GENERATIVE AI ASSIGNMENT

Submitted To

Dr. N.L. Bhanumurthy

By:

Name	ID NO
AASHUTOSH A V	2021A7PS0056H
TUSHAR RAGHANI	2021A7PS1404H
PAVAS GARG	2021A7PS2587H

Birla Institute of Technology and Science, Hyderabad Campus

TASK DEFINITION:

Image Generation and Image-to-Image Translation , and analyzing the results

Aim & Abstract

Through this project we aim to get a practical understanding of the implementation of 3 Generative Models: VAE, GAN and Diffusion Models and also the two variants of gans namely DCGan and CycleGan.

This project explores the realm of generative modeling and image-to-image translation using deep learning techniques. The objective is twofold: in Part A, we train generative models to generate images from random vectors sampled from the latent space, employing Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and Diffusion Models on the MNIST dataset. In Part B, we focus on image-to-image translation, aiming to convert source images into target images while altering specific visual properties while preserving others. This is accomplished by employing two variants of Generative Adversarial Networks (GANs) on the CelebA dataset: Deep Convolutional GAN (DCGAN) and CycleGAN. By implementing and analyzing these models, we aim to gain insights into their respective capabilities, strengths, and limitations in generating realistic images and performing effective image-to-image translations. Through this project, we contribute to the understanding and advancement of generative modeling techniques for various image generation and translation tasks.

The following pages of this report contain a brief description of our work:

Part-A

VAE

We trained a VAE to generate images of the MNIST dataset. The VAE is structured only with Linear Layers, since the simplicity of the dataset meant we weren't required to make the model more complex with ideas like Conv and Pool Layers. Our architecture involves the model learning the mean and variance of a latent variable using an Encoder, and re-constructing the images from that using a Decoder. We used the "relu" and "sigmoid" activation functions in the layers.

We also chose to use the Adam Optimizer, since it generally requires a lower learning rate, and that it converges faster.

We loaded the dataset from torchvision's datasets module, and trained the architecture as follows:

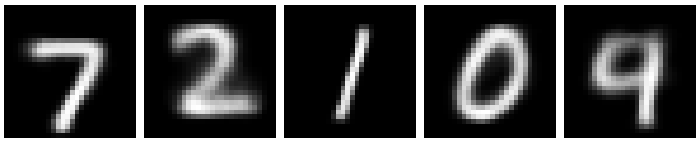
1. An encoder which transforms the shape of the data as follows:
    a. Layer 1: 784
    b. Layer 2: 512
    c. Layer 3: 256
    d. From this we learnt a mean and variance of a specific latent dimension from the given options
2. A decoder which transforms the shape of the data as follows:
    a. From the latent space of mean and variance, we propagate the layers to:
    b. Layer 1: 256
    c. Layer 2: 512
    d. Layer 3: 784, to get to the dimensions of the original image

The VAE class contains a loss function for training, which is a combination of the Binary Cross Entropy Loss to tap the reconstruction error, and the KL Divergence Loss, which taps the deviation of the latent space from a standard distribution, in our case, the Standard Normal Distribution. We try to backpropagate through this loss function,

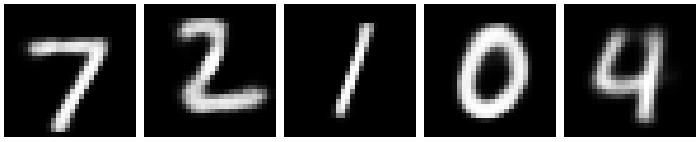
in an aim to minimize it. The sampling function samples a variable from the latent distribution and tries to reconstruct the images from that using the Decoder.

The following are the images of the reconstructions of 5 random points for latent sizes from [2, 4, 8, 16, 32, 64]:

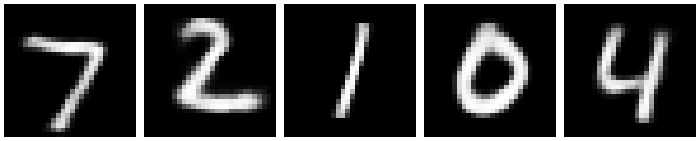
Latent Size: 2



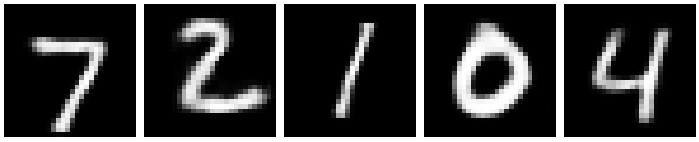
Latent Size: 4



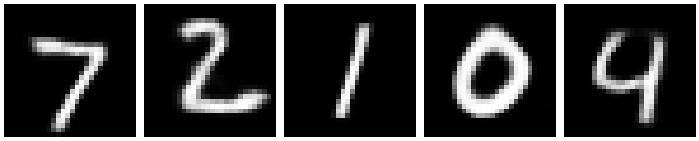
Latent Size: 8



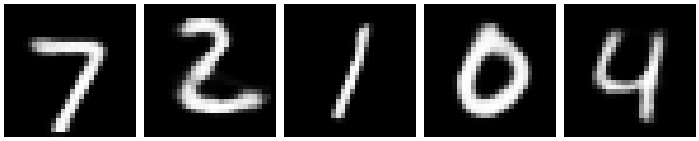
Latent Size: 16



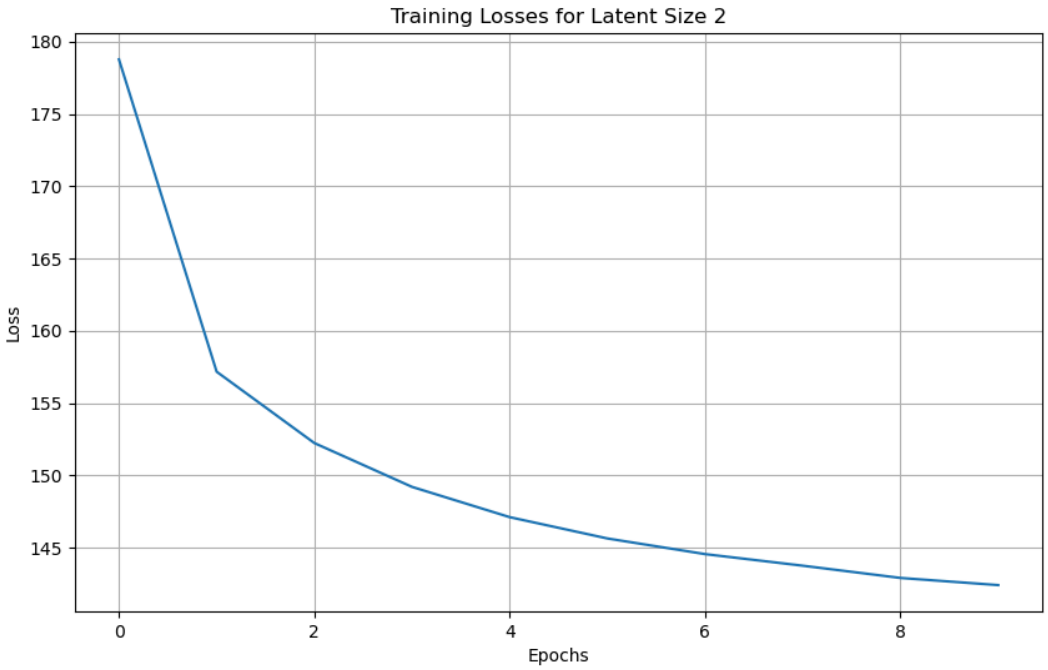
Latent Size: 32

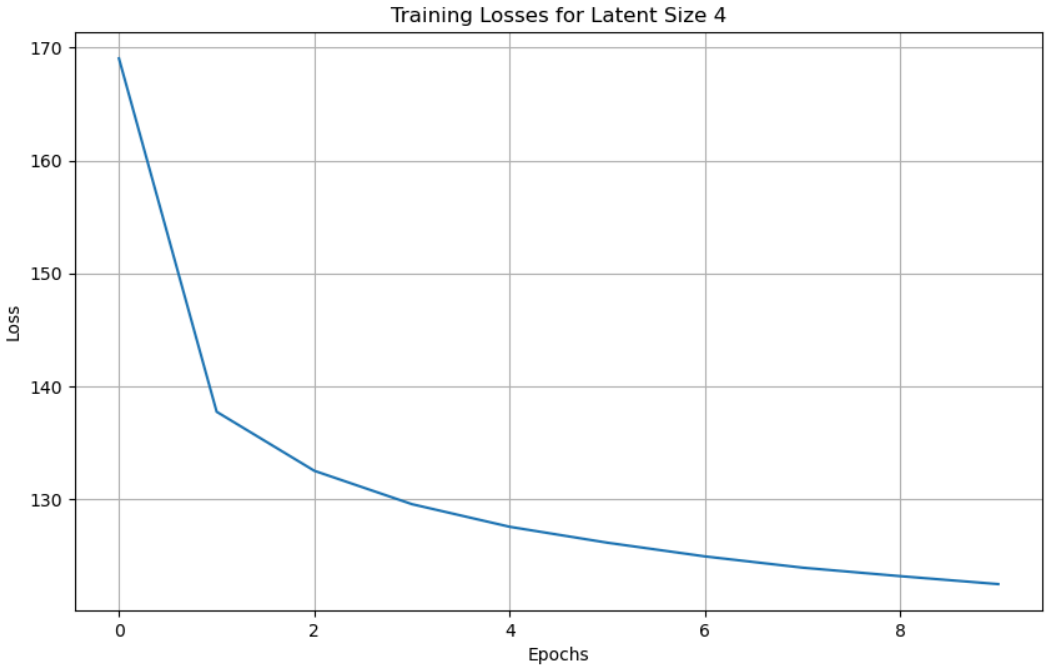


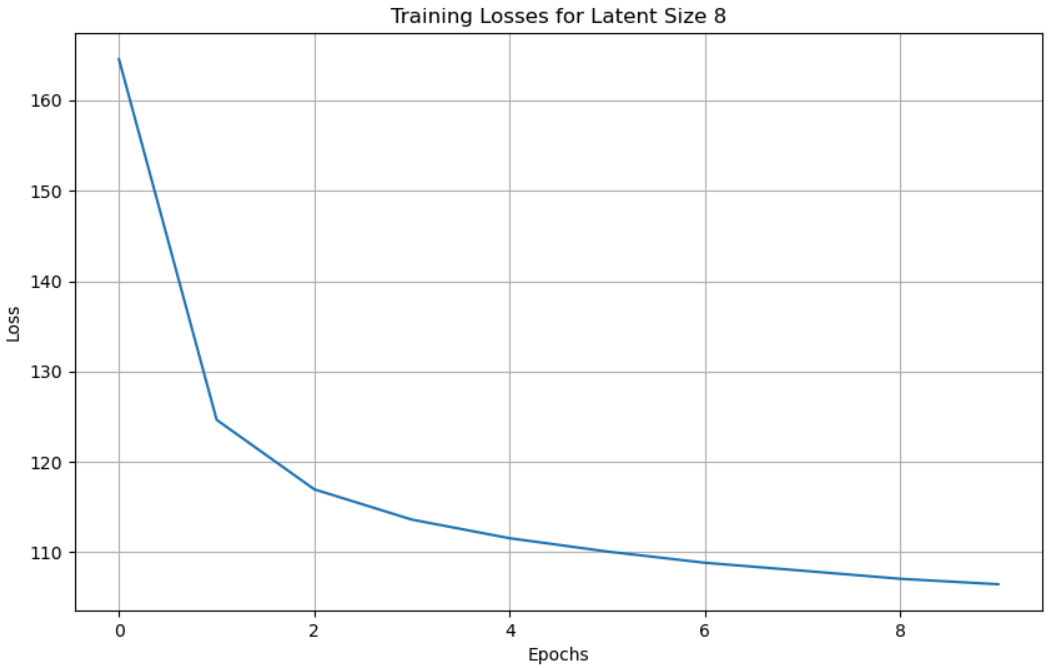
Latent Size: 64

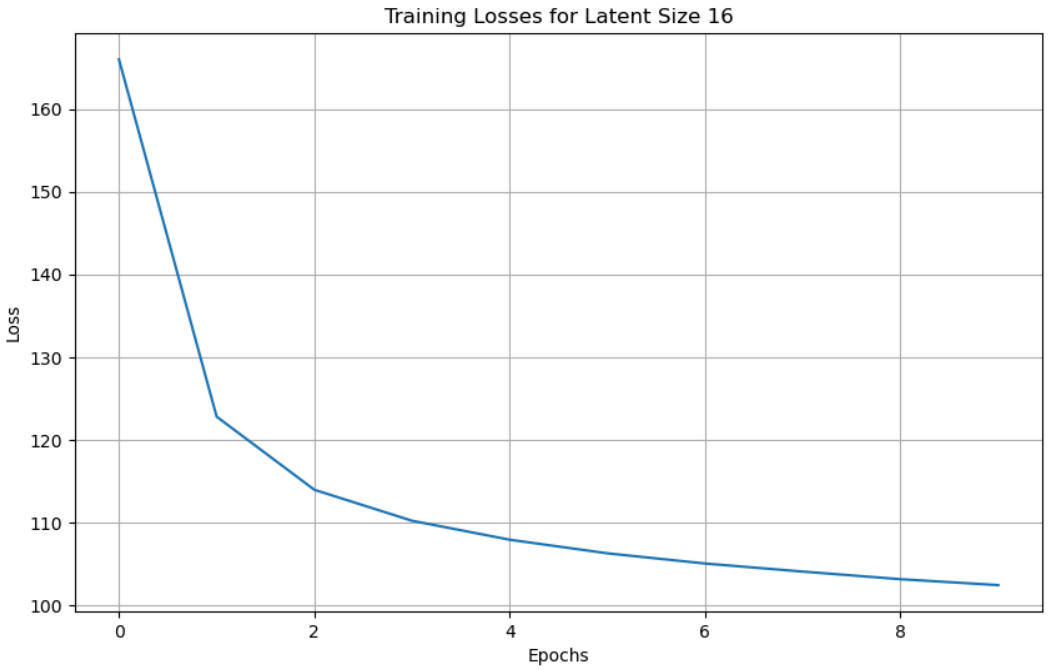


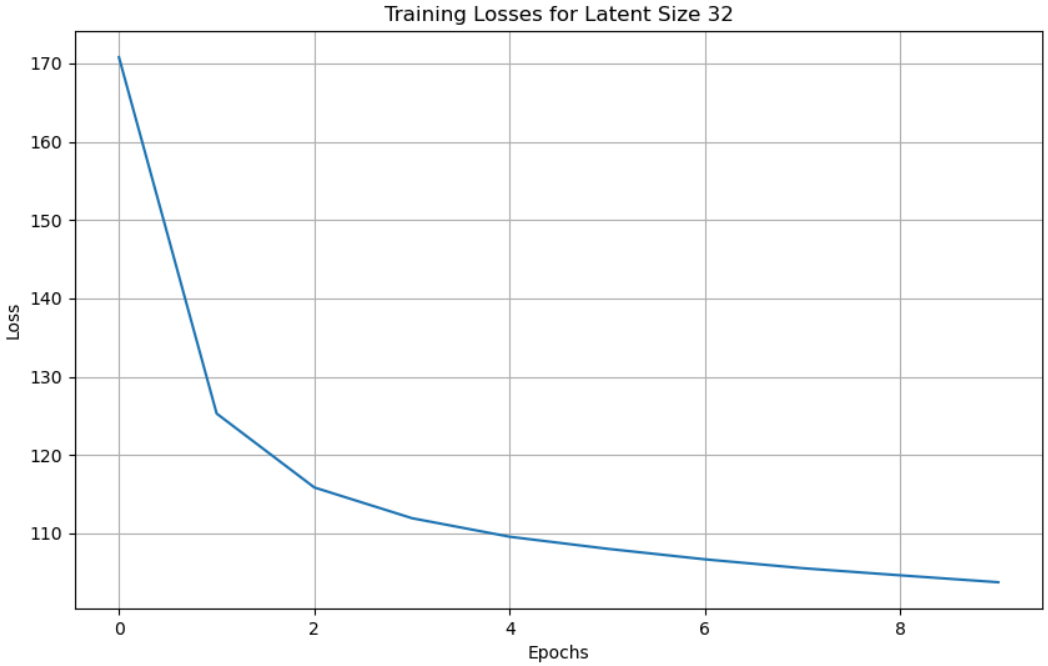
The following images also depict how the training loss(not MSE) reduces over epochs for each latent size:

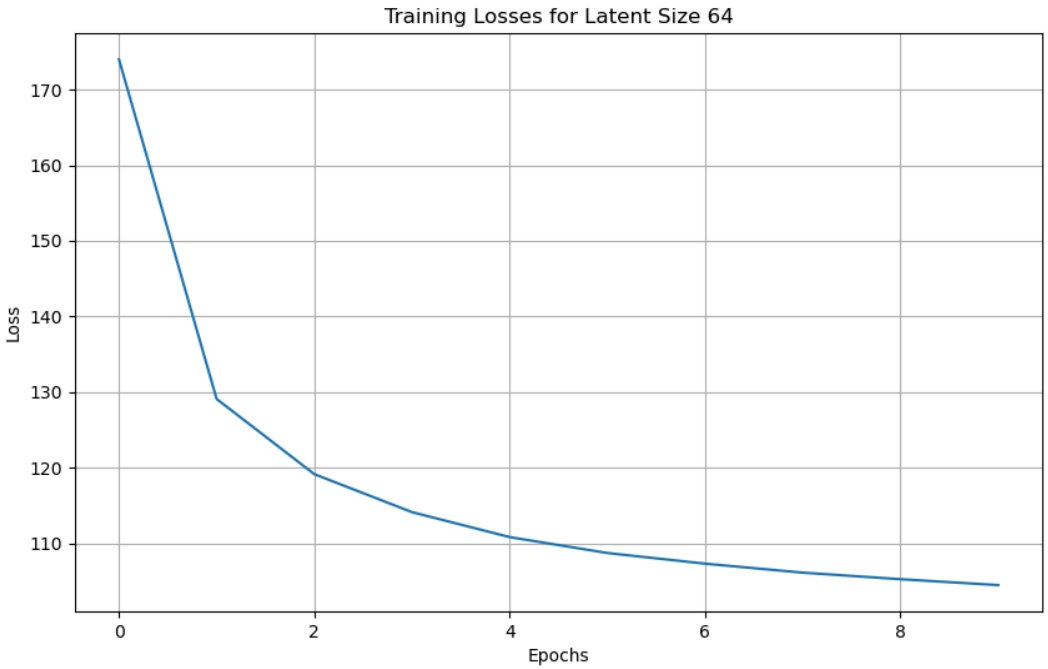





Training Losses for Latent Size 8
Loss
Epochs
160
150
140
130
120
110
0
2
4
6
8


Training Losses for Latent Size 16
Loss
Epochs
160
150
140
130
120
110
100
0
2
4
6
8


Training Losses for Latent Size 32
Loss
Epochs
170
160
150
140
130
120
110
0
2
4
6
8


Training Losses for Latent Size 64
Loss
Epochs
170
160
150
140
130
120
110
0
2
4
6
8

Mean and Variance of the latent vector space for various dimensions are given below:

Latent Size: 2, Mean: -0.04438533185515553, log(var): -6.277547416687011
Latent Size: 4, Mean: 0.01468251047830563, log(var): -5.07385148525238
Latent Size: 8, Mean: -0.013139826001715846, log(var):: -4.020021946430206
Latent Size: 16, Mean: 0.02996418400434777, log(var):: -2.3866638135910034
Latent Size: 32, Mean: 0.0076728542786440816, log(var):: -1.1874005436897277
Latent Size: 64, Mean: 0.07183750431780936, log(var):: -0.5867297291755676

GAN

The GAN is trained to generate realistic MNIST digit images.The generator learns to generate images from random noise, while the discriminator learns to distinguish between real and fake images.The training loop alternates between training the discriminator and the generator.The generator aims to minimize the discriminator's ability to distinguish fake images from real ones.The discriminator aims to maximize its ability to distinguish between real and fake images.

We used the “relu” in the layers of the generator and “relu” and “sigmoid” activation functions in the layers of the discriminator.

We also chose to use the Adam Optimizer, since it generally requires a lower learning rate, and that it converges faster.

We loaded the dataset from torchvision’s datasets module, and trained the architecture as follows:

1. Generator Architecture

   a) Linear Layer 1: Input size = latentsize, Output size = 128.
   Activation: LeakyReLU with a slope of 0.1

   b)Linear Layer 2: Input size = 128, Output size = 256.
   Activation: LeakyReLU with a slope of 0.2.

   c)Linear Layer 3: Input size = 256, Output size = 512.
   Activation: LeakyReLU with a slope of 0.3.

   d)Linear Layer 4: Input size = 512, Output size = 784 (size of MNIST images).
   Activation: Tanh (to squash the output to the range [-1, 1]).

2. Discriminator Architecture

   a)Linear Layer 1: Input size = 784, Output size = 512.
   Activation: LeakyReLU with a slope of 0.3.

   b)Linear Layer 2: Input size = 512, Output size = 256.
   Activation: LeakyReLU with a slope of 0.2.

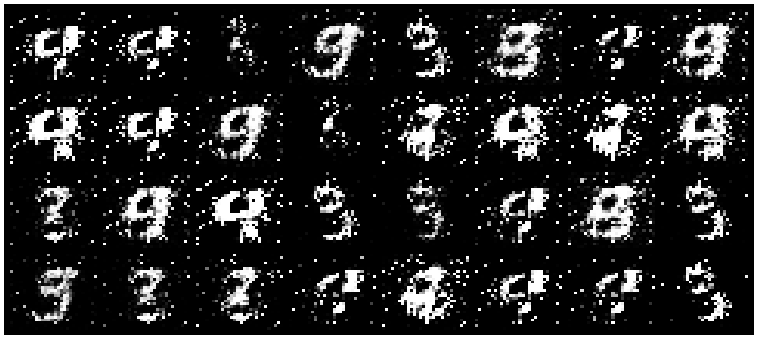
c)Linear Layer 3: Input size = 256, Output size = 128.
Activation: LeakyReLU with a slope of 0.1.

d)Linear Layer 4: Input size = 128, Output size = 1.
Activation: Sigmoid (to output a probability indicating the input image's authenticity).

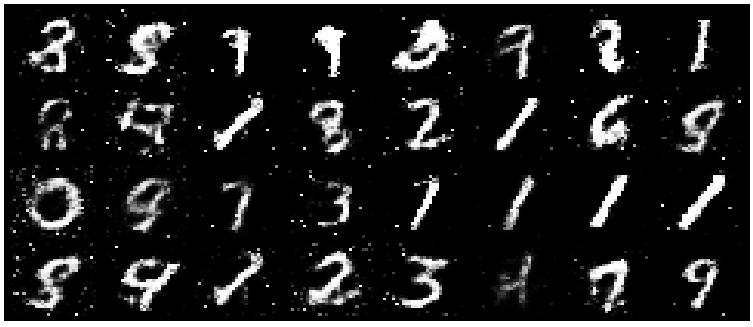
BCE loss measures the dissimilarity between two probability distributions: the predicted probabilities and the target probabilities. In the context of GANs, the BCE loss is used to train both the discriminator and the generator. For the discriminator, the BCE loss is computed separately for real and fake images, and then summed to get the total loss. For the generator, The BCE loss is computed based on the discriminator's output when fed with the generated images.The loss is then back propagated through the network, and the optimizer updates the model parameters based on the computed gradients.

The following are the images of the reconstructions of random points for latent sizes from [2, 4, 8, 16, 32, 64]:

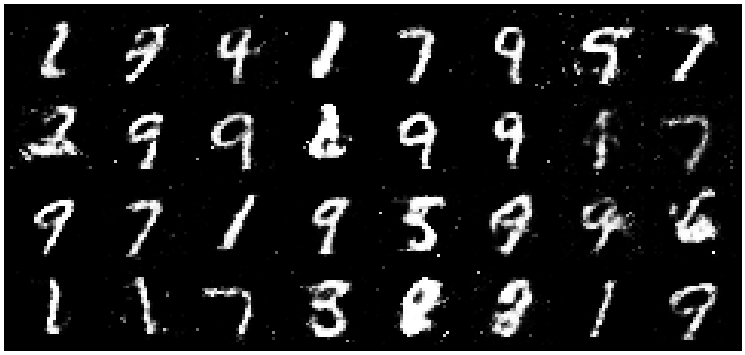
Latent Size:2



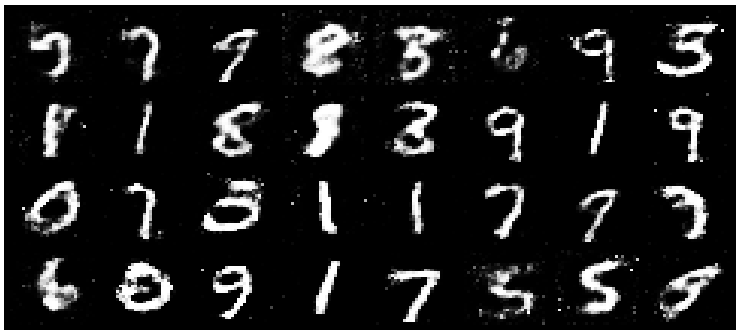
Latent Size :4



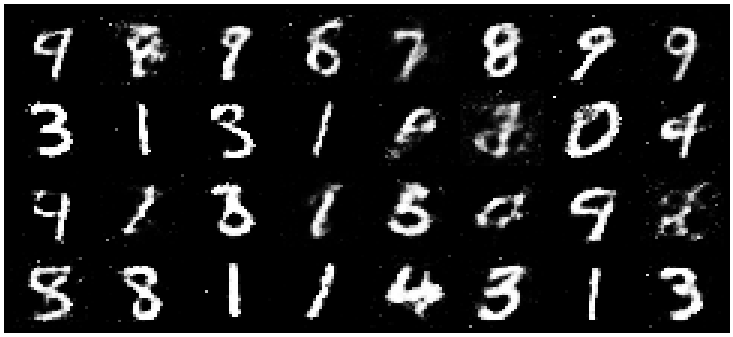
Latent Size :8



Latent Size:16



Latent Size: 32

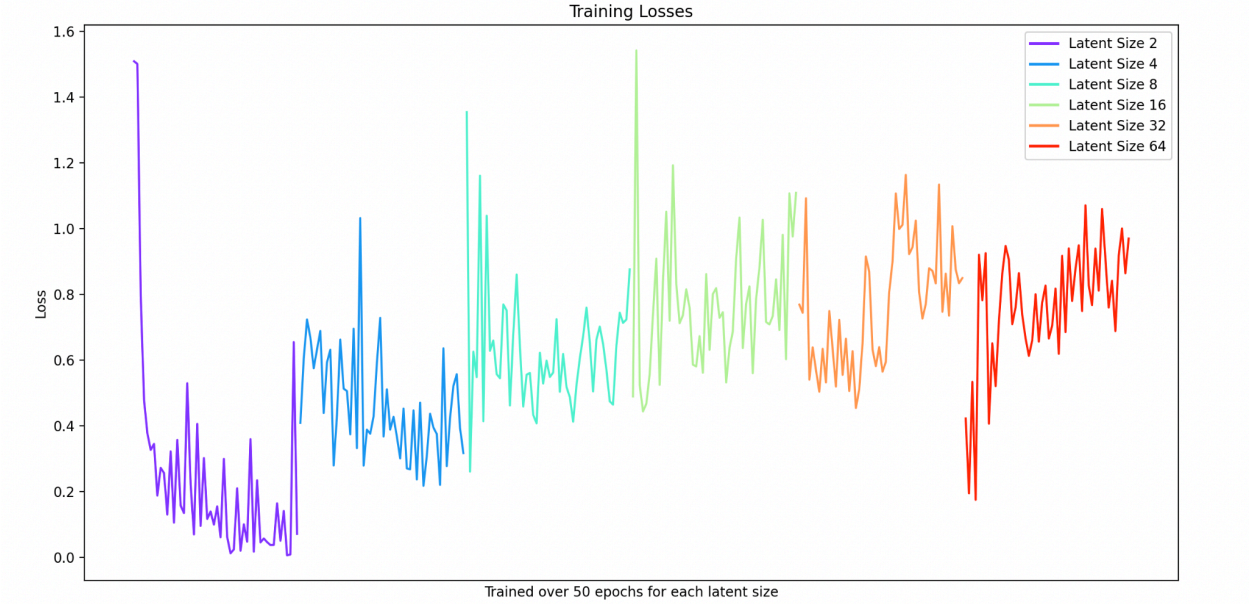


Latent Size: 64

The following images also depict how the training loss reduces over epochs for each latent size:

Generator

Discriminator



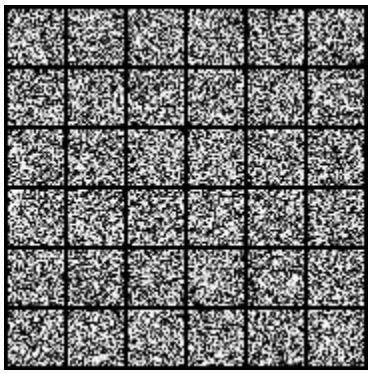
Diffusion Models

We implement a diffusion model for the MNIST dataset. Diffusion models are used for image generation and manipulation tasks. This model applies forward and reverse diffusion processes to images, allowing for the generation of high-quality samples.

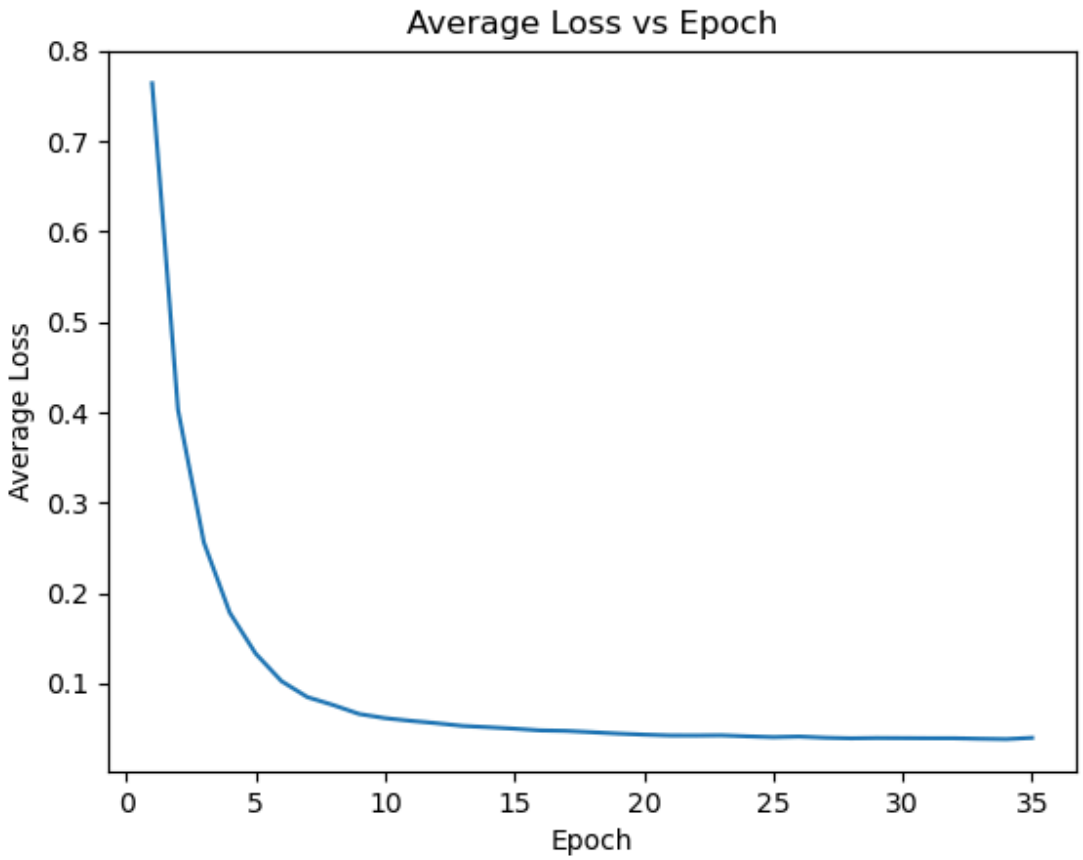
SiLU (Sigmoid-weighted Linear Unit) is used as the activation function.

We also chose to use the Adam Optimizer, since it generally requires a lower learning rate, and that it converges faster.

The architecture consists of a U-Net model with specific modifications for handling diffusion processes.
There are 2 convolutional layers with “relu” as the activation function.
The model utilizes a cosine variance schedule for the diffusion process.It also implements forward and reverse diffusion processes.
Mean Squared Error(MSE) loss is used for training.Sampling from the model is done after each epoch
These are the results obtained from the diffusion model in multiple steps:



The following image depicts how the training loss reduces over epochs:



Part-B

Cycle Gan

We implement a CycleGAN (Cycle-Consistent Generative Adversarial Network) for image-to-image translation. CycleGANs are used to learn mappings between two different domains without requiring paired data.
The activation functions used are :
Leaky ReLU: Used in both the generator and discriminator.
ReLU: Used in the generator for non-linearity.
Sigmoid: Used in the discriminator to output probabilities.
We also chose to use the Adam Optimizer, since it generally requires a lower learning rate, and that it converges faster.

The architecture is mentioned below:

Generator (Encoder-Decoder Architecture):

Encoder :
- Conv1: Convolutional layer with 64 filters
- Conv2: Convolutional layer with 128 filters
- Conv3: Convolutional layer with 256 filters
- Conv4: Convolutional layer with 512 filters

Decoder:
- Deconv1: Transposed convolutional layer with 256 filters
- Deconv2: Transposed convolutional layer with 128 filters
- Deconv3: Transposed convolutional layer with 64 filters
- Deconv4: Transposed convolutional layer with output_channels filters.

Discriminator:

Convolutional Layers:
- Input: Image with input_channels channels
- Conv1: Convolutional layer with 64 filters
- Conv2: Convolutional layer with 128 filters
- Conv3: Convolutional layer with 256 filters
- Conv4: Convolutional layer with 512 filters

Fully Connected Layers:

Linear1: Linear layer with output size 1000

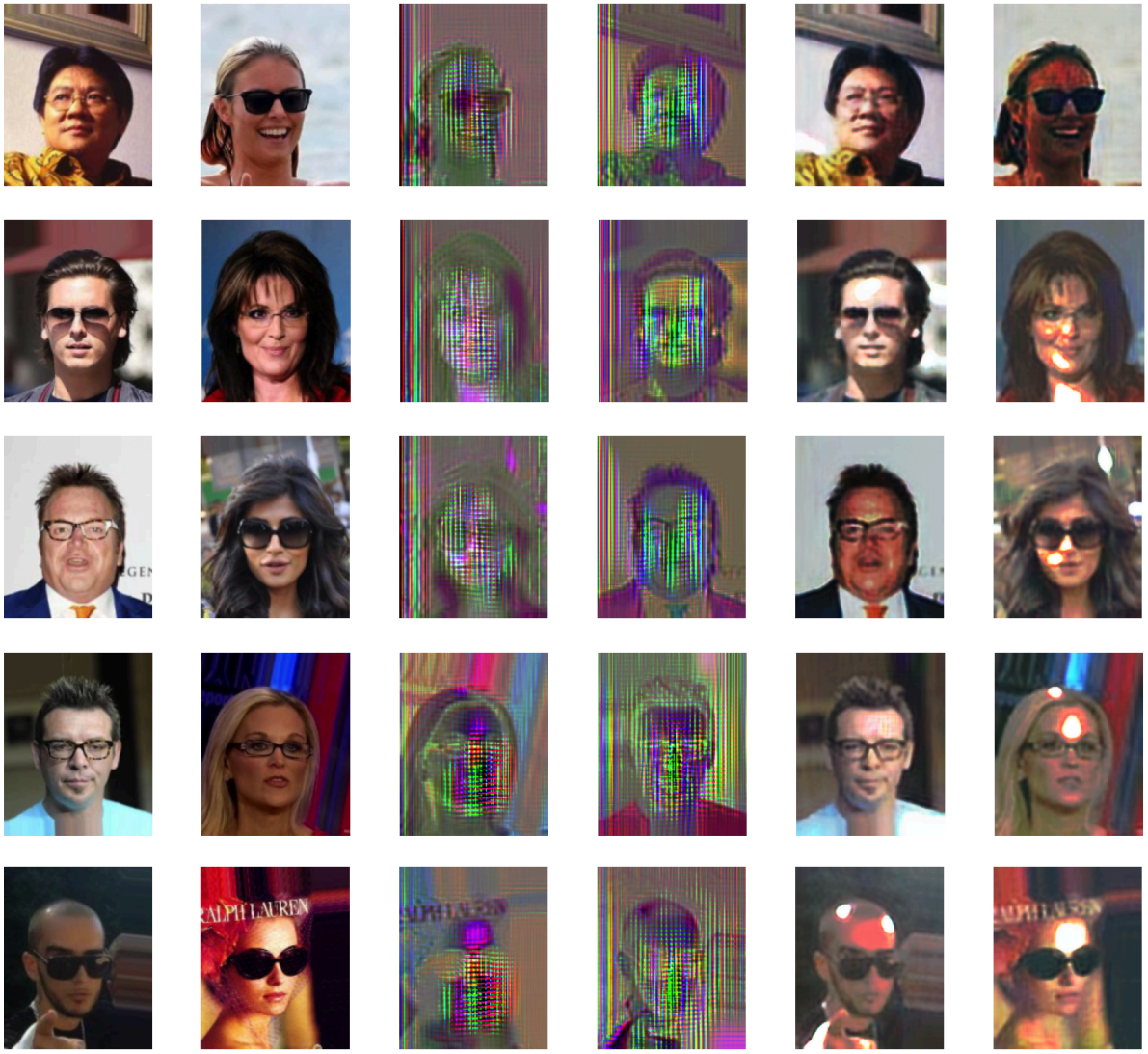
Linear2: Linear layer with output size 1 (for binary classification)

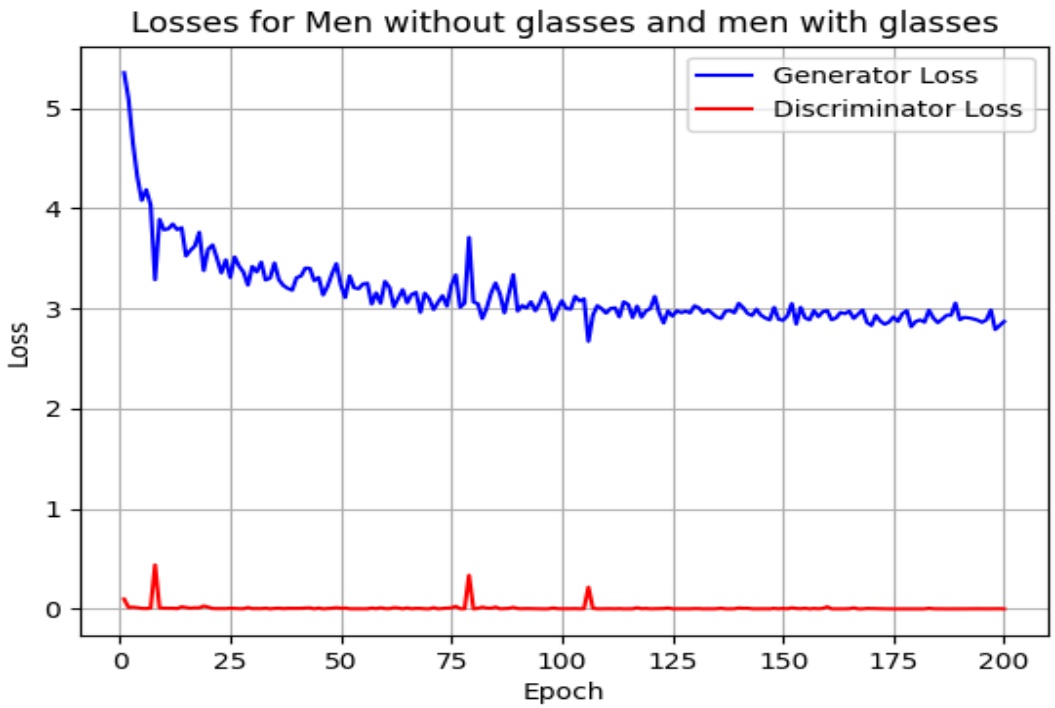
Adversarial Loss (GAN Loss): Binary Cross-Entropy (BCE) loss is used to train the generators and discriminators by minimizing the difference between real and fake predictions.

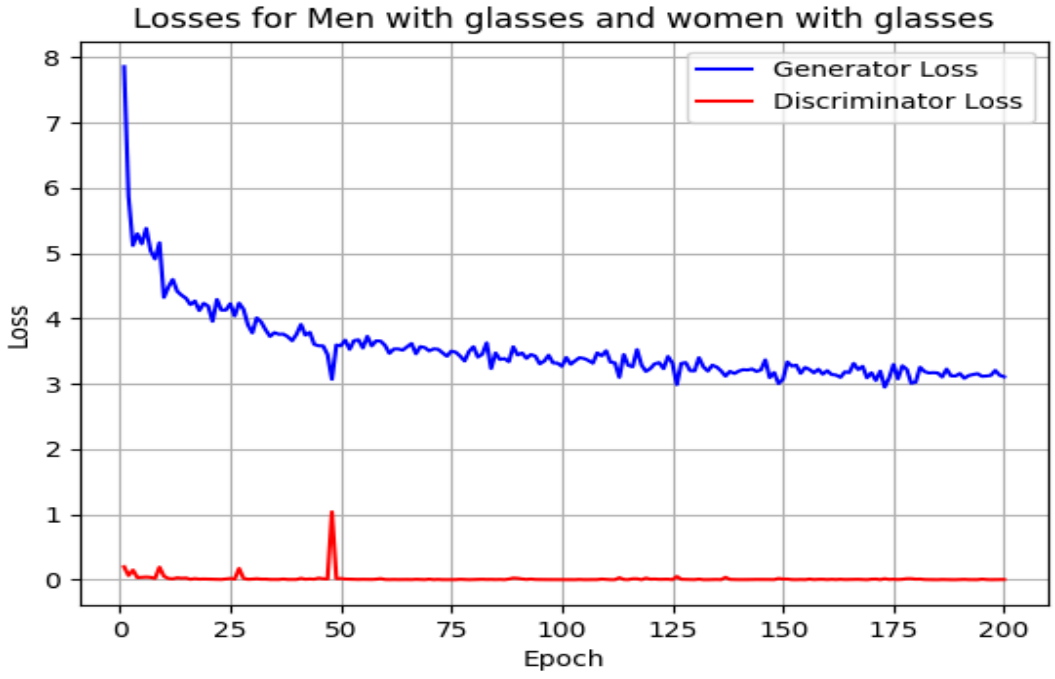
Cycle Consistency Loss: Mean Absolute Error (L1 loss) is used to enforce cycle consistency between the original and reconstructed images, ensuring that the image after translation and back-translation is close to the original image.

Men without glasses to men with glasses and vice versa:

Men with glasses to women with glasses and vice versa:




Losses for Men without glasses and men with glasses
Generator Loss
Discriminator Loss
Loss
Epoch
0
1
2
3
4
5
0
25
50
75
100
125
150
175
200


Losses for Men with glasses and women with glasses
Generator Loss
Discriminator Loss
Loss
Epoch
0
1
2
3
4
5
6
7
8
0
25
50
75
100
125
150
175
200

Deep Convolutional Gan

We implement three neural network models: a Generative Adversarial Network (GAN) Generator, a GAN Discriminator, and a CNN Encoder.

The activation functions used are :
Leaky ReLU: Used in both the discriminator and encoder models
ReLU: Used in the generator for non-linearity.
Sigmoid: Used in the final layer of discriminator to output probabilities

Architecture of Neural Networks:

1. Discriminator

   Output: Single value indicating the probability of the input image being real.

   a. Convolutional layer with 3 input channels, 64 output channels and leaky relu activation.
   b. Convolutional layer with 64 input channels, 128 output channels and leaky relu activation
   c. Convolutional layer with 128 input channels, 256 output channels and leaky relu activation
   d. Convolutional layer with 256 input channels, 512 output channels and leaky relu activation
   e. The output layer consists of a convolutional layer with 512 input channels, 1 output channel, followed by a sigmoid activation layer.

2. Generator

   Output: Generated image with the same dimensions as the input real images (64x64 pixels with 3 channels).

   a. Convolutional transpose layer with 100 input channels, 512 output channels and ReLu activation.
   b. Convolutional transpose layer with 512 input channels, 256 output channels and ReLu activation.
   c. Convolutional transpose layer with 256 input channels, 128 output channels and ReLu activation.
   d. Convolutional transpose layer with 128 input channels, 64 output channels and ReLu activation.
   e. The output layer consists of a convolutional transpose layer with 64 input channels, 3 output channels, followed by a Tanh activation.

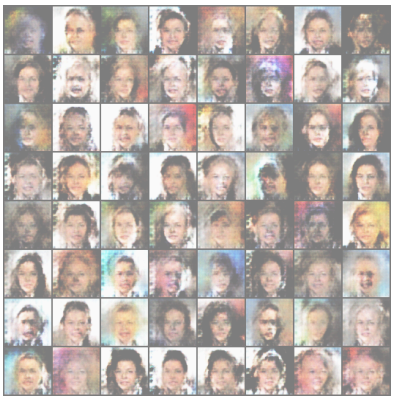
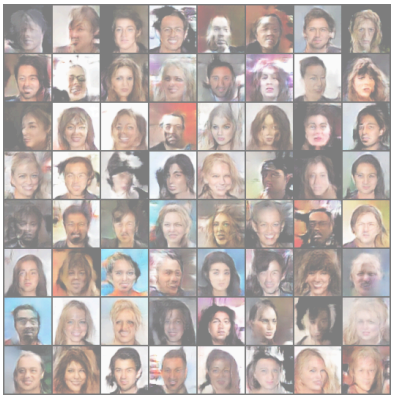
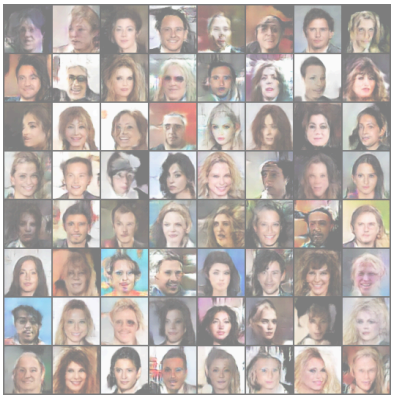
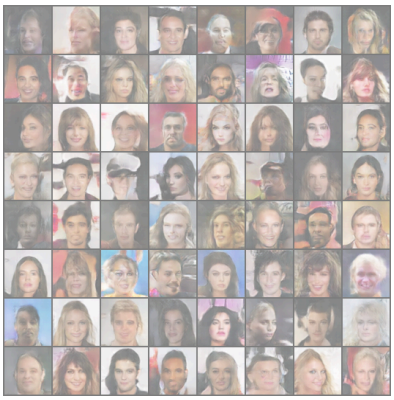
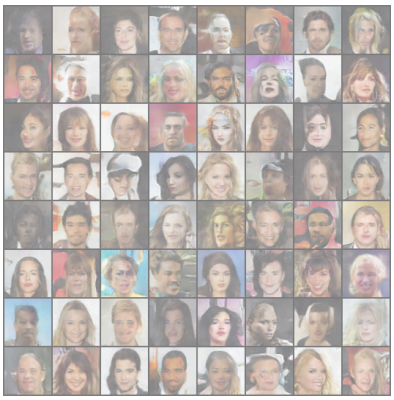
3. Encoder

   Output: Encoded representation of the input image in the form of a noise vector of dimension 100.

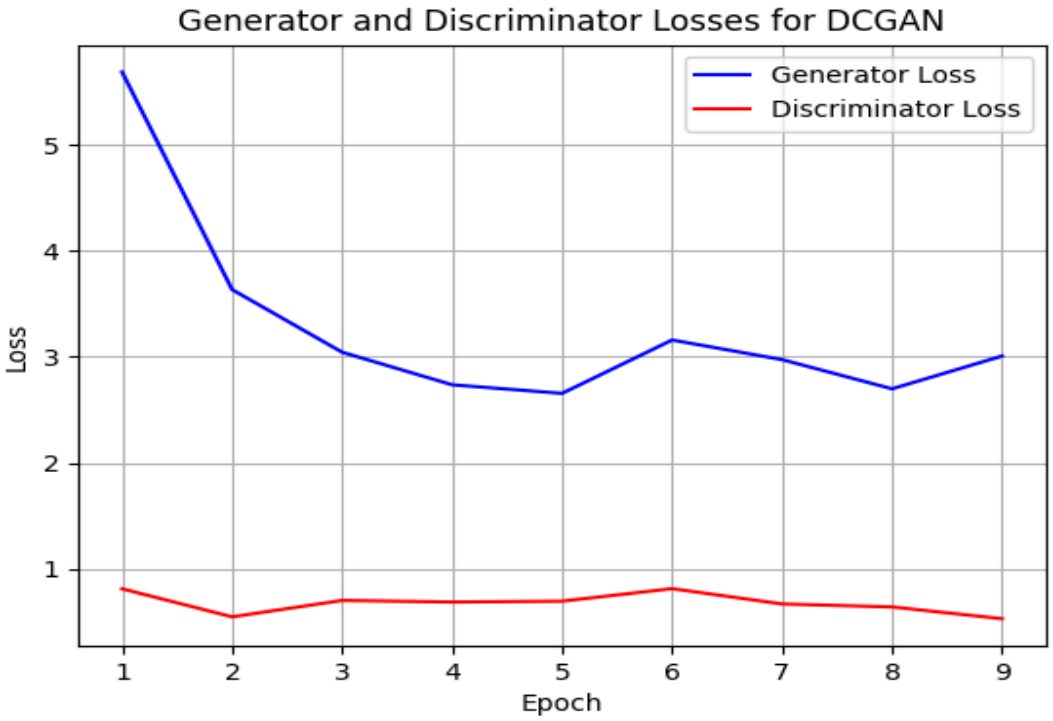
   a. Convolutional layer with 3 input channels, 64 output channels and leaky relu activation
   b. Convolutional layer with 64 input channels, 128 output channels and leaky relu activation
   c. Convolutional layer with 128 input channels,256 output channels and leaky relu activation
   d. Convolutional layer with 256 input channels, 512 output channels and leaky relu activation
   e. The output layer consists of a convolutional layer with 512 input channels, 100 output channels.

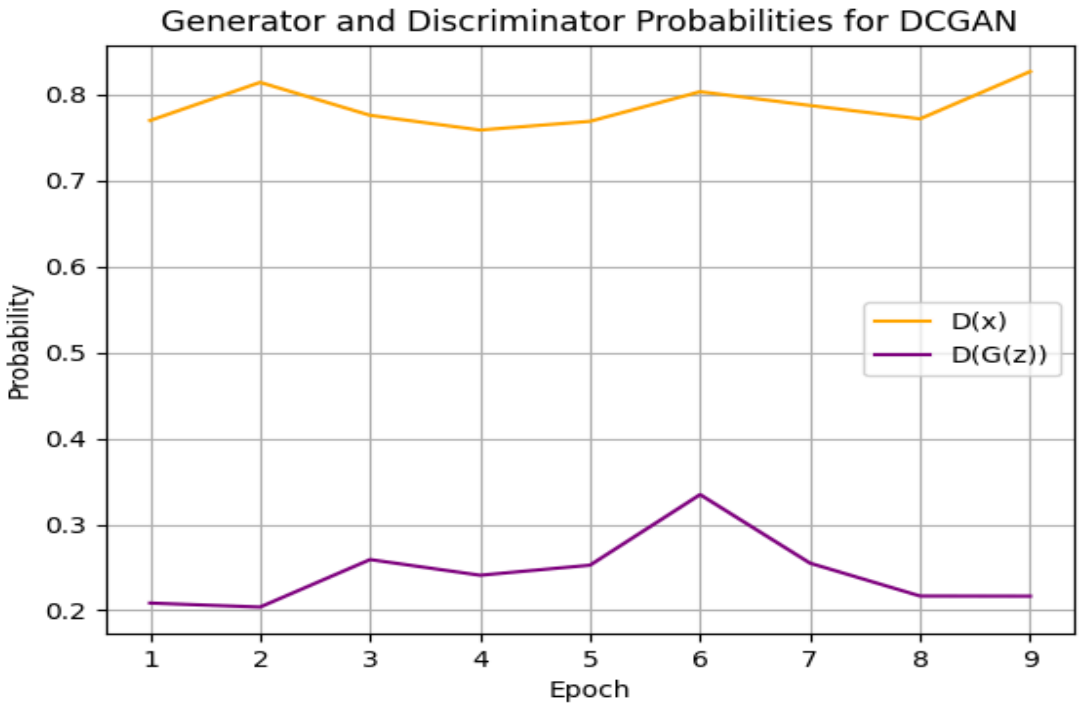
The Loss function used for training is Binary Cross-Entropy Loss.

The following are the outputs of the dcgan generator on different training epochs:

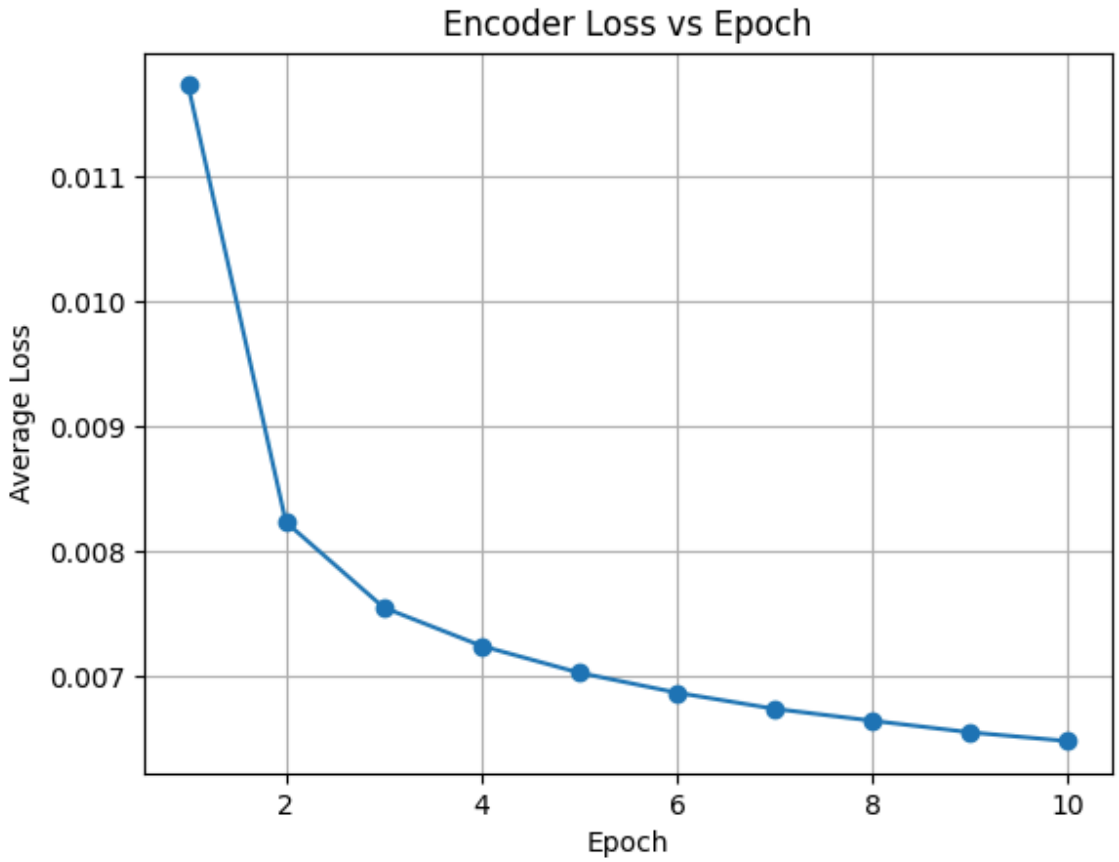
Epoch-0: 	Epoch-1: 	Epoch-2: 
Epoch-3: 	Epoch-4: 	Epoch-5: 
Epoch-6: 	Epoch-7: 	Epoch-8: 

The following image depicts how the training loss reduces over epochs:





Loss plot for encoder during training

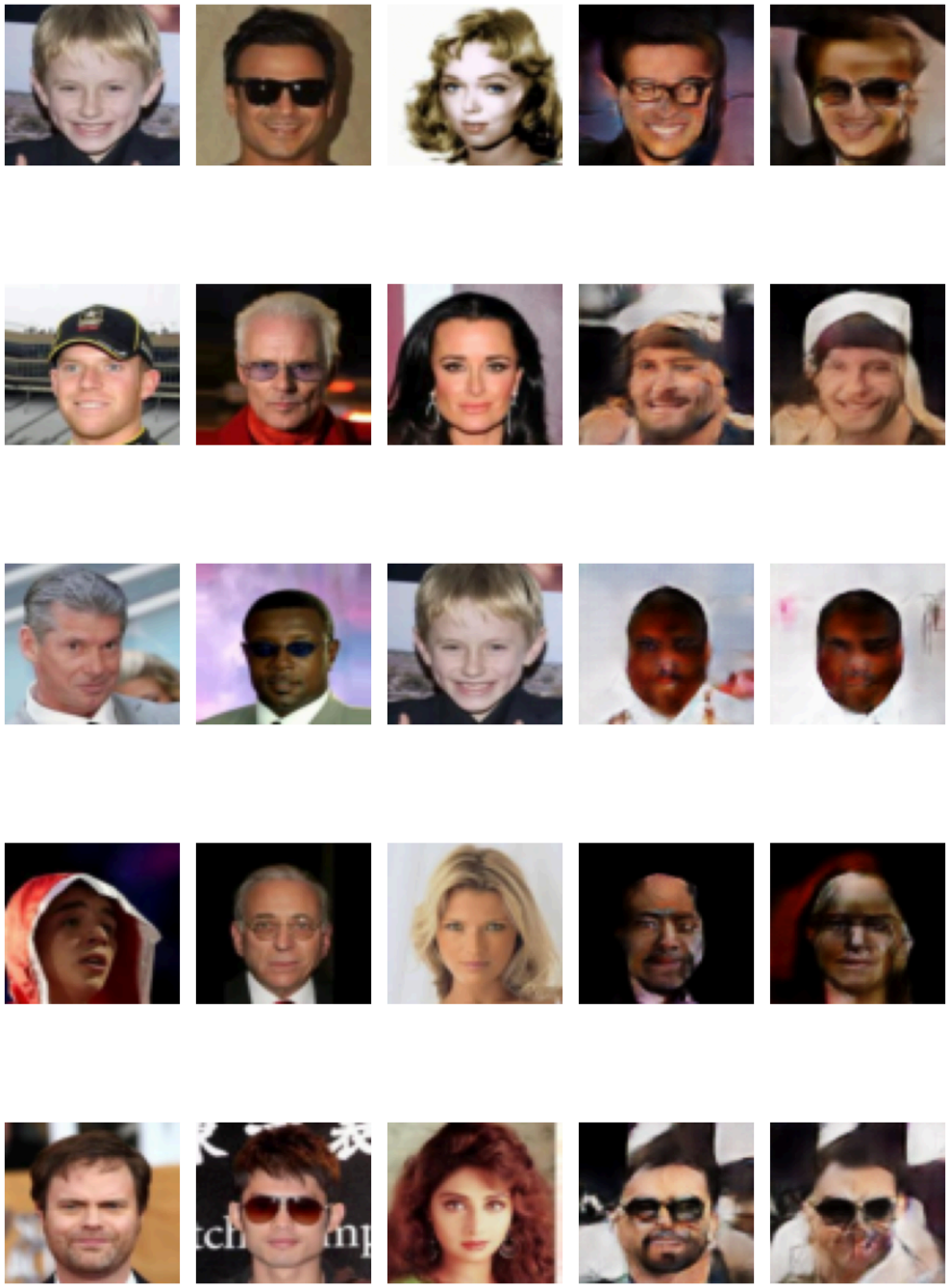


Vector Arithmetic Result:

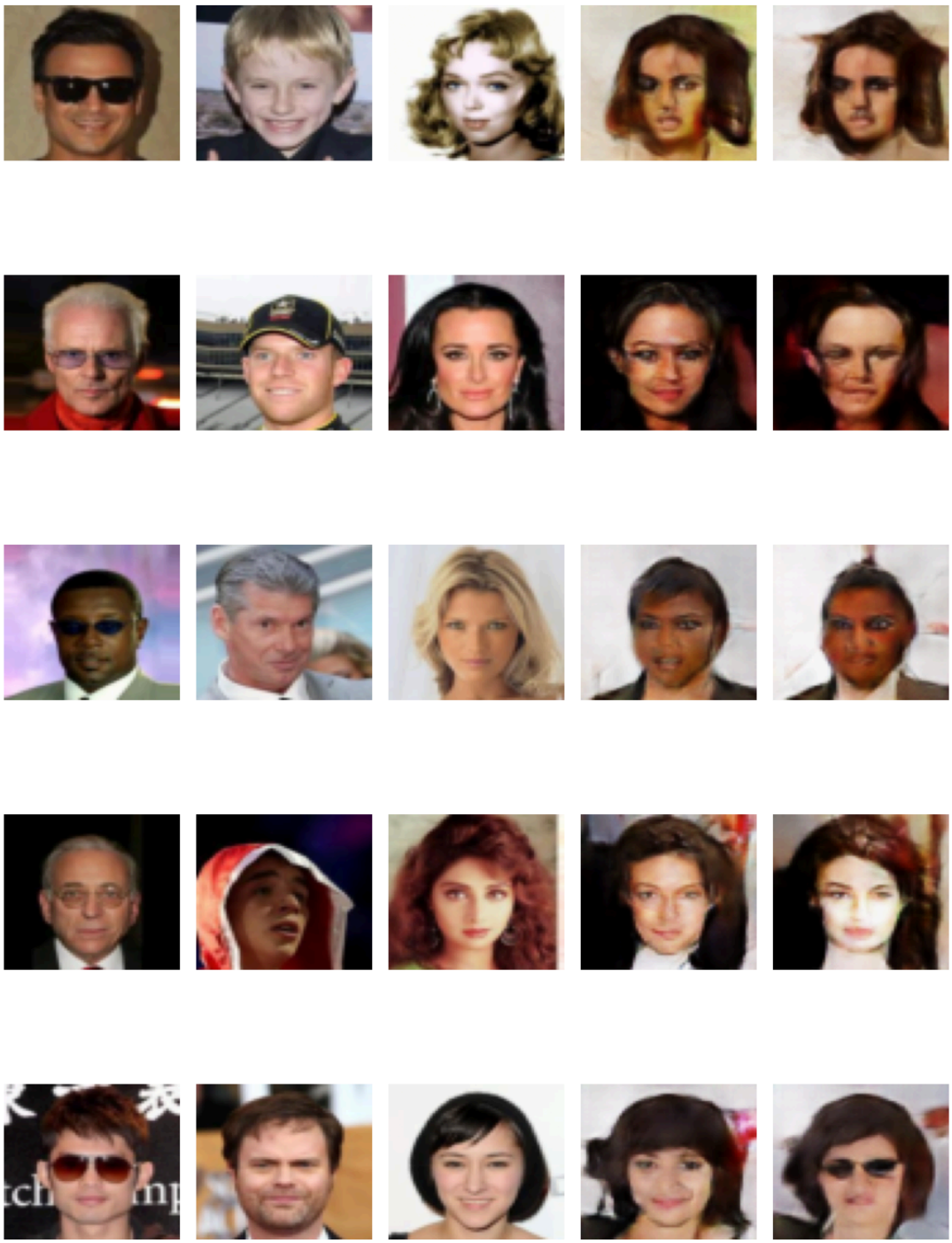
The first (n-2) columns are the inputs to the vector arithmetics , the (n-1)th column on every row is obtained by performing the arithmetics on the average of z vectors of the first columns of each row and then passing that to generator, and the last column is obtained by performing the vector arithmetic directly on image space, for first (n-2) columns and then passing it through the encoder and finally passing z-vector through the DCGAN generator.

.

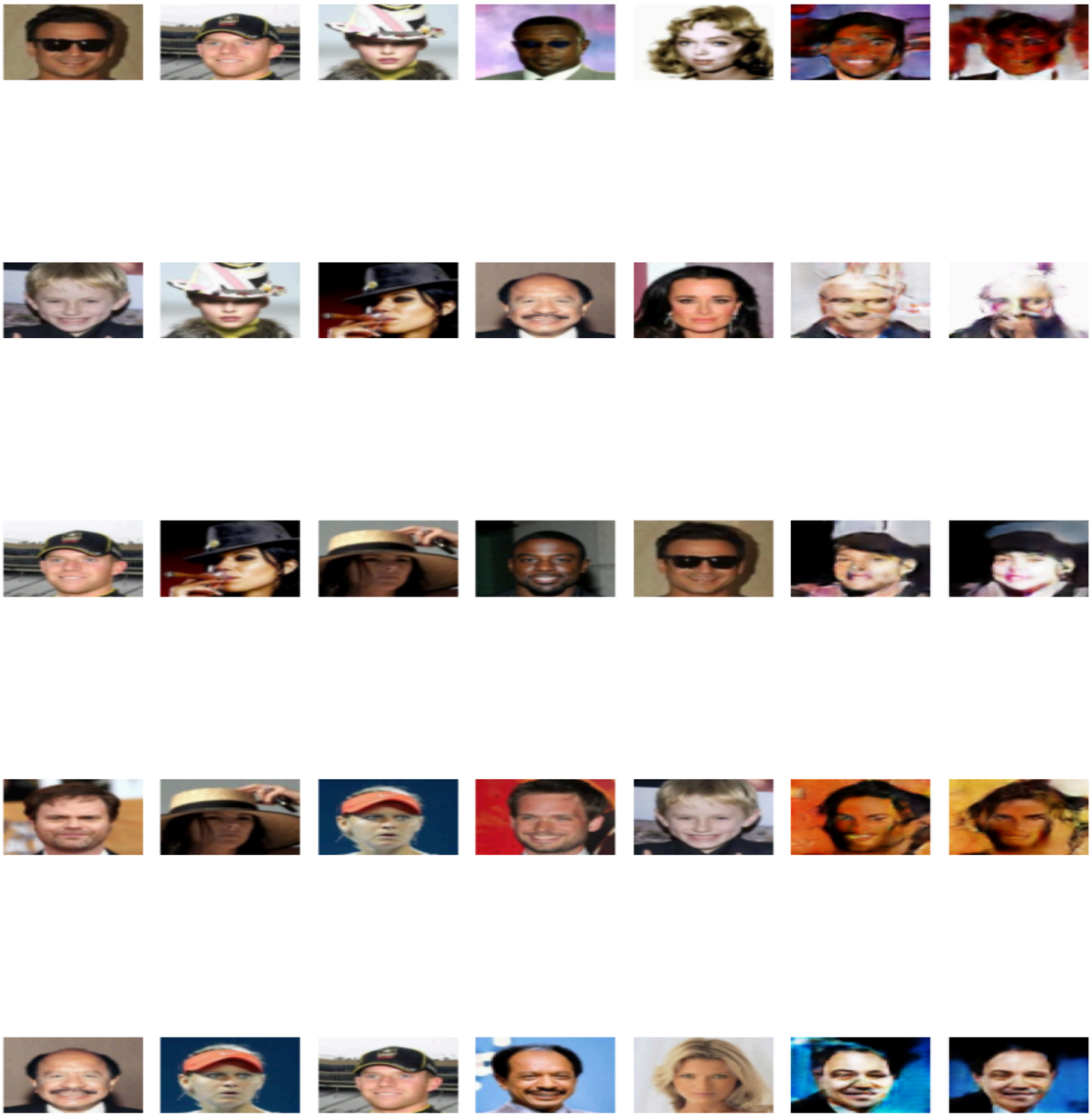
1.Men without glasses + People with glasses - People without glasses



2. Men with glasses - Men without glasses + Women without glasses



3.Smiling Men + People with Hat - People with Hat + People with Mustache - People without Mustache



4.Smiling Men + People with Hat - People without Hat + People with Mustache - People without Mustache

The following images illustrate 3 x 3 grids of the arithmetic outputs over the 3 classes asked for:

1.Men without glasses + People with glasses - People without glasses

2. Men with glasses - Men without glasses + Women without glasses

3.Smiling Men + People with Hat - People with Hat + People with Mustache - People without Mustache

Conclusion

We would like to express our sincere gratitude to Dr. NL Bhanumurthy sir, for giving us this project, and allowing us to explore and learn more about these generative models, and learn the implementation of the same. We had the opportunity to figure out little loopholes in our preconceived notions about the same as we hit stumbling blocks through the course of working on this, and therefore believe we have grown conceptually stronger as we progressed.

Sincerely
Aashutosh
Pavas
Tushar