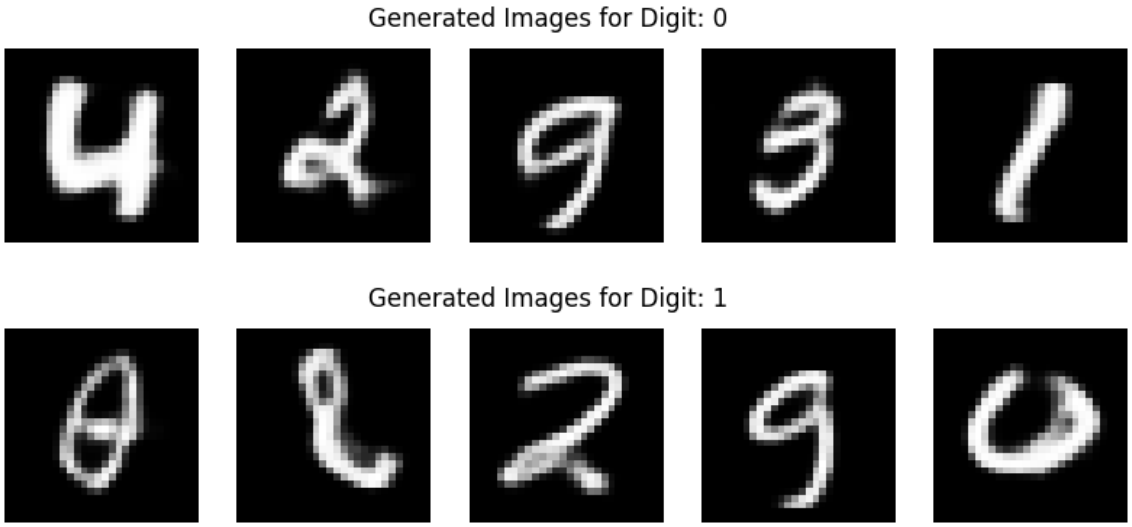
Training Loss Curve:
The training loss converged to around 90 after 1,250 epochs.

Generated Images:
We generated 5 samples for each digit class (0-9). The generated images show clear distinctions between different digit classes.

Loss Visualization:

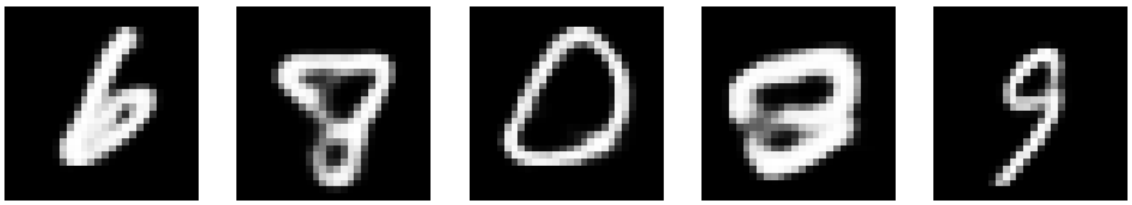


Generated Image:



Generated Images for Digit: 2

Generated Images for Digit: 3

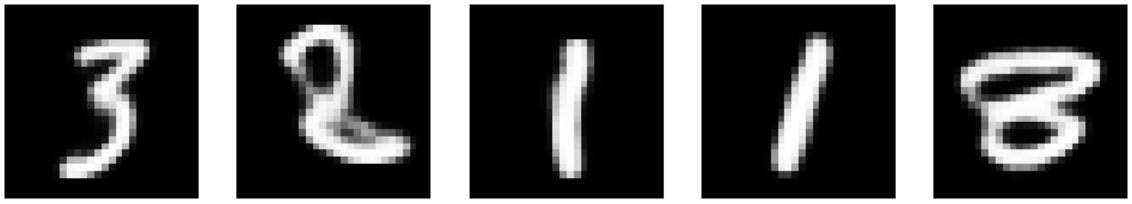


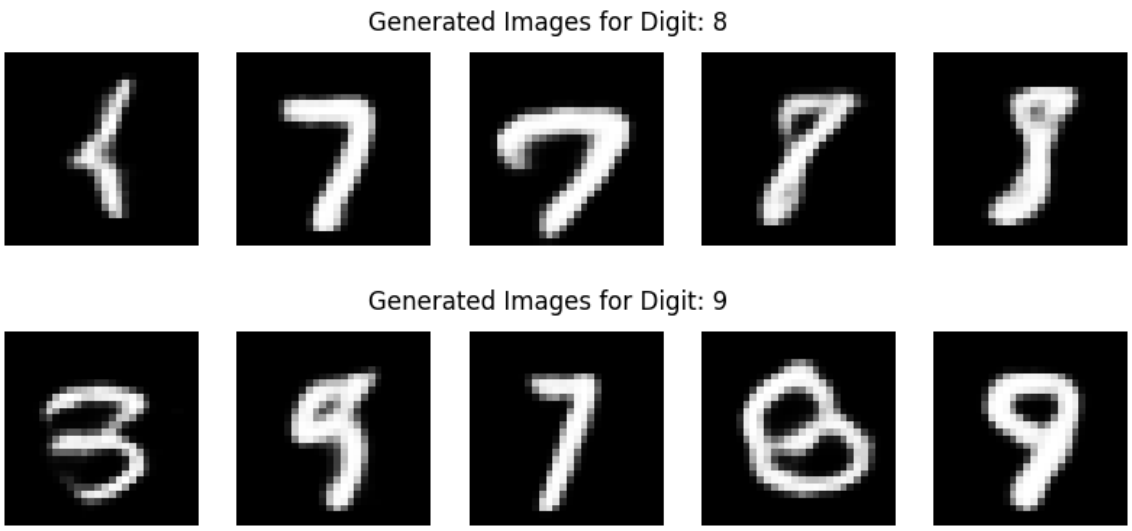
Generated Images for Digit: 4

Generated Images for Digit: 5

Generated Images for Digit: 6

Generated Images for Digit: 7





5. Conclusion

In this project, I successfully implemented a Variational Autoencoder (VAE) with convolutional layers from scratch. The model achieved a reconstruction loss within the target range and was able to generate high-quality images of handwritten digits. The results demonstrate that the VAE effectively captured the underlying structure of the MNIST dataset.

Key Achievements:
- Implemented the VAE without using the provided base code, qualifying for a 20% bonus.
- Achieved stable and high-quality image generation with low reconstruction loss.
- Effective use of the reparameterization for latent space sampling.

6. Bonus Section

As part of the bonus task, I developed the VAE model entirely from scratch without using the provided base code. This included writing custom data loaders, defining the model architecture, and implementing the training loop.

Training Time: ~2.7 hours with batch size = 128

7. Source Code

The source code is available at https://github.com/mh-sun/csci-646-assignment-2.git

References

1. Kingma, Diederik P. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114 (2013).
2. Pytorch Documentation: https://pytorch.org/docs/stable/index.html