CV2_HW4_CS4090

April 21, 2022

Name: Chandan Suri, CS4090

In collaboration with Gursifath Bhasin, GB2760

0.1 GANs : Generative Adversarial Networks

Image from here

A generative adversarial network (GAN) is a generative model composed of two neural networks: a generator and a discriminator. These two networks are trained in unsupervised way via competition. The generator creates "realistic" fake images from random noise to fool the discriminator, while the discriminator evaluates the given image for authenticity. The loss function that the generator wants to minimize and the discriminator to maximize is as follows:

min G max D L(D, G) = Ex pdata(x)[log D(x)] + Ez pz(z)[log(1 − D(G(z)))]

Here, G and D are the generator and the discriminator. The first and second term of the loss represent the correct prediction of the discriminator on the real images and on the fake images respectively.

0.2 DCGAN

- You will implement deep convolutional GAN model on the MNIST dataset with Pytorch. The input image size is 28 x 28.

- The details of the generator of DCGAN is described below.

- You will start with batch size of 128, input noise of 100 dimension and Adam optimizer with learning rate of 2e-4. You may vary these hyperparameters for better performance.

0.3 Architectures

NOTE: Referenced From: 1. https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html 2. https://stackoverflow.com/questions/41489907/generative-adversarial-networks-tanh 3. https://pytorch.org/docs/stable/generated/torch.randn.html 4. https://stackoverflow.com/questions/65046236/why-do-we-use-sigmoid-fn-when-we-make-mnist-gan-example 5. https://stats.stackexchange.com/questions/498508/why-use-tanh-function-at-the-last-layer-of-generator-in-gan

Generator:

The goal for the generator is to use layers such as convolution, maybe also upsampling layer/transposedConvolution to produce image from the given input noise vector. As this is DC-

GAN (deep convolutional GAN), we expect you to use convolution in the generator. You will get full credit if you can produce `[batchsize, 1, 28, 28]` vector (image) from the given `[batchsize, 100, 1, 1]` vector (noise).

Linear Layers that you may use:

- torch.nn.Conv2d
- torch.nn.UpsamplingBilinear2d
- torch.nn.ConvTranspose2d

Non-linear layer:

- torch.nn.LeakyReLU with slope=0.2 between all linear layers.
- torch.nn.Tanh for the last layer's activation. Can you explain why do we need this in the code comment?

You may use `view` to change the vector size: https://pytorch.org/docs/stable/generated/torch.Tensor.view.html

We recommend to use 2 Conv/TransposedConv layers. When you are increasing the feature map size, considering upsample the feature by a factor of 2 each time. If you have width of 7 in one of your feature map, to get output with width of 28, you can do upsampling with factor of 2 and upsampling 2 times.

Discriminator:

You will get full credit if you can produce an output of `[batchsize, 1]` vector (image) from the given input `[batchsize, 1, 28, 28]` vector (noise).

Linear Layers that you may use:

- torch.nn.Conv2d
- torch.nn.Linear

Non-linear Layers:

- torch.nn.LeakyReLU with slope=0.2 between all linear layers.
- torch.nn.Sigmoid for the last layer's activation. Can you explain why do we need this in the code comment?

Use Leaky ReLu as the activation function between all layers, except after the last layer use Sigmoid.

You may use `view` to change the vector size: https://pytorch.org/docs/stable/generated/torch.Tensor.view.html

As an example, you may use 2 convolution layer and one linear layer in the discriminator, you can also use other setup. Note that instead of using pooling to downsampling, you may also use stride=2 in convolution to downsample the feature.

[1]:
```python
from torchvision.transforms.transforms import Normalize
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

```python
from torchvision import datasets, transforms
from torch.autograd import Variable
from torchvision.utils import save_image
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML
import numpy as np
from torch.optim.lr_scheduler import StepLR
import torchvision.utils as vutils
from torch.utils.data import DataLoader, TensorDataset
from scipy import linalg
from scipy.stats import entropy
import tqdm
import cv2
# image input size
image_size=28

# Setting up transforms to resize and normalize
transform=transforms.Compose([transforms.Resize(image_size),
                              transforms.CenterCrop(image_size),
                              transforms.ToTensor(),
                              transforms.Normalize((0.5), (0.5))])
# batchsize of dataset
batch_size = 100

# Load MNIST Dataset
gan_train_dataset = datasets.MNIST(root='./MNIST/', train=True,
 ↪transform=transform, download=True)
gan_train_loader = torch.utils.data.DataLoader(dataset=gan_train_dataset,
 ↪batch_size=batch_size, shuffle=True)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
./MNIST/MNIST/raw/train-images-idx3-ubyte.gz

  0%|          | 0/9912422 [00:00<?, ?it/s]

Extracting ./MNIST/MNIST/raw/train-images-idx3-ubyte.gz to ./MNIST/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
./MNIST/MNIST/raw/train-labels-idx1-ubyte.gz

  0%|          | 0/28881 [00:00<?, ?it/s]

Extracting ./MNIST/MNIST/raw/train-labels-idx1-ubyte.gz to ./MNIST/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
./MNIST/MNIST/raw/t10k-images-idx3-ubyte.gz
```

```
  0%|          | 0/1648877 [00:00<?, ?it/s]

Extracting ./MNIST/MNIST/raw/t10k-images-idx3-ubyte.gz to ./MNIST/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
./MNIST/MNIST/raw/t10k-labels-idx1-ubyte.gz

  0%|          | 0/4542 [00:00<?, ?it/s]

Extracting ./MNIST/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./MNIST/MNIST/raw
```

0.4 Model Definition (TODO)

[2]:
```python
class DCGAN_Generator(nn.Module):
    def __init__(self):
        super(DCGAN_Generator,self).__init__()

        self.gen_net = nn.Sequential(
            nn.ConvTranspose2d(100, 256, 7, 1, 0, bias = False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, True),

            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias = False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, True),

            nn.ConvTranspose2d(128, 1, 4, 2, 1, bias = False),
            nn.Tanh()
        )


    def forward(self, input):

        out = self.gen_net(input)

        # Explain why Tanh is needed for the last layer
        '''
        Answer: There are many facets to the reasoning here:
        1. As our inputs don't range just between 0 and 1 (as we generate␣
↪through
        noise that follows a normal distribution with mean 0 and std of 1)
        , so, we prefer to use tanh than sigmoid here if we want to introduce␣
↪some
        non-linearity in the network through the activation.
        2. Along with these reasons, a bounded activation would allow the model␣
↪to
        learn more quickly to saturate and cover the color space of the training
```

```
        distribution.
        3. Also, tanh is symmetric w.r.t zero here which means that it would␣
↪treat
        darker colors and lighter colors in a symmetric way.
        '''

        return out


class DCGAN_Discriminator(nn.Module):
    def __init__(self):
        super(DCGAN_Discriminator, self).__init__()

        self.disc_net = nn.Sequential(
            nn.Conv2d(1, 128, 4, 2, 1, bias = False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, True),

            nn.Conv2d(128, 64, 4, 2, 1, bias = False),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, True),

            nn.Flatten(),

            # This is added to create a single output
            # and the number is actually the number of nodes coming out of
            # the previous layer
            nn.Linear(64*7*7, 1),
            nn.Sigmoid()
        )


    def forward(self, input):

        out = self.disc_net(input)

        # Explain why Sigmoid is needed for the last layer
        '''
        Answer: There are many facets to the reasoning here:
        1. As we are doing Binary Classification here from the discriminator,
        Real or Fake, for which we would need to calculate the losses using the
        logits/probabilities for the classes predicted and the target variable,
        we need to use Sigmoid function which gives us the inputs as needed for
        our problem statement and use case at hand.
        2. Also, as our inputs range just between 0 and 1, so, to introduce
        non-linearity through an activation function, we would use the sigmoid
        activation rather than tanh here.
```

```
        '''

        return out


# Code that check size
g=DCGAN_Generator()
batchsize=2
z=torch.zeros((batchsize, 100, 1, 1))
out = g(z)
print(out.size()) # You should expect size [batchsize, 1, 28, 28]

d=DCGAN_Discriminator()
x=torch.zeros((batchsize, 1, 28, 28))
out = d(x)
print(out.size()) # You should expect size [batchsize, 1]
```

```
torch.Size([2, 1, 28, 28])
torch.Size([2, 1])
```

GAN loss (TODO)

[3]:
```
import torch

def loss_discriminator(D, real, G, noise, Valid_label, Fake_label, criterion,␣
 ↪optimizerD):

    '''
    1. Forward real images into the discriminator
    2. Compute loss between Valid_label and dicriminator output on real images
    3. Forward noise into the generator to get fake images
    4. Forward fake images to the discriminator
    5. Compute loss between Fake_label and discriminator output on fake images␣
 ↪(and remember to detach the gradient from the fake images using detach()!)
    6. sum real loss and fake loss as the loss_D
    7. we also need to output fake images generate by G(noise) for␣
 ↪loss_generator computation
    '''
    # Step-1
    output = D(real).view(-1)
    # Step-2
    loss_D_real = criterion(output, Valid_label)

    # Step-3
    fake_imgs = G(noise)
    # Step-4
    output = D(fake_imgs.detach()).view(-1)
    # Step-5
```

```python
    loss_D_fake = criterion(output, Fake_label)

    # Step-6
    loss_D = loss_D_real + loss_D_fake

    # Step-7
    return loss_D, fake_imgs

def loss_generator(netD, netG, fake, Valid_label, criterion, optimizerG):
    '''
    1. Forward fake images to the discriminator
    2. Compute loss between valid labels and discriminator output on fake images
    '''
    # Step-1
    output = netD(fake).view(-1)
    # Step-2
    loss_G = criterion(output, Valid_label)

    return loss_G
```

[4]:

```python
import torchvision.utils as vutils
from torch.optim.lr_scheduler import StepLR
import pdb

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Number of channels
nc = 1
# Size of z latent vector (i.e. size of generator input)
nz = 100

netG = DCGAN_Generator().to(device)
netD = DCGAN_Discriminator().to(device)

from torchsummary import summary
print(summary(netG,(100,1,1)))
print(summary(netD,(1, 28, 28)))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1            [-1, 256, 7, 7]       1,254,400
       BatchNorm2d-2            [-1, 256, 7, 7]             512
         LeakyReLU-3            [-1, 256, 7, 7]               0
   ConvTranspose2d-4          [-1, 128, 14, 14]         524,288
       BatchNorm2d-5          [-1, 128, 14, 14]             256
         LeakyReLU-6          [-1, 128, 14, 14]               0
   ConvTranspose2d-7            [-1, 1, 28, 28]           2,048
```

```
             Tanh-8            [-1, 1, 28, 28]               0
================================================================
Total params: 1,781,504
Trainable params: 1,781,504
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.87
Params size (MB): 6.80
Estimated Total Size (MB): 7.67
----------------------------------------------------------------
None
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 128, 14, 14]           2,048
       BatchNorm2d-2          [-1, 128, 14, 14]             256
         LeakyReLU-3          [-1, 128, 14, 14]               0
            Conv2d-4             [-1, 64, 7, 7]         131,072
       BatchNorm2d-5             [-1, 64, 7, 7]             128
         LeakyReLU-6             [-1, 64, 7, 7]               0
           Flatten-7                 [-1, 3136]               0
            Linear-8                    [-1, 1]           3,137
           Sigmoid-9                    [-1, 1]               0
================================================================
Total params: 136,641
Trainable params: 136,641
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.67
Params size (MB): 0.52
Estimated Total Size (MB): 1.19
----------------------------------------------------------------
None
```

TRAINING

[5]:
```
import torchvision.utils as vutils
from torch.optim.lr_scheduler import StepLR
import pdb

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Number of channels
nc = 1
# Size of z latent vector (i.e. size of generator input)
nz = 100
```

```
# Create the generator and discriminator
netG = DCGAN_Generator().to(device)
netD = DCGAN_Discriminator().to(device)

# Initialize BCELoss function
criterion = nn.BCELoss()

# Create latent vector to test the generator performance
fixed_noise = torch.randn(36, nz, 1, 1, device=device)

# Establish convention for real and fake labels during training
real_label = 1
fake_label = 0

learning_rate = 0.0002
beta1 = 0.5

# Setup Adam optimizers for both G and D

################################
# Please fill in your code here:

optimizerD = optim.Adam(netD.parameters(), lr=learning_rate, betas=(beta1, 0.
↪999))
optimizerG = optim.Adam(netG.parameters(), lr=learning_rate, betas=(beta1, 0.
↪999))

################################



img_list = []
real_img_list = []
G_losses = []
D_losses = []
iters = 0
num_epochs = 50 # Changed this to get better results


def load_param(num_eps):
    model_saved = torch.load('/content/gan_{}.pt'.format(num_eps))
    netG.load_state_dict(model_saved['netG'])
    netD.load_state_dict(model_saved['netD'])

# GAN Training Loop
for epoch in range(num_epochs):
```

```
    for i, data in enumerate(gan_train_loader, 0):
        real = data[0].to(device)
        b_size = real.size(0)
        noise = torch.randn(b_size, nz, 1, 1, device=device)

        Valid_label = torch.full((b_size,), real_label, dtype=torch.float,␣
↪device=device)
        Fake_label = torch.full((b_size,), fake_label, dtype=torch.float,␣
↪device=device)

        ############################
        # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
        ###########################
        netD.zero_grad()

        loss_D, fake_imgs = loss_discriminator(netD, real, netG, noise,␣
↪Valid_label, Fake_label, criterion, optimizerD)

        loss_D.backward()
        optimizerD.step()

        ############################
        # (2) Update G network: maximize log(D(G(z)))
        ###########################
        netG.zero_grad()

        loss_G = loss_generator(netD, netG, fake_imgs, Valid_label, criterion,␣
↪optimizerG)

        loss_G.backward()
        optimizerG.step()

        # Output training stats
        if i % 50 == 0:
            print('[%d/%d][%d/%d]\tLoss_D: %.4f\tLoss_G: %.4f\t'
                  % (epoch, num_epochs, i, len(gan_train_loader),
                     loss_D.item(), loss_G.item()))

        # Save Losses for plotting later
        G_losses.append(loss_G.item())
        D_losses.append(loss_D.item())

        # Check how the generator is doing by saving G's output on fixed_noise
        if (iters % 500 == 0) or ((epoch == num_epochs-1) and (i ==␣
↪len(gan_train_loader)-1)):
            with torch.no_grad():
                fake = netG(fixed_noise).detach().cpu()
```

```
            img_list.append(vutils.make_grid(fake, padding=2, normalize=True))

        iters += 1



plt.title("Generator and Discriminator Loss During Training")
plt.plot(G_losses,label="G")
plt.plot(D_losses,label="D")
plt.xlabel("iterations")
plt.ylabel("Loss")
plt.legend()
plt.show()


checkpoint = {'netG': netG.state_dict(),
              'netD': netD.state_dict()}
torch.save(checkpoint, 'gan_{}.pt'.format(num_epochs))
```

```
[0/50][0/600]   Loss_D: 1.5680  Loss_G: 0.8732
[0/50][50/600]  Loss_D: 0.8541  Loss_G: 1.4793
[0/50][100/600] Loss_D: 1.0419  Loss_G: 0.6862
[0/50][150/600] Loss_D: 1.0198  Loss_G: 0.9973
[0/50][200/600] Loss_D: 1.0158  Loss_G: 1.6525
[0/50][250/600] Loss_D: 0.8398  Loss_G: 1.4010
[0/50][300/600] Loss_D: 0.7039  Loss_G: 1.2722
[0/50][350/600] Loss_D: 0.5747  Loss_G: 1.9331
[0/50][400/600] Loss_D: 0.5572  Loss_G: 1.5470
[0/50][450/600] Loss_D: 0.6471  Loss_G: 1.7121
[0/50][500/600] Loss_D: 0.5147  Loss_G: 1.8375
[0/50][550/600] Loss_D: 0.4706  Loss_G: 1.8235
[1/50][0/600]   Loss_D: 0.4593  Loss_G: 1.7749
[1/50][50/600]  Loss_D: 0.5812  Loss_G: 1.6918
[1/50][100/600] Loss_D: 0.5394  Loss_G: 1.4933
[1/50][150/600] Loss_D: 0.6323  Loss_G: 0.6902
[1/50][200/600] Loss_D: 0.7331  Loss_G: 1.4061
[1/50][250/600] Loss_D: 0.5846  Loss_G: 1.7611
[1/50][300/600] Loss_D: 0.7295  Loss_G: 1.5103
[1/50][350/600] Loss_D: 0.4963  Loss_G: 1.5023
[1/50][400/600] Loss_D: 0.7561  Loss_G: 0.8506
[1/50][450/600] Loss_D: 0.6414  Loss_G: 1.9196
[1/50][500/600] Loss_D: 0.7856  Loss_G: 1.4955
[1/50][550/600] Loss_D: 0.8247  Loss_G: 2.4358
[2/50][0/600]   Loss_D: 0.7197  Loss_G: 1.4258
[2/50][50/600]  Loss_D: 0.5601  Loss_G: 1.8779
[2/50][100/600] Loss_D: 0.5649  Loss_G: 1.7796
[2/50][150/600] Loss_D: 0.4427  Loss_G: 2.0067
[2/50][200/600] Loss_D: 0.8113  Loss_G: 1.0601
```

```
[2/50][250/600] Loss_D: 0.8420  Loss_G: 3.4616
[2/50][300/600] Loss_D: 0.7018  Loss_G: 2.3584
[2/50][350/600] Loss_D: 0.5479  Loss_G: 2.5846
[2/50][400/600] Loss_D: 0.6443  Loss_G: 1.6570
[2/50][450/600] Loss_D: 0.6682  Loss_G: 1.4708
[2/50][500/600] Loss_D: 0.8416  Loss_G: 1.5236
[2/50][550/600] Loss_D: 0.8563  Loss_G: 1.1306
[3/50][0/600]   Loss_D: 0.7467  Loss_G: 1.1430
[3/50][50/600]  Loss_D: 1.1255  Loss_G: 2.1863
[3/50][100/600] Loss_D: 0.7164  Loss_G: 1.9378
[3/50][150/600] Loss_D: 0.6312  Loss_G: 1.8080
[3/50][200/600] Loss_D: 0.5837  Loss_G: 1.7076
[3/50][250/600] Loss_D: 0.5916  Loss_G: 1.2347
[3/50][300/600] Loss_D: 0.8338  Loss_G: 0.9688
[3/50][350/600] Loss_D: 1.2971  Loss_G: 2.7516
[3/50][400/600] Loss_D: 0.6095  Loss_G: 1.7204
[3/50][450/600] Loss_D: 0.6601  Loss_G: 1.3771
[3/50][500/600] Loss_D: 0.6651  Loss_G: 1.7613
[3/50][550/600] Loss_D: 0.4865  Loss_G: 2.0187
[4/50][0/600]   Loss_D: 0.4695  Loss_G: 1.8346
[4/50][50/600]  Loss_D: 0.7469  Loss_G: 1.0282
[4/50][100/600] Loss_D: 0.9725  Loss_G: 3.1240
[4/50][150/600] Loss_D: 0.5153  Loss_G: 1.9743
[4/50][200/600] Loss_D: 0.8244  Loss_G: 2.6224
[4/50][250/600] Loss_D: 0.5597  Loss_G: 1.9392
[4/50][300/600] Loss_D: 0.5925  Loss_G: 2.5154
[4/50][350/600] Loss_D: 0.6282  Loss_G: 1.8843
[4/50][400/600] Loss_D: 1.0299  Loss_G: 2.2887
[4/50][450/600] Loss_D: 0.7497  Loss_G: 1.7623
[4/50][500/600] Loss_D: 0.6163  Loss_G: 1.7632
[4/50][550/600] Loss_D: 0.6402  Loss_G: 1.3164
[5/50][0/600]   Loss_D: 0.7663  Loss_G: 1.1434
[5/50][50/600]  Loss_D: 0.7085  Loss_G: 2.0353
[5/50][100/600] Loss_D: 0.5353  Loss_G: 1.5692
[5/50][150/600] Loss_D: 0.5225  Loss_G: 1.5850
[5/50][200/600] Loss_D: 0.6587  Loss_G: 1.5748
[5/50][250/600] Loss_D: 0.5477  Loss_G: 1.8129
[5/50][300/600] Loss_D: 0.5301  Loss_G: 1.8204
[5/50][350/600] Loss_D: 0.5406  Loss_G: 1.7260
[5/50][400/600] Loss_D: 0.6253  Loss_G: 2.3640
[5/50][450/600] Loss_D: 0.5149  Loss_G: 1.5431
[5/50][500/600] Loss_D: 0.5446  Loss_G: 1.4730
[5/50][550/600] Loss_D: 0.5942  Loss_G: 1.8001
[6/50][0/600]   Loss_D: 0.5778  Loss_G: 2.4183
[6/50][50/600]  Loss_D: 0.9447  Loss_G: 0.7604
[6/50][100/600] Loss_D: 0.8141  Loss_G: 0.7797
[6/50][150/600] Loss_D: 0.4591  Loss_G: 2.0422
[6/50][200/600] Loss_D: 0.5944  Loss_G: 1.9065
```

```
[6/50][250/600] Loss_D: 1.1560  Loss_G: 0.9100
[6/50][300/600] Loss_D: 0.5470  Loss_G: 1.9697
[6/50][350/600] Loss_D: 0.7468  Loss_G: 0.8224
[6/50][400/600] Loss_D: 0.5075  Loss_G: 2.6210
[6/50][450/600] Loss_D: 0.5725  Loss_G: 1.9285
[6/50][500/600] Loss_D: 1.0137  Loss_G: 1.3215
[6/50][550/600] Loss_D: 0.6190  Loss_G: 2.4138
[7/50][0/600]   Loss_D: 0.9757  Loss_G: 1.3992
[7/50][50/600]  Loss_D: 0.5559  Loss_G: 1.6948
[7/50][100/600] Loss_D: 0.6236  Loss_G: 1.1428
[7/50][150/600] Loss_D: 0.4608  Loss_G: 1.9500
[7/50][200/600] Loss_D: 0.5825  Loss_G: 1.2220
[7/50][250/600] Loss_D: 0.7078  Loss_G: 1.5409
[7/50][300/600] Loss_D: 0.8064  Loss_G: 0.5129
[7/50][350/600] Loss_D: 0.4742  Loss_G: 2.0002
[7/50][400/600] Loss_D: 0.5666  Loss_G: 2.5095
[7/50][450/600] Loss_D: 0.4478  Loss_G: 2.3693
[7/50][500/600] Loss_D: 0.4590  Loss_G: 1.8243
[7/50][550/600] Loss_D: 0.4973  Loss_G: 2.4245
[8/50][0/600]   Loss_D: 0.7239  Loss_G: 1.9520
[8/50][50/600]  Loss_D: 0.8380  Loss_G: 0.9201
[8/50][100/600] Loss_D: 0.4183  Loss_G: 2.1505
[8/50][150/600] Loss_D: 0.5030  Loss_G: 1.8009
[8/50][200/600] Loss_D: 0.7264  Loss_G: 1.1231
[8/50][250/600] Loss_D: 0.4455  Loss_G: 1.9563
[8/50][300/600] Loss_D: 0.5998  Loss_G: 2.3868
[8/50][350/600] Loss_D: 0.6192  Loss_G: 2.0435
[8/50][400/600] Loss_D: 0.6469  Loss_G: 1.7227
[8/50][450/600] Loss_D: 0.5065  Loss_G: 2.9479
[8/50][500/600] Loss_D: 0.5106  Loss_G: 2.2250
[8/50][550/600] Loss_D: 0.7189  Loss_G: 2.7596
[9/50][0/600]   Loss_D: 0.6857  Loss_G: 1.1795
[9/50][50/600]  Loss_D: 0.4736  Loss_G: 2.1352
[9/50][100/600] Loss_D: 0.5780  Loss_G: 2.6128
[9/50][150/600] Loss_D: 0.4336  Loss_G: 2.4512
[9/50][200/600] Loss_D: 0.4159  Loss_G: 2.6024
[9/50][250/600] Loss_D: 0.9202  Loss_G: 0.7810
[9/50][300/600] Loss_D: 0.5433  Loss_G: 1.4477
[9/50][350/600] Loss_D: 1.1141  Loss_G: 3.3660
[9/50][400/600] Loss_D: 0.4418  Loss_G: 2.9481
[9/50][450/600] Loss_D: 0.5587  Loss_G: 2.1139
[9/50][500/600] Loss_D: 0.8284  Loss_G: 2.3588
[9/50][550/600] Loss_D: 0.4412  Loss_G: 1.9901
[10/50][0/600]  Loss_D: 0.5160  Loss_G: 1.4835
[10/50][50/600] Loss_D: 0.5135  Loss_G: 2.8555
[10/50][100/600]        Loss_D: 0.5093  Loss_G: 2.4332
[10/50][150/600]        Loss_D: 0.8908  Loss_G: 4.1359
[10/50][200/600]        Loss_D: 0.6449  Loss_G: 1.2802
```

```
[10/50][250/600]        Loss_D: 0.6190  Loss_G: 2.3388
[10/50][300/600]        Loss_D: 0.8450  Loss_G: 2.2125
[10/50][350/600]        Loss_D: 0.6131  Loss_G: 1.6909
[10/50][400/600]        Loss_D: 0.5714  Loss_G: 1.6612
[10/50][450/600]        Loss_D: 0.4141  Loss_G: 2.4889
[10/50][500/600]        Loss_D: 0.6252  Loss_G: 1.7478
[10/50][550/600]        Loss_D: 0.7324  Loss_G: 0.7077
[11/50][0/600]  Loss_D: 0.9154  Loss_G: 3.2214
[11/50][50/600] Loss_D: 0.4208  Loss_G: 2.2572
[11/50][100/600]        Loss_D: 0.6015  Loss_G: 2.4370
[11/50][150/600]        Loss_D: 0.6133  Loss_G: 2.0986
[11/50][200/600]        Loss_D: 0.5487  Loss_G: 2.7824
[11/50][250/600]        Loss_D: 0.7259  Loss_G: 3.1681
[11/50][300/600]        Loss_D: 0.6101  Loss_G: 2.9995
[11/50][350/600]        Loss_D: 0.6723  Loss_G: 1.4365
[11/50][400/600]        Loss_D: 0.6147  Loss_G: 2.0612
[11/50][450/600]        Loss_D: 0.7283  Loss_G: 1.4230
[11/50][500/600]        Loss_D: 0.5265  Loss_G: 2.5009
[11/50][550/600]        Loss_D: 0.5370  Loss_G: 2.4616
[12/50][0/600]  Loss_D: 0.6134  Loss_G: 2.3842
[12/50][50/600] Loss_D: 0.4849  Loss_G: 2.3870
[12/50][100/600]        Loss_D: 0.8420  Loss_G: 2.9040
[12/50][150/600]        Loss_D: 0.5166  Loss_G: 1.7637
[12/50][200/600]        Loss_D: 0.6453  Loss_G: 1.9602
[12/50][250/600]        Loss_D: 0.5065  Loss_G: 1.9799
[12/50][300/600]        Loss_D: 0.5314  Loss_G: 2.3480
[12/50][350/600]        Loss_D: 0.8205  Loss_G: 3.3575
[12/50][400/600]        Loss_D: 0.3985  Loss_G: 2.2193
[12/50][450/600]        Loss_D: 0.5392  Loss_G: 1.9862
[12/50][500/600]        Loss_D: 0.7159  Loss_G: 1.6564
[12/50][550/600]        Loss_D: 0.4407  Loss_G: 2.7636
[13/50][0/600]  Loss_D: 0.4477  Loss_G: 2.2256
[13/50][50/600] Loss_D: 0.5763  Loss_G: 1.9199
[13/50][100/600]        Loss_D: 0.5879  Loss_G: 1.5820
[13/50][150/600]        Loss_D: 0.8220  Loss_G: 2.9888
[13/50][200/600]        Loss_D: 0.4443  Loss_G: 2.1864
[13/50][250/600]        Loss_D: 0.5201  Loss_G: 2.2581
[13/50][300/600]        Loss_D: 0.5409  Loss_G: 0.9988
[13/50][350/600]        Loss_D: 0.4180  Loss_G: 2.3652
[13/50][400/600]        Loss_D: 0.5411  Loss_G: 2.1003
[13/50][450/600]        Loss_D: 0.3635  Loss_G: 2.8073
[13/50][500/600]        Loss_D: 0.4991  Loss_G: 2.4565
[13/50][550/600]        Loss_D: 0.5492  Loss_G: 2.8100
[14/50][0/600]  Loss_D: 1.1791  Loss_G: 0.3085
[14/50][50/600] Loss_D: 0.4686  Loss_G: 2.4309
[14/50][100/600]        Loss_D: 0.5037  Loss_G: 0.9371
[14/50][150/600]        Loss_D: 0.4244  Loss_G: 2.5694
[14/50][200/600]        Loss_D: 0.4888  Loss_G: 1.8678
```

```
[14/50][250/600]        Loss_D: 0.6060  Loss_G: 1.9319
[14/50][300/600]        Loss_D: 0.5269  Loss_G: 2.0915
[14/50][350/600]        Loss_D: 0.5891  Loss_G: 2.2497
[14/50][400/600]        Loss_D: 0.5099  Loss_G: 1.9081
[14/50][450/600]        Loss_D: 0.5432  Loss_G: 2.0123
[14/50][500/600]        Loss_D: 0.5089  Loss_G: 1.8210
[14/50][550/600]        Loss_D: 0.4007  Loss_G: 2.5216
[15/50][0/600]  Loss_D: 0.4090  Loss_G: 2.3912
[15/50][50/600] Loss_D: 0.4126  Loss_G: 2.7302
[15/50][100/600]        Loss_D: 0.2593  Loss_G: 2.8944
[15/50][150/600]        Loss_D: 0.5037  Loss_G: 2.1723
[15/50][200/600]        Loss_D: 0.6851  Loss_G: 1.1594
[15/50][250/600]        Loss_D: 0.6233  Loss_G: 3.6549
[15/50][300/600]        Loss_D: 0.6081  Loss_G: 2.1778
[15/50][350/600]        Loss_D: 0.4454  Loss_G: 1.8431
[15/50][400/600]        Loss_D: 0.7657  Loss_G: 1.4584
[15/50][450/600]        Loss_D: 0.5282  Loss_G: 2.7681
[15/50][500/600]        Loss_D: 0.4853  Loss_G: 2.2664
[15/50][550/600]        Loss_D: 0.5464  Loss_G: 2.0667
[16/50][0/600]  Loss_D: 0.3650  Loss_G: 2.1825
[16/50][50/600] Loss_D: 0.4608  Loss_G: 2.6197
[16/50][100/600]        Loss_D: 0.5209  Loss_G: 1.1353
[16/50][150/600]        Loss_D: 0.3581  Loss_G: 2.3860
[16/50][200/600]        Loss_D: 0.5978  Loss_G: 1.8348
[16/50][250/600]        Loss_D: 0.6517  Loss_G: 1.6790
[16/50][300/600]        Loss_D: 0.7297  Loss_G: 0.9317
[16/50][350/600]        Loss_D: 0.4783  Loss_G: 1.9181
[16/50][400/600]        Loss_D: 0.4156  Loss_G: 2.5709
[16/50][450/600]        Loss_D: 0.3198  Loss_G: 2.4515
[16/50][500/600]        Loss_D: 0.4511  Loss_G: 2.7146
[16/50][550/600]        Loss_D: 0.4560  Loss_G: 2.4058
[17/50][0/600]  Loss_D: 0.3842  Loss_G: 2.3754
[17/50][50/600] Loss_D: 0.4562  Loss_G: 3.0873
[17/50][100/600]        Loss_D: 1.0836  Loss_G: 4.2010
[17/50][150/600]        Loss_D: 0.6746  Loss_G: 2.8064
[17/50][200/600]        Loss_D: 0.6271  Loss_G: 3.0622
[17/50][250/600]        Loss_D: 0.4099  Loss_G: 2.1811
[17/50][300/600]        Loss_D: 0.4947  Loss_G: 2.7384
[17/50][350/600]        Loss_D: 0.4168  Loss_G: 3.0599
[17/50][400/600]        Loss_D: 0.4412  Loss_G: 2.7751
[17/50][450/600]        Loss_D: 0.5118  Loss_G: 2.4599
[17/50][500/600]        Loss_D: 0.6439  Loss_G: 2.5736
[17/50][550/600]        Loss_D: 0.5484  Loss_G: 2.2172
[18/50][0/600]  Loss_D: 0.4781  Loss_G: 2.0604
[18/50][50/600] Loss_D: 0.4487  Loss_G: 2.3756
[18/50][100/600]        Loss_D: 0.4613  Loss_G: 1.8476
[18/50][150/600]        Loss_D: 0.6545  Loss_G: 1.3752
[18/50][200/600]        Loss_D: 0.3753  Loss_G: 2.5228
```

```
[18/50][250/600]        Loss_D: 0.8594  Loss_G: 3.5642
[18/50][300/600]        Loss_D: 0.3813  Loss_G: 2.6581
[18/50][350/600]        Loss_D: 0.4225  Loss_G: 2.5186
[18/50][400/600]        Loss_D: 0.4506  Loss_G: 2.5827
[18/50][450/600]        Loss_D: 0.5712  Loss_G: 2.7535
[18/50][500/600]        Loss_D: 0.3870  Loss_G: 2.8117
[18/50][550/600]        Loss_D: 0.5489  Loss_G: 1.3144
[19/50][0/600]  Loss_D: 0.3847  Loss_G: 1.5133
[19/50][50/600] Loss_D: 0.4420  Loss_G: 2.0989
[19/50][100/600]        Loss_D: 0.7285  Loss_G: 1.2114
[19/50][150/600]        Loss_D: 0.4386  Loss_G: 2.6242
[19/50][200/600]        Loss_D: 0.4943  Loss_G: 2.2584
[19/50][250/600]        Loss_D: 0.4580  Loss_G: 2.6593
[19/50][300/600]        Loss_D: 0.4468  Loss_G: 2.2591
[19/50][350/600]        Loss_D: 0.5018  Loss_G: 1.6970
[19/50][400/600]        Loss_D: 0.4474  Loss_G: 1.9971
[19/50][450/600]        Loss_D: 0.7919  Loss_G: 5.4139
[19/50][500/600]        Loss_D: 0.5171  Loss_G: 2.9695
[19/50][550/600]        Loss_D: 0.4902  Loss_G: 3.2663
[20/50][0/600]  Loss_D: 0.3468  Loss_G: 2.6855
[20/50][50/600] Loss_D: 0.4483  Loss_G: 2.3953
[20/50][100/600]        Loss_D: 0.3076  Loss_G: 2.5926
[20/50][150/600]        Loss_D: 0.4806  Loss_G: 2.0785
[20/50][200/600]        Loss_D: 0.3695  Loss_G: 2.7140
[20/50][250/600]        Loss_D: 0.6151  Loss_G: 1.9065
[20/50][300/600]        Loss_D: 0.4619  Loss_G: 2.7809
[20/50][350/600]        Loss_D: 0.4249  Loss_G: 2.5803
[20/50][400/600]        Loss_D: 0.9134  Loss_G: 0.9734
[20/50][450/600]        Loss_D: 0.5461  Loss_G: 3.0596
[20/50][500/600]        Loss_D: 0.3644  Loss_G: 2.7626
[20/50][550/600]        Loss_D: 0.4132  Loss_G: 2.0365
[21/50][0/600]  Loss_D: 0.3752  Loss_G: 2.3820
[21/50][50/600] Loss_D: 0.3490  Loss_G: 1.9043
[21/50][100/600]        Loss_D: 0.4020  Loss_G: 2.7695
[21/50][150/600]        Loss_D: 0.6093  Loss_G: 1.5078
[21/50][200/600]        Loss_D: 0.3587  Loss_G: 2.3946
[21/50][250/600]        Loss_D: 0.4425  Loss_G: 1.7601
[21/50][300/600]        Loss_D: 0.4073  Loss_G: 1.9388
[21/50][350/600]        Loss_D: 0.4610  Loss_G: 2.1884
[21/50][400/600]        Loss_D: 0.4678  Loss_G: 2.8628
[21/50][450/600]        Loss_D: 1.1613  Loss_G: 0.5835
[21/50][500/600]        Loss_D: 0.4447  Loss_G: 2.2785
[21/50][550/600]        Loss_D: 0.5107  Loss_G: 4.0338
[22/50][0/600]  Loss_D: 0.4632  Loss_G: 3.4068
[22/50][50/600] Loss_D: 0.9521  Loss_G: 4.4375
[22/50][100/600]        Loss_D: 0.5719  Loss_G: 1.8787
[22/50][150/600]        Loss_D: 0.4713  Loss_G: 2.8289
[22/50][200/600]        Loss_D: 0.6529  Loss_G: 2.4574
```

```
[22/50][250/600]        Loss_D: 0.5635  Loss_G: 1.4811
[22/50][300/600]        Loss_D: 0.6406  Loss_G: 3.1498
[22/50][350/600]        Loss_D: 0.4578  Loss_G: 3.1000
[22/50][400/600]        Loss_D: 0.2703  Loss_G: 2.3436
[22/50][450/600]        Loss_D: 0.4154  Loss_G: 2.3268
[22/50][500/600]        Loss_D: 0.6092  Loss_G: 2.8286
[22/50][550/600]        Loss_D: 0.6320  Loss_G: 1.0916
[23/50][0/600]  Loss_D: 0.4400  Loss_G: 3.1943
[23/50][50/600] Loss_D: 0.5092  Loss_G: 1.3804
[23/50][100/600]        Loss_D: 0.6050  Loss_G: 2.3896
[23/50][150/600]        Loss_D: 0.4067  Loss_G: 2.5373
[23/50][200/600]        Loss_D: 0.3574  Loss_G: 2.2258
[23/50][250/600]        Loss_D: 0.4669  Loss_G: 2.1299
[23/50][300/600]        Loss_D: 0.3434  Loss_G: 2.5590
[23/50][350/600]        Loss_D: 0.4865  Loss_G: 2.6968
[23/50][400/600]        Loss_D: 0.5011  Loss_G: 2.5833
[23/50][450/600]        Loss_D: 0.7573  Loss_G: 1.5433
[23/50][500/600]        Loss_D: 0.4163  Loss_G: 2.0754
[23/50][550/600]        Loss_D: 0.3347  Loss_G: 3.0486
[24/50][0/600]  Loss_D: 1.1936  Loss_G: 3.6934
[24/50][50/600] Loss_D: 0.5378  Loss_G: 0.9680
[24/50][100/600]        Loss_D: 0.4388  Loss_G: 2.6490
[24/50][150/600]        Loss_D: 0.5724  Loss_G: 2.1959
[24/50][200/600]        Loss_D: 0.4244  Loss_G: 2.1329
[24/50][250/600]        Loss_D: 0.4455  Loss_G: 3.0406
[24/50][300/600]        Loss_D: 0.3321  Loss_G: 2.4134
[24/50][350/600]        Loss_D: 0.5988  Loss_G: 1.7616
[24/50][400/600]        Loss_D: 0.3427  Loss_G: 2.2826
[24/50][450/600]        Loss_D: 0.4813  Loss_G: 2.8834
[24/50][500/600]        Loss_D: 0.5007  Loss_G: 2.4519
[24/50][550/600]        Loss_D: 0.4260  Loss_G: 3.0648
[25/50][0/600]  Loss_D: 0.4607  Loss_G: 3.1425
[25/50][50/600] Loss_D: 0.2991  Loss_G: 2.6936
[25/50][100/600]        Loss_D: 0.5063  Loss_G: 2.0340
[25/50][150/600]        Loss_D: 0.4259  Loss_G: 3.1865
[25/50][200/600]        Loss_D: 0.3900  Loss_G: 2.2875
[25/50][250/600]        Loss_D: 0.3750  Loss_G: 2.2474
[25/50][300/600]        Loss_D: 0.3889  Loss_G: 1.9969
[25/50][350/600]        Loss_D: 0.5852  Loss_G: 3.1533
[25/50][400/600]        Loss_D: 0.3487  Loss_G: 2.3261
[25/50][450/600]        Loss_D: 0.5378  Loss_G: 1.7190
[25/50][500/600]        Loss_D: 0.4487  Loss_G: 2.7538
[25/50][550/600]        Loss_D: 0.3041  Loss_G: 3.2458
[26/50][0/600]  Loss_D: 0.4503  Loss_G: 2.2842
[26/50][50/600] Loss_D: 0.3489  Loss_G: 2.6564
[26/50][100/600]        Loss_D: 0.6579  Loss_G: 2.8939
[26/50][150/600]        Loss_D: 0.3353  Loss_G: 2.2434
[26/50][200/600]        Loss_D: 0.4773  Loss_G: 3.1342
```

```
[26/50][250/600]        Loss_D: 0.3662  Loss_G: 2.8379
[26/50][300/600]        Loss_D: 0.5131  Loss_G: 3.4970
[26/50][350/600]        Loss_D: 0.3792  Loss_G: 2.6981
[26/50][400/600]        Loss_D: 0.4323  Loss_G: 2.5704
[26/50][450/600]        Loss_D: 0.3947  Loss_G: 2.3975
[26/50][500/600]        Loss_D: 0.2977  Loss_G: 3.4269
[26/50][550/600]        Loss_D: 0.5514  Loss_G: 2.5667
[27/50][0/600]  Loss_D: 0.6379  Loss_G: 1.1618
[27/50][50/600] Loss_D: 0.6599  Loss_G: 3.7506
[27/50][100/600]        Loss_D: 0.4977  Loss_G: 1.8961
[27/50][150/600]        Loss_D: 0.1920  Loss_G: 3.8254
[27/50][200/600]        Loss_D: 0.6461  Loss_G: 3.0744
[27/50][250/600]        Loss_D: 0.4106  Loss_G: 2.4324
[27/50][300/600]        Loss_D: 0.4349  Loss_G: 1.9785
[27/50][350/600]        Loss_D: 0.4464  Loss_G: 2.2109
[27/50][400/600]        Loss_D: 0.5049  Loss_G: 2.7734
[27/50][450/600]        Loss_D: 0.6087  Loss_G: 1.1244
[27/50][500/600]        Loss_D: 0.4084  Loss_G: 1.7516
[27/50][550/600]        Loss_D: 0.5655  Loss_G: 2.4316
[28/50][0/600]  Loss_D: 0.3783  Loss_G: 2.8445
[28/50][50/600] Loss_D: 0.5295  Loss_G: 3.6603
[28/50][100/600]        Loss_D: 0.4289  Loss_G: 1.7051
[28/50][150/600]        Loss_D: 0.5941  Loss_G: 1.5720
[28/50][200/600]        Loss_D: 0.4873  Loss_G: 2.4065
[28/50][250/600]        Loss_D: 0.3905  Loss_G: 2.7666
[28/50][300/600]        Loss_D: 0.3130  Loss_G: 3.4284
[28/50][350/600]        Loss_D: 0.4086  Loss_G: 2.4225
[28/50][400/600]        Loss_D: 0.5132  Loss_G: 1.6603
[28/50][450/600]        Loss_D: 0.3748  Loss_G: 3.0326
[28/50][500/600]        Loss_D: 0.5273  Loss_G: 3.4994
[28/50][550/600]        Loss_D: 0.4206  Loss_G: 1.7373
[29/50][0/600]  Loss_D: 0.3478  Loss_G: 3.4001
[29/50][50/600] Loss_D: 0.5086  Loss_G: 3.8521
[29/50][100/600]        Loss_D: 0.5763  Loss_G: 2.9643
[29/50][150/600]        Loss_D: 0.9782  Loss_G: 1.3463
[29/50][200/600]        Loss_D: 0.2830  Loss_G: 3.0713
[29/50][250/600]        Loss_D: 0.3920  Loss_G: 3.0967
[29/50][300/600]        Loss_D: 0.3733  Loss_G: 2.6111
[29/50][350/600]        Loss_D: 0.4270  Loss_G: 2.9006
[29/50][400/600]        Loss_D: 0.4295  Loss_G: 3.3327
[29/50][450/600]        Loss_D: 0.5538  Loss_G: 3.8329
[29/50][500/600]        Loss_D: 0.4733  Loss_G: 2.9616
[29/50][550/600]        Loss_D: 0.3111  Loss_G: 2.9044
[30/50][0/600]  Loss_D: 0.4345  Loss_G: 2.0259
[30/50][50/600] Loss_D: 0.6200  Loss_G: 3.9743
[30/50][100/600]        Loss_D: 0.4313  Loss_G: 3.2020
[30/50][150/600]        Loss_D: 0.3279  Loss_G: 3.0166
[30/50][200/600]        Loss_D: 0.3512  Loss_G: 3.5357
```

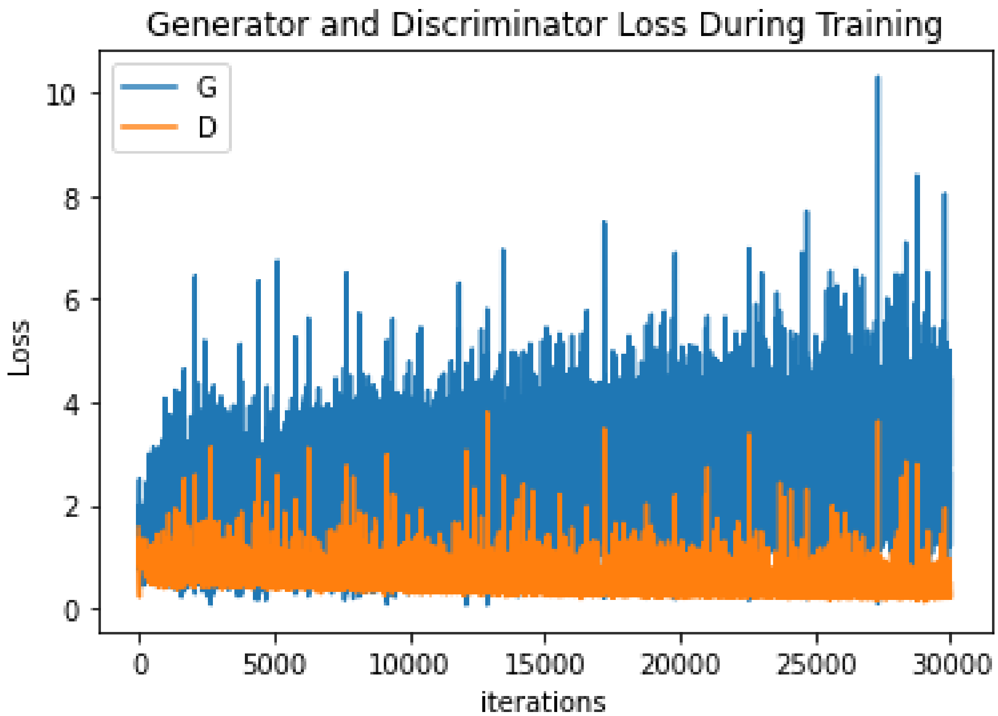
```
[30/50][250/600]        Loss_D: 0.5063  Loss_G: 2.4043
[30/50][300/600]        Loss_D: 0.5311  Loss_G: 1.3589
[30/50][350/600]        Loss_D: 0.4467  Loss_G: 3.5378
[30/50][400/600]        Loss_D: 0.4281  Loss_G: 1.9756
[30/50][450/600]        Loss_D: 0.4603  Loss_G: 4.0650
[30/50][500/600]        Loss_D: 0.3081  Loss_G: 3.1654
[30/50][550/600]        Loss_D: 0.5886  Loss_G: 1.0939
[31/50][0/600]  Loss_D: 0.8689  Loss_G: 4.5905
[31/50][50/600] Loss_D: 0.3414  Loss_G: 3.1185
[31/50][100/600]        Loss_D: 0.2474  Loss_G: 3.6269
[31/50][150/600]        Loss_D: 0.3096  Loss_G: 2.9890
[31/50][200/600]        Loss_D: 0.3116  Loss_G: 2.0922
[31/50][250/600]        Loss_D: 0.5200  Loss_G: 3.3630
[31/50][300/600]        Loss_D: 0.3644  Loss_G: 2.8833
[31/50][350/600]        Loss_D: 1.6041  Loss_G: 4.9094
[31/50][400/600]        Loss_D: 0.3132  Loss_G: 2.9788
[31/50][450/600]        Loss_D: 0.3865  Loss_G: 2.5429
[31/50][500/600]        Loss_D: 0.4075  Loss_G: 2.4534
[31/50][550/600]        Loss_D: 0.5046  Loss_G: 3.5429
[32/50][0/600]  Loss_D: 0.2764  Loss_G: 2.5847
[32/50][50/600] Loss_D: 0.2628  Loss_G: 2.8947
[32/50][100/600]        Loss_D: 0.3731  Loss_G: 2.0577
[32/50][150/600]        Loss_D: 0.3320  Loss_G: 2.7404
[32/50][200/600]        Loss_D: 1.0678  Loss_G: 1.3551
[32/50][250/600]        Loss_D: 0.3645  Loss_G: 3.4444
[32/50][300/600]        Loss_D: 0.2835  Loss_G: 3.0432
[32/50][350/600]        Loss_D: 0.2536  Loss_G: 2.4499
[32/50][400/600]        Loss_D: 0.3275  Loss_G: 2.7927
[32/50][450/600]        Loss_D: 1.0435  Loss_G: 0.7323
[32/50][500/600]        Loss_D: 0.3904  Loss_G: 2.7155
[32/50][550/600]        Loss_D: 0.3482  Loss_G: 2.6860
[33/50][0/600]  Loss_D: 0.3934  Loss_G: 3.0200
[33/50][50/600] Loss_D: 0.4311  Loss_G: 2.4619
[33/50][100/600]        Loss_D: 0.4472  Loss_G: 3.2822
[33/50][150/600]        Loss_D: 0.6425  Loss_G: 1.4607
[33/50][200/600]        Loss_D: 0.4673  Loss_G: 2.0442
[33/50][250/600]        Loss_D: 0.6093  Loss_G: 3.3594
[33/50][300/600]        Loss_D: 0.3678  Loss_G: 2.3336
[33/50][350/600]        Loss_D: 0.4184  Loss_G: 1.7541
[33/50][400/600]        Loss_D: 0.3495  Loss_G: 2.5977
[33/50][450/600]        Loss_D: 0.7288  Loss_G: 1.6039
[33/50][500/600]        Loss_D: 0.2965  Loss_G: 2.7350
[33/50][550/600]        Loss_D: 0.4555  Loss_G: 1.4545
[34/50][0/600]  Loss_D: 0.3652  Loss_G: 2.6418
[34/50][50/600] Loss_D: 0.8202  Loss_G: 1.6545
[34/50][100/600]        Loss_D: 0.2907  Loss_G: 3.0217
[34/50][150/600]        Loss_D: 0.5128  Loss_G: 3.0366
[34/50][200/600]        Loss_D: 0.3619  Loss_G: 2.8879
```

```
[34/50][250/600]	Loss_D: 0.3836	Loss_G: 3.0205
[34/50][300/600]	Loss_D: 0.3622	Loss_G: 2.3906
[34/50][350/600]	Loss_D: 0.2583	Loss_G: 3.2764
[34/50][400/600]	Loss_D: 0.3155	Loss_G: 2.9346
[34/50][450/600]	Loss_D: 0.4238	Loss_G: 2.5269
[34/50][500/600]	Loss_D: 0.4223	Loss_G: 3.0099
[34/50][550/600]	Loss_D: 0.5615	Loss_G: 3.7028
[35/50][0/600]	Loss_D: 0.4051	Loss_G: 2.3831
[35/50][50/600]	Loss_D: 0.3993	Loss_G: 1.9620
[35/50][100/600]	Loss_D: 0.3837	Loss_G: 2.8485
[35/50][150/600]	Loss_D: 0.5369	Loss_G: 1.1946
[35/50][200/600]	Loss_D: 0.5355	Loss_G: 4.3950
[35/50][250/600]	Loss_D: 0.2782	Loss_G: 3.1907
[35/50][300/600]	Loss_D: 0.4606	Loss_G: 3.6238
[35/50][350/600]	Loss_D: 0.4234	Loss_G: 1.9958
[35/50][400/600]	Loss_D: 0.3367	Loss_G: 3.1958
[35/50][450/600]	Loss_D: 0.3010	Loss_G: 2.8204
[35/50][500/600]	Loss_D: 0.7831	Loss_G: 1.2004
[35/50][550/600]	Loss_D: 0.3775	Loss_G: 2.8842
[36/50][0/600]	Loss_D: 0.6196	Loss_G: 3.0022
[36/50][50/600]	Loss_D: 0.4513	Loss_G: 3.4217
[36/50][100/600]	Loss_D: 0.3493	Loss_G: 2.7002
[36/50][150/600]	Loss_D: 0.2066	Loss_G: 2.9066
[36/50][200/600]	Loss_D: 0.3736	Loss_G: 2.8409
[36/50][250/600]	Loss_D: 0.4067	Loss_G: 2.0511
[36/50][300/600]	Loss_D: 0.3782	Loss_G: 3.6178
[36/50][350/600]	Loss_D: 0.4185	Loss_G: 2.5789
[36/50][400/600]	Loss_D: 0.2481	Loss_G: 3.1238
[36/50][450/600]	Loss_D: 0.3644	Loss_G: 3.7502
[36/50][500/600]	Loss_D: 0.5213	Loss_G: 2.7446
[36/50][550/600]	Loss_D: 0.4592	Loss_G: 2.2412
[37/50][0/600]	Loss_D: 0.4160	Loss_G: 2.1919
[37/50][50/600]	Loss_D: 0.4353	Loss_G: 2.3490
[37/50][100/600]	Loss_D: 0.4088	Loss_G: 1.6267
[37/50][150/600]	Loss_D: 0.3623	Loss_G: 2.2698
[37/50][200/600]	Loss_D: 0.3693	Loss_G: 2.3021
[37/50][250/600]	Loss_D: 0.5287	Loss_G: 1.3885
[37/50][300/600]	Loss_D: 0.3488	Loss_G: 3.1347
[37/50][350/600]	Loss_D: 0.3630	Loss_G: 2.8821
[37/50][400/600]	Loss_D: 0.3213	Loss_G: 2.8725
[37/50][450/600]	Loss_D: 0.3983	Loss_G: 1.9511
[37/50][500/600]	Loss_D: 0.5234	Loss_G: 1.6110
[37/50][550/600]	Loss_D: 0.8884	Loss_G: 4.2840
[38/50][0/600]	Loss_D: 0.3729	Loss_G: 3.1187
[38/50][50/600]	Loss_D: 0.4320	Loss_G: 2.7468
[38/50][100/600]	Loss_D: 0.3374	Loss_G: 2.7341
[38/50][150/600]	Loss_D: 0.3782	Loss_G: 2.5960
[38/50][200/600]	Loss_D: 0.3604	Loss_G: 3.3073
```

```
[38/50][250/600]        Loss_D: 0.4785  Loss_G: 3.3855
[38/50][300/600]        Loss_D: 0.4315  Loss_G: 4.0737
[38/50][350/600]        Loss_D: 0.6209  Loss_G: 2.3096
[38/50][400/600]        Loss_D: 0.3882  Loss_G: 2.9599
[38/50][450/600]        Loss_D: 0.5241  Loss_G: 1.6130
[38/50][500/600]        Loss_D: 0.5954  Loss_G: 3.6850
[38/50][550/600]        Loss_D: 0.3635  Loss_G: 3.2555
[39/50][0/600]  Loss_D: 0.4620  Loss_G: 1.8979
[39/50][50/600] Loss_D: 0.3697  Loss_G: 2.6793
[39/50][100/600]        Loss_D: 0.2963  Loss_G: 2.5822
[39/50][150/600]        Loss_D: 0.3988  Loss_G: 2.6256
[39/50][200/600]        Loss_D: 0.3062  Loss_G: 2.2848
[39/50][250/600]        Loss_D: 0.5799  Loss_G: 1.6071
[39/50][300/600]        Loss_D: 0.5138  Loss_G: 3.1001
[39/50][350/600]        Loss_D: 0.4303  Loss_G: 3.3637
[39/50][400/600]        Loss_D: 1.2605  Loss_G: 0.7932
[39/50][450/600]        Loss_D: 0.3828  Loss_G: 2.3003
[39/50][500/600]        Loss_D: 0.3983  Loss_G: 1.8306
[39/50][550/600]        Loss_D: 0.2377  Loss_G: 3.4827
[40/50][0/600]  Loss_D: 0.3497  Loss_G: 2.9943
[40/50][50/600] Loss_D: 0.3828  Loss_G: 3.0228
[40/50][100/600]        Loss_D: 0.4290  Loss_G: 3.4715
[40/50][150/600]        Loss_D: 0.6010  Loss_G: 0.8889
[40/50][200/600]        Loss_D: 0.3811  Loss_G: 3.1070
[40/50][250/600]        Loss_D: 0.3665  Loss_G: 2.4469
[40/50][300/600]        Loss_D: 0.2657  Loss_G: 2.9776
[40/50][350/600]        Loss_D: 0.3270  Loss_G: 2.2996
[40/50][400/600]        Loss_D: 0.3180  Loss_G: 2.8596
[40/50][450/600]        Loss_D: 0.3360  Loss_G: 1.9991
[40/50][500/600]        Loss_D: 0.4300  Loss_G: 4.0851
[40/50][550/600]        Loss_D: 0.4734  Loss_G: 3.9404
[41/50][0/600]  Loss_D: 0.3528  Loss_G: 3.4866
[41/50][50/600] Loss_D: 0.3737  Loss_G: 3.0201
[41/50][100/600]        Loss_D: 0.7469  Loss_G: 3.5014
[41/50][150/600]        Loss_D: 0.6690  Loss_G: 3.5134
[41/50][200/600]        Loss_D: 0.2876  Loss_G: 2.8294
[41/50][250/600]        Loss_D: 0.4412  Loss_G: 3.8998
[41/50][300/600]        Loss_D: 0.3474  Loss_G: 4.0863
[41/50][350/600]        Loss_D: 0.4137  Loss_G: 2.6298
[41/50][400/600]        Loss_D: 0.3332  Loss_G: 2.3180
[41/50][450/600]        Loss_D: 0.7884  Loss_G: 1.3356
[41/50][500/600]        Loss_D: 0.4272  Loss_G: 1.8199
[41/50][550/600]        Loss_D: 0.3224  Loss_G: 2.4811
[42/50][0/600]  Loss_D: 0.5255  Loss_G: 3.1990
[42/50][50/600] Loss_D: 0.5070  Loss_G: 3.5041
[42/50][100/600]        Loss_D: 0.3403  Loss_G: 2.4938
[42/50][150/600]        Loss_D: 0.3783  Loss_G: 4.6584
[42/50][200/600]        Loss_D: 0.3537  Loss_G: 3.4901
```

```
[42/50][250/600]        Loss_D: 0.4538  Loss_G: 1.5073
[42/50][300/600]        Loss_D: 0.2776  Loss_G: 3.0560
[42/50][350/600]        Loss_D: 0.2829  Loss_G: 2.2972
[42/50][400/600]        Loss_D: 0.4503  Loss_G: 2.8642
[42/50][450/600]        Loss_D: 0.4558  Loss_G: 3.5721
[42/50][500/600]        Loss_D: 0.4410  Loss_G: 2.9038
[42/50][550/600]        Loss_D: 0.7229  Loss_G: 4.7295
[43/50][0/600]  Loss_D: 0.3984  Loss_G: 4.6831
[43/50][50/600] Loss_D: 0.3118  Loss_G: 2.6163
[43/50][100/600]        Loss_D: 0.6643  Loss_G: 2.5711
[43/50][150/600]        Loss_D: 0.2515  Loss_G: 3.7695
[43/50][200/600]        Loss_D: 0.3989  Loss_G: 2.9739
[43/50][250/600]        Loss_D: 0.3593  Loss_G: 1.9063
[43/50][300/600]        Loss_D: 0.4481  Loss_G: 1.9534
[43/50][350/600]        Loss_D: 0.4579  Loss_G: 2.3176
[43/50][400/600]        Loss_D: 0.5154  Loss_G: 3.3844
[43/50][450/600]        Loss_D: 0.4674  Loss_G: 3.8261
[43/50][500/600]        Loss_D: 0.2979  Loss_G: 2.4485
[43/50][550/600]        Loss_D: 0.5502  Loss_G: 3.7772
[44/50][0/600]  Loss_D: 0.2417  Loss_G: 2.7986
[44/50][50/600] Loss_D: 0.3715  Loss_G: 2.4556
[44/50][100/600]        Loss_D: 0.4538  Loss_G: 3.3152
[44/50][150/600]        Loss_D: 0.7957  Loss_G: 1.4152
[44/50][200/600]        Loss_D: 0.3138  Loss_G: 3.5790
[44/50][250/600]        Loss_D: 0.3789  Loss_G: 2.4691
[44/50][300/600]        Loss_D: 0.3978  Loss_G: 1.8773
[44/50][350/600]        Loss_D: 0.3948  Loss_G: 2.3810
[44/50][400/600]        Loss_D: 0.4049  Loss_G: 2.3045
[44/50][450/600]        Loss_D: 0.2649  Loss_G: 2.2568
[44/50][500/600]        Loss_D: 0.3354  Loss_G: 3.4026
[44/50][550/600]        Loss_D: 0.4050  Loss_G: 3.3628
[45/50][0/600]  Loss_D: 0.2532  Loss_G: 2.7829
[45/50][50/600] Loss_D: 0.3550  Loss_G: 3.5631
[45/50][100/600]        Loss_D: 0.3185  Loss_G: 3.4133
[45/50][150/600]        Loss_D: 0.3679  Loss_G: 3.7935
[45/50][200/600]        Loss_D: 0.3040  Loss_G: 3.0477
[45/50][250/600]        Loss_D: 0.3765  Loss_G: 2.8988
[45/50][300/600]        Loss_D: 0.3360  Loss_G: 2.3916
[45/50][350/600]        Loss_D: 0.3614  Loss_G: 3.0630
[45/50][400/600]        Loss_D: 0.3288  Loss_G: 2.9440
[45/50][450/600]        Loss_D: 0.5104  Loss_G: 3.6972
[45/50][500/600]        Loss_D: 0.3421  Loss_G: 3.8595
[45/50][550/600]        Loss_D: 0.4026  Loss_G: 1.8263
[46/50][0/600]  Loss_D: 0.2482  Loss_G: 3.3445
[46/50][50/600] Loss_D: 0.5996  Loss_G: 4.2720
[46/50][100/600]        Loss_D: 0.2575  Loss_G: 3.1200
[46/50][150/600]        Loss_D: 0.4688  Loss_G: 1.6901
[46/50][200/600]        Loss_D: 0.3923  Loss_G: 2.2267
```

```
[46/50][250/600]	Loss_D: 0.5835	Loss_G: 1.7922
[46/50][300/600]	Loss_D: 0.8190	Loss_G: 1.2642
[46/50][350/600]	Loss_D: 0.4624	Loss_G: 3.8113
[46/50][400/600]	Loss_D: 0.6133	Loss_G: 2.4126
[46/50][450/600]	Loss_D: 0.5923	Loss_G: 1.6736
[46/50][500/600]	Loss_D: 0.4448	Loss_G: 1.6860
[46/50][550/600]	Loss_D: 0.2382	Loss_G: 3.9120
[47/50][0/600]	Loss_D: 0.4586	Loss_G: 3.8590
[47/50][50/600]	Loss_D: 0.4774	Loss_G: 2.8442
[47/50][100/600]	Loss_D: 0.3341	Loss_G: 3.4906
[47/50][150/600]	Loss_D: 0.2103	Loss_G: 3.3713
[47/50][200/600]	Loss_D: 0.3644	Loss_G: 2.3649
[47/50][250/600]	Loss_D: 0.4324	Loss_G: 1.9970
[47/50][300/600]	Loss_D: 0.7659	Loss_G: 1.6945
[47/50][350/600]	Loss_D: 0.2646	Loss_G: 2.9827
[47/50][400/600]	Loss_D: 0.3491	Loss_G: 3.5705
[47/50][450/600]	Loss_D: 0.4345	Loss_G: 4.0074
[47/50][500/600]	Loss_D: 0.3407	Loss_G: 2.9295
[47/50][550/600]	Loss_D: 0.4654	Loss_G: 1.8896
[48/50][0/600]	Loss_D: 0.4462	Loss_G: 2.5714
[48/50][50/600]	Loss_D: 0.3556	Loss_G: 2.9966
[48/50][100/600]	Loss_D: 0.2191	Loss_G: 2.7932
[48/50][150/600]	Loss_D: 0.2010	Loss_G: 3.1783
[48/50][200/600]	Loss_D: 0.3863	Loss_G: 2.4399
[48/50][250/600]	Loss_D: 0.3241	Loss_G: 2.5135
[48/50][300/600]	Loss_D: 0.3415	Loss_G: 1.8424
[48/50][350/600]	Loss_D: 0.4284	Loss_G: 3.5664
[48/50][400/600]	Loss_D: 0.1710	Loss_G: 3.7601
[48/50][450/600]	Loss_D: 0.3622	Loss_G: 3.2441
[48/50][500/600]	Loss_D: 0.7117	Loss_G: 1.2641
[48/50][550/600]	Loss_D: 0.4622	Loss_G: 2.4253
[49/50][0/600]	Loss_D: 0.8841	Loss_G: 5.4495
[49/50][50/600]	Loss_D: 0.5757	Loss_G: 1.9121
[49/50][100/600]	Loss_D: 0.3230	Loss_G: 2.2311
[49/50][150/600]	Loss_D: 0.5445	Loss_G: 2.5606
[49/50][200/600]	Loss_D: 0.5118	Loss_G: 1.9560
[49/50][250/600]	Loss_D: 0.2729	Loss_G: 2.8002
[49/50][300/600]	Loss_D: 0.5223	Loss_G: 4.3253
[49/50][350/600]	Loss_D: 0.2614	Loss_G: 3.1697
[49/50][400/600]	Loss_D: 1.7870	Loss_G: 0.4338
[49/50][450/600]	Loss_D: 0.2505	Loss_G: 2.7221
[49/50][500/600]	Loss_D: 0.2571	Loss_G: 2.6830
[49/50][550/600]	Loss_D: 0.3396	Loss_G: 3.8744
```



0.5 Qualitative Visualisations

[6]:
```
# Test GAN on a random sample and display on 6X6 grid
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(8,8))
plt.axis("off")
ims = [[plt.imshow(np.transpose(i,(1,2,0)), animated=True)] for i in img_list]
ani = animation.ArtistAnimation(fig, ims, interval=1000, repeat_delay=1000,␣
↪blit=True)

HTML(ani.to_jshtml())
```

[6]: <IPython.core.display.HTML object>