```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318

Enter your authorization code:
··········
Mounted at /content/drive
```

```
%cd /content/drive/My Drive/
```

```
/content/drive/My Drive
```

```
!apt-get install p7zip-full
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
p7zip-full is already the newest version (16.02+dfsg-6).
0 upgraded, 0 newly installed, 0 to remove and 28 not upgraded.
```

```
from __future__ import print_function
#%matplotlib inline
import argparse
import os
import random
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torchvision.utils as vutils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML
import zipfile
import cv2
import unzip

# Set random seem for reproducibility
manualSeed = 999
#manualSeed = random.randint(1, 10000) # use if you want new results
print("Random Seed: ", manualSeed)
random.seed(manualSeed)
torch.manual_seed(manualSeed)
```

```
```

```
Random Seed:  999
```

```
# drive = '/content/sample_data/square_data/squares/'
# for i in range(5000):
#     x = random.randrange(0,5)
#     img = np.zeros((64,64,3), np.uint8)
#     color = (random.randrange(1,255), random.randrange(1,255), random.randrange(1,255))

#     img[42-x:42+x, 42-x:42+x, :] = tuple(reversed(color))
#     rotationMat = cv2.getRotationMatrix2D((32,32), random.randrange(0,90), 1)
#     img = cv2.warpAffine(img, rotationMat,(img.shape[0],img.shape[1]))
#     trans_mat = np.float32([[1,0,random.randrange(-20,20)], [0,1,random.randrange(-20,20)]])
#     img = cv2.warpAffine(img, trans_mat, (img.shape[0],img.shape[1]))
#     # cv2.war
#     file = drive + 'square'+ str(i) + '.png'
#     plt.imsave(file, img)
```

Inputs

Let's define some inputs for the run:

- **dataroot** - the path to the root of the dataset folder. We will talk more about the dataset in the n
- **workers** - the number of worker threads for loading the data with the DataLoader
- **batch_size** - the batch size used in training. The DCGAN paper uses a batch size of 128
- **image_size** - the spatial size of the images used for training. This implementation defaults to 6
  size is desired, the structures of D and G must be changed. See `here <https://github.com/pytorch/examples/issues/70>` __ for more details
- **nc** - number of color channels in the input images. For color images this is 3
- **nz** - length of latent vector
- **ngf** - relates to the depth of feature maps carried through the generator
- **ndf** - sets the depth of feature maps propagated through the discriminator
- **num_epochs** - number of training epochs to run. Training for longer will probably lead to better r
  also take much longer
- **lr** - learning rate for training. As described in the DCGAN paper, this number should be 0.0002
- **beta1** - beta1 hyperparameter for Adam optimizers. As described in paper, this number should b
- **ngpu** - number of GPUs available. If this is 0, code will run in CPU mode. If this number is greate
  run on that number of GPUs

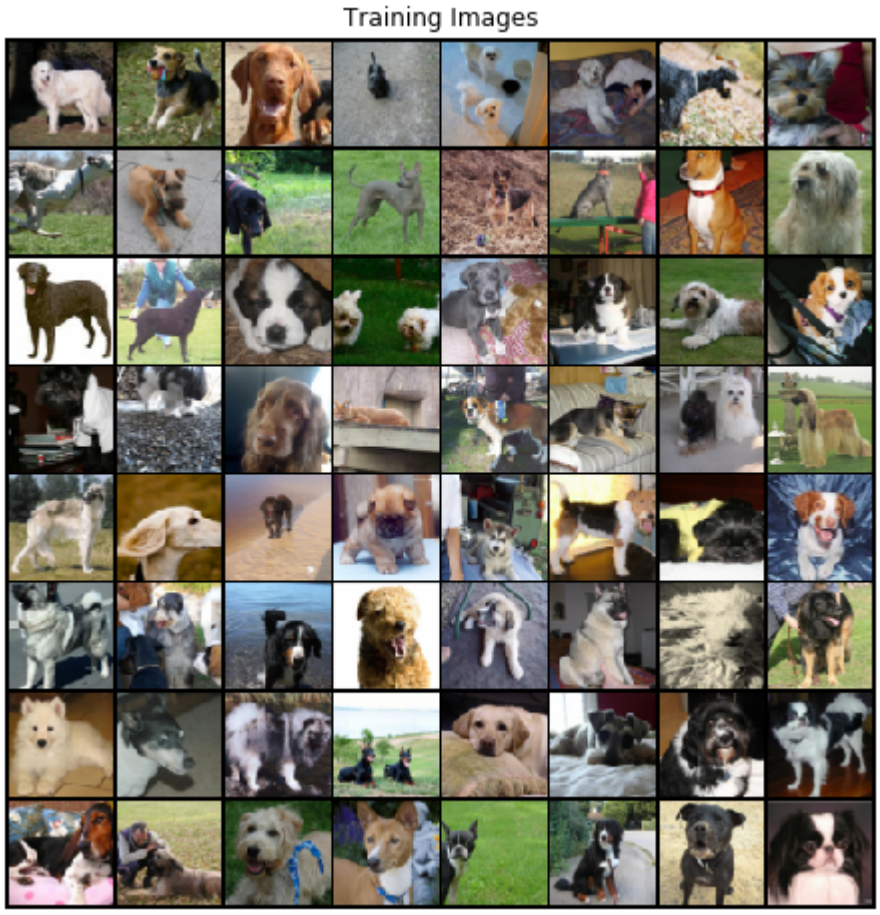
```
# Root directory for dataset
dataroot = "/content/sample_data/Images"

# Number of workers for dataloader
workers = 4

# Batch size during training
batch_size = 128

# Spatial size of training images. All images will be resized to this
```

```
#   size using a transformer.
image_size = 64

# Number of channels in the training images. For color images this is 3
nc = 3

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

# Size of feature maps in generator
ngf = 64

# Size of feature maps in discriminator
ndf = 64

# Number of training epochs
num_epochs = 100

# Learning rate for optimizers
lr = 0.0005

# Beta1 hyperparam for Adam optimizers
beta1 = 0.5

# Number of GPUs available. Use 0 for CPU mode.
ngpu = 1
```

```
# We can use an image folder dataset the way we have it setup.
# Create the dataset
dataset = dset.ImageFolder(root=dataroot,
                           transform=transforms.Compose([
                               transforms.Resize(image_size),
                               transforms.CenterCrop(image_size),
                               transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                           ]))
# Create the dataloader
dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
                                         shuffle=True, num_workers=workers)

# Decide which device we want to run on
device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else "cpu")
```

```
# Plot some training images
real_batch = next(iter(dataloader))
plt.figure(figsize=(8,8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(real_batch[0].to(device)[:64], padding=2, normalize=True).c
```

```
<matplotlib.image.AxesImage at 0x7f44ad9bae10>
```




```
# custom weights initialization called on netG and netD
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)


# Generator Code

class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # state size. (ngf*8) x 4 x 4
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # state size. (ngf*4) x 8 x 8
            nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # state size. (ngf*2) x 16 x 16
            nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
```

```
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # state size. (ngf) x 32 x 32
            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. (nc) x 64 x 64
        )

    def forward(self, input):
        return self.main(input)
```

Now, we can instantiate the generator and apply the `weights_init` function. Check out the printed m the generator object is structured.

```
# Create the generator
netG = Generator(ngpu).to(device)

# Handle multi-gpu if desired
if (device.type == 'cuda') and (ngpu > 1):
    netG = nn.DataParallel(netG, list(range(ngpu)))

# Apply the weights_init function to randomly initialize all weights
#  to mean=0, stdev=0.2.
netG.apply(weights_init)

# Print the model
print(netG)
```

```
Generator(
  (main): Sequential(
    (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), bi
    (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), bi
    (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), bia
    (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
    (13): Tanh()
  )
)
```

Discriminator Code

```
class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 64 x 64
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
```

```
            # state size. (ndf) x 32 x 32
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*4) x 8 x 8
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*8) x 4 x 4
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

Now, as with the generator, we can create the discriminator, apply the `weights_init` function, and pri structure.

```
# Create the Discriminator
netD = Discriminator(ngpu).to(device)

# Handle multi-gpu if desired
if (device.type == 'cuda') and (ngpu > 1):
    netD = nn.DataParallel(netD, list(range(ngpu)))

# Apply the weights_init function to randomly initialize all weights
#  to mean=0, stdev=0.2.
netD.apply(weights_init)

# Print the model
print(netD)
```

```
Discriminator(
  (main): Sequential(
    (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)
    (12): Sigmoid()
  )
)
```

```
# Initialize BCELoss function
criterion = nn.BCELoss()

# Create batch of latent vectors that we will use to visualize
```

```
#  the progression of the generator
fixed_noise = torch.randn(64, nz, 1, 1, device=device)

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

# Setup Adam optimizers for both G and D
optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1, 0.999))



# Training Loop

# Lists to keep track of progress
img_list = []
G_losses = []
D_losses = []
iters = 0

print("Starting Training Loop...")
# For each epoch
for epoch in range(num_epochs):
    # For each batch in the dataloader
    for i, data in enumerate(dataloader, 0):

        ############################
        # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
        ###########################
        ## Train with all-real batch
        netD.zero_grad()
        # Format batch
        real_cpu = data[0].to(device)
        b_size = real_cpu.size(0)
        label = torch.full((b_size,), real_label, device=device)
        # Forward pass real batch through D
        output = netD(real_cpu).view(-1)
        # Calculate loss on all-real batch
        errD_real = criterion(output, label)
        # Calculate gradients for D in backward pass
        errD_real.backward()
        D_x = output.mean().item()

        ## Train with all-fake batch
        # Generate batch of latent vectors
        noise = torch.randn(b_size, nz, 1, 1, device=device)
        # Generate fake image batch with G
        fake = netG(noise)
        label.fill_(fake_label)
        # Classify all fake batch with D
        output = netD(fake.detach()).view(-1)
        # Calculate D's loss on the all-fake batch
        errD_fake = criterion(output, label)
        # Calculate the gradients for this batch
        errD_fake.backward()
        D_G_z1 = output.mean().item()
        # Add the gradients from the all-real and all-fake batches
        errD = errD_real + errD_fake
        # Update D
        optimizerD.step()

        ############################
        # (2) Update G network: maximize log(D(G(z)))
        ###########################
        netG.zero_grad()
        label.fill_(real_label)  # fake labels are real for generator cost
        # Since we just updated D, perform another forward pass of all-fake batch through D
        output = netD(fake).view(-1)
        # Calculate G's loss based on this output
```

```
        # Calculate G's loss based on this output
        errG = criterion(output, label)
        # Calculate gradients for G
        errG.backward()
        D_G_z2 = output.mean().item()
        # Update G
        optimizerG.step()

        # Output training stats
        if i % 50 == 0:
            print('[%d/%d][%d/%d]\tLoss_D: %.4f\tLoss_G: %.4f\tD(x): %.4f\tD(G(z)): %.4f / %.4f'
                  % (epoch, num_epochs, i, len(dataloader),
                     errD.item(), errG.item(), D_x, D_G_z1, D_G_z2))

        # Save Losses for plotting later
        G_losses.append(errG.item())
        D_losses.append(errD.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(dataloader)-1)):
            with torch.no_grad():
                fake = netG(fixed_noise).detach().cpu()
            img_list.append(vutils.make_grid(fake, padding=2, normalize=True))

        iters += 1
```

```
Starting Training Loop...
[0/100][0/161]  Loss_D: 0.8100  Loss_G: 0.9598  D(x): 0.5468    D(G(z)): 0.1011 / 0.4484
[0/100][50/161] Loss_D: 0.6963  Loss_G: 4.5809  D(x): 0.9340    D(G(z)): 0.4086 / 0.0147
[0/100][100/161]        Loss_D: 0.4431  Loss_G: 3.7177  D(x): 0.9215    D(G(z)): 0.2729
[0/100][150/161]        Loss_D: 0.5768  Loss_G: 3.2273  D(x): 0.8737    D(G(z)): 0.3138
[1/100][0/161]  Loss_D: 0.3516  Loss_G: 3.5345  D(x): 0.7531    D(G(z)): 0.0361 / 0.0431
[1/100][50/161] Loss_D: 0.4609  Loss_G: 2.5573  D(x): 0.7198    D(G(z)): 0.0835 / 0.1096
[1/100][100/161]        Loss_D: 0.5839  Loss_G: 2.6688  D(x): 0.7289    D(G(z)): 0.1752
[1/100][150/161]        Loss_D: 0.4623  Loss_G: 2.9039  D(x): 0.8494    D(G(z)): 0.2287
[2/100][0/161]  Loss_D: 0.3297  Loss_G: 3.8357  D(x): 0.8941    D(G(z)): 0.1747 / 0.0310
[2/100][50/161] Loss_D: 0.3638  Loss_G: 3.3285  D(x): 0.9308    D(G(z)): 0.2318 / 0.0476
[2/100][100/161]        Loss_D: 0.8392  Loss_G: 1.2146  D(x): 0.5120    D(G(z)): 0.0776
[2/100][150/161]        Loss_D: 1.2937  Loss_G: 5.4068  D(x): 0.9462    D(G(z)): 0.6285
[3/100][0/161]  Loss_D: 0.5849  Loss_G: 3.4913  D(x): 0.8777    D(G(z)): 0.3216 / 0.0417
[3/100][50/161] Loss_D: 0.3511  Loss_G: 3.0541  D(x): 0.8938    D(G(z)): 0.1927 / 0.0659
[3/100][100/161]        Loss_D: 0.7528  Loss_G: 1.2757  D(x): 0.6596    D(G(z)): 0.2001
[3/100][150/161]        Loss_D: 0.4854  Loss_G: 3.4287  D(x): 0.9146    D(G(z)): 0.3031
[4/100][0/161]  Loss_D: 0.4562  Loss_G: 2.4405  D(x): 0.7134    D(G(z)): 0.0842 / 0.1193
[4/100][50/161] Loss_D: 0.7954  Loss_G: 1.3949  D(x): 0.5174    D(G(z)): 0.0330 / 0.3059
[4/100][100/161]        Loss_D: 1.6129  Loss_G: 0.8149  D(x): 0.2614    D(G(z)): 0.0186
[4/100][150/161]        Loss_D: 0.4013  Loss_G: 3.3253  D(x): 0.8842    D(G(z)): 0.2189
[5/100][0/161]  Loss_D: 0.4969  Loss_G: 2.1798  D(x): 0.7131    D(G(z)): 0.1049 / 0.1540
[5/100][50/161] Loss_D: 4.8455  Loss_G: 0.7697  D(x): 0.0148    D(G(z)): 0.0011 / 0.5658
[5/100][100/161]        Loss_D: 0.5555  Loss_G: 3.1644  D(x): 0.8333    D(G(z)): 0.2747
[5/100][150/161]        Loss_D: 0.6307  Loss_G: 1.9482  D(x): 0.6377    D(G(z)): 0.0845
[6/100][0/161]  Loss_D: 0.4062  Loss_G: 2.7437  D(x): 0.8414    D(G(z)): 0.1796 / 0.0900
[6/100][50/161] Loss_D: 0.5699  Loss_G: 2.0237  D(x): 0.6634    D(G(z)): 0.1071 / 0.1820
[6/100][100/161]        Loss_D: 0.5627  Loss_G: 3.4666  D(x): 0.8351    D(G(z)): 0.2787
[6/100][150/161]        Loss_D: 0.6430  Loss_G: 1.9308  D(x): 0.6293    D(G(z)): 0.0891
[7/100][0/161]  Loss_D: 0.4245  Loss_G: 1.9482  D(x): 0.7481    D(G(z)): 0.1002 / 0.1769
[7/100][50/161] Loss_D: 0.7366  Loss_G: 4.6704  D(x): 0.9202    D(G(z)): 0.4196 / 0.0135
[7/100][100/161]        Loss_D: 1.2626  Loss_G: 0.2991  D(x): 0.3717    D(G(z)): 0.0202
[7/100][150/161]        Loss_D: 0.7244  Loss_G: 2.6214  D(x): 0.7478    D(G(z)): 0.2983
[8/100][0/161]  Loss_D: 0.4011  Loss_G: 3.3844  D(x): 0.8610    D(G(z)): 0.1993 / 0.0474
[8/100][50/161] Loss_D: 0.4771  Loss_G: 2.2528  D(x): 0.7506    D(G(z)): 0.1425 / 0.1361
[8/100][100/161]        Loss_D: 0.5973  Loss_G: 2.6121  D(x): 0.8132    D(G(z)): 0.2853
[8/100][150/161]        Loss_D: 0.7265  Loss_G: 1.9010  D(x): 0.5651    D(G(z)): 0.0382
[9/100][0/161]  Loss_D: 0.3778  Loss_G: 2.8879  D(x): 0.8644    D(G(z)): 0.1881 / 0.0752
[9/100][50/161] Loss_D: 0.4798  Loss_G: 1.8994  D(x): 0.7270    D(G(z)): 0.1082 / 0.2057
[9/100][100/161]        Loss_D: 0.3882  Loss_G: 3.2982  D(x): 0.9309    D(G(z)): 0.2509
[9/100][150/161]        Loss_D: 0.6445  Loss_G: 2.2010  D(x): 0.7109    D(G(z)): 0.1894
[10/100][0/161] Loss_D: 0.3115  Loss_G: 2.7996  D(x): 0.8949    D(G(z)): 0.1630 / 0.0909
[10/100][50/161]        Loss_D: 0.4068  Loss_G: 2.5914  D(x): 0.8343    D(G(z)): 0.1760
[10/100][100/161]       Loss_D: 0.4239  Loss_G: 2.5728  D(x): 0.8313    D(G(z)): 0.1804
[10/100][150/161]       Loss_D: 1.1135  Loss_G: 4.0758  D(x): 0.8886    D(G(z)): 0.5451
[11/100][0/161] Loss_D: 0.5972  Loss_G: 2.0738  D(x): 0.7020    D(G(z)): 0.1612 / 0.1659
[11/100][50/161]        Loss_D: 0.4419  Loss_G: 2.4177  D(x): 0.7445    D(G(z)): 0.1041
[11/100][100/161]       Loss_D: 0.3374  Loss_G: 2.6633  D(x): 0.8862    D(G(z)): 0.1787
[11/100][150/161]       Loss_D: 0.4328  Loss_G: 2.0877  D(x): 0.7341    D(G(z)): 0.0802
[12/100][0/161] Loss_D: 0.5737  Loss_G: 1.1143  D(x): 0.6111    D(G(z)): 0.0277 / 0.4123
[12/100][50/161]        Loss_D: 0.4993  Loss_G: 2.4096  D(x): 0.6807    D(G(z)): 0.0762
[12/100][100/161]       Loss_D: 0.6172  Loss_G: 3.4375  D(x): 0.8630    D(G(z)): 0.3242
[12/100][150/161]       Loss_D: 0.3863  Loss_G: 2.9132  D(x): 0.8467    D(G(z)): 0.1798
[13/100][0/161] Loss_D: 0.3378  Loss_G: 3.1575  D(x): 0.8989    D(G(z)): 0.1874 / 0.0574
[13/100][50/161]        Loss_D: 0.4934  Loss_G: 2.0075  D(x): 0.7240    D(G(z)): 0.1176
[13/100][100/161]       Loss_D: 0.7196  Loss_G: 4.5532  D(x): 0.8899    D(G(z)): 0.4082
[13/100][150/161]       Loss_D: 0.9585  Loss_G: 1.0420  D(x): 0.4673    D(G(z)): 0.0555
```

```
[14/100][0/161] Loss_D: 0.3544  Loss_G: 3.8067  D(x): 0.9223    D(G(z)): 0.2091 / 0.0346
[14/100][50/161]        Loss_D: 0.3162  Loss_G: 2.8970  D(x): 0.8649    D(G(z)): 0.1432
[14/100][100/161]       Loss_D: 0.4305  Loss_G: 3.9975  D(x): 0.8946    D(G(z)): 0.2392
[14/100][150/161]       Loss_D: 0.3893  Loss_G: 3.4480  D(x): 0.8790    D(G(z)): 0.1952
[15/100][0/161] Loss_D: 0.2914  Loss_G: 3.0229  D(x): 0.8451    D(G(z)): 0.1013 / 0.0691
[15/100][50/161]        Loss_D: 0.5258  Loss_G: 1.4038  D(x): 0.6677    D(G(z)): 0.0735
[15/100][100/161]       Loss_D: 0.9909  Loss_G: 1.9857  D(x): 0.5736    D(G(z)): 0.1960
[15/100][150/161]       Loss_D: 0.3505  Loss_G: 3.1201  D(x): 0.8342    D(G(z)): 0.1302
[16/100][0/161] Loss_D: 0.4912  Loss_G: 1.8795  D(x): 0.7521    D(G(z)): 0.1538 / 0.1946
[16/100][50/161]        Loss_D: 0.3483  Loss_G: 2.7029  D(x): 0.8274    D(G(z)): 0.1237
[16/100][100/161]       Loss_D: 0.4016  Loss_G: 3.3015  D(x): 0.9240    D(G(z)): 0.2455
[16/100][150/161]       Loss_D: 0.4928  Loss_G: 3.2516  D(x): 0.8821    D(G(z)): 0.2720
[17/100][0/161] Loss_D: 0.6529  Loss_G: 1.4271  D(x): 0.5850    D(G(z)): 0.0496 / 0.3023
[17/100][50/161]        Loss_D: 0.4060  Loss_G: 2.6064  D(x): 0.8561    D(G(z)): 0.1943
[17/100][100/161]       Loss_D: 1.5500  Loss_G: 4.1865  D(x): 0.7915    D(G(z)): 0.6107
[17/100][150/161]       Loss_D: 0.5390  Loss_G: 2.2428  D(x): 0.6872    D(G(z)): 0.1031
[18/100][0/161] Loss_D: 0.4133  Loss_G: 2.2259  D(x): 0.8119    D(G(z)): 0.1512 / 0.1454
[18/100][50/161]        Loss_D: 0.5940  Loss_G: 4.1822  D(x): 0.9694    D(G(z)): 0.3789
[18/100][100/161]       Loss_D: 0.4309  Loss_G: 2.3518  D(x): 0.7655    D(G(z)): 0.1174
[18/100][150/161]       Loss_D: 0.3315  Loss_G: 2.3597  D(x): 0.8243    D(G(z)): 0.1055
[19/100][0/161] Loss_D: 0.3602  Loss_G: 4.1574  D(x): 0.9561    D(G(z)): 0.2480 / 0.0222
[19/100][50/161]        Loss_D: 1.9111  Loss_G: 5.1599  D(x): 0.9570    D(G(z)): 0.7610
[19/100][100/161]       Loss_D: 0.3967  Loss_G: 2.7722  D(x): 0.8515    D(G(z)): 0.1880
[19/100][150/161]       Loss_D: 0.4159  Loss_G: 1.9492  D(x): 0.7309    D(G(z)): 0.0707
[20/100][0/161] Loss_D: 0.2756  Loss_G: 3.3697  D(x): 0.9078    D(G(z)): 0.1494 / 0.0486
[20/100][50/161]        Loss_D: 0.3743  Loss_G: 2.2132  D(x): 0.8164    D(G(z)): 0.1374
[20/100][100/161]       Loss_D: 0.3573  Loss_G: 2.5387  D(x): 0.8442    D(G(z)): 0.1485
[20/100][150/161]       Loss_D: 0.3181  Loss_G: 3.2135  D(x): 0.9115    D(G(z)): 0.1846
[21/100][0/161] Loss_D: 0.7129  Loss_G: 5.8813  D(x): 0.9637    D(G(z)): 0.4319 / 0.0046
[21/100][50/161]        Loss_D: 0.4488  Loss_G: 2.8387  D(x): 0.8975    D(G(z)): 0.2469
[21/100][100/161]       Loss_D: 0.4369  Loss_G: 2.9535  D(x): 0.8719    D(G(z)): 0.2241
[21/100][150/161]       Loss_D: 0.3364  Loss_G: 2.0728  D(x): 0.8025    D(G(z)): 0.0941
[22/100][0/161] Loss_D: 0.3219  Loss_G: 3.3504  D(x): 0.8943    D(G(z)): 0.1714 / 0.0502
[22/100][50/161]        Loss_D: 0.3870  Loss_G: 1.6277  D(x): 0.7666    D(G(z)): 0.0918
[22/100][100/161]       Loss_D: 1.5678  Loss_G: 4.5470  D(x): 0.9381    D(G(z)): 0.6533
[22/100][150/161]       Loss_D: 0.3485  Loss_G: 3.0940  D(x): 0.8501    D(G(z)): 0.1447
[23/100][0/161] Loss_D: 0.3104  Loss_G: 3.0113  D(x): 0.8515    D(G(z)): 0.1213 / 0.0742
[23/100][50/161]        Loss_D: 0.3607  Loss_G: 2.4891  D(x): 0.8075    D(G(z)): 0.1179
[23/100][100/161]       Loss_D: 0.9413  Loss_G: 5.6846  D(x): 0.9877    D(G(z)): 0.5384
[23/100][150/161]       Loss_D: 0.3940  Loss_G: 2.9335  D(x): 0.8582    D(G(z)): 0.1874
[24/100][0/161] Loss_D: 0.3667  Loss_G: 3.1611  D(x): 0.8919    D(G(z)): 0.1960 / 0.0625
[24/100][50/161]        Loss_D: 0.3539  Loss_G: 2.2563  D(x): 0.7904    D(G(z)): 0.0836
[24/100][100/161]       Loss_D: 1.3719  Loss_G: 0.6154  D(x): 0.3823    D(G(z)): 0.0651
[24/100][150/161]       Loss_D: 0.4485  Loss_G: 2.8565  D(x): 0.8454    D(G(z)): 0.2154
[25/100][0/161] Loss_D: 0.5795  Loss_G: 4.2793  D(x): 0.9461    D(G(z)): 0.3462 / 0.0204
[25/100][50/161]        Loss_D: 0.3708  Loss_G: 2.6107  D(x): 0.8250    D(G(z)): 0.1413
[25/100][100/161]       Loss_D: 0.4612  Loss_G: 3.5890  D(x): 0.9277    D(G(z)): 0.2787
[25/100][150/161]       Loss_D: 0.3610  Loss_G: 3.4062  D(x): 0.9062    D(G(z)): 0.2079
[26/100][0/161] Loss_D: 0.2721  Loss_G: 3.5776  D(x): 0.9390    D(G(z)): 0.1711 / 0.0387
[26/100][50/161]        Loss_D: 2.2831  Loss_G: 0.2214  D(x): 0.2625    D(G(z)): 0.1062
[26/100][100/161]       Loss_D: 0.3730  Loss_G: 2.7806  D(x): 0.7836    D(G(z)): 0.0964
[26/100][150/161]       Loss_D: 0.3297  Loss_G: 2.8581  D(x): 0.8962    D(G(z)): 0.1762
[27/100][0/161] Loss_D: 0.2885  Loss_G: 3.0864  D(x): 0.9179    D(G(z)): 0.1677 / 0.0637
[27/100][50/161]        Loss_D: 0.2758  Loss_G: 3.6182  D(x): 0.8761    D(G(z)): 0.1171
[27/100][100/161]       Loss_D: 0.3016  Loss_G: 2.7493  D(x): 0.8473    D(G(z)): 0.1119
[27/100][150/161]       Loss_D: 1.4117  Loss_G: 7.2502  D(x): 0.9811    D(G(z)): 0.6687
[28/100][0/161] Loss_D: 1.0521  Loss_G: 3.2744  D(x): 0.6394    D(G(z)): 0.3294 / 0.0617
[28/100][50/161]        Loss_D: 0.3688  Loss_G: 3.4191  D(x): 0.9232    D(G(z)): 0.2241
```

```
[28/100][100/161]       Loss_D: 0.2514  Loss_G: 3.4468  D(x): 0.9135    D(G(z)): 0.1359
[28/100][150/161]       Loss_D: 0.5360  Loss_G: 1.9681  D(x): 0.6512    D(G(z)): 0.0455
[29/100][0/161] Loss_D: 0.2172  Loss_G: 3.0462  D(x): 0.9216    D(G(z)): 0.1162 / 0.0698
[29/100][50/161]        Loss_D: 0.2604  Loss_G: 3.4587  D(x): 0.9033    D(G(z)): 0.1281
[29/100][100/161]       Loss_D: 0.2788  Loss_G: 2.9256  D(x): 0.8774    D(G(z)): 0.1193
[29/100][150/161]       Loss_D: 0.8995  Loss_G: 1.9343  D(x): 0.5767    D(G(z)): 0.1529
[30/100][0/161] Loss_D: 0.4888  Loss_G: 3.0780  D(x): 0.7877    D(G(z)): 0.1731 / 0.0736
[30/100][50/161]        Loss_D: 0.4005  Loss_G: 2.9689  D(x): 0.8845    D(G(z)): 0.2156
[30/100][100/161]       Loss_D: 0.3153  Loss_G: 3.0021  D(x): 0.8560    D(G(z)): 0.1279
[30/100][150/161]       Loss_D: 0.5292  Loss_G: 4.1446  D(x): 0.9274    D(G(z)): 0.3191
[31/100][0/161] Loss_D: 0.3551  Loss_G: 4.3526  D(x): 0.9555    D(G(z)): 0.2341 / 0.0196
[31/100][50/161]        Loss_D: 0.2579  Loss_G: 2.7767  D(x): 0.8528    D(G(z)): 0.0747
[31/100][100/161]       Loss_D: 0.4186  Loss_G: 1.7467  D(x): 0.7293    D(G(z)): 0.0659
[31/100][150/161]       Loss_D: 0.4153  Loss_G: 2.4227  D(x): 0.7047    D(G(z)): 0.0316
[32/100][0/161] Loss_D: 0.2435  Loss_G: 2.7260  D(x): 0.8584    D(G(z)): 0.0747 / 0.0993
[32/100][50/161]        Loss_D: 0.2620  Loss_G: 3.0170  D(x): 0.8748    D(G(z)): 0.1090
[32/100][100/161]       Loss_D: 0.5651  Loss_G: 2.6390  D(x): 0.7352    D(G(z)): 0.1493
[32/100][150/161]       Loss_D: 0.3230  Loss_G: 3.1147  D(x): 0.8270    D(G(z)): 0.1045
[33/100][0/161] Loss_D: 0.2457  Loss_G: 3.2423  D(x): 0.9205    D(G(z)): 0.1358 / 0.0554
[33/100][50/161]        Loss_D: 0.3355  Loss_G: 3.5691  D(x): 0.9191    D(G(z)): 0.1937
[33/100][100/161]       Loss_D: 0.1873  Loss_G: 3.3758  D(x): 0.9116    D(G(z)): 0.0832
[33/100][150/161]       Loss_D: 0.2571  Loss_G: 3.1728  D(x): 0.8919    D(G(z)): 0.1159
[34/100][0/161] Loss_D: 0.2082  Loss_G: 3.2461  D(x): 0.9140    D(G(z)): 0.1021 / 0.0596
[34/100][50/161]        Loss_D: 0.3242  Loss_G: 2.2852  D(x): 0.7697    D(G(z)): 0.0315
[34/100][100/161]       Loss_D: 0.3301  Loss_G: 2.2457  D(x): 0.7932    D(G(z)): 0.0702
[34/100][150/161]       Loss_D: 0.4858  Loss_G: 3.3671  D(x): 0.8939    D(G(z)): 0.2676
[35/100][0/161] Loss_D: 0.2975  Loss_G: 2.9650  D(x): 0.8916    D(G(z)): 0.1429 / 0.0808
[35/100][50/161]        Loss_D: 0.2808  Loss_G: 3.2691  D(x): 0.8461    D(G(z)): 0.0864
[35/100][100/161]       Loss_D: 0.2661  Loss_G: 3.1920  D(x): 0.8836    D(G(z)): 0.1173
[35/100][150/161]       Loss_D: 0.2650  Loss_G: 2.5467  D(x): 0.8251    D(G(z)): 0.0558
[36/100][0/161] Loss_D: 0.1729  Loss_G: 3.2838  D(x): 0.9110    D(G(z)): 0.0697 / 0.0591
[36/100][50/161]        Loss_D: 0.2144  Loss_G: 3.4127  D(x): 0.8892    D(G(z)): 0.0823
[36/100][100/161]       Loss_D: 0.9553  Loss_G: 3.4623  D(x): 0.8102    D(G(z)): 0.4188
[36/100][150/161]       Loss_D: 0.3239  Loss_G: 3.1418  D(x): 0.8970    D(G(z)): 0.1684
[37/100][0/161] Loss_D: 0.4797  Loss_G: 4.1344  D(x): 0.9317    D(G(z)): 0.2875 / 0.0229
[37/100][50/161]        Loss_D: 0.2556  Loss_G: 2.5658  D(x): 0.8550    D(G(z)): 0.0794
[37/100][100/161]       Loss_D: 0.3122  Loss_G: 2.8919  D(x): 0.8711    D(G(z)): 0.1396
[37/100][150/161]       Loss_D: 0.2137  Loss_G: 3.6202  D(x): 0.9253    D(G(z)): 0.1131
[38/100][0/161] Loss_D: 0.2120  Loss_G: 3.4505  D(x): 0.8878    D(G(z)): 0.0797 / 0.0516
[38/100][50/161]        Loss_D: 0.8800  Loss_G: 4.8780  D(x): 0.8384    D(G(z)): 0.3912
[38/100][100/161]       Loss_D: 0.2629  Loss_G: 3.0774  D(x): 0.8651    D(G(z)): 0.0963
[38/100][150/161]       Loss_D: 0.2038  Loss_G: 3.3257  D(x): 0.8969    D(G(z)): 0.0824
[39/100][0/161] Loss_D: 0.1773  Loss_G: 3.6386  D(x): 0.9050    D(G(z)): 0.0679 / 0.0372
[39/100][50/161]        Loss_D: 0.3131  Loss_G: 3.0762  D(x): 0.8801    D(G(z)): 0.1451
[39/100][100/161]       Loss_D: 0.2293  Loss_G: 3.2647  D(x): 0.9111    D(G(z)): 0.1107
[39/100][150/161]       Loss_D: 0.1456  Loss_G: 3.9095  D(x): 0.9265    D(G(z)): 0.0618
[40/100][0/161] Loss_D: 0.2754  Loss_G: 4.1447  D(x): 0.9407    D(G(z)): 0.1719 / 0.0232
[40/100][50/161]        Loss_D: 0.2673  Loss_G: 3.2434  D(x): 0.8706    D(G(z)): 0.1082
[40/100][100/161]       Loss_D: 0.3082  Loss_G: 1.9497  D(x): 0.7852    D(G(z)): 0.0449
[40/100][150/161]       Loss_D: 1.8133  Loss_G: 1.8963  D(x): 0.5261    D(G(z)): 0.5001
[41/100][0/161] Loss_D: 2.3553  Loss_G: 0.8780  D(x): 0.2661    D(G(z)): 0.1776 / 0.5425
[41/100][50/161]        Loss_D: 1.2300  Loss_G: 4.7306  D(x): 0.9013    D(G(z)): 0.5536
[41/100][100/161]       Loss_D: 0.3906  Loss_G: 2.6680  D(x): 0.8464    D(G(z)): 0.1742
[41/100][150/161]       Loss_D: 0.3319  Loss_G: 3.0882  D(x): 0.8757    D(G(z)): 0.1622
[42/100][0/161] Loss_D: 0.3136  Loss_G: 4.2159  D(x): 0.9494    D(G(z)): 0.2048 / 0.0213
[42/100][50/161]        Loss_D: 0.2601  Loss_G: 2.7243  D(x): 0.8418    D(G(z)): 0.0668
[42/100][100/161]       Loss_D: 0.2002  Loss_G: 3.2181  D(x): 0.9264    D(G(z)): 0.1082
[42/100][150/161]       Loss_D: 0.2225  Loss_G: 3.6192  D(x): 0.9081    D(G(z)): 0.1082
```

```
[42/100][150/161]       Loss_D: 0.2225  Loss_G: 3.6192  D(x): 0.9081    D(G(z)): 0.1082
[43/100][0/161] Loss_D: 0.2122  Loss_G: 3.6416  D(x): 0.8799    D(G(z)): 0.0686 / 0.0416
[43/100][50/161]        Loss_D: 0.2042  Loss_G: 3.2277  D(x): 0.9225    D(G(z)): 0.1025
[43/100][100/161]       Loss_D: 0.2329  Loss_G: 3.9943  D(x): 0.9515    D(G(z)): 0.1482
[43/100][150/161]       Loss_D: 0.6993  Loss_G: 1.9119  D(x): 0.6927    D(G(z)): 0.1881
[44/100][0/161] Loss_D: 0.5648  Loss_G: 4.6800  D(x): 0.9333    D(G(z)): 0.3230 / 0.0154
[44/100][50/161]        Loss_D: 0.2111  Loss_G: 3.6193  D(x): 0.9227    D(G(z)): 0.1103
[44/100][100/161]       Loss_D: 0.2505  Loss_G: 3.0106  D(x): 0.8468    D(G(z)): 0.0645
[44/100][150/161]       Loss_D: 1.5893  Loss_G: 1.5647  D(x): 0.4430    D(G(z)): 0.2619
[45/100][0/161] Loss_D: 0.7991  Loss_G: 3.5663  D(x): 0.7763    D(G(z)): 0.3198 / 0.0477
[45/100][50/161]        Loss_D: 0.2520  Loss_G: 3.2349  D(x): 0.8777    D(G(z)): 0.0991
[45/100][100/161]       Loss_D: 0.1929  Loss_G: 3.1303  D(x): 0.9144    D(G(z)): 0.0871
[45/100][150/161]       Loss_D: 0.3918  Loss_G: 2.0553  D(x): 0.7454    D(G(z)): 0.0630
[46/100][0/161] Loss_D: 0.1807  Loss_G: 3.8025  D(x): 0.9178    D(G(z)): 0.0802 / 0.0339
[46/100][50/161]        Loss_D: 0.2051  Loss_G: 3.9145  D(x): 0.9370    D(G(z)): 0.1197
[46/100][100/161]       Loss_D: 0.1743  Loss_G: 3.9165  D(x): 0.9308    D(G(z)): 0.0847
[46/100][150/161]       Loss_D: 0.2438  Loss_G: 3.5440  D(x): 0.9143    D(G(z)): 0.1302
[47/100][0/161] Loss_D: 0.1383  Loss_G: 4.0218  D(x): 0.9329    D(G(z)): 0.0613 / 0.0294
[47/100][50/161]        Loss_D: 0.1520  Loss_G: 3.8182  D(x): 0.9452    D(G(z)): 0.0850
[47/100][100/161]       Loss_D: 0.2021  Loss_G: 3.4132  D(x): 0.9123    D(G(z)): 0.0914
[47/100][150/161]       Loss_D: 5.2012  Loss_G: 0.0026  D(x): 0.0342    D(G(z)): 0.0079
[48/100][0/161] Loss_D: 1.7627  Loss_G: 5.5343  D(x): 0.8745    D(G(z)): 0.6765 / 0.0154
[48/100][50/161]        Loss_D: 0.2045  Loss_G: 3.9913  D(x): 0.8978    D(G(z)): 0.0753
[48/100][100/161]       Loss_D: 0.2457  Loss_G: 3.5835  D(x): 0.9313    D(G(z)): 0.1371
[48/100][150/161]       Loss_D: 0.1757  Loss_G: 3.9380  D(x): 0.9461    D(G(z)): 0.1006
[49/100][0/161] Loss_D: 0.2756  Loss_G: 4.2279  D(x): 0.9414    D(G(z)): 0.1717 / 0.0229
[49/100][50/161]        Loss_D: 0.1861  Loss_G: 3.6761  D(x): 0.9133    D(G(z)): 0.0806
[49/100][100/161]       Loss_D: 0.1304  Loss_G: 3.9364  D(x): 0.9333    D(G(z)): 0.0560
[49/100][150/161]       Loss_D: 0.1227  Loss_G: 3.8845  D(x): 0.9307    D(G(z)): 0.0464
[50/100][0/161] Loss_D: 0.1127  Loss_G: 4.2785  D(x): 0.9444    D(G(z)): 0.0472 / 0.0273
[50/100][50/161]        Loss_D: 0.1425  Loss_G: 4.1065  D(x): 0.9546    D(G(z)): 0.0865
[50/100][100/161]       Loss_D: 0.1825  Loss_G: 4.2270  D(x): 0.9599    D(G(z)): 0.1208
[50/100][150/161]       Loss_D: 1.0231  Loss_G: 2.0651  D(x): 0.6397    D(G(z)): 0.2911
[51/100][0/161] Loss_D: 0.6083  Loss_G: 2.9686  D(x): 0.7926    D(G(z)): 0.2216 / 0.0784
[51/100][50/161]        Loss_D: 0.3675  Loss_G: 2.5897  D(x): 0.8131    D(G(z)): 0.1150
[51/100][100/161]       Loss_D: 0.2211  Loss_G: 3.5900  D(x): 0.9010    D(G(z)): 0.0985
[51/100][150/161]       Loss_D: 0.3311  Loss_G: 4.8422  D(x): 0.9391    D(G(z)): 0.2073
[52/100][0/161] Loss_D: 0.2361  Loss_G: 3.1658  D(x): 0.8374    D(G(z)): 0.0417 / 0.0776
[52/100][50/161]        Loss_D: 0.5582  Loss_G: 1.6389  D(x): 0.6183    D(G(z)): 0.0109
[52/100][100/161]       Loss_D: 0.2725  Loss_G: 4.3042  D(x): 0.9520    D(G(z)): 0.1816
[52/100][150/161]       Loss_D: 4.7542  Loss_G: 10.0843 D(x): 0.9964    D(G(z)): 0.9649
[53/100][0/161] Loss_D: 2.3075  Loss_G: 6.4784  D(x): 0.9364    D(G(z)): 0.7687 / 0.0076
[53/100][50/161]        Loss_D: 0.2150  Loss_G: 3.9405  D(x): 0.9640    D(G(z)): 0.1488
[53/100][100/161]       Loss_D: 0.1826  Loss_G: 3.7916  D(x): 0.9278    D(G(z)): 0.0939
[53/100][150/161]       Loss_D: 0.2579  Loss_G: 2.6639  D(x): 0.8840    D(G(z)): 0.1097
[54/100][0/161] Loss_D: 0.1327  Loss_G: 4.0275  D(x): 0.9166    D(G(z)): 0.0391 / 0.0296
[54/100][50/161]        Loss_D: 0.1434  Loss_G: 3.6999  D(x): 0.9522    D(G(z)): 0.0837
[54/100][100/161]       Loss_D: 0.1420  Loss_G: 4.6802  D(x): 0.9661    D(G(z)): 0.0938
[54/100][150/161]       Loss_D: 1.2261  Loss_G: 0.7929  D(x): 0.4816    D(G(z)): 0.1042
[55/100][0/161] Loss_D: 0.3341  Loss_G: 3.0412  D(x): 0.8035    D(G(z)): 0.0681 / 0.0909
[55/100][50/161]        Loss_D: 0.2526  Loss_G: 4.2977  D(x): 0.9510    D(G(z)): 0.1598
[55/100][100/161]       Loss_D: 0.2166  Loss_G: 3.0401  D(x): 0.8568    D(G(z)): 0.0460
[55/100][150/161]       Loss_D: 0.1755  Loss_G: 3.6999  D(x): 0.9195    D(G(z)): 0.0781
[56/100][0/161] Loss_D: 0.1040  Loss_G: 4.0644  D(x): 0.9468    D(G(z)): 0.0451 / 0.0278
[56/100][50/161]        Loss_D: 0.1747  Loss_G: 3.2306  D(x): 0.8716    D(G(z)): 0.0288
[56/100][100/161]       Loss_D: 0.1492  Loss_G: 4.0860  D(x): 0.9611    D(G(z)): 0.0973
[56/100][150/161]       Loss_D: 3.0611  Loss_G: 4.2588  D(x): 0.8910    D(G(z)): 0.8616
[57/100][0/161] Loss_D: 1.4793  Loss_G: 1.1818  D(x): 0.4207    D(G(z)): 0.1502 / 0.4373
```

```
[57/100][50/161]	Loss_D: 0.4034	Loss_G: 2.9087	D(x): 0.7962	D(G(z)): 0.1200
[57/100][100/161]	Loss_D: 0.2057	Loss_G: 3.4774	D(x): 0.9017	D(G(z)): 0.0871
[57/100][150/161]	Loss_D: 0.1689	Loss_G: 4.0520	D(x): 0.9668	D(G(z)): 0.1186
[58/100][0/161] Loss_D: 0.1937	Loss_G: 3.8698	D(x): 0.9227	D(G(z)): 0.0935 / 0.0368
[58/100][50/161]	Loss_D: 0.1835	Loss_G: 3.2757	D(x): 0.8780	D(G(z)): 0.0438
[58/100][100/161]	Loss_D: 0.1198	Loss_G: 3.8948	D(x): 0.9523	D(G(z)): 0.0641
[58/100][150/161]	Loss_D: 0.1319	Loss_G: 3.8544	D(x): 0.9622	D(G(z)): 0.0812
[59/100][0/161] Loss_D: 0.1010	Loss_G: 4.2193	D(x): 0.9630	D(G(z)): 0.0579 / 0.0236
[59/100][50/161]	Loss_D: 0.1136	Loss_G: 3.8082	D(x): 0.9385	D(G(z)): 0.0458
[59/100][100/161]	Loss_D: 0.1093	Loss_G: 4.4188	D(x): 0.9834	D(G(z)): 0.0835
[59/100][150/161]	Loss_D: 0.1687	Loss_G: 3.2691	D(x): 0.9110	D(G(z)): 0.0632
[60/100][0/161] Loss_D: 0.0849	Loss_G: 4.6524	D(x): 0.9502	D(G(z)): 0.0311 / 0.0165
[60/100][50/161]	Loss_D: 0.2798	Loss_G: 6.4457	D(x): 0.9776	D(G(z)): 0.1940
[60/100][100/161]	Loss_D: 2.4755	Loss_G: 3.1127	D(x): 0.8233	D(G(z)): 0.7698
[60/100][150/161]	Loss_D: 0.6298	Loss_G: 1.3119	D(x): 0.6380	D(G(z)): 0.0578
[61/100][0/161] Loss_D: 0.3911	Loss_G: 3.9757	D(x): 0.8817	D(G(z)): 0.1703 / 0.0371
[61/100][50/161]	Loss_D: 0.1980	Loss_G: 3.6707	D(x): 0.9056	D(G(z)): 0.0797
[61/100][100/161]	Loss_D: 0.2287	Loss_G: 2.8302	D(x): 0.8585	D(G(z)): 0.0603
[61/100][150/161]	Loss_D: 0.3453	Loss_G: 5.4239	D(x): 0.9796	D(G(z)): 0.2398
[62/100][0/161] Loss_D: 0.1686	Loss_G: 3.8938	D(x): 0.8745	D(G(z)): 0.0225 / 0.0337
[62/100][50/161]	Loss_D: 0.1464	Loss_G: 3.7855	D(x): 0.9220	D(G(z)): 0.0555
[62/100][100/161]	Loss_D: 0.1526	Loss_G: 3.4037	D(x): 0.9247	D(G(z)): 0.0666
[62/100][150/161]	Loss_D: 0.1174	Loss_G: 3.8994	D(x): 0.9493	D(G(z)): 0.0583
[63/100][0/161] Loss_D: 0.0980	Loss_G: 4.6666	D(x): 0.9697	D(G(z)): 0.0620 / 0.0139
[63/100][50/161]	Loss_D: 0.1535	Loss_G: 3.8241	D(x): 0.9349	D(G(z)): 0.0742
[63/100][100/161]	Loss_D: 0.1211	Loss_G: 4.5078	D(x): 0.9646	D(G(z)): 0.0759
[63/100][150/161]	Loss_D: 0.8183	Loss_G: 2.6612	D(x): 0.7447	D(G(z)): 0.2953
[64/100][0/161] Loss_D: 0.7566	Loss_G: 4.2882	D(x): 0.9124	D(G(z)): 0.3932 / 0.0275
[64/100][50/161]	Loss_D: 0.2793	Loss_G: 3.8689	D(x): 0.9129	D(G(z)): 0.1456
[64/100][100/161]	Loss_D: 0.1973	Loss_G: 3.3425	D(x): 0.9010	D(G(z)): 0.0712
[64/100][150/161]	Loss_D: 0.1426	Loss_G: 3.7510	D(x): 0.9367	D(G(z)): 0.0657
[65/100][0/161] Loss_D: 0.1724	Loss_G: 3.5825	D(x): 0.8746	D(G(z)): 0.0281 / 0.0494
[65/100][50/161]	Loss_D: 0.1062	Loss_G: 3.9866	D(x): 0.9662	D(G(z)): 0.0656
[65/100][100/161]	Loss_D: 0.1066	Loss_G: 4.1841	D(x): 0.9555	D(G(z)): 0.0535
[65/100][150/161]	Loss_D: 0.1224	Loss_G: 4.2504	D(x): 0.9365	D(G(z)): 0.0498
[66/100][0/161] Loss_D: 0.3571	Loss_G: 6.6641	D(x): 0.9867	D(G(z)): 0.2568 / 0.0023
[66/100][50/161]	Loss_D: 0.1141	Loss_G: 4.0237	D(x): 0.9223	D(G(z)): 0.0290
[66/100][100/161]	Loss_D: 0.1413	Loss_G: 4.2737	D(x): 0.9513	D(G(z)): 0.0763
[66/100][150/161]	Loss_D: 3.0074	Loss_G: 0.0013	D(x): 0.1193	D(G(z)): 0.0196
[67/100][0/161] Loss_D: 1.9815	Loss_G: 3.2906	D(x): 0.8029	D(G(z)): 0.6976 / 0.0891
[67/100][50/161]	Loss_D: 0.3223	Loss_G: 3.9087	D(x): 0.8683	D(G(z)): 0.1295
[67/100][100/161]	Loss_D: 0.2029	Loss_G: 3.3293	D(x): 0.9150	D(G(z)): 0.0914
[67/100][150/161]	Loss_D: 0.1983	Loss_G: 3.2729	D(x): 0.8805	D(G(z)): 0.0558
[68/100][0/161] Loss_D: 0.1378	Loss_G: 3.7139	D(x): 0.9317	D(G(z)): 0.0580 / 0.0431
[68/100][50/161]	Loss_D: 0.1354	Loss_G: 3.6628	D(x): 0.9284	D(G(z)): 0.0527
[68/100][100/161]	Loss_D: 0.1571	Loss_G: 3.7116	D(x): 0.9294	D(G(z)): 0.0726
[68/100][150/161]	Loss_D: 0.1225	Loss_G: 4.4644	D(x): 0.9568	D(G(z)): 0.0682
[69/100][0/161] Loss_D: 0.0800	Loss_G: 4.1908	D(x): 0.9610	D(G(z)): 0.0377 / 0.0266
[69/100][50/161]	Loss_D: 0.0998	Loss_G: 3.9751	D(x): 0.9458	D(G(z)): 0.0395
[69/100][100/161]	Loss_D: 0.1126	Loss_G: 4.0938	D(x): 0.9666	D(G(z)): 0.0697
[69/100][150/161]	Loss_D: 0.1572	Loss_G: 3.5554	D(x): 0.8867	D(G(z)): 0.0306
[70/100][0/161] Loss_D: 0.1594	Loss_G: 5.0148	D(x): 0.9732	D(G(z)): 0.1086 / 0.0111
[70/100][50/161]	Loss_D: 0.0965	Loss_G: 4.4237	D(x): 0.9759	D(G(z)): 0.0665
[70/100][100/161]	Loss_D: 2.3298	Loss_G: 0.9234	D(x): 0.3444	D(G(z)): 0.2662
[70/100][150/161]	Loss_D: 1.2302	Loss_G: 3.2183	D(x): 0.7618	D(G(z)): 0.4816
[71/100][0/161] Loss_D: 0.6796	Loss_G: 1.6093	D(x): 0.6699	D(G(z)): 0.1337 / 0.2802
[71/100][50/161]	Loss_D: 0.3623	Loss_G: 2.8814	D(x): 0.7774	D(G(z)): 0.0582
[71/100][100/161]	Loss_D: 0.2865	Loss_G: 4.9405	D(x): 0.9774	D(G(z)): 0.2023
```

```
[71/100][150/161]       Loss_D: 0.1626  Loss_G: 4.0041  D(x): 0.9448    D(G(z)): 0.0879
[72/100][0/161] Loss_D: 0.1484  Loss_G: 3.6390  D(x): 0.9390    D(G(z)): 0.0726 / 0.0428
[72/100][50/161]        Loss_D: 0.1798  Loss_G: 3.1706  D(x): 0.9054    D(G(z)): 0.0671
[72/100][100/161]       Loss_D: 0.1403  Loss_G: 3.6506  D(x): 0.9302    D(G(z)): 0.0583
[72/100][150/161]       Loss_D: 0.1044  Loss_G: 4.1882  D(x): 0.9487    D(G(z)): 0.0467
[73/100][0/161] Loss_D: 0.0902  Loss_G: 4.2956  D(x): 0.9478    D(G(z)): 0.0327 / 0.0257
[73/100][50/161]        Loss_D: 0.1278  Loss_G: 4.6614  D(x): 0.9664    D(G(z)): 0.0806
[73/100][100/161]       Loss_D: 0.1092  Loss_G: 4.1640  D(x): 0.9465    D(G(z)): 0.0498
[73/100][150/161]       Loss_D: 0.1066  Loss_G: 4.3517  D(x): 0.9540    D(G(z)): 0.0540
[74/100][0/161] Loss_D: 0.1483  Loss_G: 5.1025  D(x): 0.9793    D(G(z)): 0.1103 / 0.0093
[74/100][50/161]        Loss_D: 0.7173  Loss_G: 0.0443  D(x): 0.5692    D(G(z)): 0.0125
[74/100][100/161]       Loss_D: 0.3634  Loss_G: 3.2263  D(x): 0.7917    D(G(z)): 0.0735
[74/100][150/161]       Loss_D: 0.2154  Loss_G: 3.3624  D(x): 0.8929    D(G(z)): 0.0752
[75/100][0/161] Loss_D: 0.1264  Loss_G: 4.6237  D(x): 0.9157    D(G(z)): 0.0310 / 0.0221
[75/100][50/161]        Loss_D: 0.1923  Loss_G: 2.7573  D(x): 0.8573    D(G(z)): 0.0262
[75/100][100/161]       Loss_D: 0.1317  Loss_G: 3.9437  D(x): 0.9481    D(G(z)): 0.0669
[75/100][150/161]       Loss_D: 0.1306  Loss_G: 4.4124  D(x): 0.9735    D(G(z)): 0.0868
[76/100][0/161] Loss_D: 0.0938  Loss_G: 5.0034  D(x): 0.9776    D(G(z)): 0.0655 / 0.0104
[76/100][50/161]        Loss_D: 0.1915  Loss_G: 2.4002  D(x): 0.8559    D(G(z)): 0.0259
[76/100][100/161]       Loss_D: 0.1026  Loss_G: 4.5317  D(x): 0.9700    D(G(z)): 0.0643
[76/100][150/161]       Loss_D: 0.1147  Loss_G: 3.9053  D(x): 0.9470    D(G(z)): 0.0541
[77/100][0/161] Loss_D: 0.1011  Loss_G: 4.2633  D(x): 0.9597    D(G(z)): 0.0541 / 0.0253
[77/100][50/161]        Loss_D: 0.0887  Loss_G: 4.6375  D(x): 0.9728    D(G(z)): 0.0562
[77/100][100/161]       Loss_D: 0.0826  Loss_G: 4.2337  D(x): 0.9461    D(G(z)): 0.0250
[77/100][150/161]       Loss_D: 0.0932  Loss_G: 4.0861  D(x): 0.9401    D(G(z)): 0.0280
[78/100][0/161] Loss_D: 0.1024  Loss_G: 5.1864  D(x): 0.9738    D(G(z)): 0.0656 / 0.0104
[78/100][50/161]        Loss_D: 0.0864  Loss_G: 3.9110  D(x): 0.9388    D(G(z)): 0.0206
[78/100][100/161]       Loss_D: 0.0620  Loss_G: 4.8196  D(x): 0.9710    D(G(z)): 0.0303
[78/100][150/161]       Loss_D: 0.0687  Loss_G: 4.3069  D(x): 0.9664    D(G(z)): 0.0324
[79/100][0/161] Loss_D: 0.0693  Loss_G: 5.4533  D(x): 0.9729    D(G(z)): 0.0370 / 0.0084
[79/100][50/161]        Loss_D: 0.0881  Loss_G: 5.2084  D(x): 0.9922    D(G(z)): 0.0715
[79/100][100/161]       Loss_D: 0.0857  Loss_G: 4.2052  D(x): 0.9442    D(G(z)): 0.0251
[79/100][150/161]       Loss_D: 0.1021  Loss_G: 5.9619  D(x): 0.9894    D(G(z)): 0.0817
[80/100][0/161] Loss_D: 2.4322  Loss_G: 1.2583  D(x): 0.3416    D(G(z)): 0.3336 / 0.4737
[80/100][50/161]        Loss_D: 1.4974  Loss_G: 1.3617  D(x): 0.6086    D(G(z)): 0.5139
[80/100][100/161]       Loss_D: 1.9370  Loss_G: 0.8662  D(x): 0.2938    D(G(z)): 0.0352
[80/100][150/161]       Loss_D: 0.8040  Loss_G: 1.7487  D(x): 0.6137    D(G(z)): 0.1075
[81/100][0/161] Loss_D: 0.3952  Loss_G: 3.6449  D(x): 0.7943    D(G(z)): 0.0909 / 0.0508
[81/100][50/161]        Loss_D: 0.2043  Loss_G: 3.9878  D(x): 0.9095    D(G(z)): 0.0848
[81/100][100/161]       Loss_D: 0.2725  Loss_G: 3.1655  D(x): 0.8523    D(G(z)): 0.0743
[81/100][150/161]       Loss_D: 0.1928  Loss_G: 3.1379  D(x): 0.8844    D(G(z)): 0.0433
[82/100][0/161] Loss_D: 0.1546  Loss_G: 4.4513  D(x): 0.9753    D(G(z)): 0.1041 / 0.0185
[82/100][50/161]        Loss_D: 0.2982  Loss_G: 6.0806  D(x): 0.9774    D(G(z)): 0.2057
[82/100][100/161]       Loss_D: 0.1261  Loss_G: 5.1911  D(x): 0.9868    D(G(z)): 0.0928
[82/100][150/161]       Loss_D: 0.1009  Loss_G: 4.1103  D(x): 0.9341    D(G(z)): 0.0291
[83/100][0/161] Loss_D: 0.1237  Loss_G: 5.0797  D(x): 0.9802    D(G(z)): 0.0905 / 0.0111
[83/100][50/161]        Loss_D: 0.0962  Loss_G: 4.6482  D(x): 0.9671    D(G(z)): 0.0568
[83/100][100/161]       Loss_D: 0.1077  Loss_G: 4.1587  D(x): 0.9498    D(G(z)): 0.0503
[83/100][150/161]       Loss_D: 0.2150  Loss_G: 6.8076  D(x): 0.9492    D(G(z)): 0.1279
[84/100][0/161] Loss_D: 2.1903  Loss_G: 1.1071  D(x): 0.3666    D(G(z)): 0.2890 / 0.5252
[84/100][50/161]        Loss_D: 1.7990  Loss_G: 3.8496  D(x): 0.8605    D(G(z)): 0.7010
[84/100][100/161]       Loss_D: 0.3043  Loss_G: 4.4955  D(x): 0.8935    D(G(z)): 0.1364
[84/100][150/161]       Loss_D: 0.3915  Loss_G: 3.8761  D(x): 0.7956    D(G(z)): 0.0883
[85/100][0/161] Loss_D: 0.2173  Loss_G: 4.1857  D(x): 0.9329    D(G(z)): 0.1190 / 0.0252
[85/100][50/161]        Loss_D: 0.2039  Loss_G: 3.4358  D(x): 0.9060    D(G(z)): 0.0839
[85/100][100/161]       Loss_D: 0.1272  Loss_G: 4.0672  D(x): 0.9551    D(G(z)): 0.0720
[85/100][150/161]       Loss_D: 0.1463  Loss_G: 4.0033  D(x): 0.9070    D(G(z)): 0.0399
[86/100][0/161] Loss_D: 0.1149  Loss_G: 4.4225  D(x): 0.9770    D(G(z)): 0.0803 / 0.0195
```

```
[86/100][0/161] Loss_D: 0.1149  Loss_G: 4.4225  D(x): 0.9770    D(G(z)): 0.0805 / 0.019
[86/100][50/161]       Loss_D: 0.0771  Loss_G: 4.3311  D(x): 0.9720    D(G(z)): 0.0441
[86/100][100/161]      Loss_D: 0.1346  Loss_G: 3.8800  D(x): 0.9330    D(G(z)): 0.0583
[86/100][150/161]      Loss_D: 0.0778  Loss_G: 4.6705  D(x): 0.9557    D(G(z)): 0.0302
[87/100][0/161] Loss_D: 0.0716  Loss_G: 4.5778  D(x): 0.9589    D(G(z)): 0.0278 / 0.020
[87/100][50/161]       Loss_D: 0.1281  Loss_G: 3.4883  D(x): 0.9009    D(G(z)): 0.0188
[87/100][100/161]      Loss_D: 0.0489  Loss_G: 4.6324  D(x): 0.9844    D(G(z)): 0.0314
[87/100][150/161]      Loss_D: 0.0826  Loss_G: 4.6818  D(x): 0.9791    D(G(z)): 0.0549
[88/100][0/161] Loss_D: 0.0758  Loss_G: 4.9387  D(x): 0.9878    D(G(z)): 0.0583 / 0.013
[88/100][50/161]       Loss_D: 0.0985  Loss_G: 5.4013  D(x): 0.9844    D(G(z)): 0.0734
[88/100][100/161]      Loss_D: 0.0822  Loss_G: 4.3424  D(x): 0.9513    D(G(z)): 0.0299
[88/100][150/161]      Loss_D: 0.1266  Loss_G: 3.6090  D(x): 0.8998    D(G(z)): 0.0157
[89/100][0/161] Loss_D: 0.0435  Loss_G: 4.8609  D(x): 0.9871    D(G(z)): 0.0293 / 0.013
[89/100][50/161]       Loss_D: 0.0622  Loss_G: 4.4689  D(x): 0.9709    D(G(z)): 0.0306
[89/100][100/161]      Loss_D: 0.1817  Loss_G: 6.2353  D(x): 0.9939    D(G(z)): 0.1439
[89/100][150/161]      Loss_D: 1.1168  Loss_G: 1.4523  D(x): 0.6006    D(G(z)): 0.3339
[90/100][0/161] Loss_D: 0.9074  Loss_G: 2.5720  D(x): 0.6910    D(G(z)): 0.2766 / 0.122
[90/100][50/161]       Loss_D: 0.2981  Loss_G: 3.6818  D(x): 0.8707    D(G(z)): 0.1009
[90/100][100/161]      Loss_D: 0.1423  Loss_G: 4.8103  D(x): 0.9774    D(G(z)): 0.0995
[90/100][150/161]      Loss_D: 0.2086  Loss_G: 3.5856  D(x): 0.8570    D(G(z)): 0.0343
[91/100][0/161] Loss_D: 0.1070  Loss_G: 4.4915  D(x): 0.9500    D(G(z)): 0.0505 / 0.021
[91/100][50/161]       Loss_D: 0.0994  Loss_G: 5.0027  D(x): 0.9790    D(G(z)): 0.0698
[91/100][100/161]      Loss_D: 0.1200  Loss_G: 4.1215  D(x): 0.9196    D(G(z)): 0.0309
[91/100][150/161]      Loss_D: 0.0760  Loss_G: 4.5070  D(x): 0.9557    D(G(z)): 0.0281
[92/100][0/161] Loss_D: 0.0602  Loss_G: 4.7300  D(x): 0.9677    D(G(z)): 0.0255 / 0.020
[92/100][50/161]       Loss_D: 0.0678  Loss_G: 4.7170  D(x): 0.9688    D(G(z)): 0.0332
[92/100][100/161]      Loss_D: 0.0812  Loss_G: 5.2247  D(x): 0.9882    D(G(z)): 0.0631
[92/100][150/161]      Loss_D: 0.0717  Loss_G: 4.3838  D(x): 0.9480    D(G(z)): 0.0161
[93/100][0/161] Loss_D: 0.0728  Loss_G: 4.8076  D(x): 0.9888    D(G(z)): 0.0567 / 0.012
[93/100][50/161]       Loss_D: 0.0613  Loss_G: 4.8884  D(x): 0.9711    D(G(z)): 0.0298
[93/100][100/161]      Loss_D: 0.0775  Loss_G: 4.3596  D(x): 0.9494    D(G(z)): 0.0232
[93/100][150/161]      Loss_D: 2.3133  Loss_G: 0.5577  D(x): 0.2486    D(G(z)): 0.0843
[94/100][0/161] Loss_D: 0.5694  Loss_G: 2.6833  D(x): 0.7198    D(G(z)): 0.0758 / 0.150
[94/100][50/161]       Loss_D: 0.2332  Loss_G: 3.8332  D(x): 0.8611    D(G(z)): 0.0521
[94/100][100/161]      Loss_D: 0.1141  Loss_G: 4.3357  D(x): 0.9201    D(G(z)): 0.0232
[94/100][150/161]      Loss_D: 0.0864  Loss_G: 4.8479  D(x): 0.9688    D(G(z)): 0.0481
[95/100][0/161] Loss_D: 0.0527  Loss_G: 5.0035  D(x): 0.9758    D(G(z)): 0.0266 / 0.014
[95/100][50/161]       Loss_D: 0.0940  Loss_G: 4.5648  D(x): 0.9534    D(G(z)): 0.0416
[95/100][100/161]      Loss_D: 0.0635  Loss_G: 4.9686  D(x): 0.9572    D(G(z)): 0.0182
[95/100][150/161]      Loss_D: 0.1063  Loss_G: 4.8353  D(x): 0.9793    D(G(z)): 0.0735
[96/100][0/161] Loss_D: 0.0667  Loss_G: 5.0134  D(x): 0.9564    D(G(z)): 0.0198 / 0.015
[96/100][50/161]       Loss_D: 0.0652  Loss_G: 4.5096  D(x): 0.9648    D(G(z)): 0.0269
[96/100][100/161]      Loss_D: 0.0843  Loss_G: 4.9425  D(x): 0.9729    D(G(z)): 0.0510
[96/100][150/161]      Loss_D: 0.0773  Loss_G: 4.1287  D(x): 0.9607    D(G(z)): 0.0339
[97/100][0/161] Loss_D: 0.0507  Loss_G: 4.7642  D(x): 0.9808    D(G(z)): 0.0298 / 0.014
[97/100][50/161]       Loss_D: 0.0654  Loss_G: 4.8765  D(x): 0.9732    D(G(z)): 0.0322
[97/100][100/161]      Loss_D: 0.0806  Loss_G: 4.6489  D(x): 0.9755    D(G(z)): 0.0509
[97/100][150/161]      Loss_D: 0.0666  Loss_G: 4.2086  D(x): 0.9540    D(G(z)): 0.0180
[98/100][0/161] Loss_D: 0.0713  Loss_G: 4.8351  D(x): 0.9633    D(G(z)): 0.0313 / 0.015
[98/100][50/161]       Loss_D: 0.0673  Loss_G: 4.6903  D(x): 0.9570    D(G(z)): 0.0217
[98/100][100/161]      Loss_D: 0.0615  Loss_G: 4.7208  D(x): 0.9559    D(G(z)): 0.0149
[98/100][150/161]      Loss_D: 0.0397  Loss_G: 5.5651  D(x): 0.9794    D(G(z)): 0.0180
[99/100][0/161] Loss_D: 0.0596  Loss_G: 5.5022  D(x): 0.9899    D(G(z)): 0.0461 / 0.007
[99/100][50/161]       Loss_D: 0.0480  Loss_G: 5.4659  D(x): 0.9854    D(G(z)): 0.0309
[99/100][100/161]      Loss_D: 1.6554  Loss_G: 2.0721  D(x): 0.6630    D(G(z)): 0.5826
[99/100][150/161]      Loss_D: 1.4941  Loss_G: 1.1523  D(x): 0.5356    D(G(z)): 0.4741
```

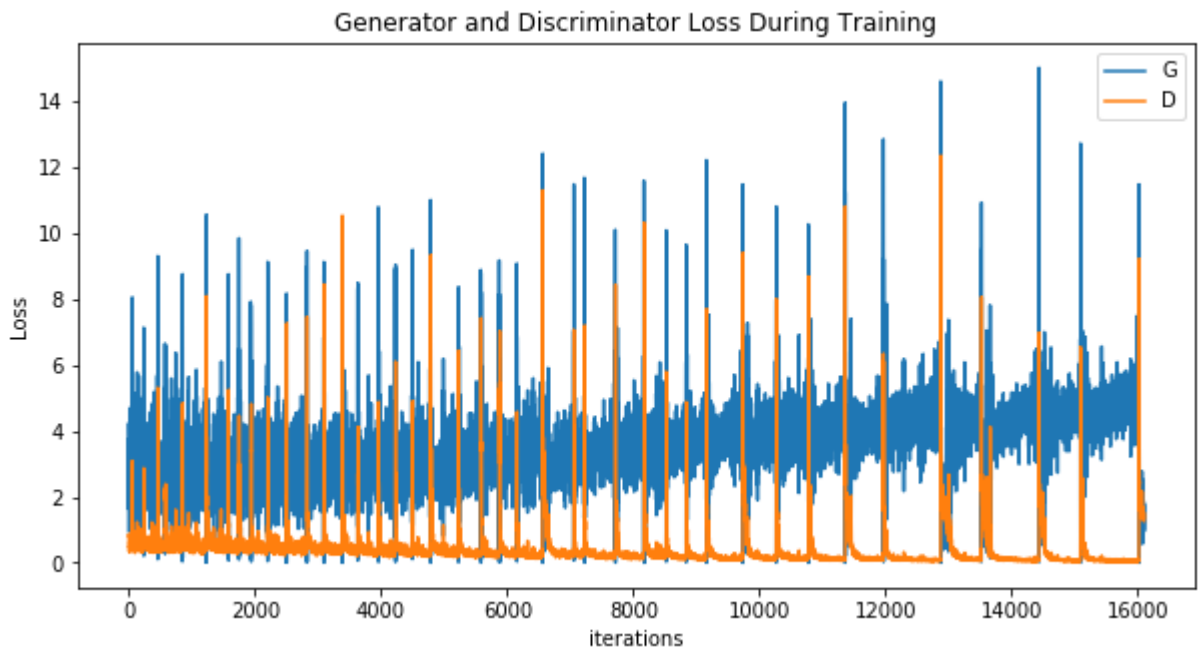
Results

Finally, lets check out how we did. Here, we will look at three different results. First, we will see how D changed during training. Second, we will visualize G's output on the fixed_noise batch for every epoch will look at a batch of real data next to a batch of fake data from G.

Loss versus training iteration

Below is a plot of D & G's losses versus training iterations.

```
plt.figure(figsize=(10,5))
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()
```

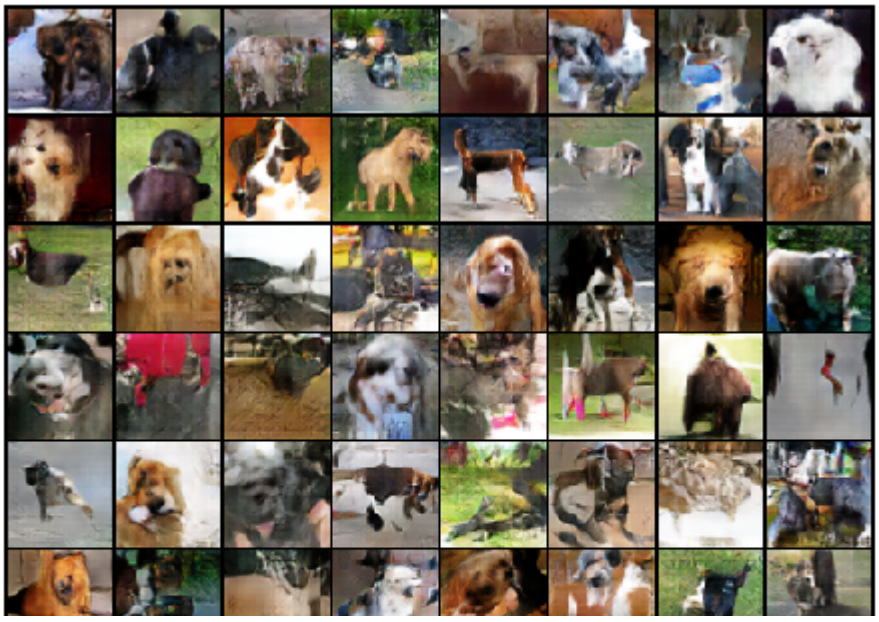


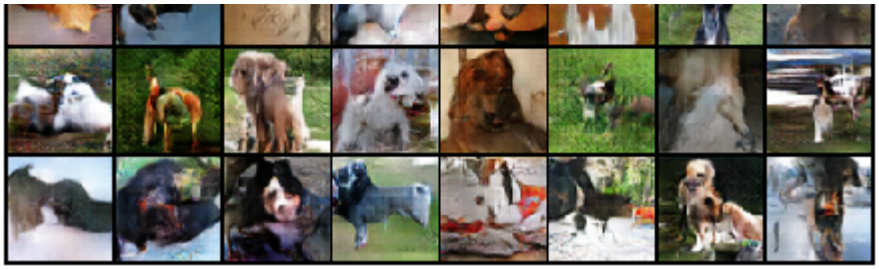
Visualization of G's progression

Remember how we saved the generator's output on the fixed_noise batch after every epoch of trainin visualize the training progression of G with an animation. Press the play button to start the animation

```
#%%capture
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())
```

Once Loop Reflect





Real Images vs. Fake Images

Finally, lets take a look at some real images and fake images side by side.

```
# Grab a batch of real images from the dataloader
real_batch = next(iter(dataloader))

# Plot the real images
plt.figure(figsize=(15,15))
plt.subplot(1,2,1)
plt.axis("off")
plt.title("Real Images")
plt.imshow(np.transpose(vutils.make_grid(real_batch[0].to(device)[:64], padding=5, normalize=True).c

# Plot the fake images from the last epoch
plt.subplot(1,2,2)
plt.axis("off")
plt.title("Fake Images")
plt.imshow(np.transpose(img_list[-4],(1,2,0)))
plt.show()
```

Where to Go Next

We have reached the end of our journey, but there are several places you could go from here. You cou

- Train for longer to see how good the results get
- Modify this model to take a different dataset and possibly change the size of the images and th architecture
- Check out some other cool GAN projects `here <https://github.com/nashory/gans-awesome-a`
- Create GANs that generate `music <https://deepmind.com/blog/wavenet-generative-model-ra`