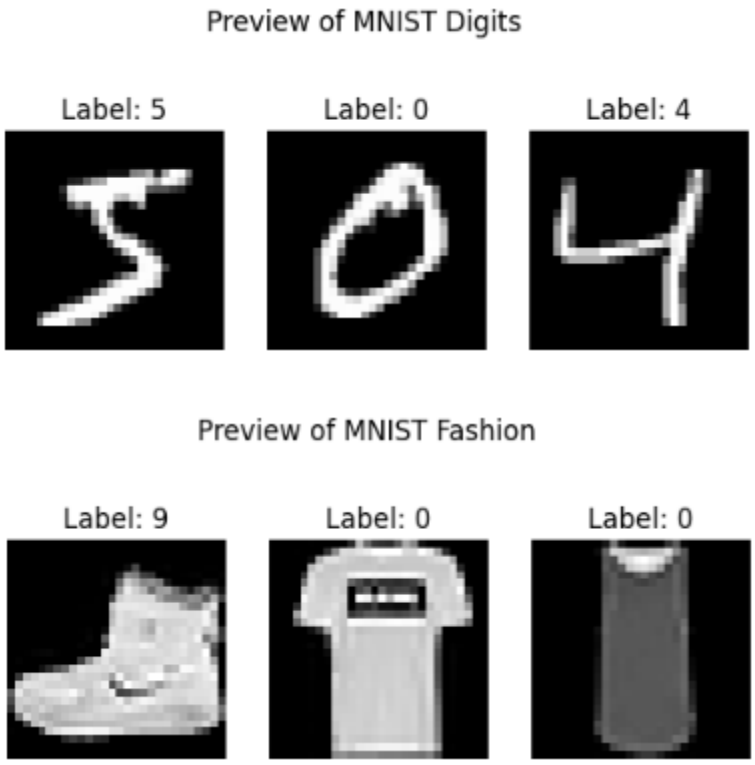
Part 1: Exploratory Data Analysis (EDA)

1. **Load Data**
   Loaded the datasets (MNIST digits and MNIST fashion) as tensors and normalized them.
2. **Preview the datasets**



3. **Dataset Analysis**
   a. **Number of samples**

```
Total samples in MNIST digits dataset: 70000
Division:
        Training data: 60000
        Testing Data: 10000


Total samples in MNIST Fashion dataset: 70000
Division:
        Training data: 60000
        Testing Data: 10000
```

   b. **Number of classes and labels**

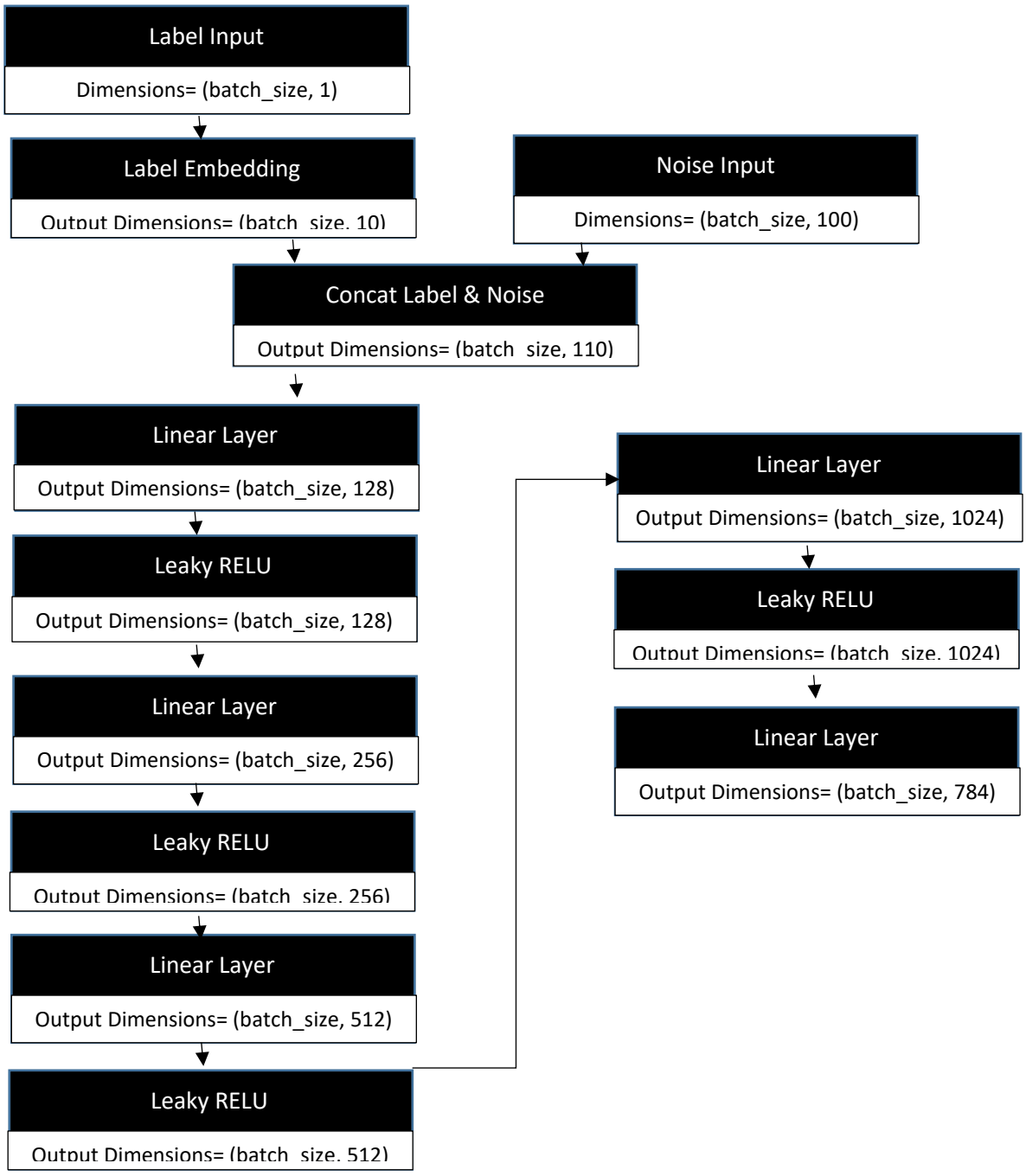
```
Number of classes in MNIST Digits: 10
        Labels: [0 1 2 3 4 5 6 7 8 9]


Number of classes in MNIST Fashion: 10
        Labels: [0 1 2 3 4 5 6 7 8 9]
```

Part 2: Implementing Generative Adversarial Networks (GANs)

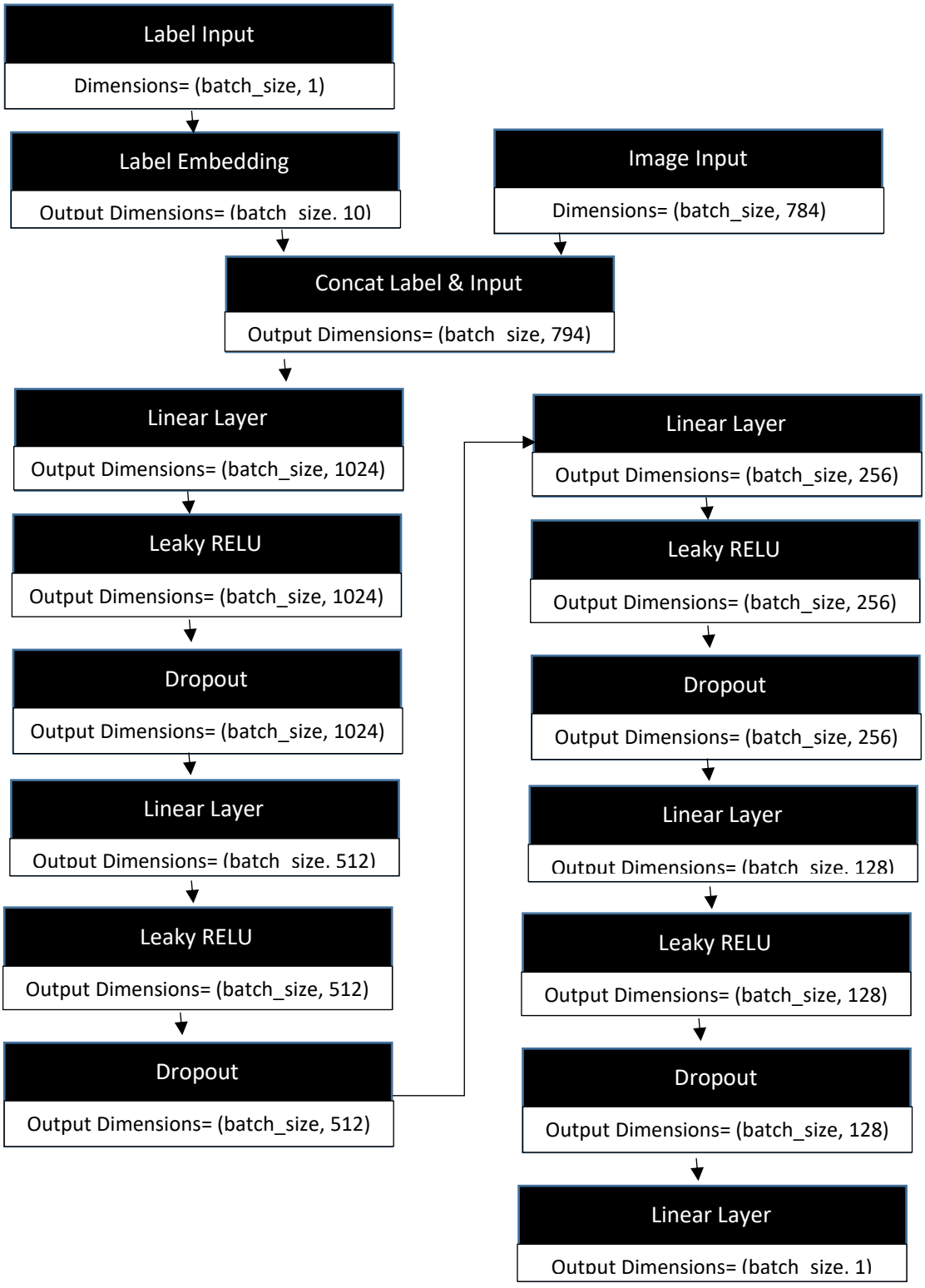
1. For MNIST Digits Data

Since we were required to print particular digit, so I implemented CGAN (Conditional GAN) for MNIST Digits dataset with linear layer, Leaky RELU(0.2) to add non-linearity and prevent dying neurons, and dropout layer to prevent overfitting. Computed binary cross entropy loss for discriminator.

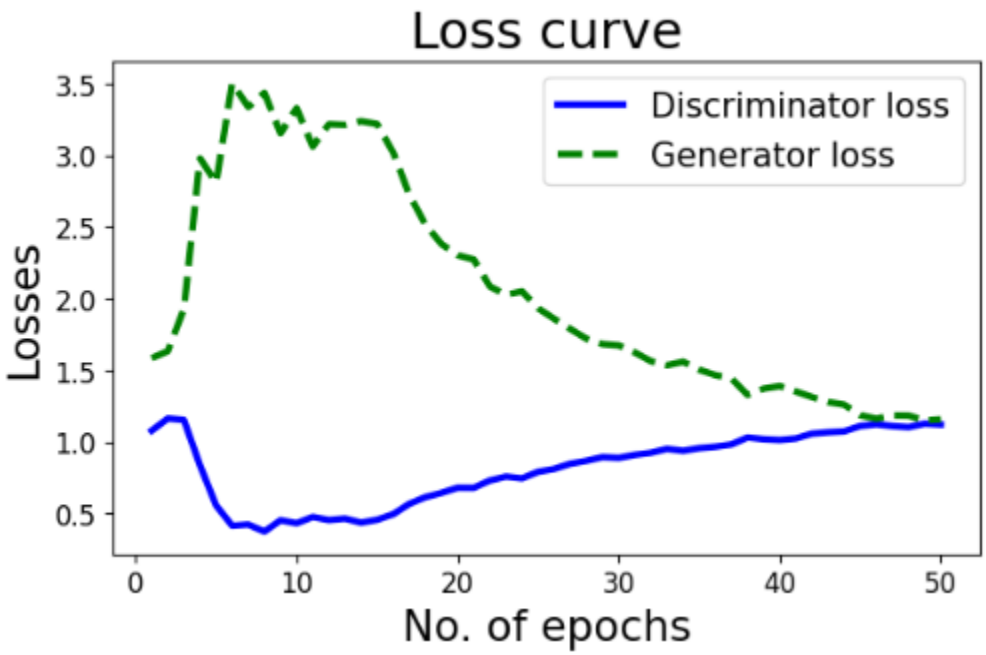
a. Generator



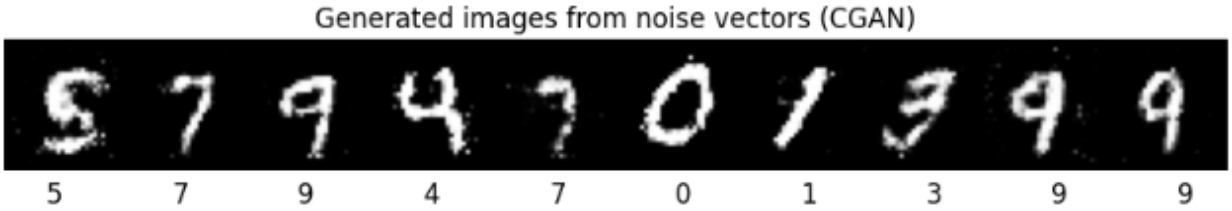
b. Discriminator Architecture



c. Loss Curves



d. 10 generated images

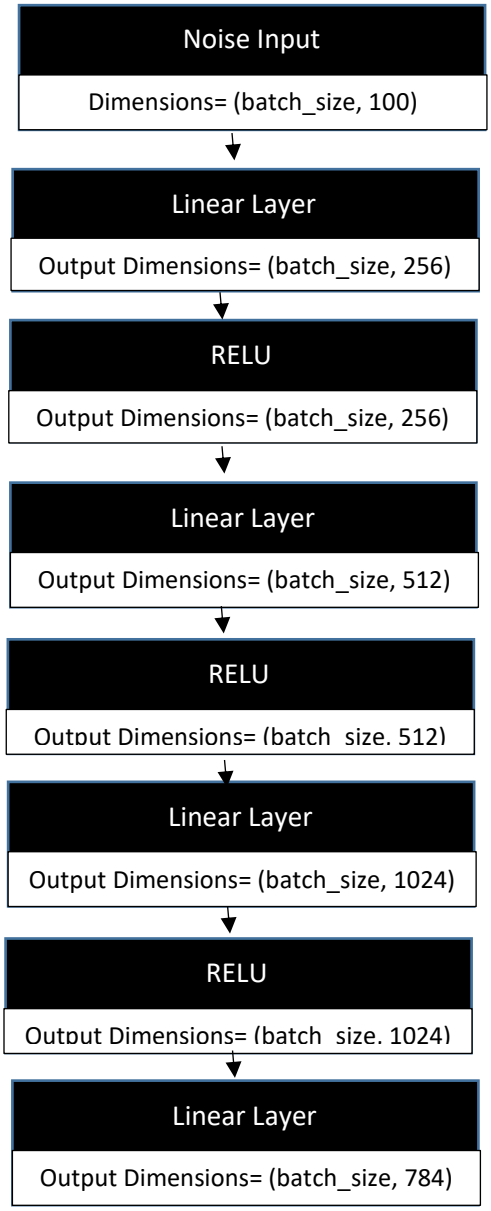


e. 5 images of last digit of roll num i.e '1'

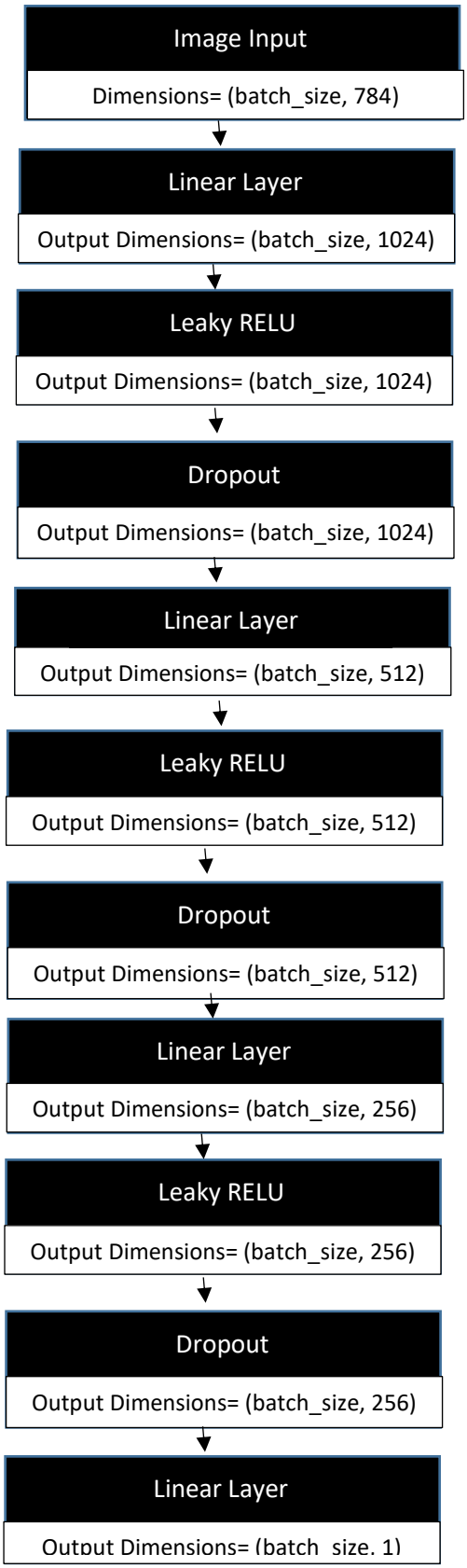
2. For MNIST Fashion Data

Implemented GAN for MNIST Fashion Data with linear layer, RELU to add non-linearity and dropout layer to prevent overfitting. Computed binary cross entropy loss for discriminator.

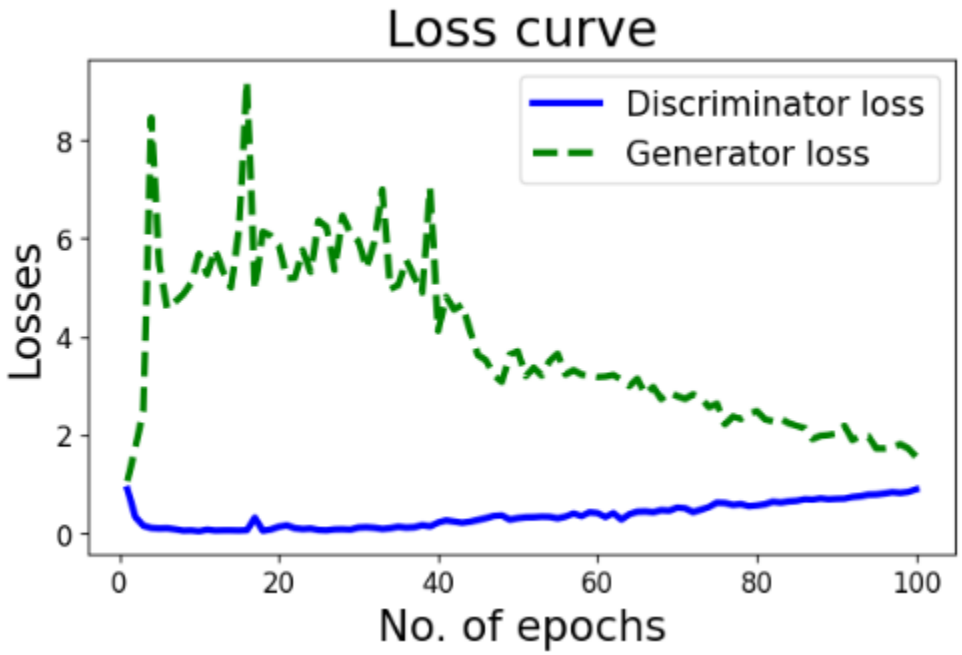
a. Generator Architecture



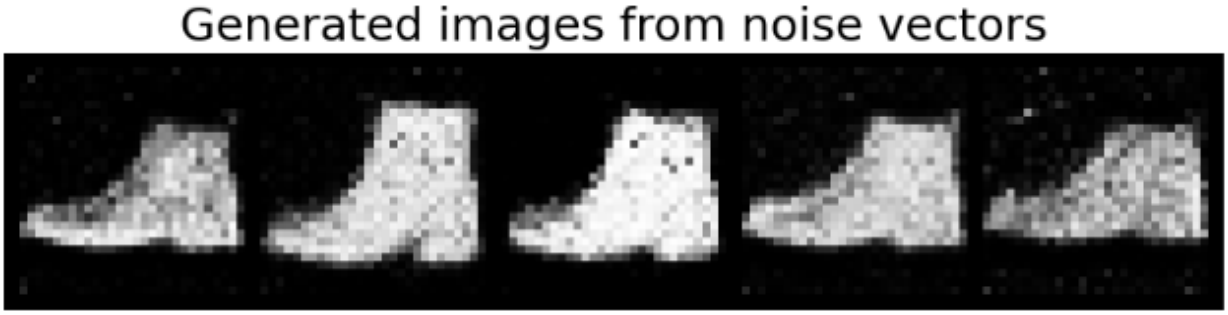
b. Discriminator Architecture



c. Loss Curves



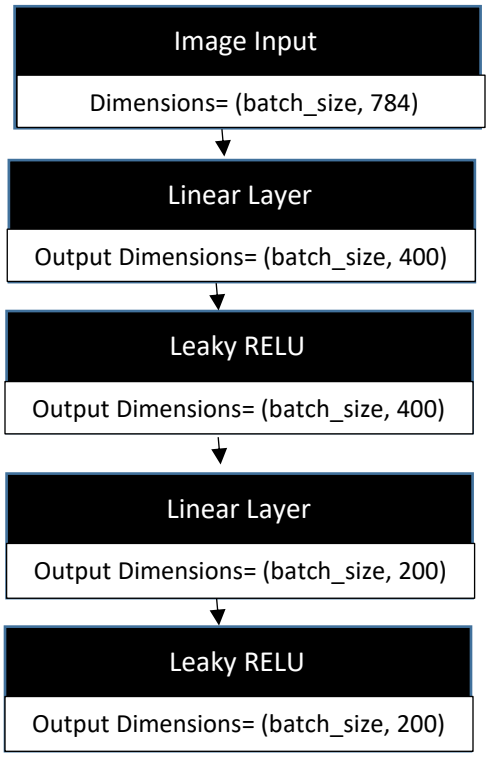
d. Generated images of class 'Ankle Boot'



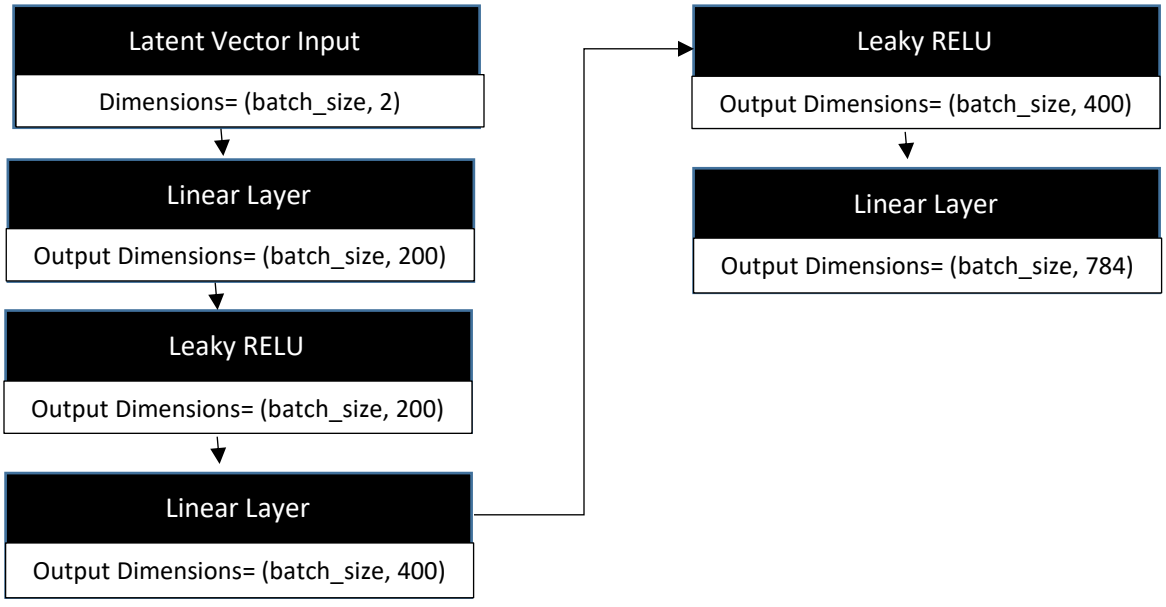
Part 3: Implementing Variational Autoencoder (VAE)

Implemented encoder and decoder using linear layer and Leaky RELU(0.2) to add non-linearity and prevent dying neurons. Used reconstruction loss to measure how well the model reconstructs the input from the latent space, and KL divergence to measure how much the learned latent distribution deviates from the prior distribution.

Encoder:

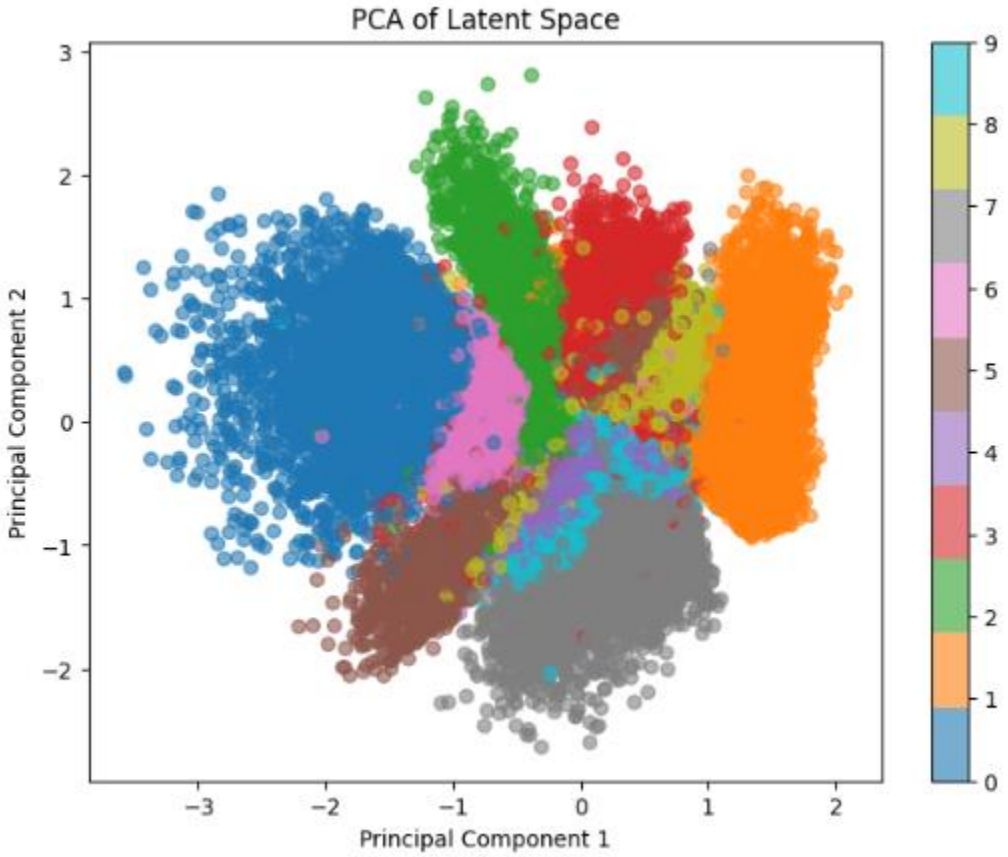


Decoder:



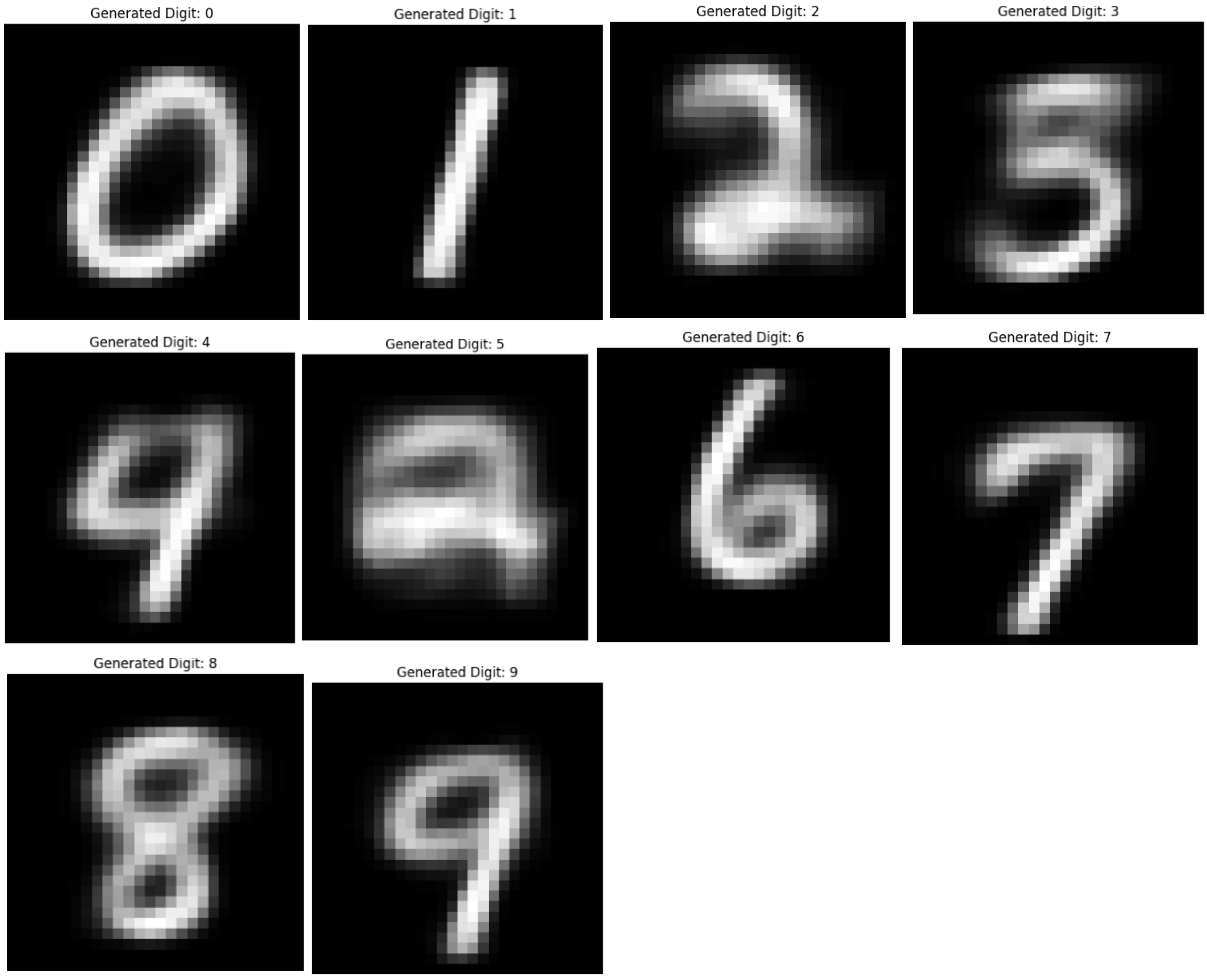
1) MNIST Digits Dataset

a) Latent Space Representation Using PCA:

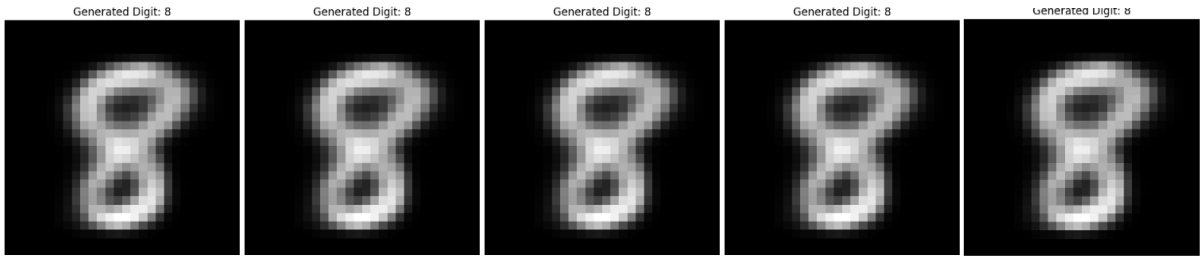


Since all labels have separate clusters so this shows that model has learned their unique pattern and representation.

b) 10 Digits Images:

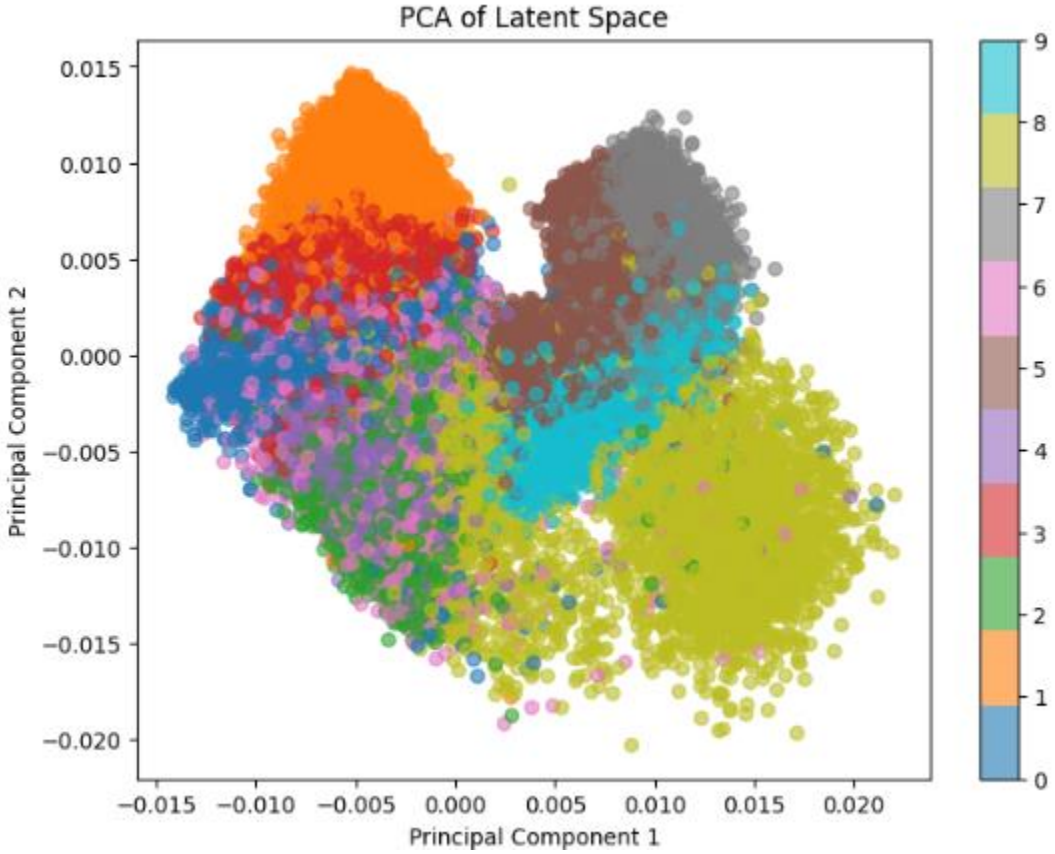


c) 5 images of second last roll num digit i.e ‘8’



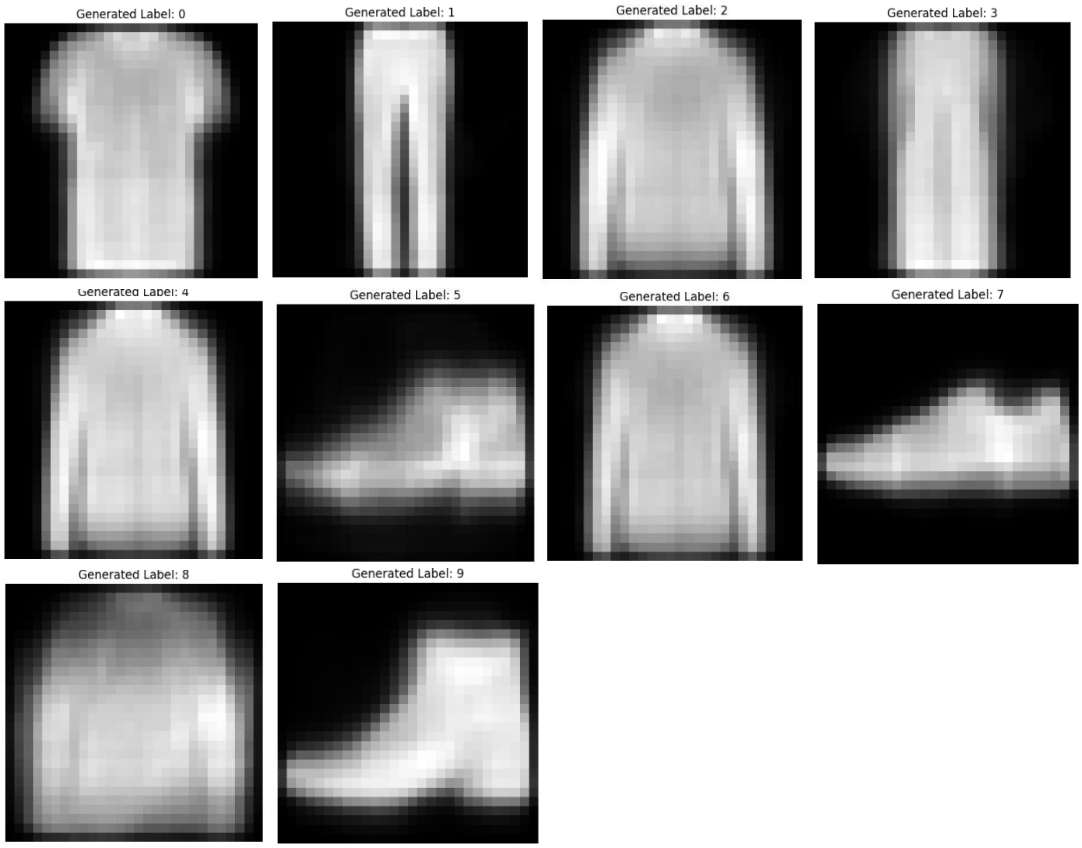
2) MNIST Fashion Dataset

a) Latent Space Representation Using PCA:



Since all labels have separate clusters so this shows that model has learned their unique pattern and representation.

b) 10 Images:



Part 4: Comparison and Analysis

a) Image Quality

GAN produced better quality digits as the generated images were sharp and had variation. Images generated by VAE were comparatively blurrier.

b) Training stability

GAN was harder to train as it has min-max game between generator and discriminator. If discriminator becomes too strong then generator receives no meaningful gradients. If the generator becomes too strong then discriminator fails to guide learning, leading to mode collapse.

c) Latent Space Representation

GANs have unstructured and non-continuous latent space. The generator maps random noise to realistic samples, but without an explicit constraint on how points in the latent space are structured. Interpolating between points in latent space might generate unrealistic images. Similar images might not be close in latent space, making representation learning harder.

VAE have structured and continuous latent space. VAEs enforce a structured latent space using KL divergence to make the learned distribution resemble a standard normal distribution.

Potential Improvements:

We can use larger latent vector in GAN to increase variability in generated images. We can use different optimizer and learning rates in GAN. We can also change Leaky RELU penalty factor as well as the Dropout value.

For VAE, we can increase the size of latent vector to store better representation. We can use ELU instead of Leaky RELU for smoother gradient flow. We can also use different optimizer and learning rates.