ADRL (E9 333) Assignment 1 Report

Sasanka Sekhar Sahu, Adarsh Shah
sasankasahu@iisc.ac.in, adarshshah@iisc.ac.in

October 5, 2022

1 VAE: Problem 1

In this problem, we were asked to train a Vannila Variational Autoencoder on two datasets: CelebA & dSprites. For CelebA dataset, we transformed & centercropped every image into $3 \times 128 \times 128$. The dimention of latent space is kept to 128×1. The architechture details for encoder & decoder are mentioned in Figure 1 & 2.

```
Encoder(
(convs): ModuleList(
|    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
|    (1): LeakyReLU(negative_slope=0.01)
|    (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
|    (3): LeakyReLU(negative_slope=0.01)
|    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
|    (5): LeakyReLU(negative_slope=0.01)
|    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
|    (7): LeakyReLU(negative_slope=0.01)
|    (8): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
|    (9): LeakyReLU(negative_slope=0.01)
)
(mean): Linear(in_features=8192, out_features=128, bias=True)
(std): Linear(in_features=8192, out_features=128, bias=True)
)
```

```
==========================================================
Layer (type:depth-idx)   Output Shape          Param #
==========================================================
Encoder                  [128, 128]            --
├─ModuleList: 1-1        --                    --
│    └─Conv2d: 2-1       [128, 32, 64, 64]     896
│    └─LeakyReLU: 2-2    [128, 32, 64, 64]     --
│    └─Conv2d: 2-3       [128, 64, 32, 32]     18,496
│    └─LeakyReLU: 2-4    [128, 64, 32, 32]     --
│    └─Conv2d: 2-5       [128, 128, 16, 16]    73,856
│    └─LeakyReLU: 2-6    [128, 128, 16, 16]    --
│    └─Conv2d: 2-7       [128, 256, 8, 8]      295,168
│    └─LeakyReLU: 2-8    [128, 256, 8, 8]      --
│    └─Conv2d: 2-9       [128, 512, 4, 4]      1,180,160
│    └─LeakyReLU: 2-10   [128, 512, 4, 4]      --
├─Linear: 1-2            [128, 128]            1,048,704
├─Linear: 1-3            [128, 128]            1,048,704
==========================================================
Total params: 3,665,984
Trainable params: 3,665,984
```

Figure 1: (Left) Encoder Architecture. (Right) Input, Intermediate & Output shapes along with number of parameters in Encoder.

```
Decoder(
(in_decoder): Linear(in_features=128, out_features=2048, bias=True)
(convs): ModuleList(
|   (0): ConvTranspose2d(512, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
|   (1): LeakyReLU(negative_slope=0.01)
|   (2): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
|   (3): LeakyReLU(negative_slope=0.01)
|   (4): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
|   (5): LeakyReLU(negative_slope=0.01)
|   (6): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
|   (7): LeakyReLU(negative_slope=0.01)
|   (8): ConvTranspose2d(32, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
|   (9): LeakyReLU(negative_slope=0.01)
|   (10): ConvTranspose2d(16, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
|   (11): LeakyReLU(negative_slope=0.01)
|   (12): Conv2d(16, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
|   (13): Sigmoid()
)
)
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Decoder                                  [128, 3, 128, 128]        --
├─Linear: 1-1                            [128, 2048]               264,192
├─ModuleList: 1-2                        --                        --
│    └─ConvTranspose2d: 2-1              [128, 256, 4, 4]          1,179,904
│    └─LeakyReLU: 2-2                    [128, 256, 4, 4]          --
│    └─ConvTranspose2d: 2-3              [128, 128, 8, 8]          295,040
│    └─LeakyReLU: 2-4                    [128, 128, 8, 8]          --
│    └─ConvTranspose2d: 2-5              [128, 64, 16, 16]         73,792
│    └─LeakyReLU: 2-6                    [128, 64, 16, 16]         --
│    └─ConvTranspose2d: 2-7              [128, 32, 32, 32]         18,464
│    └─LeakyReLU: 2-8                    [128, 32, 32, 32]         --
│    └─ConvTranspose2d: 2-9              [128, 16, 64, 64]         4,624
│    └─LeakyReLU: 2-10                   [128, 16, 64, 64]         --
│    └─ConvTranspose2d: 2-11             [128, 16, 128, 128]       2,320
│    └─LeakyReLU: 2-12                   [128, 16, 128, 128]       --
│    └─Conv2d: 2-13                      [128, 3, 128, 128]        435
│    └─Sigmoid: 2-14                     [128, 3, 128, 128]        --
==========================================================================================
Total params: 1,838,771
Trainable params: 1,838,771
```

Figure 2: (Left) Decoder Architecture. (Right) Input, Intermediate & Output shapes along with number of parameters in Decoder.

The batch size used was 128. Adam optimizer was used with default learning rate & parameters. The model was trained for around 30 epochs. The genrated images from the decoder as shown in Figure 3.

Figure 3: 10 × 10 grid of images generated from the Decoder after training for 30 epochs on the CelebA dataset.

For dSprites dataset, the images are of dimentions 64 × 64. Apart from slight modification to incorporate for the different input size, Encoder & Decoder architecture are similar to CelebA. All the hyperparameters are same as CelebA. The results are shown in Figure 4.

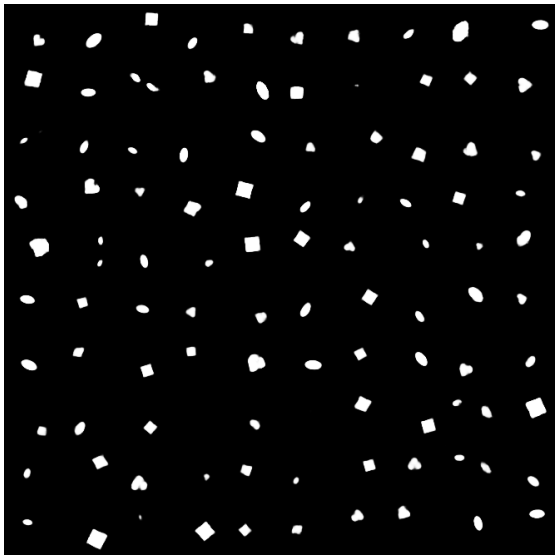

Figure 4: 10 × 10 grid of images generated from the Decoder after training for 30 epochs on the CelebA dataset.

Further, we needed to calculate the marginal likelihood scores in both the cases. For CelebA, we calculated the marginals for 10 random images and

the calculated values are **1.4381e-23, 2.9066e-24, 6.0882e-28, 6.6265e-24, 2.0169e-25, 1.3046e-29, 1.0164e-21, 5.5556e-22, 6.4115e-29, 7.4224e-28**. Similarly, we calculated marginals for 10 random images in dSprites dataset. The values are as follows: **0.0253, 0.0263, 0.0208, 0.0269, 0.0237, 0.0242, 0.0263, 0.0227, 0.0253, 0.0246**.

2 VAE: Problem 3

The VQ-VAE is implemented to encode tiny-image dataset. The encoder compresses the 64x64x3 images to 32x32x1 latents. The latents are further quantized to 512 bins. Hence, we achived compression ratio of $\frac{64\times64\times3\times256}{32\times32\times512}$. We also tried quantization bins of size 256 and 128. The reconstruction from 512 quantized bins is better from 128 and 256 as evident in the Figures 5,6 and 7

Figure 5: 10 × Reconstructed image from codebook of size 128 bins.

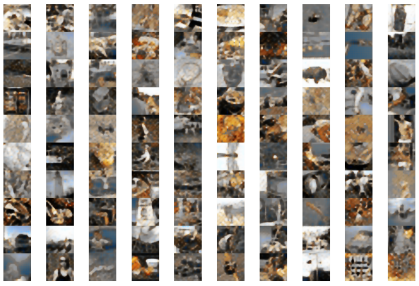

Figure 6: 10 × Reconstructed image from codebook of size 256 bins.

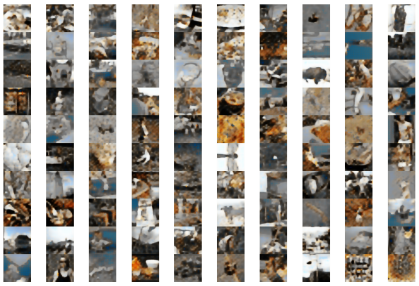

Figure 7: 10 × Reconstructed image from codebook of size 512 bins.

We generated quantized latents dataset by performing a single pass through tiny image dataset. We fitted a GMM with 100 components and a simple VAE on it. The following images are generated from vq-vae's decoder after sampling points from GMM and latent's vae.

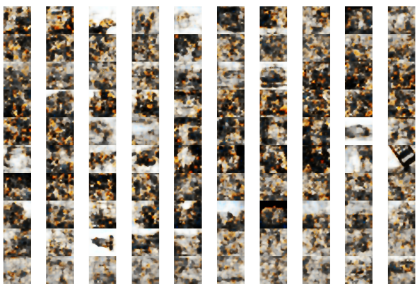

Figure 8: 10 × Images generated using latents sampled from GMM.

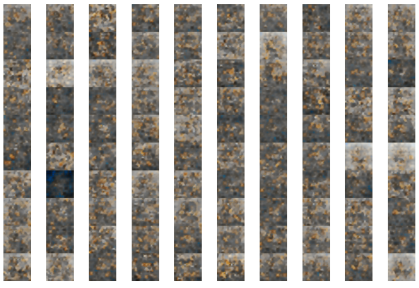

Figure 9: 10 × Images generated using latents sampled from VAE.

3 GAN: Problem 1

In this problem we needed to train a DCGAN on bitmoji Dataset. The images in this dataset are of dimention $3 \times 64 \times 64$. The architecture of the Generator & Discriminator are shown in Figure 10 & 11.

```
Generator(
  (layers): ModuleList(
    |  (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    |  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    |  (2): ReLU(inplace=True)
    |  (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False
    |  (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    |  (5): ReLU(inplace=True)
    |  (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False
    |  (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    |  (8): ReLU(inplace=True)
    |  (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    |  (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    |  (11): ReLU(inplace=True)
    |  (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    |  (13): Tanh()
  )
)
```

```
==========================================================
Layer (type:depth-idx)      Output Shape           Param #
==========================================================
Discriminator               [128, 1, 1, 1]         --
├─ModuleList: 1-1           --                     --
    └─Conv2d: 2-1           [128, 64, 32, 32]      3,072
    └─LeakyReLU: 2-2        [128, 64, 32, 32]      --
    └─Conv2d: 2-3           [128, 128, 16, 16]     131,072
    └─BatchNorm2d: 2-4      [128, 128, 16, 16]     256
    └─LeakyReLU: 2-5        [128, 128, 16, 16]     --
    └─Conv2d: 2-6           [128, 256, 8, 8]       524,288
    └─BatchNorm2d: 2-7      [128, 256, 8, 8]       512
    └─LeakyReLU: 2-8        [128, 256, 8, 8]       --
    └─Conv2d: 2-9           [128, 512, 4, 4]       2,097,152
    └─BatchNorm2d: 2-10     [128, 512, 4, 4]       1,024
    └─LeakyReLU: 2-11       [128, 512, 4, 4]       --
    └─Conv2d: 2-12          [128, 1, 1, 1]         8,192
==========================================================
Total params: 2,765,568
Trainable params: 2,765,568
```

Figure 10: 10×10 grid of images generated from the Decoder after training for 30 epochs on the CelebA dataset.

```
Discriminator(
  (layers): ModuleList(
    |  (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    |  (1): LeakyReLU(negative_slope=0.2, inplace=True)
    |  (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    |  (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    |  (4): LeakyReLU(negative_slope=0.2, inplace=True)
    |  (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    |  (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    |  (7): LeakyReLU(negative_slope=0.2, inplace=True)
    |  (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    |  (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    |  (10): LeakyReLU(negative_slope=0.2, inplace=True)
    |  (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
  )
)
```

```
==========================================================
Layer (type:depth-idx)      Output Shape           Param #
==========================================================
Generator                   [128, 3, 64, 64]       --
├─ModuleList: 1-1           --                     --
    └─ConvTranspose2d: 2-1  [128, 512, 4, 4]       819,200
    └─BatchNorm2d: 2-2      [128, 512, 4, 4]       1,024
    └─ReLU: 2-3             [128, 512, 4, 4]       --
    └─ConvTranspose2d: 2-4  [128, 256, 8, 8]       2,097,152
    └─BatchNorm2d: 2-5      [128, 256, 8, 8]       512
    └─ReLU: 2-6             [128, 256, 8, 8]       --
    └─ConvTranspose2d: 2-7  [128, 128, 16, 16]     524,288
    └─BatchNorm2d: 2-8      [128, 128, 16, 16]     256
    └─ReLU: 2-9             [128, 128, 16, 16]     --
    └─ConvTranspose2d: 2-10 [128, 64, 32, 32]      131,072
    └─BatchNorm2d: 2-11     [128, 64, 32, 32]      128
    └─ReLU: 2-12            [128, 64, 32, 32]      --
    └─ConvTranspose2d: 2-13 [128, 3, 64, 64]       3,072
    └─Tanh: 2-14            [128, 3, 64, 64]       --
==========================================================
Total params: 3,576,704
Trainable params: 3,576,704
```

Figure 11: Plots of a sample CT Scan & the reconstructed images from 4x & 8x Sinogram.

The batch size used was 128. The Adam optimizer was used with default learning rate 0.0002, $\beta_1 = 0.2$ and $\beta_2 = 0.999$. The model was trained for around 10 epochs but the best images obtained were after epoch 5. The results are shown in Figure 12.

We also needed to calculate the FID score between 1000 generated samples and 1000 real samples. We calculated the FID score at every epoch of training, the FID score initially reduced however, it fluctuated towards the end. The best FID score achieved in DCGAN was **131.7**.

Figure 12: 10×10 grid of images generated from the Generator of DCGAN after training for 5 epochs on the Bitmoji dataset.

The loss curve for generator & discriminator is shown in Figure 13 below.

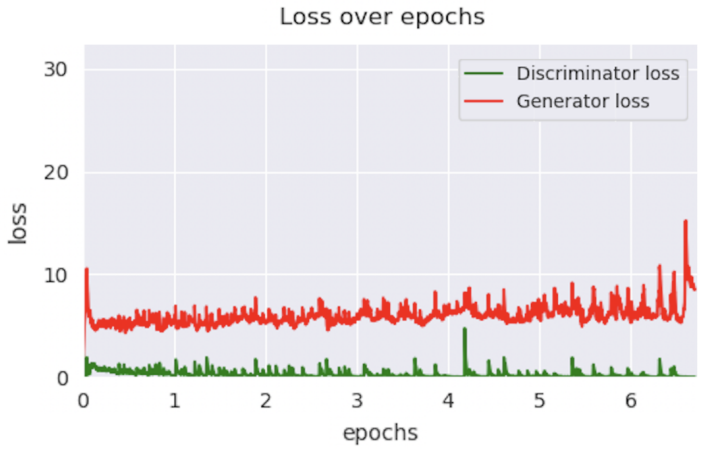

Figure 13: Generator loss (red) & Discriminator loss (blue) over epochs.

As observed from the loss curve, the generator loss was high in the beginning but it quickly reduced to counter the discriminator. Both losses remained stable till 7th epoch after which generator loss shot up which may be due to mode collapse.

4 GAN: Problem 4

In this problem, we needed to train LSGAN and BiGAN. For LSGAN, the architecture is exactly same as DCGAN except the fact that we use MSE Loss function instead of Cross Entropy and Sigmoid was removed from the Discriminator. The results are shown in Figure 14.

Figure 14: 10×10 grid of images generated from the Generator of LSGAN after training for 13 epochs on the Bitmoji dataset.

For BiGAN, the generator architecture is exactly the same as DCGAN. However, the discriminator was slightly modified to incoporate for the encodings and an additional encoder was used whose architecture is shown in Figure 15.

```
Encoder(
  (convs): ModuleList(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): Conv2d(512, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (13): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU(inplace=True)
    (15): Conv2d(512, 100, kernel_size=(1, 1), stride=(1, 1))
  )
)
```

```
==========================================================================
Layer (type:depth-idx)          Output Shape              Param #
==========================================================================
Encoder                         [128, 100, 1, 1]          --
├─ModuleList: 1-1               --                        --
│    └─Conv2d: 2-1              [128, 64, 32, 32]         3,072
│    └─BatchNorm2d: 2-2         [128, 64, 32, 32]         128
│    └─ReLU: 2-3                [128, 64, 32, 32]         --
│    └─Conv2d: 2-4              [128, 128, 16, 16]        131,072
│    └─BatchNorm2d: 2-5         [128, 128, 16, 16]        256
│    └─ReLU: 2-6                [128, 128, 16, 16]        --
│    └─Conv2d: 2-7              [128, 256, 8, 8]          524,288
│    └─BatchNorm2d: 2-8         [128, 256, 8, 8]          512
│    └─ReLU: 2-9                [128, 256, 8, 8]          --
│    └─Conv2d: 2-10             [128, 512, 4, 4]          2,097,152
│    └─BatchNorm2d: 2-11        [128, 512, 4, 4]          1,024
│    └─ReLU: 2-12               [128, 512, 4, 4]          --
│    └─Conv2d: 2-13             [128, 512, 1, 1]          4,194,304
│    └─BatchNorm2d: 2-14        [128, 512, 1, 1]          1,024
│    └─ReLU: 2-15               [128, 512, 1, 1]          --
│    └─Conv2d: 2-16             [128, 100, 1, 1]          51,300
==========================================================================
Total params: 7,004,132
Trainable params: 7,004,132
```

Figure 15: (Left) Encoder Architecture in BiGAN. (Right) Input, Intermediate & Output shapes along with number of parameters in Encoder of BiGAN.

The results are shown in Figure 16.

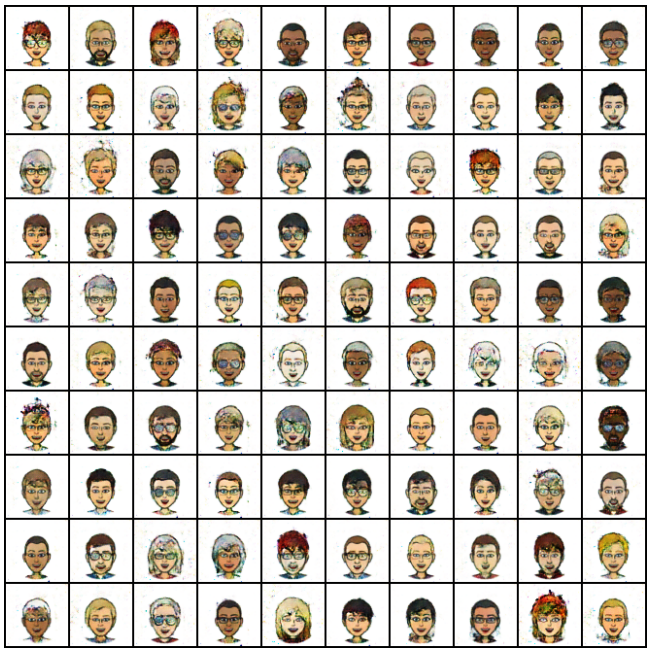

Figure 16: 10 × 10 grid of images generated from the Generator of BiGAN after training for 19 epochs on the Bitmoji dataset.

The best FID score calculated in LSGAN was **129.3** and BiGAN was **133.1**. Further, we needed to perform latent traversal. The results are shown in Figure 17-22.

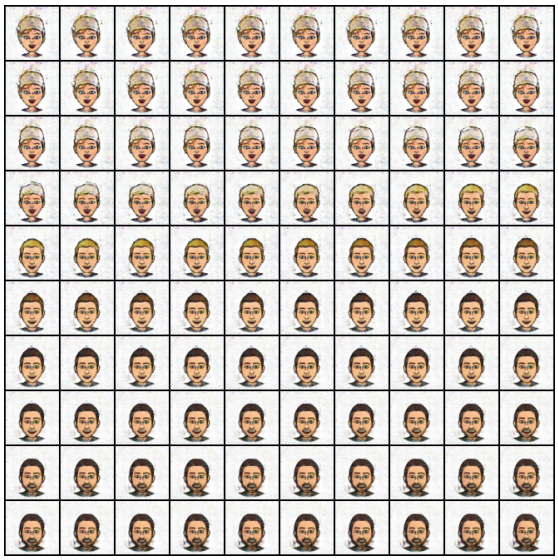

Figure 17: Latent Traversal on DCGAN.

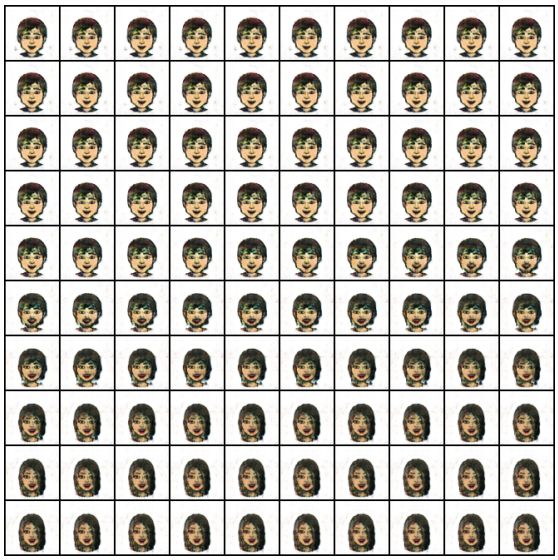

Figure 18: Latent Traversal on DCGAN.

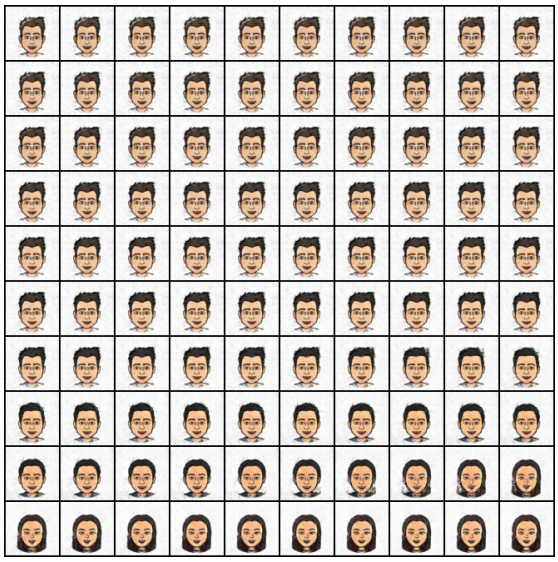

Figure 19: Latent Traversal on LSGAN.

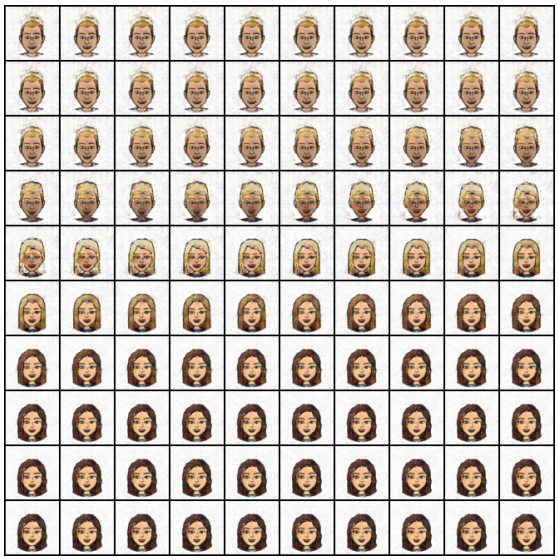

Figure 20: Latent Traversal on LSGAN.

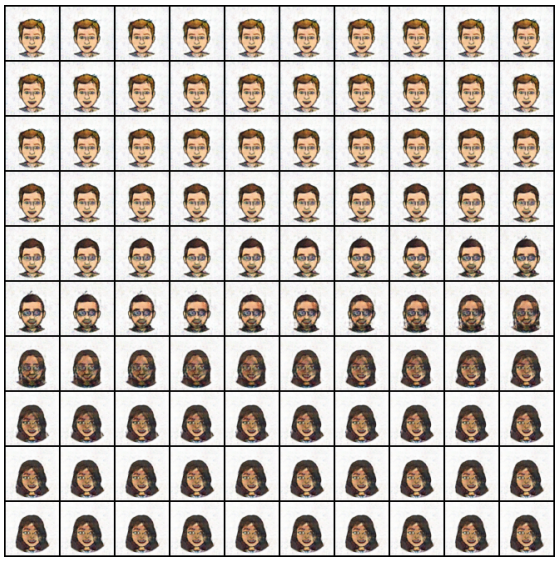

Figure 21: Latent Traversal on BiGAN.

Figure 22: Latent Traversal on BiGAN.