Lab06 Report

st122134

In this report, three parts are divided to show.

1) Reproducing the vanilla GAN and DCGAN results on MNIST and CIFAR. Get the training and test loss for the generator and discriminator over time, plot them, and interpret them.

2) Develop your own GAN to model data generated as follows:

$$\theta \sim \mathcal{U}(0, 2\pi) \; r \qquad \sim \mathcal{N}(0, 1)$$

$$\mathbf{x} \leftarrow \begin{cases} \begin{bmatrix} (10+r)\cos\theta \\ (10+r)\sin\theta + 10 \end{bmatrix} & \frac{1}{2}\pi \le \theta \le \frac{3}{2}\pi \\ \begin{bmatrix} (10+r)\cos\theta \\ (10+r)\sin\theta - 10 \end{bmatrix} & \text{otherwise} \end{cases}$$

You should create a PyTorch DataSet that generates the 2D data in the **init**() method, outputs a sample in the **getitem**() method, and returns the dataset size in the **len**() method. Use the vanilla GAN approach above with an appropriate structure for the generator. Can your GAN generate a convincing facsimile of a set of samples from the actual distribution?

3) Use the DCGAN (or an improvement to it) to build a generator for a face image set of your choice. Can you get realistic faces that are not in the training set?

1.1 VanillaGAN On MINIST data

The following steps are implemented for vanillaGAN On MINIST data

- Set up tensorboard:a Logger class with a lot of useful tricks to indicate training progress and visualize results.
- Downloaded the MNIST dataset and load the dataset
- The simpleest "vanilla" GAN netwrok architecture is built by Generator and discriminator which has the responsibity to classify its input as real or fake. When a fake sample from the generator is given, it should ouptut 0 for fake:
- Set up the optimizers

  The optimizers used are:

    d_optimizer = optim.Adam(discriminator.parameters(), lr=0.0002)

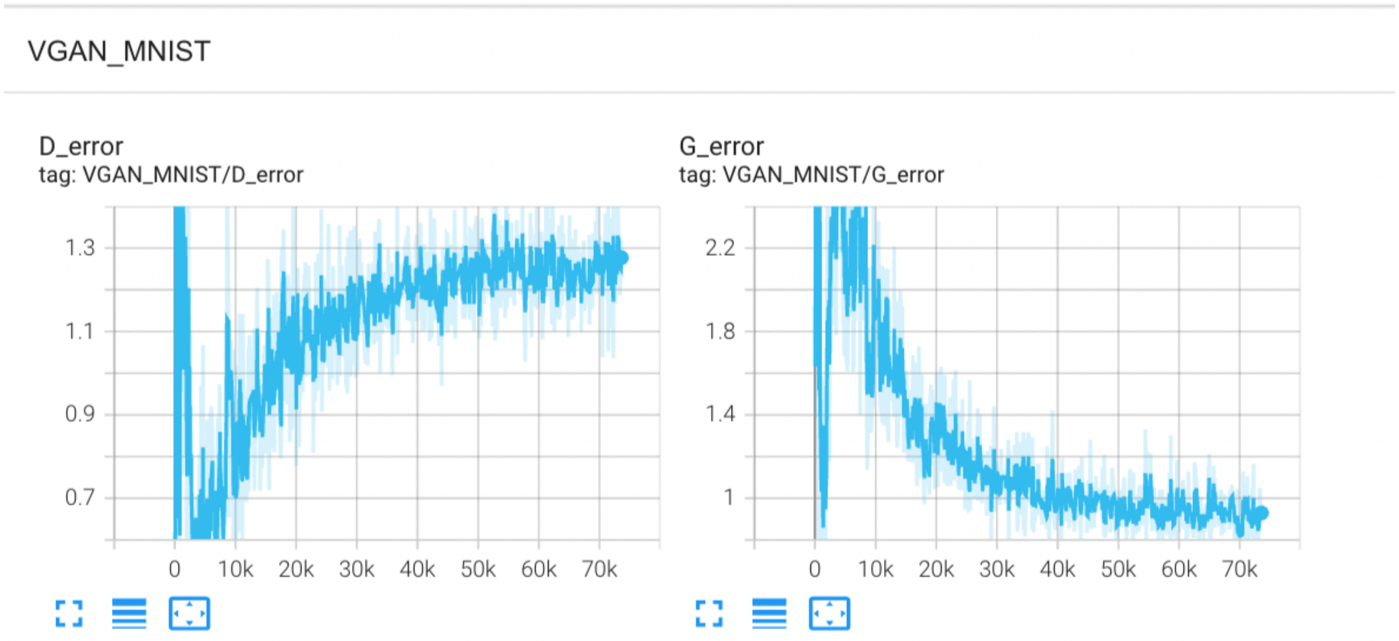
    g_optimizer = optim.Adam(generator.parameters(), lr=0.0002)

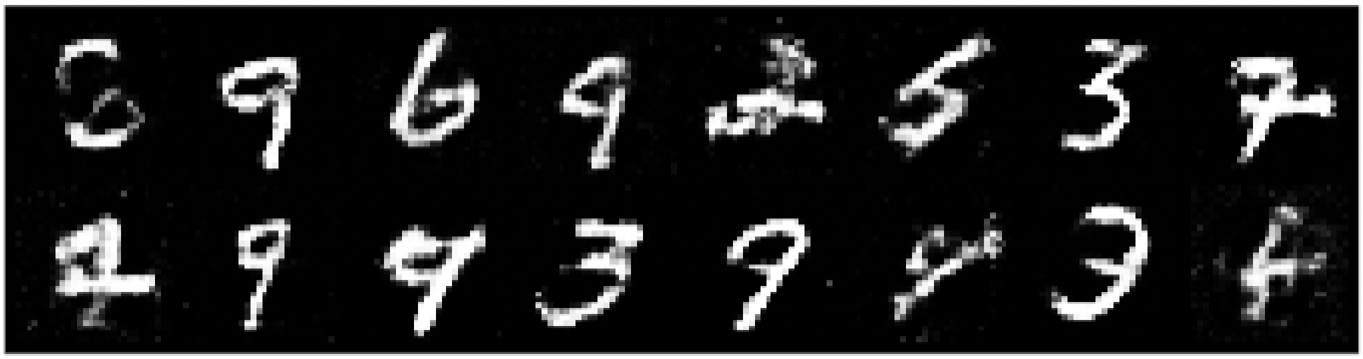
- In training, the targets for the discriminator may be 0 or 1 depending on whether we're giving it real or fake data:
- Generate some noise vectors to use as inputs to the generator.
- Then,training the model
- The results after training for 200 epochs is as follows:

  Epoch: [99/100], Batch Num: [500/600]

  Discriminator Loss: 1.1777, Generator Loss: 0.8801

  D(x): 0.6245, D(G(z)): 0.4490





1.2 DCGAN On CIFAR data

The following steps are implemented for DCGAN On CIFAR data

- Set up tensorboard:a Logger class with a lot of useful tricks to indicate training progress and visualize results.
- Downloaded the CIFAR10 dataset and load the dataset

- The DCGAN netwrok architecture is built by Generator and Discriminator which has the responsibity to classify its input as real or fake. When a fake sample from the generator is given, it should ouptut 0 for fake:
- Set up the optimizers

  The optimizers used are:

  d_optimizer = optim.Adam(discriminator.parameters(), lr=0.0002)

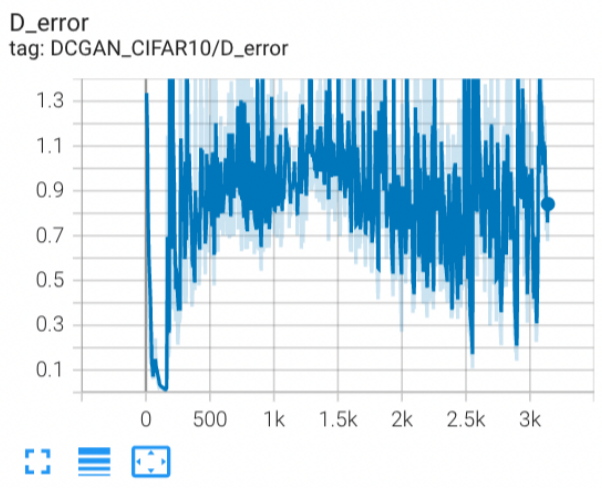
  g_optimizer = optim.Adam(generator.parameters(), lr=0.0002)
- In training, the targets for the discriminator may be 0 or 1 depending on whether we're giving it real or fake data:
- Generate some noise vectors to use as inputs to the generator.
- Then,training the model
- The results after training for 200 epochs is as follows:

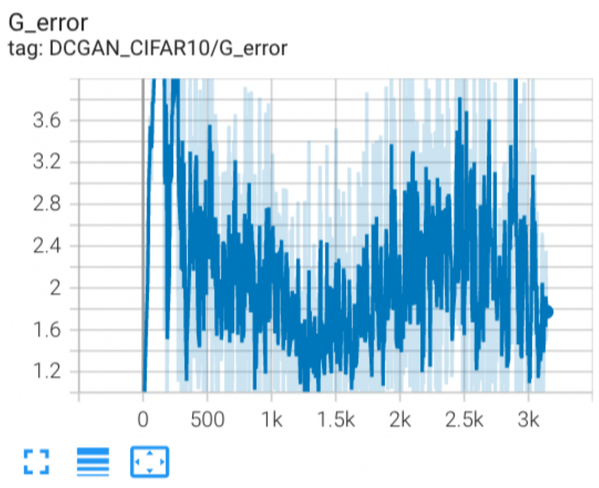
  Epoch: [29/30], Batch Num: [500/600]

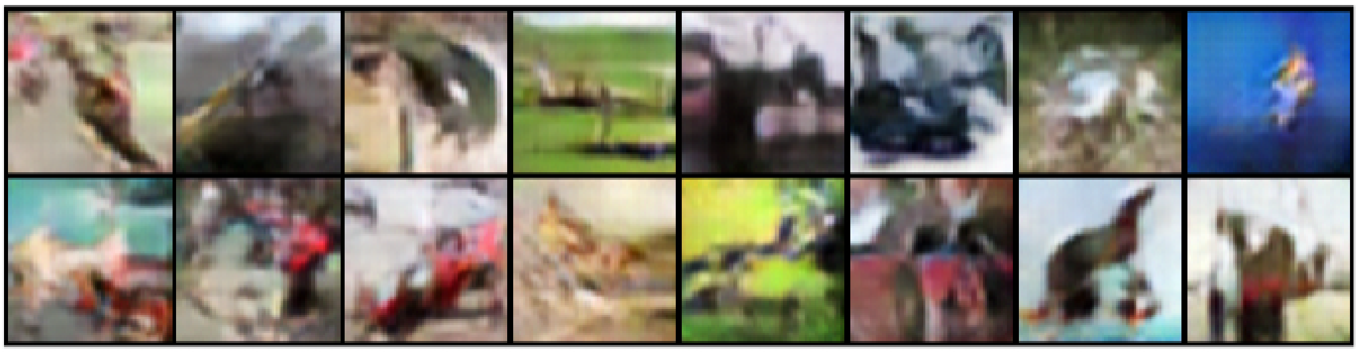
  Discriminator Loss: 0.8223, Generator Loss: 2.1527

  D(x): 0.6267, D(G(z)): 0.0373

DCGAN_CIFAR10







2.1 Custom dataset class to generate num_sample number of the 'S-shape' data

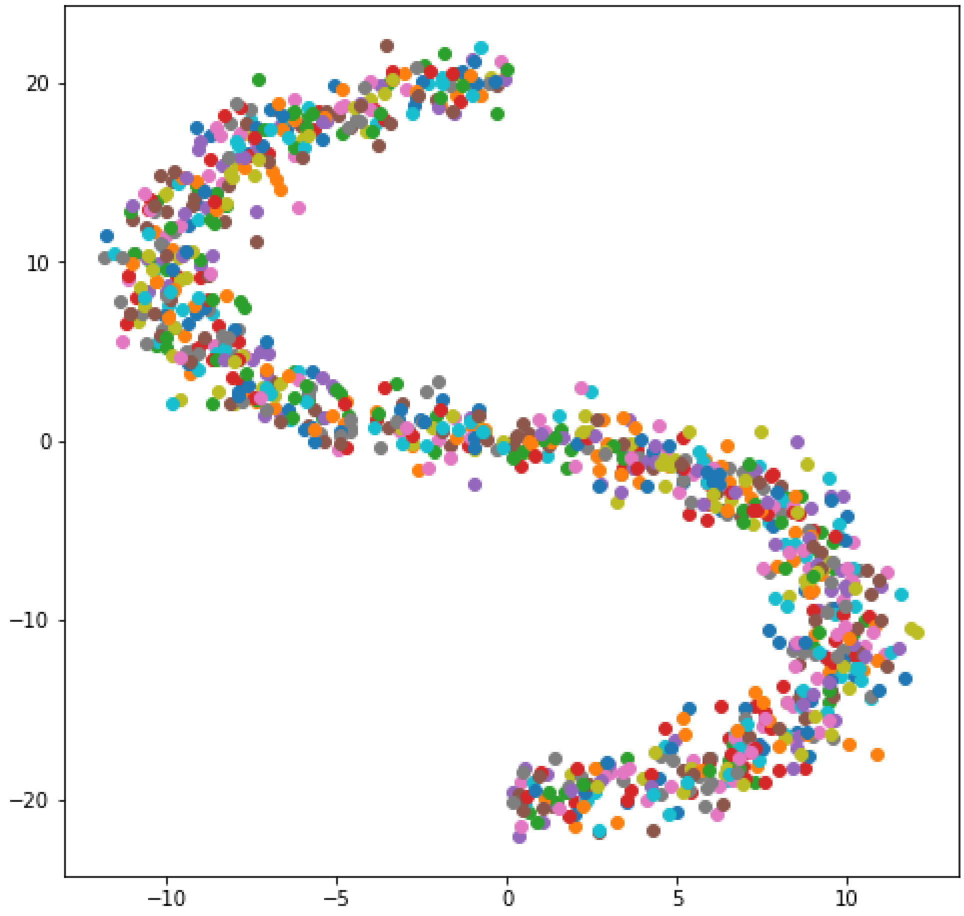
In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
from torch.utils.data import DataLoader, Dataset
import torchvision
from torchvision import transforms


class s_dataset(Dataset):

    def __init__(self, num_sample = 1000, transform=None):

        pi = np.pi
        self.data = torch.zeros([num_sample,2])

        for i in range(num_sample):
            theta = torch.FloatTensor(1).uniform_(0, 2*pi)
            r = torch.randn(1)
            x = (10+r) * torch.cos(theta)


            if 0.5*pi <= theta and theta <= 1.5*pi:
                y = ((10+r) * torch.sin(theta)) + 10
            else:
                y = ((10+r) * torch.sin(theta)) - 10

            self.data[i,0] = x
            self.data[i,1] = y

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        data = self.data[index]
        return data
```

2.2 Get the dataset and plot

In [3]:

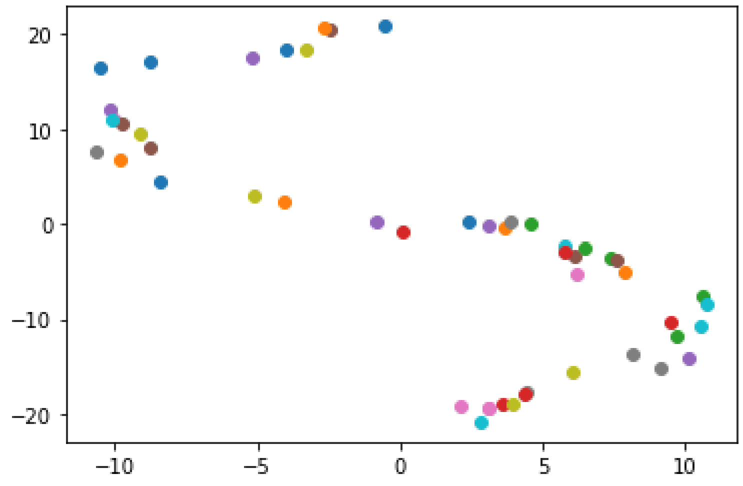
```
train_dataset = s_dataset(num_sample = 100000, transform = None)

plt.figure(figsize = (8,8))
for i in range(1000):
    data = train_dataset.__getitem__(i)
    plt.scatter(data[0], data[1])
plt.show()
```



2.3 Load the data into dataloader then plot some examples

In [4]:

```
data_loader = torch.utils.data.DataLoader(train_dataset, batch_size=100, shuffle=Tru
num_batches = len(data_loader)

for i, data in enumerate(data_loader):
    plt.scatter(data[i,0], data[i,1])
    if i == 50:
        break
plt.figure(figsize=(8,8))
plt.show()
```



```
<Figure size 576x576 with 0 Axes>
```

The following steps are implemented for DCGAN on S-shape data which is generated above:

- Set up tensorboard:a Logger class with a lot of useful tricks to indicate training progress and visualize results.
- The DCGAN netwrok architecture is built by Generator and Discriminator
- Set up the optimizers

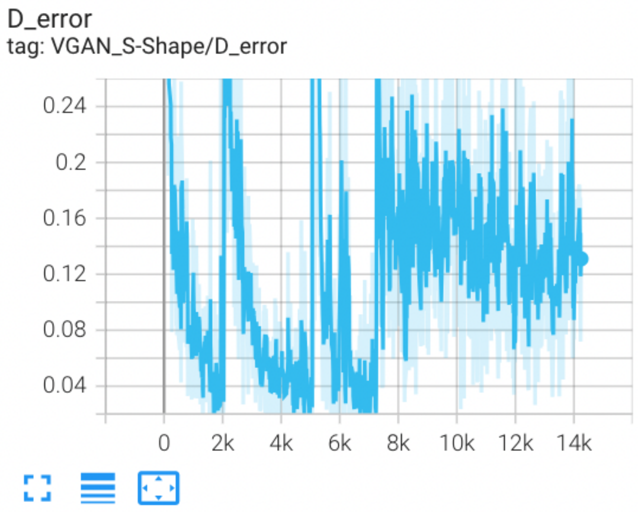
  The optimizers used are:

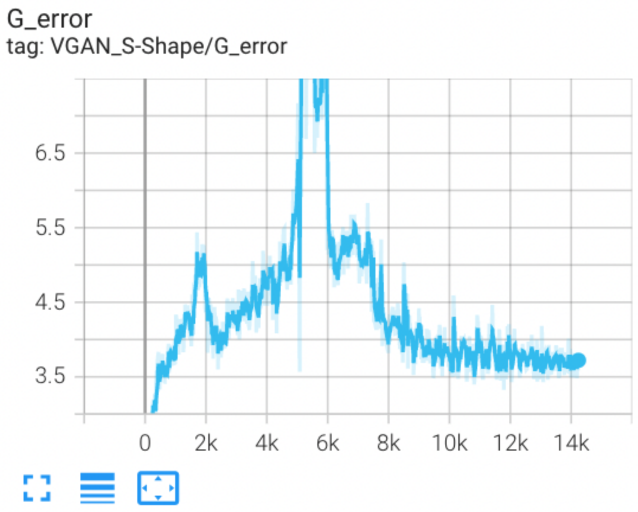
  d_optimizer = optim.Adam(discriminator.parameters(), lr=0.0002)

  g_optimizer = optim.Adam(generator.parameters(), lr=0.0002)
- In training, the targets for the discriminator may be 0 or 1 depending on whether we're giving it real or fake data:
- Generate some noise vectors to use as inputs to the generator.
- Then,training the model
- The results after training for 200 epochs is as follows:

```
Epoch: [29/30], Batch Num: [900/1000]

Discriminator Loss: 0.2187, Generator Loss: 3.7196

D(x): 0.9511, D(G(z)): 0.0268
```

VGAN_S-Shape





3. DCGAN to build a generator for a face image dataset

Celebrity Face Dataset

The face dataset I used is from here : [https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html (https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html)](https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html)

3.1 Download and load the dataset

In [ ]:

```
transform = transforms.Compose([
    transforms.Resize(64),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5,], std=[0.5,]),
])

data = torchvision.datasets.ImageFolder('../face_DCGAN/Face_Dataset/img', transform
#data = torch.utils.data.Dataset ('../face_DCGAN/face_data/img', transform = transfo

# Load Dataset and attach a DataLoader
batch_size = 10
data_loader = torch.utils.data.DataLoader(data,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          num_workers=2)
num_batches = len(data_loader)
print(num_batches)
```

The following steps are implemented for DCGAN on celeba image data

- Set up tensorboard:a Logger class with a lot of useful tricks to indicate training progress and visualize results.
- The DCGAN netwrok architecture is built by Generator and Discriminator which has the responsibity to classify its input as real or fake. When a fake sample from the generator is given, it should ouptut 0 for fake:
- Set up the optimizers

  The optimizers used are:

  d_optimizer = optim.Adam(discriminator.parameters(), lr=0.0002)

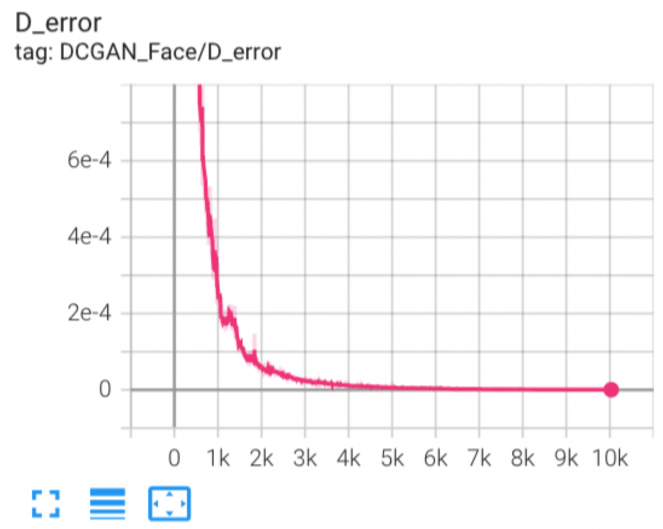
  g_optimizer = optim.Adam(generator.parameters(), lr=0.0002)
- In training, the targets for the discriminator may be 0 or 1 depending on whether we're giving it real or fake data:
- Generate some noise vectors to use as inputs to the generator.
- Then,training the model
- The results after training for 200 epochs is as follows:

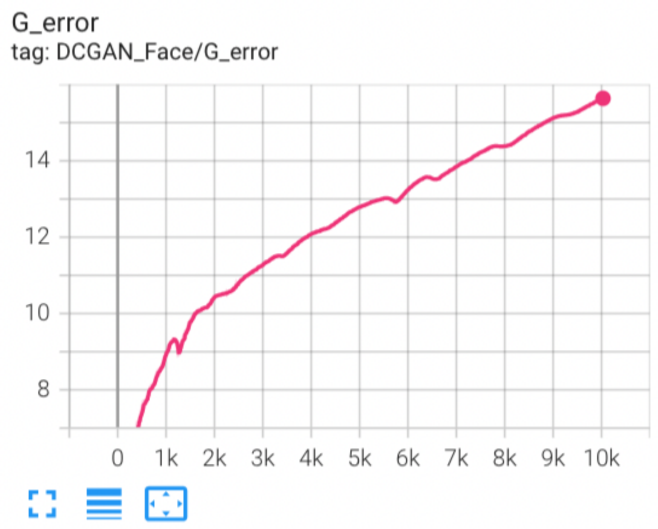
  Epoch: [199/200], Batch Num: [500/600]

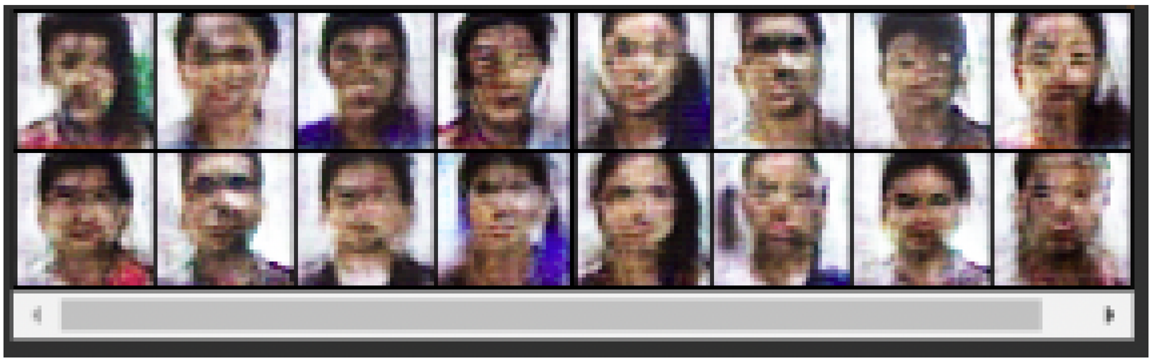
  Discriminator Loss: 1.2012, Generator Loss: 0.9787

  D(x): 0.6136, D(G(z)): 0.4570

DCGAN_Face







In [ ]: