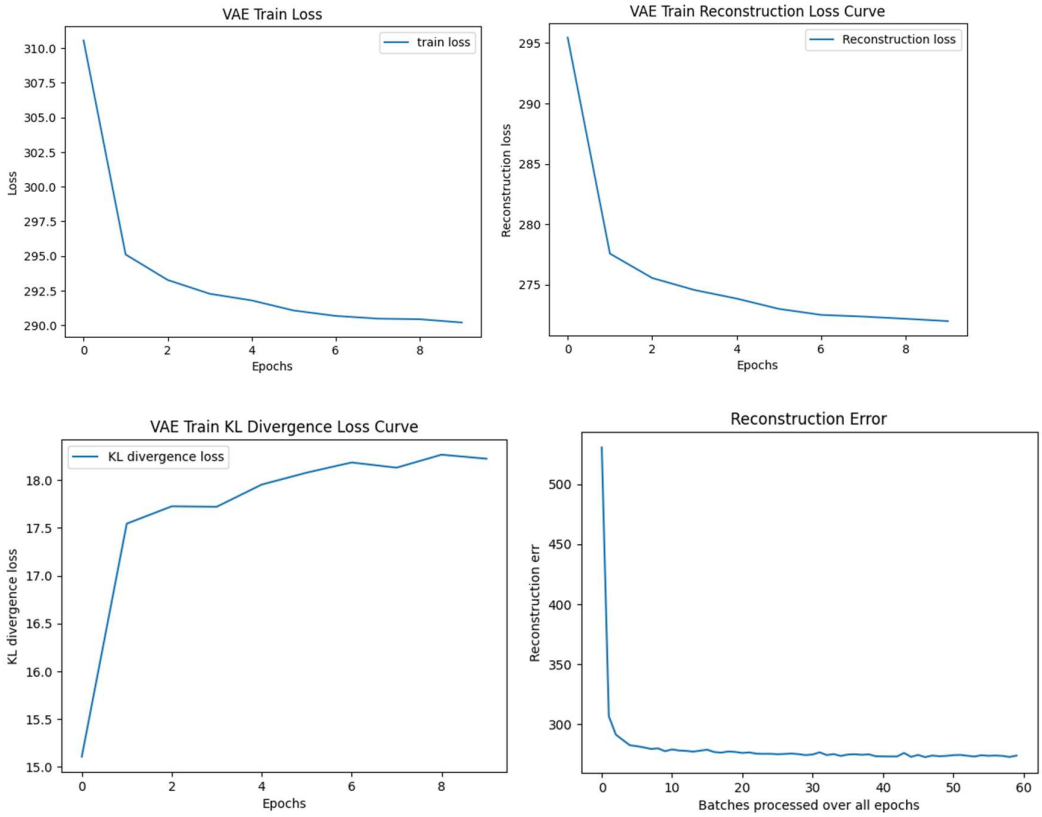
VAE GANS Results

VAE Implementation Results

First, we ran our model on default parameters –

- **zdim = 5**,
- Weight between reconstruction losses = 1:1
- Optimizer = “Adam”,
- Learning Rate = 1e-3,
- Epochs = 10



The Final reconstruction error at end of all epochs is 273.7904052734375

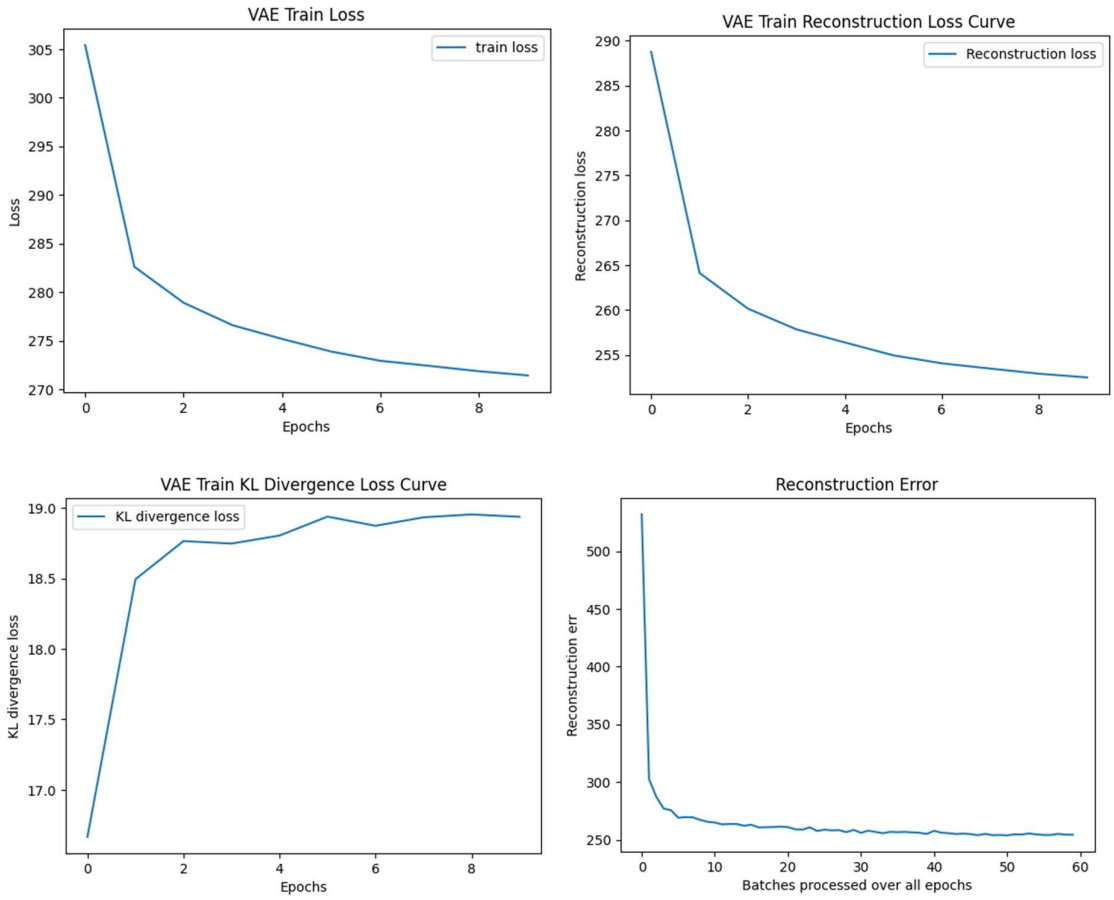
zdim =5

IS SCORE :

```
print('real images IS_score:')       => (4.111614656251431, 0.0)
print('generated images IS_score:') => (2.491037379650279, 0.0)
```

Parameter Tuning:

Next, we ran our model by changing the zdim to **50** since 5 dimensions can be too less for model.




```
The Final reconstruction error at end of all epochs is 254.33189392089844
```

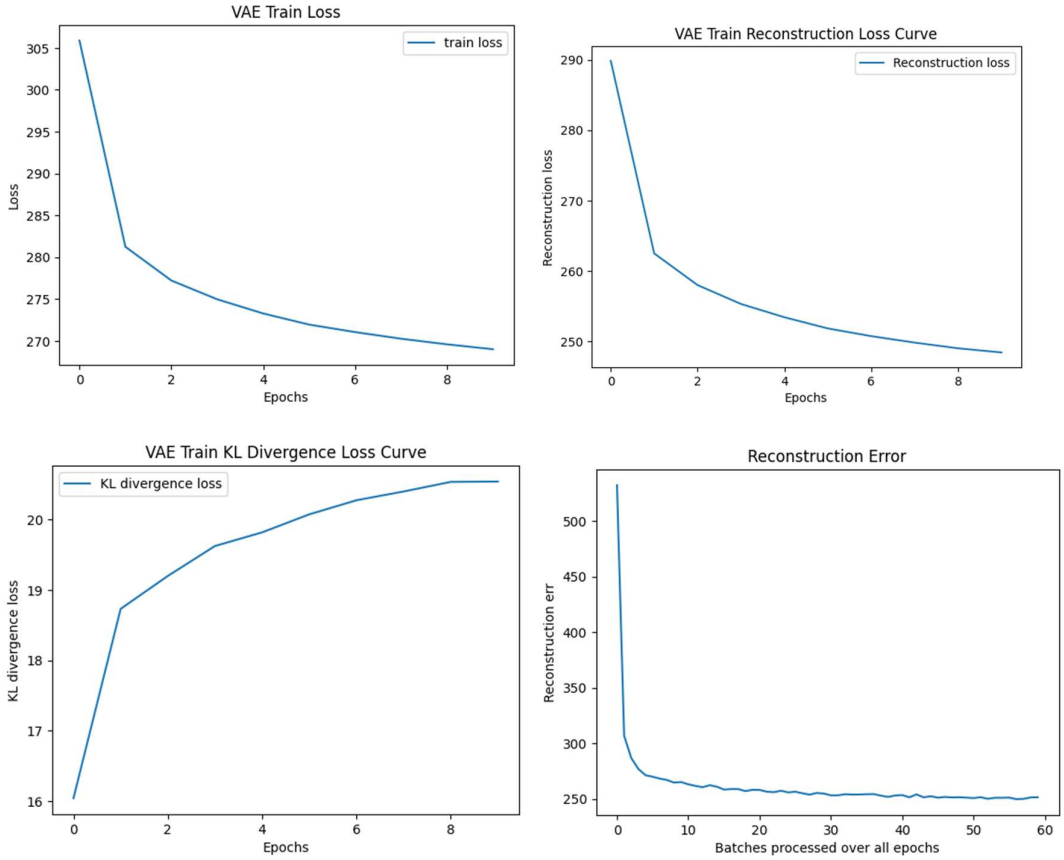
zdim = 50

IS SCORE :

```
print('real images IS_score:')       => (4.217926705523307, 0.0)
print('generated images IS_score:') => (2.6211442134526766, 0.0)
```

The results are little better than 5 dimensions. IS Score is also little higher which is a good sign. We tried again with **zdim = 200.** Results are below:

zdim = 200



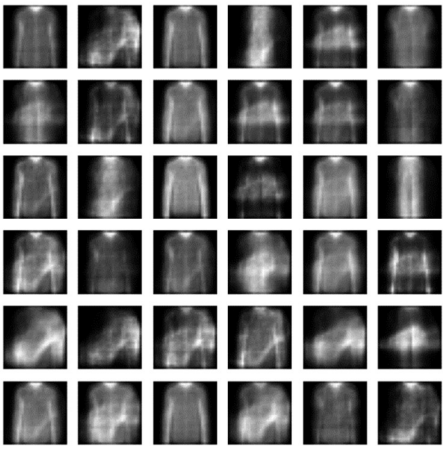
The Final reconstruction error at end of all epochs is 251.50616455078125

IS SCORE :

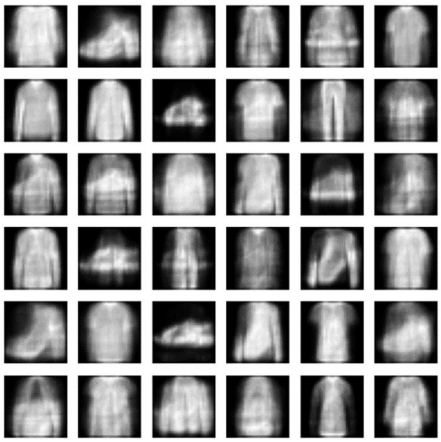
This calculation threw cuda out of memory error due to lack of resources. But we have demonstrated that going from 5 to 50 dimensions, our IS score improved which is desirable.

Qualitative Analysis Results:

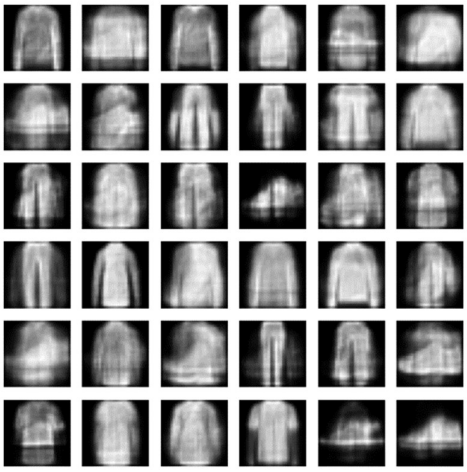
1) We sampled 36 noise vectors from standard Gaussian distribution and plotted the decoder output for those vectors



zdim = 5



zdim = 50

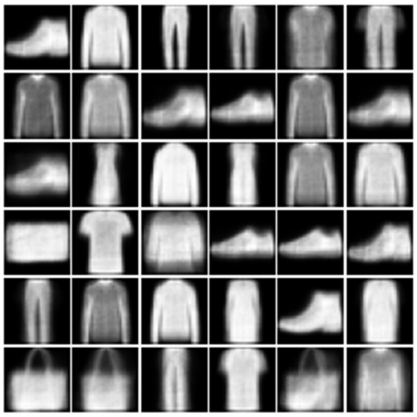


Zdim = 200

2) We sample 36 images from our dataset and visualize the original and reconstructed images.

Original Images

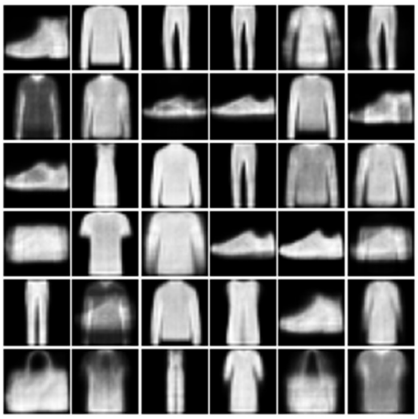
Reconstructed Images



zdim = 5

Original Images

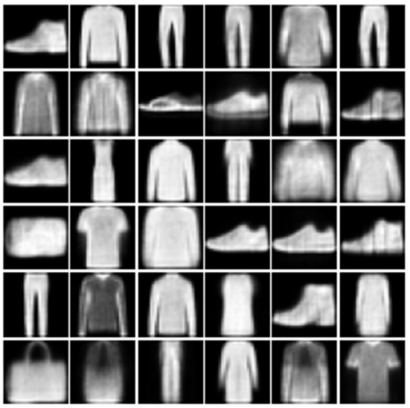
Reconstructed Images



zdim = 50

Original Images

Reconstructed Images



Zdim = 200

As seen, zdim = 200 are much better in reproducing outputs. For example, row 2, column 3 slippers are best represented with 200 dimensions.

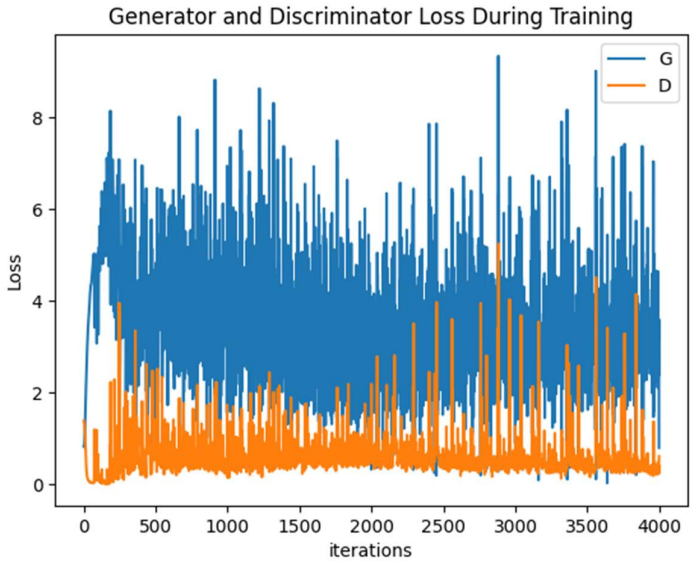
Model tuning discussion:

Overall, our model works better when we increase our zdim which is expected as image is compressed to higher number of latent dimensions. We tested from zdim = [5,50,200] and found best results with 200 dimensions. The images reproduced were far better with 200 dimensions. The losses and reconstruction error are improving as we increase the dimensions. In future, if we increase more dimensions, we might get even better results but due to limited resources (IS Score crashing for zdim = 200), we limit our observations to 200 latent dimensions

GAN Implementation Results

First, we ran our model on default parameters –

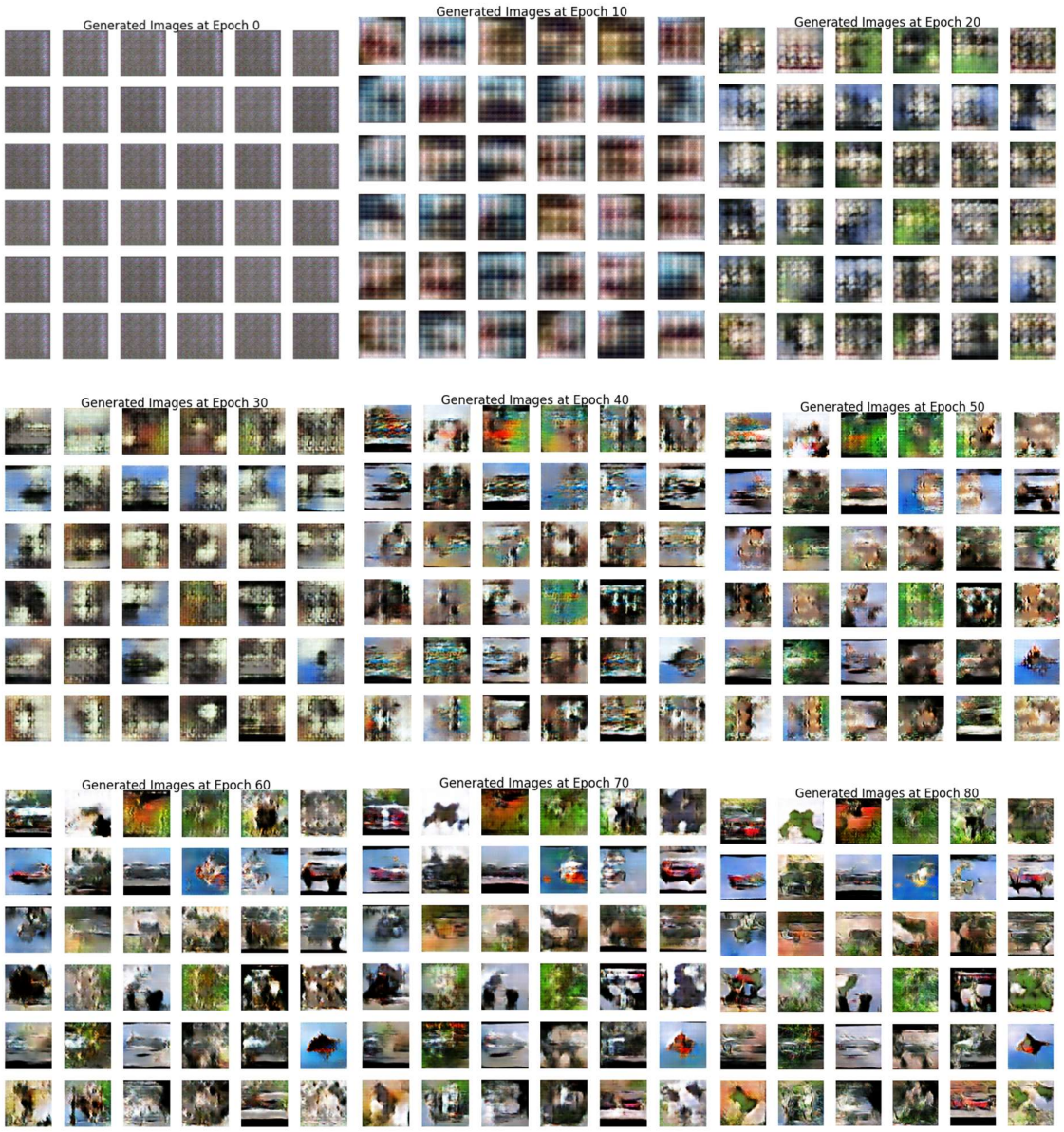
- nc = 100,
- **learning_rate = 0.0002**
- batch size = 128
- Epochs = 100

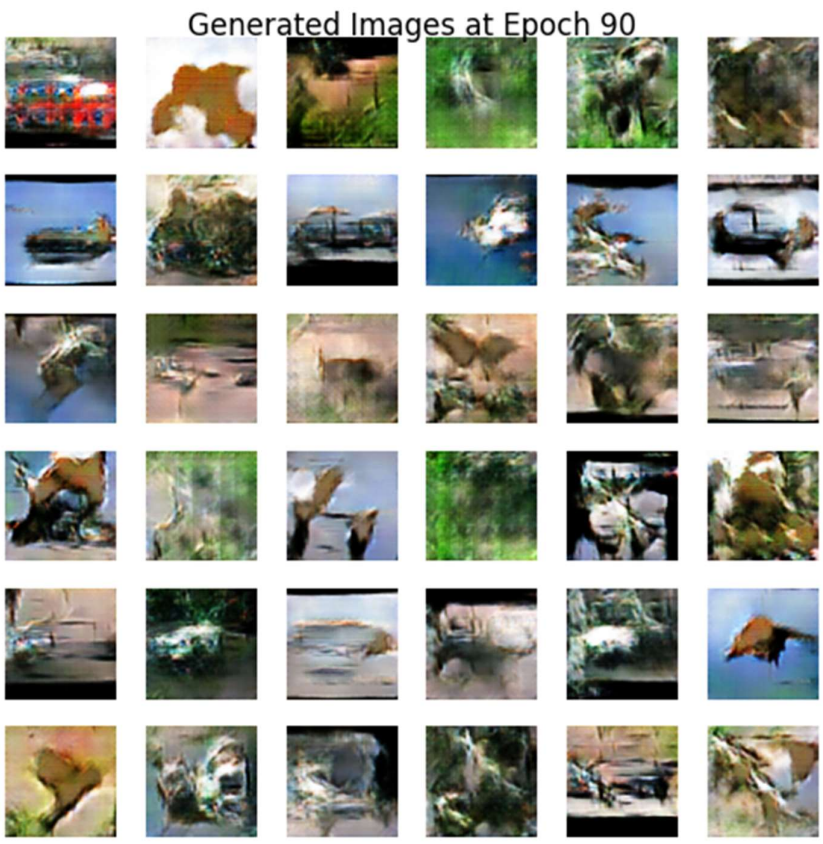


Qualitative Analysis Results:

1) We sampled 36 noise vectors from standard Gaussian distribution and plotted the generator output for those vectors

2) Visualisation at different epochs





Quantitative Analysis Results

FID Score –

```
! python -m pytorch_fid '/content/STL_10_real_1'
'/content/STL_10_real_1'
! python -m pytorch_fid '/content/STL_10_real_1'
'/content/STL_10_real_2'
! python -m pytorch_fid '/content/STL_10_real_1' '/content/STL_10_fake'
```

FID: -2.2051696589642233e-05

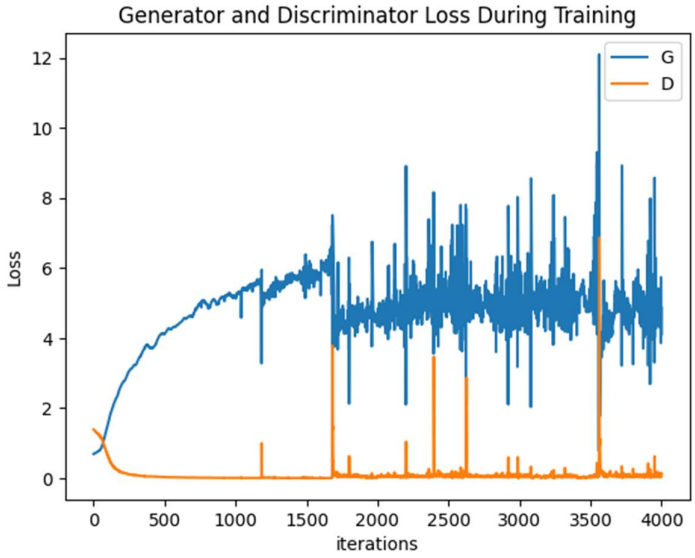
FID: 37.67575937923124

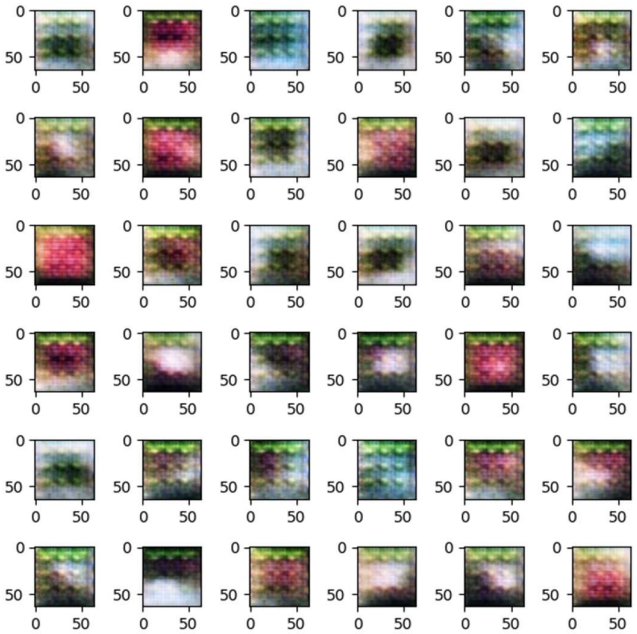
FID: 210.80015423096992

IS SCORE –

IS score for real data set 1: (14.623364188788035, 0.0)
IS score for generated data set: (2.511533595059272, 0.0)

We tried decreasing the learning rate to **0.00002** and following results were observed:





FID Score –

```
! python -m pytorch_fid '/content/STL_10_real_1'
'/content/STL_10_real_1'
! python -m pytorch_fid '/content/STL_10_real_1'
'/content/STL_10_real_2'
! python -m pytorch_fid '/content/STL_10_real_1' '/content/STL_10_fake'
```

FID: -2.2051696589642233e-05

FID: 37.67575937923124

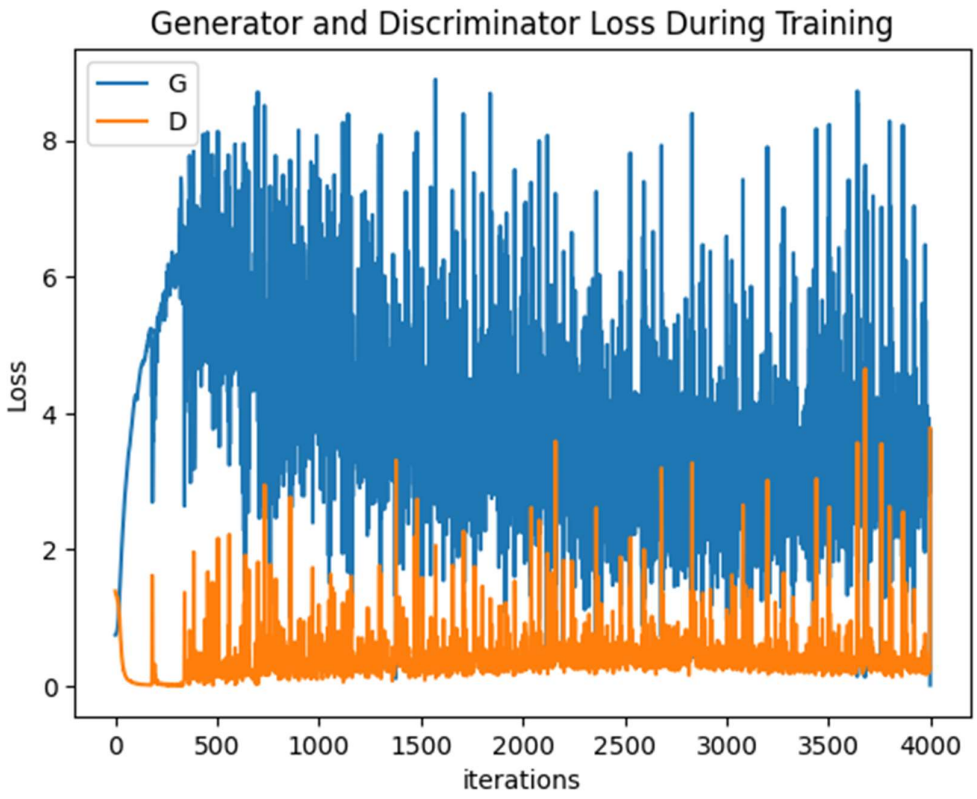
```
FID:  210.80015423096992
```

IS SCORE –

```
IS score for real data set 1:      (14.623364188788035, 0.0)
IS score for generated data set:   (2.511533595059272, 0.0)
```

The plots and result shows that learning rate of 0.00002 might not be a good choice. We resort to just lower learning rate than 0.0002 i.e 0.0001

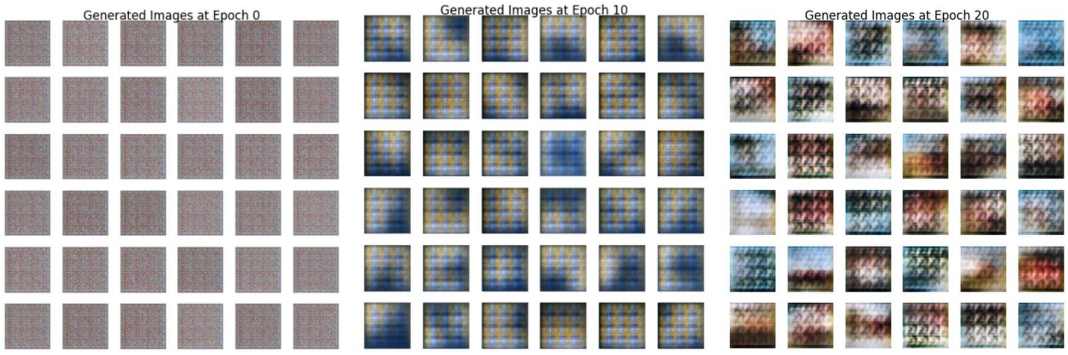
LR = 0.0001

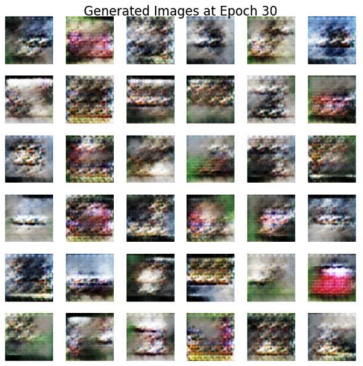


Qualitative analysis results :

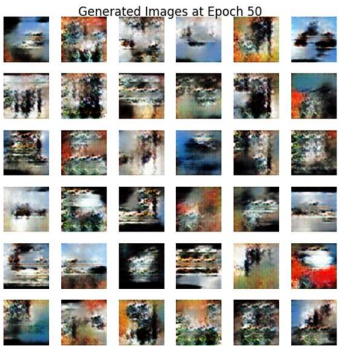
1) Visualization with random samples at lr = 0.0001

1) Visualization at different epochs for lr = 0.0001

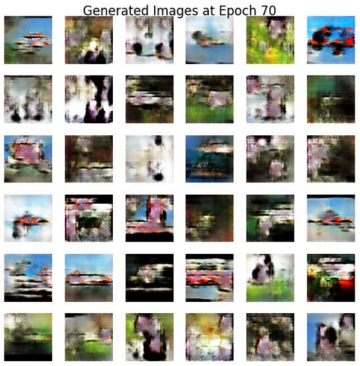


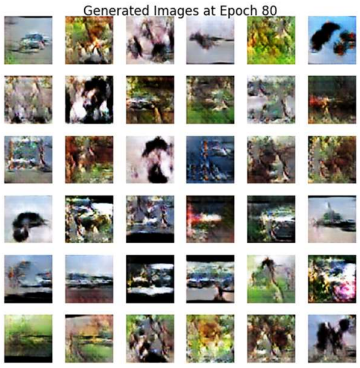

Generated Images at Epoch 30

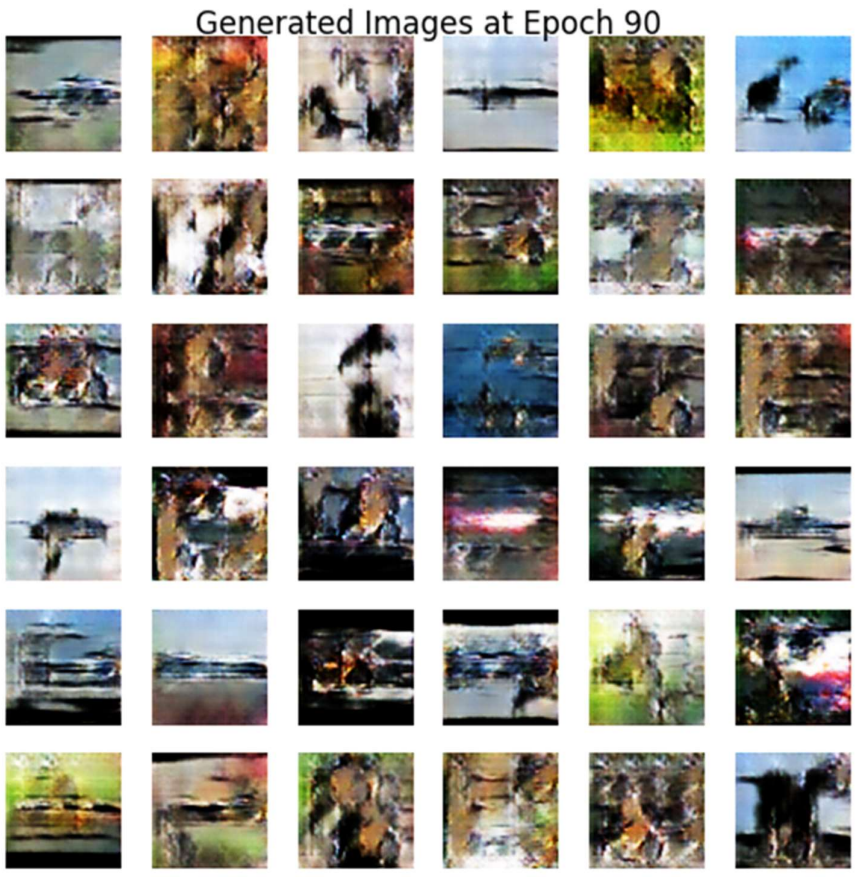

Generated Images at Epoch 40


Generated Images at Epoch 50


Generated Images at Epoch 60


Generated Images at Epoch 70


Generated Images at Epoch 80


Generated Images at Epoch 90

```
! python -m pytorch_fid '/content/STL_10_real_1'
'/content/STL_10_real_1'
! python -m pytorch_fid '/content/STL_10_real_1'
'/content/STL_10_real_2'
! python -m pytorch_fid '/content/STL_10_real_1' '/content/STL_10_fake'
```

```
FID:  -2.1626007026043226e-05
FID:  37.67575937923124
FID:  199.49929070056172
```

IS SCORE –

```
IS score for real data set 1:      (14.623364188788035, 0.0)
IS score for generated data set:   (2.4117110555928383, 0.0)
```

Model tuning discussion:

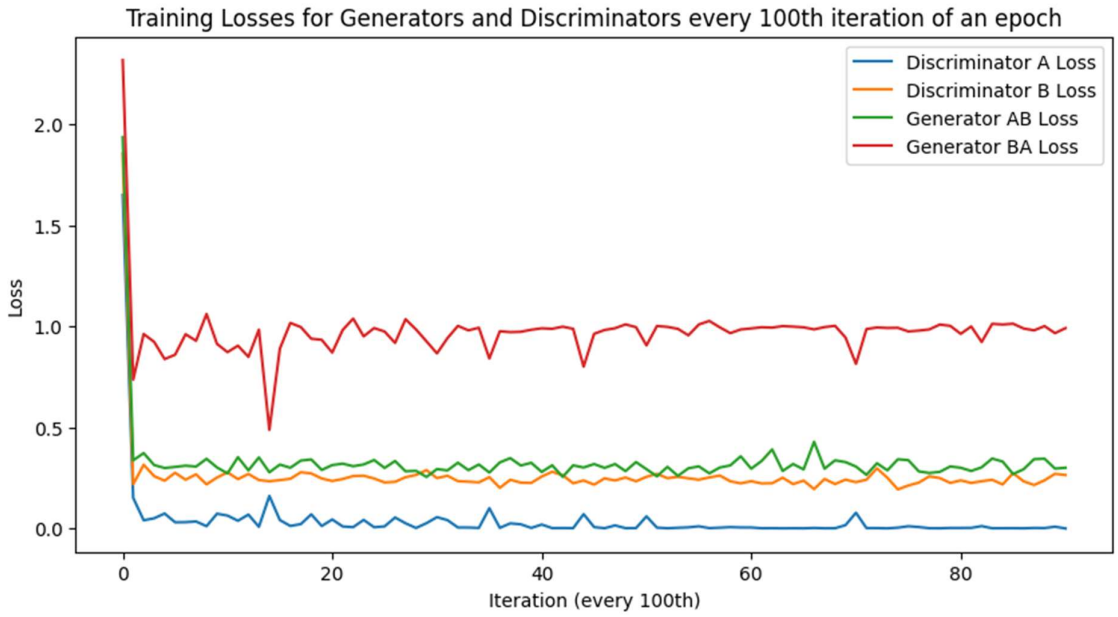
The plots and result (**reduced FID Score** compared to learning rate = 0.0002) shows that learning rate of 0.0001 performs the best compared to 0.0002 or 0.00002. In future, we could have reduced learning rate even more and could have found an ideal learning rate between 0.00002 and 0.0001.

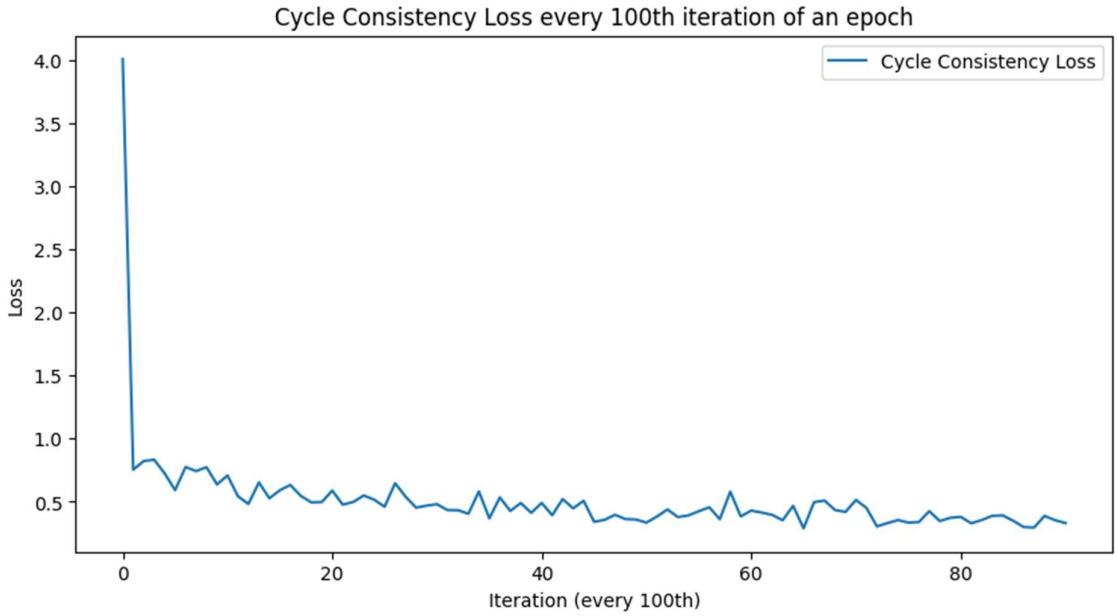
CycleGAN Implementation Results

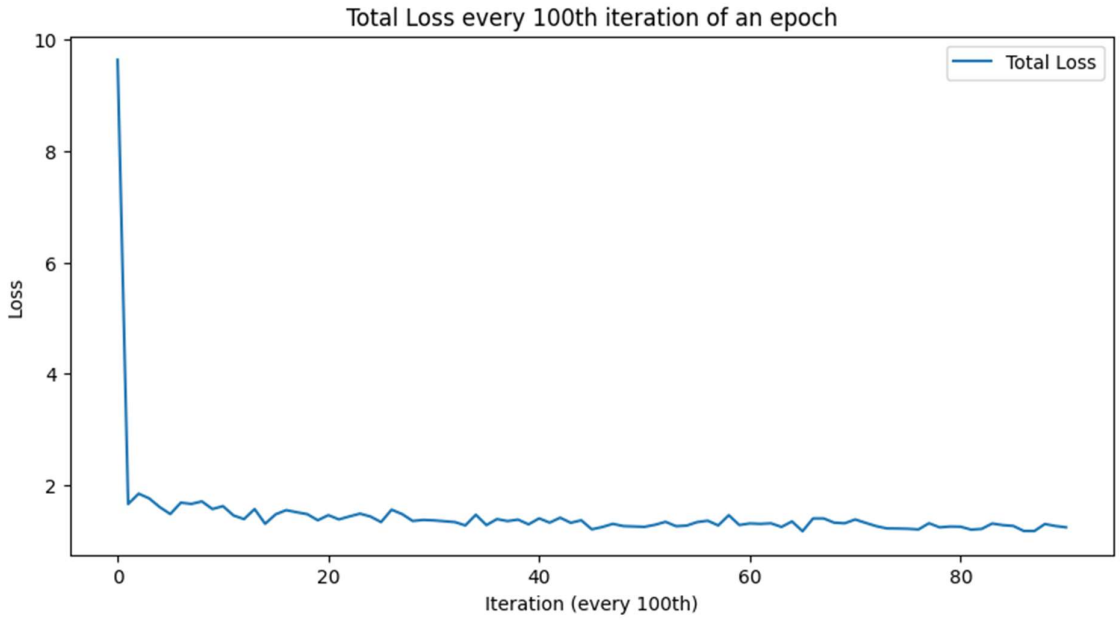
First, we ran our model on default parameters –

- num_epochs = 2
- **learning_rate = 0.0002**
- batch_size = 1

For plotting, we are storing losses every 100th iteration. The model is supposed to run for 99,000 epochs but due to computational limit, our training stopped at 91,000 iterations. This is very close to full 2 epochs training and hence our results are acceptable.







Visualizing real edge, fake shoe, real shoe, fake edge in 2-by-2 grid figure :

The third image indicate that model outputs are not perfect. We need to improve our model

FID SCORE:

```
! python -m pytorch_fid '/content/real_A_1' '/content/real_A_2'
! python -m pytorch_fid '/content/real_A_1' '/content/gen_A'
! python -m pytorch_fid '/content/real_B_1' '/content/real_B_2'
! python -m pytorch_fid '/content/real_B_1' '/content/gen_B'
```

```
FID:  50.10597859917954
FID:  99.80046131409844
FID:  58.80420301905852
FID:  123.79387242137818
```

IS SCORE:

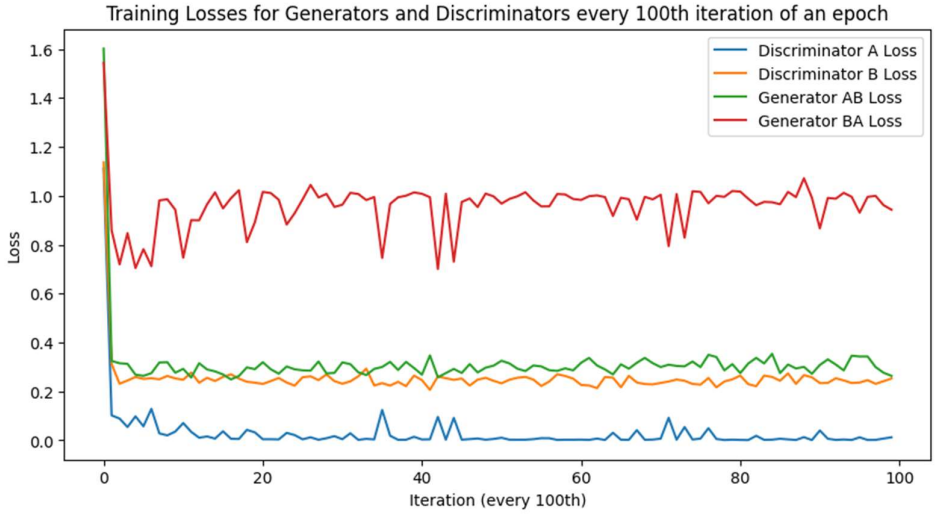
```
print('IS score for real_edge_1 data set:')=> (1.001543486827918, 0.0)
print('IS score for gen_edge data set:')=> (1.0010872079538045, 0.0)
print('IS score for real_shoe_1:')=> (3.539735974730432, 0.0)
print('IS score for gen_shoe data set:')=> (1.9373810161357972, 0.0)
```

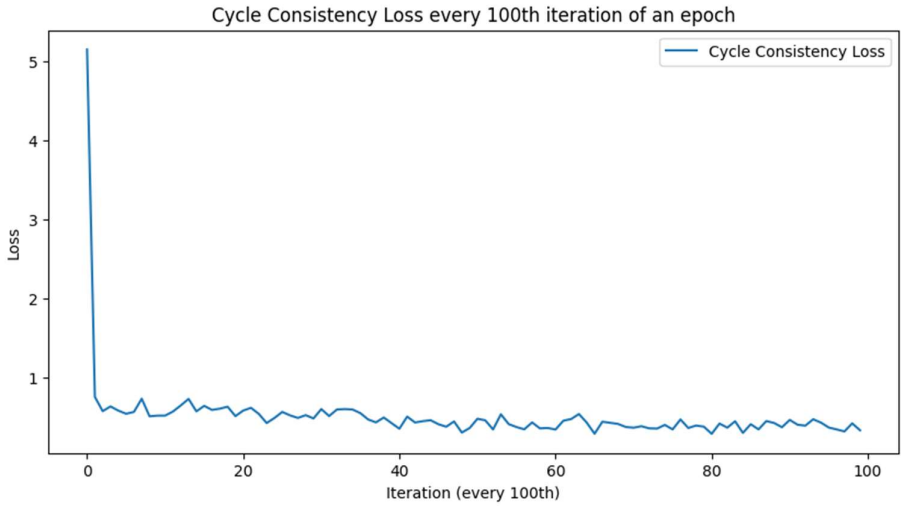
Parameter Tuning - Based on the loss plots we observe that the discriminator loss is much less than generator loss meaning that discriminator is out training generator. To avoid this, we introduce two learning rates. We reduce lr for discriminator and keep the generator lr same. Slower lr of discriminator can prevent it from becoming too accurate too quickly and generator can then become better in finding the configurations to fool discriminator

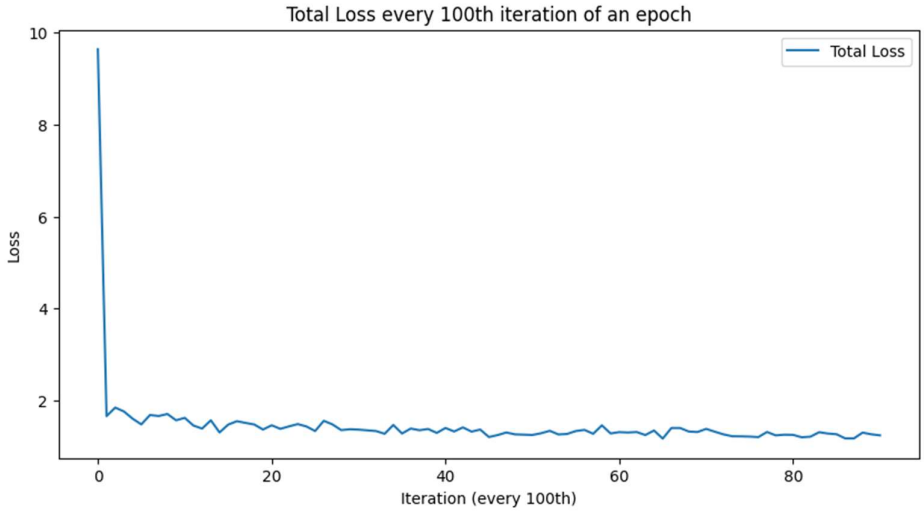
Discriminator LR = 0.0001

Generator LR = 0.0002

The training took place for complete 2 epochs this time. The results are as follows :


Training Losses for Generators and Discriminators every 100th iteration of an epoch
Discriminator A Loss
Discriminator B Loss
Generator AB Loss
Generator BA Loss
Loss
Iteration (every 100th)


Cycle Consistency Loss every 100th iteration of an epoch
Cycle Consistency Loss
Loss
Iteration (every 100th)


Total Loss every 100th iteration of an epoch
Total Loss
Loss
Iteration (every 100th)

Visualizing real edge, fake shoe, real shoe, fake edge in 2-by-2 grid figure :

These images look much better in terms of outline and representation compared to last model. Let's take a look at quantitative analysis.

FID SCORE:

```
! python -m pytorch_fid '/content/real_A_1' '/content/real_A_2'
! python -m pytorch_fid '/content/real_A_1' '/content/gen_A'
! python -m pytorch_fid '/content/real_B_1' '/content/real_B_2'
! python -m pytorch_fid '/content/real_B_1' '/content/gen_B'
```

```
FID:  49.445948773750985
FID:  72.18188015626862
FID:  59.58885942844975
FID:  161.0324641672321
```

IS SCORE:

```
print('IS score for real_edge_1 data set:')=> (1.0031845324999558, 0.0)
print('IS score for gen_edge data set:')=> (1.0043012541355028, 0.0)
print('IS score for real_shoe_1:')=> (3.450902136659301, 0.0)
print('IS score for gen_shoe data set:')=> (3.437892180669382, 0.0)
```

Model tuning discussion:

The plots and result obtained by using two different learning rates (**reduced FID Score** and **increased IS Score)** compared to just one constant learning rate = 0.0002 shows that the model was able to perform better. As explained before, slower lr of discriminator prevented it from becoming too accurate too quickly and generator became better in finding the configurations to fool discriminator. However, we also see that the improvement was observed in only one type of translation. This could be because we did not change generator's learning rate and modified only discriminator's lr. In future, we could try playing with lr of both entities and we can expect better performance in such case.