dlnd_face_generation0

December 4, 2021

1 Face Generation

In this project, you'll define and train a DCGAN on a dataset of faces. Your goal is to get a generator network to generate *new* images of faces that look as realistic as possible!

The project will be broken down into a series of tasks from **loading in data to defining and training adversarial networks**. At the end of the notebook, you'll be able to visualize the results of your trained Generator to see how it performs; your generated samples should look like fairly realistic faces with small amounts of noise.

1.0.1 Get the Data

You'll be using the CelebFaces Attributes Dataset (CelebA) to train your adversarial networks.

This dataset is more complex than the number datasets (like MNIST or SVHN) you've been working with, and so, you should prepare to define deeper networks and train them for a longer time to get good results. It is suggested that you utilize a GPU for training.

1.0.2 Pre-processed Data

Since the project's main focus is on building the GANs, we've done *some* of the pre-processing for you. Each of the CelebA images has been cropped to remove parts of the image that don't include a face, then resized down to 64x64x3 NumPy images. Some sample data is show below.

If you are working locally, you can download this data by clicking here

This is a zip file that you'll need to extract in the home directory of this notebook for further loading and processing. After extracting the data, you should be left with a directory of data `processed_celeba_small/`

```
In [31]: # can comment out after executing
         #!unzip processed_celeba_small.zip

In [32]: data_dir = 'processed_celeba_small/'

         """
         DON'T MODIFY ANYTHING IN THIS CELL
         """
         import pickle as pkl
         import matplotlib.pyplot as plt
```

```
import numpy as np
import problem_unittests as tests
#import helper

%matplotlib inline
```

1.1 Visualize the CelebA Data

The CelebA dataset contains over 200,000 celebrity images with annotations. Since you're going to be generating faces, you won't need the annotations, you'll only need the images. Note that these are color images with 3 color channels (RGB) each.

1.1.1 Pre-process and Load the Data

Since the project's main focus is on building the GANs, we've done *some* of the pre-processing for you. Each of the CelebA images has been cropped to remove parts of the image that don't include a face, then resized down to 64x64x3 NumPy images. This *pre-processed* dataset is a smaller subset of the very large CelebA data.

> There are a few other steps that you'll need to **transform** this data and create a **DataLoader**.

Exercise: Complete the following `get_dataloader` function, such that it satisfies these requirements:

- Your images should be square, Tensor images of size `image_size x image_size` in the x and y dimension.
- Your function should return a DataLoader that shuffles and batches these Tensor images.

ImageFolder To create a dataset given a directory of images, it's recommended that you use PyTorch's ImageFolder wrapper, with a root directory `processed_celeba_small/` and data transformation passed in.

```
In [33]: # necessary imports
         import torch
         from torchvision import datasets
         from torchvision import transforms
```

```
In [34]: def get_dataloader(batch_size, image_size, data_dir='processed_celeba_small/'):
             """
             Batch the neural network data using DataLoader
             :param batch_size: The size of each batch; the number of images in a batch
             :param img_size: The square size of the image data (x, y)
             :param data_dir: Directory where image data is located
             :return: DataLoader with batched data
             """

             # TODO: Implement function and return a dataloader
             transform = transforms.Compose([transforms.Resize(image_size),transforms.ToTensor()
```

```
    dataset = datasets.ImageFolder(data_dir, transform)

    dataloader = torch.utils.data.DataLoader(dataset=dataset, batch_size=batch_size, sh

    return dataloader
```

1.2 Create a DataLoader

Exercise: Create a DataLoader `celeba_train_loader` with appropriate hyperparameters. Call the above function and create a dataloader to view images. * You can decide on any reasonable `batch_size` parameter * Your `image_size` **must be** `32`. Resizing the data to a smaller size will make for faster training, while still creating convincing images of faces!

In [35]:
```
# Define function hyperparameters
batch_size = 20
img_size = 32

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
# Call your function and get a dataloader
celeba_train_loader = get_dataloader(batch_size, img_size)
```

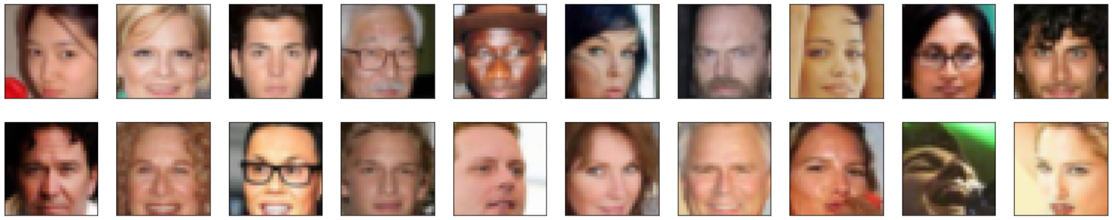
Next, you can view some images! You should seen square images of somewhat-centered faces.

Note: You'll need to convert the Tensor images into a NumPy type and transpose the dimensions to correctly display an image, suggested `imshow` code is below, but it may not be perfect.

In [36]:
```
# helper display function
def imshow(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
# obtain one batch of training images
dataiter = iter(celeba_train_loader)
images, _ = dataiter.next() # _ for no labels

# plot the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(20, 4))
plot_size=20
for idx in np.arange(plot_size):
    ax = fig.add_subplot(2, plot_size/2, idx+1, xticks=[], yticks=[])
    imshow(images[idx])
```



Exercise: Pre-process your image data and scale it to a pixel range of -1 to 1 You need to do a bit of pre-processing; you know that the output of a `tanh` activated generator will contain pixel values in a range from -1 to 1, and so, we need to rescale our training images to a range of -1 to 1. (Right now, they are in a range from 0-1.)

In [37]:
```
# TODO: Complete the scale function
def scale(x, feature_range=(-1, 1)):
    ''' Scale takes in an image x and returns that image, scaled
       with a feature_range of pixel values from -1 to 1.
       This function assumes that the input x is already scaled from 0-1.'''
    # assume x is scaled to (0, 1)
    # scale to feature_range and return scaled x

    min , max = feature_range

    x = x * (max-min) + min

    return x
```

In [38]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
# check scaled range
# should be close to -1 to 1
img = images[0]
scaled_img = scale(img)

print('Min: ', scaled_img.min())
print('Max: ', scaled_img.max())
```

```
Min:  tensor(-1.)
Max:  tensor(0.9373)
```

---

2 Define the Model

A GAN is comprised of two adversarial networks, a discriminator and a generator.

2.1 Discriminator

Your first task will be to define the discriminator. This is a convolutional classifier like you've built before, only without any maxpooling layers. To deal with this complex data, it's suggested you use a deep network with **normalization**. You are also allowed to create any helper functions that may be useful.

Exercise: Complete the Discriminator class

- The inputs to the discriminator are 32x32x3 tensor images
- The output should be a single value that will indicate whether a given image is real or fake

In [39]:
```
import torch.nn as nn
import torch.nn.functional as F
```

2.2 Helper Function

In [40]:
```
# helper conv function
def conv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True):
    """Creates a convolutional layer, with optional batch normalization.
    """
    layers = []
    conv_layer = nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                           kernel_size=kernel_size, stride=stride, padding=padding, bia

    layers.append(conv_layer)

    if batch_norm:
        layers.append(nn.BatchNorm2d(out_channels))
    return nn.Sequential(*layers)


# helper deconv function
def deconv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True
    """Creates a transpose convolutional layer, with optional batch normalization.
    """
    layers = []
    # append transpose conv layer
    layers.append(nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride, pa
    # optional batch norm layer
    if batch_norm:
        layers.append(nn.BatchNorm2d(out_channels))
    return nn.Sequential(*layers)
```

In [41]:
```
class Discriminator(nn.Module):

    def __init__(self, conv_dim):
        """
        Initialize the Discriminator Module
```

```python
        :param conv_dim: The depth of the first convolutional layer
        """
        super(Discriminator, self).__init__()

        # complete init function
        self.conv_dim = conv_dim

        self.conv1 = nn.Conv2d(3, conv_dim, stride=2, padding=1, bias=False, kernel_siz
        self.batch_norm1 = nn.BatchNorm2d(conv_dim)

        self.conv2 = nn.Conv2d(conv_dim, conv_dim*2, stride=2, padding=1, bias=False, k
        self.batch_norm2 = nn.BatchNorm2d(conv_dim*2)

        self.conv3 = nn.Conv2d(conv_dim*2, conv_dim*4, stride=2, padding=1, bias=False,
        self.batch_norm3 = nn.BatchNorm2d(conv_dim*4)

        self.conv4 = nn.Conv2d(conv_dim*4, conv_dim*8, stride=2, padding=1, bias=False,
        self.batch_norm4 = nn.BatchNorm2d(conv_dim*8)

        self.conv5 = nn.Conv2d(conv_dim*8, conv_dim*16, stride=2, padding=1, bias=False
        self.fc = nn.Linear(conv_dim*4*4, 1)



    def forward(self, x):
        """
        Forward propagation of the neural network
        :param x: The input to the neural network
        :return: Discriminator logits; the output of the neural network
        """
        # define feedforward behavior

        x = F.leaky_relu(self.batch_norm1(self.conv1(x)), 0.2)

        x = F.leaky_relu(self.batch_norm2(self.conv2(x)), 0.2)

        x = F.leaky_relu(self.batch_norm3(self.conv3(x)), 0.2)

        x = F.leaky_relu(self.batch_norm4(self.conv4(x)), 0.2)

        x = self.conv5(x)

        x = x.view(-1, self.conv_dim*4*4)

        # output layer
        x = F.sigmoid(self.fc(x))

        return x
```

```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_discriminator(Discriminator)
```

```
Tests Passed
```

2.3 Generator

The generator should upsample an input and generate a *new* image of the same size as our training data `32x32x3`. This should be mostly transpose convolutional layers with normalization applied to the outputs.

Exercise: Complete the Generator class

- The inputs to the generator are vectors of some length `z_size`
- The output should be a image of shape `32x32x3`

In [42]:
```
class Generator(nn.Module):

    def __init__(self, z_size, conv_dim):
        """
        Initialize the Generator Module
        :param z_size: The length of the input latent vector, z
        :param conv_dim: The depth of the inputs to the *last* transpose convolutional
        """
        super(Generator, self).__init__()

        # complete init function

        self.conv_dim = conv_dim

        self.t_conv1 = nn.ConvTranspose2d(conv_dim, conv_dim*8, stride=2, padding=1, bi
        self.batch_norm1 = nn.BatchNorm2d(conv_dim*8)

        self.t_conv2 = nn.ConvTranspose2d(conv_dim*8, conv_dim*4, stride=2, padding=1,
        self.batch_norm2 = nn.BatchNorm2d(conv_dim*4)

        self.t_conv3 = nn.ConvTranspose2d(conv_dim*4, conv_dim*2, stride=2, padding=1,
        self.batch_norm3 = nn.BatchNorm2d(conv_dim*2)

        self.t_conv4 = nn.ConvTranspose2d(conv_dim*2, 3, stride=2, padding=1, bias=Fals

        self.fc = nn.Linear(z_size, conv_dim*4)
```

```
    def forward(self, x):
        """
        Forward propagation of the neural network
        :param x: The input to the neural network
        :return: A 32x32x3 Tensor image as output
        """
        # define feedforward behavior

        batch_s = x.shape[0]

        x = self.fc(x)

        x = x.view(batch_s, self.conv_dim, 2, 2)

        x = F.relu(self.batch_norm1(self.t_conv1(x)))

        x = F.relu(self.batch_norm2(self.t_conv2(x)))

        x = F.relu(self.batch_norm3(self.t_conv3(x)))

        x = self.t_conv4(x)

        # output layer
        x = F.tanh(x)


        return x

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_generator(Generator)
```

```
Tests Passed
```

2.4 Initialize the weights of your networks

To help your models converge, you should initialize the weights of the convolutional and linear layers in your model. From reading the original DCGAN paper, they say: > All weights were initialized from a zero-centered Normal distribution with standard deviation 0.02.

So, your next task will be to define a weight initialization function that does just this!

You can refer back to the lesson on weight initialization or even consult existing model code, such as that from the `networks.py` file in CycleGAN Github repository to help you complete this function.

Exercise: Complete the weight initialization function

- This should initialize only **convolutional** and **linear** layers
- Initialize the weights to a normal distribution, centered around 0, with a standard deviation of 0.02.
- The bias terms, if they exist, may be left alone or set to 0.

In [43]:
```
def weights_init_normal(m):
    """
    Applies initial weights to certain layers in a model .
    The weights are taken from a normal distribution
    with mean = 0, std dev = 0.02.
    :param m: A module or layer in a network
    """
    # classname will be something like:
    # `Conv`, `BatchNorm2d`, `Linear`, etc.
    classname = m.__class__.__name__

    # TODO: Apply initial weights to convolutional and linear layers

    if  (classname.find('Conv') != -1 or classname.find('Linear') != -1) and hasattr(m,
        nn.init.normal_(m.weight.data, 0.0, 0.02)
```

2.5 Build complete network

Define your models' hyperparameters and instantiate the discriminator and generator from the classes defined above. Make sure you've passed in the correct input arguments.

In [44]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
def build_network(d_conv_dim, g_conv_dim, z_size):
    # define discriminator and generator
    D = Discriminator(d_conv_dim)
    G = Generator(z_size=z_size, conv_dim=g_conv_dim)

    # initialize model weights
    D.apply(weights_init_normal)
    G.apply(weights_init_normal)

    print(D)
    print()
    print(G)

    return D, G
```

Exercise: Define model hyperparameters

In [45]:
```
# Define model hyperparams
d_conv_dim = 32
```

```
g_conv_dim = 32
z_size = 100

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
D, G = build_network(d_conv_dim, g_conv_dim, z_size)
```

```
Discriminator(
  (conv1): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (batch_norm1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (batch_norm2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (batch_norm3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
  (conv4): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (batch_norm4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
  (conv5): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (fc): Linear(in_features=512, out_features=1, bias=True)
)

Generator(
  (t_conv1): ConvTranspose2d(32, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=Fa
  (batch_norm1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
  (t_conv2): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=F
  (batch_norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
  (t_conv3): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=Fa
  (batch_norm3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (t_conv4): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=Fals
  (fc): Linear(in_features=100, out_features=128, bias=True)
)
```

2.5.1 Training on GPU

Check if you can train on GPU. Here, we'll set this as a boolean variable `train_on_gpu`. Later, you'll be responsible for making sure that >* Models, * Model inputs, and * Loss function arguments

Are moved to GPU, where appropriate.

In [46]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import torch

# Check for a GPU
train_on_gpu = torch.cuda.is_available()
if not train_on_gpu:
```

```
            print('No GPU found. Please use a GPU to train your neural network.')
        else:
            print('Training on GPU!')
```

```
Training on GPU!
```

2.6 Discriminator and Generator Losses

Now we need to calculate the losses for both types of adversarial networks.

2.6.1 Discriminator Losses

- For the discriminator, the total loss is the sum of the losses for real and fake images, `d_loss = d_real_loss + d_fake_loss`.
- Remember that we want the discriminator to output 1 for real images and 0 for fake images, so we need to set up the losses to reflect that.

2.6.2 Generator Loss

The generator loss will look similar only with flipped labels. The generator's goal is to get the discriminator to *think* its generated images are *real*.

Exercise: Complete real and fake loss functions **You may choose to use either cross entropy or a least squares error loss to complete the following `real_loss` and `fake_loss` functions.**

```
In [47]: import random
         def real_loss(D_out, smooth=False):

             batch_size = D_out.size(0)

             labels = torch.ones(batch_size)*0.9

             if train_on_gpu:
                 labels = labels.cuda()


             criterion = nn.BCELoss()

             loss = criterion(D_out.squeeze(), labels)

             return loss

         def fake_loss(D_out):

             batch_size = D_out.size(0)
```

```
        labels = torch.zeros(batch_size)

        if train_on_gpu:
            labels = labels.cuda()

        criterion = nn.BCELoss()

        loss = criterion(D_out.squeeze(), labels)

        return loss
```

2.7 Optimizers

Exercise: Define optimizers for your Discriminator (D) and Generator (G) Define optimizers for your models with appropriate hyperparameters.

```
In [48]: import torch.optim as optim

         d_optimizer = optim.Adam(D.parameters(), lr=0.0005, betas=(0.1, 0.999))
         g_optimizer = optim.Adam(G.parameters(), lr=0.0005, betas=(0.1, 0.999))
```

2.8 Training

Training will involve alternating between training the discriminator and the generator. You'll use your functions `real_loss` and `fake_loss` to help you calculate the discriminator losses.

- You should train the discriminator by alternating on real and fake images
- Then the generator, which tries to trick the discriminator and should have an opposing loss function

Saving Samples You've been given some code to print out some loss statistics and save some generated "fake" samples.

Exercise: Complete the training function Keep in mind that, if you've moved your models to GPU, you'll also have to move any model inputs to GPU.

```
In [49]: def train(D, G, n_epochs, print_every=50):
             '''Trains adversarial networks for some number of epochs
                param, D: the discriminator network
                param, G: the generator network
                param, n_epochs: number of epochs to train for
                param, print_every: when to print and record the models' losses
                return: D and G losses'''

             # move models to GPU
             if train_on_gpu:
```

```python
    D.cuda()
    G.cuda()

# keep track of loss and generated, "fake" samples
samples = []
losses = []

# Get some fixed data for sampling. These are images that are held
# constant throughout training, and allow us to inspect the model's performance
sample_size=16
fixed_z = np.random.uniform(-1, 1, size=(sample_size, z_size))
fixed_z = torch.from_numpy(fixed_z).float()
# move z to GPU if available
if train_on_gpu:
    fixed_z = fixed_z.cuda()

# epoch training loop
for epoch in range(n_epochs):

    # batch training loop
    for batch_i, (real_images, _) in enumerate(celeba_train_loader):

        batch_size = real_images.size(0)
        real_images = scale(real_images)

        # ===============================================
        #         YOUR CODE HERE: TRAIN THE NETWORKS
        # ===============================================
        # 1. Train the discriminator on real and fake images
        if train_on_gpu:
            real_images = real_images.cuda()

        d_optimizer.zero_grad()

        D_real = D(real_images)

        d_real_loss = real_loss(D_real)

        z_flex = np.random.uniform(-1, 1, size=(batch_size, z_size))

        z_flex = torch.from_numpy(z_flex).float()

        if train_on_gpu:
            z_flex = z_flex.cuda()

        fake_images = G(z_flex)

        D_fake = D(fake_images)
```

```python
d_fake_loss = fake_loss(D_fake)

d_loss = d_real_loss + d_fake_loss

d_loss.backward()

d_optimizer.step()

# d_loss =

# 2. Train the generator with an adversarial loss

g_optimizer.zero_grad()

z_flex = np.random.uniform(-1, 1, size=(batch_size, z_size))

z_flex = torch.from_numpy(z_flex).float()

if train_on_gpu:
    z_flex = z_flex.cuda()

fake_images = G(z_flex)

D_fake = D(fake_images)

g_loss = real_loss(D_fake, True)

g_loss.backward()

g_optimizer.step()

# g_loss =


# ===============================================
#               END OF YOUR CODE
# ===============================================

# Print some loss stats
if batch_i % print_every == 0:
    # append discriminator loss and generator loss
    losses.append((d_loss.item(), g_loss.item()))
    # print discriminator and generator loss
    print('Epoch [{:5d}/{:5d}] | d_loss: {:6.4f} | g_loss: {:6.4f}'.format(
            epoch+1, n_epochs, d_loss.item(), g_loss.item()))
```

```
            ## AFTER EACH EPOCH##
            # this code assumes your generator is named G, feel free to change the name
            # generate and save sample, fake images
            G.eval() # for generating samples
            samples_z = G(fixed_z)
            samples.append(samples_z)
            G.train() # back to training mode

        # Save training generator samples
        with open('train_samples.pkl', 'wb') as f:
            pkl.dump(samples, f)

        # finally return losses
        return losses
```

Set your number of training epochs and train your GAN!

```
In [50]: # set number of epochs
         n_epochs = 10


         """
         DON'T MODIFY ANYTHING IN THIS CELL
         """
         # call training function
         losses = train(D, G, n_epochs=n_epochs)
```

```
Epoch [    1/   10] | d_loss: 1.3672 | g_loss: 1.0886
Epoch [    1/   10] | d_loss: 1.3421 | g_loss: 0.8818
Epoch [    1/   10] | d_loss: 1.3073 | g_loss: 1.1071
Epoch [    1/   10] | d_loss: 1.2262 | g_loss: 1.0877
Epoch [    1/   10] | d_loss: 1.3660 | g_loss: 0.9414
Epoch [    1/   10] | d_loss: 1.2818 | g_loss: 1.0140
Epoch [    1/   10] | d_loss: 1.4973 | g_loss: 1.3645
Epoch [    1/   10] | d_loss: 1.2064 | g_loss: 1.1135
Epoch [    1/   10] | d_loss: 1.3753 | g_loss: 1.2552
Epoch [    1/   10] | d_loss: 1.3895 | g_loss: 0.9882
Epoch [    1/   10] | d_loss: 1.3851 | g_loss: 1.1609
Epoch [    1/   10] | d_loss: 1.2684 | g_loss: 0.6966
Epoch [    1/   10] | d_loss: 1.3501 | g_loss: 1.1292
Epoch [    1/   10] | d_loss: 1.4714 | g_loss: 0.8745
Epoch [    1/   10] | d_loss: 1.3516 | g_loss: 0.8411
Epoch [    1/   10] | d_loss: 1.2655 | g_loss: 1.2381
Epoch [    1/   10] | d_loss: 1.2893 | g_loss: 1.3190
Epoch [    1/   10] | d_loss: 1.3110 | g_loss: 1.0625
Epoch [    1/   10] | d_loss: 1.3922 | g_loss: 0.9080
Epoch [    1/   10] | d_loss: 1.3250 | g_loss: 1.1447
Epoch [    1/   10] | d_loss: 1.4751 | g_loss: 1.4620
```

```
Epoch [    1/   10] | d_loss: 1.1810 | g_loss: 1.0830
Epoch [    1/   10] | d_loss: 1.4004 | g_loss: 1.0486
Epoch [    1/   10] | d_loss: 1.5283 | g_loss: 0.7456
Epoch [    1/   10] | d_loss: 1.2398 | g_loss: 0.9128
Epoch [    1/   10] | d_loss: 1.2065 | g_loss: 1.1173
Epoch [    1/   10] | d_loss: 1.1402 | g_loss: 1.1840
Epoch [    1/   10] | d_loss: 1.2978 | g_loss: 1.0721
Epoch [    1/   10] | d_loss: 1.2298 | g_loss: 1.1749
Epoch [    1/   10] | d_loss: 1.2470 | g_loss: 1.1691
Epoch [    1/   10] | d_loss: 1.4999 | g_loss: 1.1199
Epoch [    1/   10] | d_loss: 1.3228 | g_loss: 0.8521
Epoch [    1/   10] | d_loss: 1.3046 | g_loss: 0.9676
Epoch [    1/   10] | d_loss: 1.2502 | g_loss: 1.2177
Epoch [    1/   10] | d_loss: 1.3350 | g_loss: 0.9156
Epoch [    1/   10] | d_loss: 1.2001 | g_loss: 0.8926
Epoch [    1/   10] | d_loss: 1.2576 | g_loss: 0.8891
Epoch [    1/   10] | d_loss: 1.3140 | g_loss: 1.7108
Epoch [    1/   10] | d_loss: 1.4057 | g_loss: 1.1084
Epoch [    1/   10] | d_loss: 1.3621 | g_loss: 0.9693
Epoch [    1/   10] | d_loss: 1.4201 | g_loss: 1.1529
Epoch [    1/   10] | d_loss: 1.2713 | g_loss: 1.7139
Epoch [    1/   10] | d_loss: 1.2314 | g_loss: 1.2424
Epoch [    1/   10] | d_loss: 1.3641 | g_loss: 1.3670
Epoch [    1/   10] | d_loss: 1.3636 | g_loss: 0.8177
Epoch [    1/   10] | d_loss: 1.3175 | g_loss: 0.9847
Epoch [    1/   10] | d_loss: 1.3964 | g_loss: 1.4086
Epoch [    1/   10] | d_loss: 1.1211 | g_loss: 1.1305
Epoch [    1/   10] | d_loss: 1.1421 | g_loss: 1.0860
Epoch [    1/   10] | d_loss: 1.0557 | g_loss: 1.0966
Epoch [    1/   10] | d_loss: 1.1072 | g_loss: 1.0076
Epoch [    1/   10] | d_loss: 1.1771 | g_loss: 1.2768
Epoch [    1/   10] | d_loss: 1.3369 | g_loss: 0.8655
Epoch [    1/   10] | d_loss: 1.3186 | g_loss: 1.0367
Epoch [    1/   10] | d_loss: 1.2296 | g_loss: 0.8887
Epoch [    1/   10] | d_loss: 1.3247 | g_loss: 1.0291
Epoch [    1/   10] | d_loss: 1.1041 | g_loss: 1.2561
Epoch [    1/   10] | d_loss: 1.3483 | g_loss: 1.0542
Epoch [    1/   10] | d_loss: 1.4174 | g_loss: 1.1418
Epoch [    1/   10] | d_loss: 1.3411 | g_loss: 0.8333
Epoch [    1/   10] | d_loss: 1.2455 | g_loss: 1.7522
Epoch [    1/   10] | d_loss: 1.1916 | g_loss: 1.6224
Epoch [    1/   10] | d_loss: 1.3306 | g_loss: 0.9382
Epoch [    1/   10] | d_loss: 1.1401 | g_loss: 1.3180
Epoch [    1/   10] | d_loss: 1.2320 | g_loss: 0.8547
Epoch [    1/   10] | d_loss: 1.0980 | g_loss: 1.4262
Epoch [    1/   10] | d_loss: 1.2479 | g_loss: 1.5171
Epoch [    1/   10] | d_loss: 1.3520 | g_loss: 1.3944
Epoch [    1/   10] | d_loss: 1.0850 | g_loss: 1.3813
```

```
Epoch [    1/   10] | d_loss: 0.9317 | g_loss: 1.1432
Epoch [    1/   10] | d_loss: 1.2338 | g_loss: 1.3583
Epoch [    1/   10] | d_loss: 1.3027 | g_loss: 1.6450
Epoch [    1/   10] | d_loss: 1.1465 | g_loss: 1.0541
Epoch [    1/   10] | d_loss: 1.3121 | g_loss: 1.2595
Epoch [    1/   10] | d_loss: 1.2587 | g_loss: 1.3023
Epoch [    1/   10] | d_loss: 1.6852 | g_loss: 0.9090
Epoch [    1/   10] | d_loss: 1.1302 | g_loss: 1.1788
Epoch [    1/   10] | d_loss: 1.2020 | g_loss: 0.8436
Epoch [    1/   10] | d_loss: 1.2754 | g_loss: 0.7723
Epoch [    1/   10] | d_loss: 1.1880 | g_loss: 0.5999
Epoch [    1/   10] | d_loss: 1.1528 | g_loss: 1.1876
Epoch [    1/   10] | d_loss: 1.2460 | g_loss: 1.1575
Epoch [    1/   10] | d_loss: 1.3811 | g_loss: 1.0860
Epoch [    1/   10] | d_loss: 1.1452 | g_loss: 1.1577
Epoch [    1/   10] | d_loss: 1.2473 | g_loss: 1.1288
Epoch [    1/   10] | d_loss: 1.2628 | g_loss: 0.9991
Epoch [    1/   10] | d_loss: 1.2340 | g_loss: 1.6049
Epoch [    1/   10] | d_loss: 1.4918 | g_loss: 0.7308
Epoch [    1/   10] | d_loss: 1.3216 | g_loss: 1.3419
Epoch [    1/   10] | d_loss: 1.3529 | g_loss: 1.3533
Epoch [    2/   10] | d_loss: 1.1411 | g_loss: 1.5131
Epoch [    2/   10] | d_loss: 1.3307 | g_loss: 1.1558
Epoch [    2/   10] | d_loss: 1.1801 | g_loss: 1.3273
Epoch [    2/   10] | d_loss: 1.4406 | g_loss: 0.8971
Epoch [    2/   10] | d_loss: 1.2953 | g_loss: 0.9075
Epoch [    2/   10] | d_loss: 1.1273 | g_loss: 1.0304
Epoch [    2/   10] | d_loss: 1.4566 | g_loss: 0.8378
Epoch [    2/   10] | d_loss: 1.2443 | g_loss: 1.1187
Epoch [    2/   10] | d_loss: 1.3641 | g_loss: 0.7740
Epoch [    2/   10] | d_loss: 1.3697 | g_loss: 1.6206
Epoch [    2/   10] | d_loss: 1.2009 | g_loss: 1.2932
Epoch [    2/   10] | d_loss: 1.2054 | g_loss: 0.9936
Epoch [    2/   10] | d_loss: 1.0558 | g_loss: 2.0262
Epoch [    2/   10] | d_loss: 1.0004 | g_loss: 1.1284
Epoch [    2/   10] | d_loss: 1.1736 | g_loss: 1.8629
Epoch [    2/   10] | d_loss: 1.1254 | g_loss: 1.1790
Epoch [    2/   10] | d_loss: 1.0286 | g_loss: 1.2118
Epoch [    2/   10] | d_loss: 1.4907 | g_loss: 0.9720
Epoch [    2/   10] | d_loss: 1.1612 | g_loss: 1.0259
Epoch [    2/   10] | d_loss: 1.1232 | g_loss: 1.0101
Epoch [    2/   10] | d_loss: 1.1769 | g_loss: 0.8116
Epoch [    2/   10] | d_loss: 1.0048 | g_loss: 1.2042
Epoch [    2/   10] | d_loss: 1.3592 | g_loss: 1.1877
Epoch [    2/   10] | d_loss: 1.2660 | g_loss: 1.1102
Epoch [    2/   10] | d_loss: 0.9993 | g_loss: 1.7251
Epoch [    2/   10] | d_loss: 1.2107 | g_loss: 1.2333
Epoch [    2/   10] | d_loss: 1.1531 | g_loss: 0.9990
```

```
Epoch [    2/   10] | d_loss: 0.9075 | g_loss: 1.2403
Epoch [    2/   10] | d_loss: 1.2554 | g_loss: 1.1904
Epoch [    2/   10] | d_loss: 1.0282 | g_loss: 1.0729
Epoch [    2/   10] | d_loss: 1.1336 | g_loss: 1.0575
Epoch [    2/   10] | d_loss: 0.9535 | g_loss: 1.3588
Epoch [    2/   10] | d_loss: 1.1825 | g_loss: 1.5823
Epoch [    2/   10] | d_loss: 1.5623 | g_loss: 0.9093
Epoch [    2/   10] | d_loss: 1.1201 | g_loss: 0.9919
Epoch [    2/   10] | d_loss: 1.2705 | g_loss: 2.1539
Epoch [    2/   10] | d_loss: 1.1784 | g_loss: 1.3464
Epoch [    2/   10] | d_loss: 1.1960 | g_loss: 0.8887
Epoch [    2/   10] | d_loss: 1.1347 | g_loss: 1.2501
Epoch [    2/   10] | d_loss: 1.3046 | g_loss: 1.8206
Epoch [    2/   10] | d_loss: 1.0166 | g_loss: 1.6089
Epoch [    2/   10] | d_loss: 1.4041 | g_loss: 1.7583
Epoch [    2/   10] | d_loss: 1.2788 | g_loss: 1.1545
Epoch [    2/   10] | d_loss: 1.1798 | g_loss: 1.6525
Epoch [    2/   10] | d_loss: 1.1333 | g_loss: 1.4349
Epoch [    2/   10] | d_loss: 1.2359 | g_loss: 0.7339
Epoch [    2/   10] | d_loss: 1.2618 | g_loss: 1.7236
Epoch [    2/   10] | d_loss: 1.1358 | g_loss: 1.2800
Epoch [    2/   10] | d_loss: 1.1729 | g_loss: 0.9888
Epoch [    2/   10] | d_loss: 1.2793 | g_loss: 1.2015
Epoch [    2/   10] | d_loss: 1.0723 | g_loss: 1.2559
Epoch [    2/   10] | d_loss: 1.1544 | g_loss: 0.9833
Epoch [    2/   10] | d_loss: 1.1143 | g_loss: 0.7342
Epoch [    2/   10] | d_loss: 1.4485 | g_loss: 2.8123
Epoch [    2/   10] | d_loss: 1.3108 | g_loss: 1.5610
Epoch [    2/   10] | d_loss: 1.3001 | g_loss: 1.4948
Epoch [    2/   10] | d_loss: 1.0258 | g_loss: 1.0540
Epoch [    2/   10] | d_loss: 1.0193 | g_loss: 1.1398
Epoch [    2/   10] | d_loss: 1.4206 | g_loss: 0.7746
Epoch [    2/   10] | d_loss: 1.0473 | g_loss: 1.1509
Epoch [    2/   10] | d_loss: 1.2610 | g_loss: 1.3287
Epoch [    2/   10] | d_loss: 1.3150 | g_loss: 1.3756
Epoch [    2/   10] | d_loss: 0.9603 | g_loss: 1.1721
Epoch [    2/   10] | d_loss: 1.0668 | g_loss: 1.8175
Epoch [    2/   10] | d_loss: 1.1023 | g_loss: 0.8590
Epoch [    2/   10] | d_loss: 1.2278 | g_loss: 1.0417
Epoch [    2/   10] | d_loss: 0.9903 | g_loss: 1.7132
Epoch [    2/   10] | d_loss: 0.9511 | g_loss: 1.2121
Epoch [    2/   10] | d_loss: 1.2575 | g_loss: 1.4350
Epoch [    2/   10] | d_loss: 1.2326 | g_loss: 0.9766
Epoch [    2/   10] | d_loss: 1.2038 | g_loss: 1.2484
Epoch [    2/   10] | d_loss: 1.6012 | g_loss: 0.8423
Epoch [    2/   10] | d_loss: 1.0029 | g_loss: 1.3464
Epoch [    2/   10] | d_loss: 1.1588 | g_loss: 1.7149
Epoch [    2/   10] | d_loss: 1.0459 | g_loss: 1.3538
```

```
Epoch [    2/   10] | d_loss: 1.0022 | g_loss: 1.6057
Epoch [    2/   10] | d_loss: 1.1957 | g_loss: 1.3472
Epoch [    2/   10] | d_loss: 0.9552 | g_loss: 1.9001
Epoch [    2/   10] | d_loss: 0.8936 | g_loss: 1.5590
Epoch [    2/   10] | d_loss: 1.0533 | g_loss: 1.0887
Epoch [    2/   10] | d_loss: 1.2081 | g_loss: 0.8361
Epoch [    2/   10] | d_loss: 1.2661 | g_loss: 1.8617
Epoch [    2/   10] | d_loss: 1.2017 | g_loss: 0.9899
Epoch [    2/   10] | d_loss: 0.9712 | g_loss: 1.2098
Epoch [    2/   10] | d_loss: 1.1134 | g_loss: 0.9540
Epoch [    2/   10] | d_loss: 1.1393 | g_loss: 1.0611
Epoch [    2/   10] | d_loss: 0.8355 | g_loss: 1.1846
Epoch [    2/   10] | d_loss: 1.3635 | g_loss: 0.8841
Epoch [    2/   10] | d_loss: 1.0337 | g_loss: 1.0671
Epoch [    2/   10] | d_loss: 1.4958 | g_loss: 0.9028
Epoch [    3/   10] | d_loss: 0.9749 | g_loss: 1.4954
Epoch [    3/   10] | d_loss: 0.8502 | g_loss: 1.1088
Epoch [    3/   10] | d_loss: 0.9796 | g_loss: 1.8277
Epoch [    3/   10] | d_loss: 1.0112 | g_loss: 1.6665
Epoch [    3/   10] | d_loss: 1.5597 | g_loss: 2.4891
Epoch [    3/   10] | d_loss: 1.1545 | g_loss: 1.6196
Epoch [    3/   10] | d_loss: 1.3250 | g_loss: 2.2937
Epoch [    3/   10] | d_loss: 1.3831 | g_loss: 1.4257
Epoch [    3/   10] | d_loss: 1.1790 | g_loss: 1.3857
Epoch [    3/   10] | d_loss: 1.1193 | g_loss: 1.0749
Epoch [    3/   10] | d_loss: 1.0201 | g_loss: 0.9541
Epoch [    3/   10] | d_loss: 1.3519 | g_loss: 1.1638
Epoch [    3/   10] | d_loss: 0.9396 | g_loss: 1.0898
Epoch [    3/   10] | d_loss: 1.2277 | g_loss: 0.8996
Epoch [    3/   10] | d_loss: 0.9042 | g_loss: 1.7941
Epoch [    3/   10] | d_loss: 0.9506 | g_loss: 1.2510
Epoch [    3/   10] | d_loss: 1.2091 | g_loss: 1.7356
Epoch [    3/   10] | d_loss: 1.5946 | g_loss: 1.0611
Epoch [    3/   10] | d_loss: 1.0240 | g_loss: 1.0768
Epoch [    3/   10] | d_loss: 0.8342 | g_loss: 1.8790
Epoch [    3/   10] | d_loss: 1.1164 | g_loss: 1.2486
Epoch [    3/   10] | d_loss: 1.2151 | g_loss: 1.0905
Epoch [    3/   10] | d_loss: 1.1871 | g_loss: 1.5997
Epoch [    3/   10] | d_loss: 1.2114 | g_loss: 1.3998
Epoch [    3/   10] | d_loss: 1.1944 | g_loss: 1.1262
Epoch [    3/   10] | d_loss: 1.1230 | g_loss: 0.9411
Epoch [    3/   10] | d_loss: 1.2969 | g_loss: 0.8282
Epoch [    3/   10] | d_loss: 1.0586 | g_loss: 2.1297
Epoch [    3/   10] | d_loss: 0.8721 | g_loss: 1.6437
Epoch [    3/   10] | d_loss: 1.1644 | g_loss: 0.9337
Epoch [    3/   10] | d_loss: 1.1340 | g_loss: 1.4382
Epoch [    3/   10] | d_loss: 0.9278 | g_loss: 1.3241
Epoch [    3/   10] | d_loss: 0.9010 | g_loss: 1.6117
```

```
Epoch [    3/   10] | d_loss: 1.1474 | g_loss: 1.5314
Epoch [    3/   10] | d_loss: 1.0737 | g_loss: 1.0175
Epoch [    3/   10] | d_loss: 1.2960 | g_loss: 1.1198
Epoch [    3/   10] | d_loss: 1.2042 | g_loss: 1.4942
Epoch [    3/   10] | d_loss: 1.0855 | g_loss: 1.7304
Epoch [    3/   10] | d_loss: 1.2687 | g_loss: 1.2702
Epoch [    3/   10] | d_loss: 1.2324 | g_loss: 1.0542
Epoch [    3/   10] | d_loss: 1.3124 | g_loss: 1.7940
Epoch [    3/   10] | d_loss: 0.9083 | g_loss: 1.4828
Epoch [    3/   10] | d_loss: 1.0250 | g_loss: 1.7628
Epoch [    3/   10] | d_loss: 1.1446 | g_loss: 1.0654
Epoch [    3/   10] | d_loss: 1.0182 | g_loss: 0.9536
Epoch [    3/   10] | d_loss: 1.1230 | g_loss: 1.0278
Epoch [    3/   10] | d_loss: 1.0140 | g_loss: 1.9132
Epoch [    3/   10] | d_loss: 1.3588 | g_loss: 0.6708
Epoch [    3/   10] | d_loss: 1.1095 | g_loss: 1.4425
Epoch [    3/   10] | d_loss: 1.0737 | g_loss: 1.3366
Epoch [    3/   10] | d_loss: 1.0459 | g_loss: 1.6088
Epoch [    3/   10] | d_loss: 0.9479 | g_loss: 2.0012
Epoch [    3/   10] | d_loss: 1.4986 | g_loss: 0.8846
Epoch [    3/   10] | d_loss: 1.0554 | g_loss: 1.3326
Epoch [    3/   10] | d_loss: 0.9682 | g_loss: 1.4190
Epoch [    3/   10] | d_loss: 0.7255 | g_loss: 2.5005
Epoch [    3/   10] | d_loss: 1.0955 | g_loss: 1.9457
Epoch [    3/   10] | d_loss: 1.2296 | g_loss: 1.9799
Epoch [    3/   10] | d_loss: 0.8702 | g_loss: 0.9167
Epoch [    3/   10] | d_loss: 1.2017 | g_loss: 1.5151
Epoch [    3/   10] | d_loss: 0.8994 | g_loss: 2.3413
Epoch [    3/   10] | d_loss: 1.3342 | g_loss: 1.4424
Epoch [    3/   10] | d_loss: 1.3889 | g_loss: 1.0654
Epoch [    3/   10] | d_loss: 0.9939 | g_loss: 1.9272
Epoch [    3/   10] | d_loss: 0.9492 | g_loss: 1.5764
Epoch [    3/   10] | d_loss: 0.9554 | g_loss: 1.8875
Epoch [    3/   10] | d_loss: 1.5843 | g_loss: 1.0497
Epoch [    3/   10] | d_loss: 1.2108 | g_loss: 1.1362
Epoch [    3/   10] | d_loss: 1.3795 | g_loss: 1.0430
Epoch [    3/   10] | d_loss: 0.9584 | g_loss: 1.5924
Epoch [    3/   10] | d_loss: 1.0044 | g_loss: 1.3433
Epoch [    3/   10] | d_loss: 1.0313 | g_loss: 1.6026
Epoch [    3/   10] | d_loss: 0.8859 | g_loss: 1.3749
Epoch [    3/   10] | d_loss: 0.8756 | g_loss: 1.1832
Epoch [    3/   10] | d_loss: 1.2413 | g_loss: 1.2658
Epoch [    3/   10] | d_loss: 0.8954 | g_loss: 1.3421
Epoch [    3/   10] | d_loss: 1.1650 | g_loss: 1.2642
Epoch [    3/   10] | d_loss: 1.1828 | g_loss: 1.1256
Epoch [    3/   10] | d_loss: 1.1482 | g_loss: 0.7817
Epoch [    3/   10] | d_loss: 0.9402 | g_loss: 1.5207
Epoch [    3/   10] | d_loss: 1.0202 | g_loss: 1.6878
```

```
Epoch [    3/   10] | d_loss: 1.2564 | g_loss: 1.9934
Epoch [    3/   10] | d_loss: 0.9457 | g_loss: 1.2762
Epoch [    3/   10] | d_loss: 1.1041 | g_loss: 0.7847
Epoch [    3/   10] | d_loss: 1.0631 | g_loss: 1.5912
Epoch [    3/   10] | d_loss: 1.2338 | g_loss: 2.7168
Epoch [    3/   10] | d_loss: 0.9120 | g_loss: 1.5773
Epoch [    3/   10] | d_loss: 1.5467 | g_loss: 1.5629
Epoch [    3/   10] | d_loss: 1.1550 | g_loss: 1.9785
Epoch [    3/   10] | d_loss: 1.2438 | g_loss: 1.5133
Epoch [    4/   10] | d_loss: 1.4750 | g_loss: 2.8702
Epoch [    4/   10] | d_loss: 0.6072 | g_loss: 1.4901
Epoch [    4/   10] | d_loss: 1.0764 | g_loss: 0.7908
Epoch [    4/   10] | d_loss: 0.8356 | g_loss: 1.4112
Epoch [    4/   10] | d_loss: 1.6632 | g_loss: 1.0487
Epoch [    4/   10] | d_loss: 0.7687 | g_loss: 1.0168
Epoch [    4/   10] | d_loss: 0.8629 | g_loss: 1.5379
Epoch [    4/   10] | d_loss: 1.1319 | g_loss: 1.1280
Epoch [    4/   10] | d_loss: 1.0561 | g_loss: 1.3680
Epoch [    4/   10] | d_loss: 1.5976 | g_loss: 2.3532
Epoch [    4/   10] | d_loss: 0.8213 | g_loss: 1.7691
Epoch [    4/   10] | d_loss: 1.1104 | g_loss: 1.3896
Epoch [    4/   10] | d_loss: 1.0576 | g_loss: 1.0102
Epoch [    4/   10] | d_loss: 1.7827 | g_loss: 2.7244
Epoch [    4/   10] | d_loss: 1.1371 | g_loss: 0.9083
Epoch [    4/   10] | d_loss: 1.0835 | g_loss: 1.6127
Epoch [    4/   10] | d_loss: 0.8314 | g_loss: 1.9983
Epoch [    4/   10] | d_loss: 1.2960 | g_loss: 1.7327
Epoch [    4/   10] | d_loss: 0.8968 | g_loss: 1.0884
Epoch [    4/   10] | d_loss: 1.0863 | g_loss: 2.3617
Epoch [    4/   10] | d_loss: 1.4677 | g_loss: 1.2057
Epoch [    4/   10] | d_loss: 0.8588 | g_loss: 1.1154
Epoch [    4/   10] | d_loss: 0.9684 | g_loss: 1.2404
Epoch [    4/   10] | d_loss: 1.7143 | g_loss: 2.8408
Epoch [    4/   10] | d_loss: 1.4721 | g_loss: 1.8689
Epoch [    4/   10] | d_loss: 0.9581 | g_loss: 1.9336
Epoch [    4/   10] | d_loss: 1.4875 | g_loss: 1.5862
Epoch [    4/   10] | d_loss: 0.8420 | g_loss: 1.0559
Epoch [    4/   10] | d_loss: 0.9197 | g_loss: 1.6165
Epoch [    4/   10] | d_loss: 0.8185 | g_loss: 1.2632
Epoch [    4/   10] | d_loss: 1.3743 | g_loss: 1.2937
Epoch [    4/   10] | d_loss: 1.0642 | g_loss: 0.7015
Epoch [    4/   10] | d_loss: 0.8181 | g_loss: 1.9955
Epoch [    4/   10] | d_loss: 1.2121 | g_loss: 1.3168
Epoch [    4/   10] | d_loss: 1.1212 | g_loss: 1.8228
Epoch [    4/   10] | d_loss: 1.0847 | g_loss: 2.0742
Epoch [    4/   10] | d_loss: 1.1528 | g_loss: 1.3479
Epoch [    4/   10] | d_loss: 1.6599 | g_loss: 0.9825
Epoch [    4/   10] | d_loss: 0.9309 | g_loss: 1.4802
```

```
Epoch [    4/   10] | d_loss: 1.0809 | g_loss: 2.2229
Epoch [    4/   10] | d_loss: 0.6724 | g_loss: 1.3572
Epoch [    4/   10] | d_loss: 0.9892 | g_loss: 1.6207
Epoch [    4/   10] | d_loss: 1.0626 | g_loss: 1.1961
Epoch [    4/   10] | d_loss: 1.1962 | g_loss: 2.0308
Epoch [    4/   10] | d_loss: 1.0514 | g_loss: 1.1713
Epoch [    4/   10] | d_loss: 0.7584 | g_loss: 1.4701
Epoch [    4/   10] | d_loss: 1.0754 | g_loss: 2.1302
Epoch [    4/   10] | d_loss: 0.8918 | g_loss: 1.6707
Epoch [    4/   10] | d_loss: 1.0512 | g_loss: 0.9735
Epoch [    4/   10] | d_loss: 0.9611 | g_loss: 1.2512
Epoch [    4/   10] | d_loss: 0.9215 | g_loss: 1.8310
Epoch [    4/   10] | d_loss: 1.1587 | g_loss: 1.6202
Epoch [    4/   10] | d_loss: 1.2601 | g_loss: 1.2518
Epoch [    4/   10] | d_loss: 1.0736 | g_loss: 1.3041
Epoch [    4/   10] | d_loss: 1.1442 | g_loss: 1.5960
Epoch [    4/   10] | d_loss: 1.3616 | g_loss: 1.1612
Epoch [    4/   10] | d_loss: 0.9483 | g_loss: 1.2930
Epoch [    4/   10] | d_loss: 1.6049 | g_loss: 2.5226
Epoch [    4/   10] | d_loss: 0.9580 | g_loss: 1.1246
Epoch [    4/   10] | d_loss: 1.1188 | g_loss: 1.8327
Epoch [    4/   10] | d_loss: 0.9736 | g_loss: 1.2381
Epoch [    4/   10] | d_loss: 0.9526 | g_loss: 1.5222
Epoch [    4/   10] | d_loss: 0.9186 | g_loss: 1.2669
Epoch [    4/   10] | d_loss: 1.1580 | g_loss: 1.4127
Epoch [    4/   10] | d_loss: 0.9100 | g_loss: 1.6952
Epoch [    4/   10] | d_loss: 1.2877 | g_loss: 0.9177
Epoch [    4/   10] | d_loss: 0.8739 | g_loss: 1.9813
Epoch [    4/   10] | d_loss: 0.9727 | g_loss: 1.9253
Epoch [    4/   10] | d_loss: 1.5521 | g_loss: 1.7312
Epoch [    4/   10] | d_loss: 1.3410 | g_loss: 0.6925
Epoch [    4/   10] | d_loss: 0.8950 | g_loss: 1.3280
Epoch [    4/   10] | d_loss: 1.0160 | g_loss: 1.6559
Epoch [    4/   10] | d_loss: 1.2604 | g_loss: 1.5461
Epoch [    4/   10] | d_loss: 0.9914 | g_loss: 2.6337
Epoch [    4/   10] | d_loss: 1.0528 | g_loss: 1.1219
Epoch [    4/   10] | d_loss: 1.3292 | g_loss: 2.5788
Epoch [    4/   10] | d_loss: 1.3515 | g_loss: 0.9151
Epoch [    4/   10] | d_loss: 0.8512 | g_loss: 1.3085
Epoch [    4/   10] | d_loss: 1.0442 | g_loss: 1.5975
Epoch [    4/   10] | d_loss: 1.0674 | g_loss: 1.5195
Epoch [    4/   10] | d_loss: 1.8059 | g_loss: 3.2087
Epoch [    4/   10] | d_loss: 1.0624 | g_loss: 1.5757
Epoch [    4/   10] | d_loss: 1.4623 | g_loss: 2.0987
Epoch [    4/   10] | d_loss: 1.2563 | g_loss: 1.0405
Epoch [    4/   10] | d_loss: 1.2931 | g_loss: 0.6135
Epoch [    4/   10] | d_loss: 1.2906 | g_loss: 2.1331
Epoch [    4/   10] | d_loss: 0.7915 | g_loss: 1.8433
```

```
Epoch [    4/   10] | d_loss: 1.0233 | g_loss: 2.3858
Epoch [    4/   10] | d_loss: 0.7885 | g_loss: 1.6247
Epoch [    4/   10] | d_loss: 1.4405 | g_loss: 0.9376
Epoch [    5/   10] | d_loss: 1.1318 | g_loss: 0.9082
Epoch [    5/   10] | d_loss: 0.7803 | g_loss: 2.4624
Epoch [    5/   10] | d_loss: 1.3940 | g_loss: 1.0214
Epoch [    5/   10] | d_loss: 1.0232 | g_loss: 0.9746
Epoch [    5/   10] | d_loss: 1.3650 | g_loss: 2.6696
Epoch [    5/   10] | d_loss: 1.0000 | g_loss: 1.8816
Epoch [    5/   10] | d_loss: 1.0050 | g_loss: 1.1029
Epoch [    5/   10] | d_loss: 1.2160 | g_loss: 1.9930
Epoch [    5/   10] | d_loss: 0.8427 | g_loss: 1.6330
Epoch [    5/   10] | d_loss: 1.4914 | g_loss: 0.7409
Epoch [    5/   10] | d_loss: 1.2116 | g_loss: 1.1024
Epoch [    5/   10] | d_loss: 0.8809 | g_loss: 2.1050
Epoch [    5/   10] | d_loss: 0.8529 | g_loss: 2.4208
Epoch [    5/   10] | d_loss: 1.4474 | g_loss: 1.6805
Epoch [    5/   10] | d_loss: 0.7559 | g_loss: 1.8718
Epoch [    5/   10] | d_loss: 0.6650 | g_loss: 1.8591
Epoch [    5/   10] | d_loss: 1.1224 | g_loss: 1.0908
Epoch [    5/   10] | d_loss: 0.7636 | g_loss: 1.7753
Epoch [    5/   10] | d_loss: 0.7861 | g_loss: 1.7060
Epoch [    5/   10] | d_loss: 1.2727 | g_loss: 0.9213
Epoch [    5/   10] | d_loss: 0.7646 | g_loss: 1.9612
Epoch [    5/   10] | d_loss: 0.9942 | g_loss: 1.1278
Epoch [    5/   10] | d_loss: 1.0997 | g_loss: 0.8268
Epoch [    5/   10] | d_loss: 1.0850 | g_loss: 2.2054
Epoch [    5/   10] | d_loss: 1.0862 | g_loss: 1.0529
Epoch [    5/   10] | d_loss: 1.2484 | g_loss: 0.8678
Epoch [    5/   10] | d_loss: 1.3771 | g_loss: 1.5351
Epoch [    5/   10] | d_loss: 1.3312 | g_loss: 0.9755
Epoch [    5/   10] | d_loss: 0.9800 | g_loss: 1.8784
Epoch [    5/   10] | d_loss: 1.2135 | g_loss: 0.8230
Epoch [    5/   10] | d_loss: 0.9621 | g_loss: 1.5346
Epoch [    5/   10] | d_loss: 0.8643 | g_loss: 1.6710
Epoch [    5/   10] | d_loss: 0.8906 | g_loss: 1.3413
Epoch [    5/   10] | d_loss: 0.9007 | g_loss: 1.1027
Epoch [    5/   10] | d_loss: 0.9548 | g_loss: 2.4572
Epoch [    5/   10] | d_loss: 1.1130 | g_loss: 2.3750
Epoch [    5/   10] | d_loss: 1.2107 | g_loss: 1.1435
Epoch [    5/   10] | d_loss: 1.7393 | g_loss: 2.8410
Epoch [    5/   10] | d_loss: 1.1707 | g_loss: 0.7529
Epoch [    5/   10] | d_loss: 1.6233 | g_loss: 0.9072
Epoch [    5/   10] | d_loss: 1.1644 | g_loss: 0.6178
Epoch [    5/   10] | d_loss: 1.3627 | g_loss: 1.1879
Epoch [    5/   10] | d_loss: 1.1955 | g_loss: 1.0738
Epoch [    5/   10] | d_loss: 0.7556 | g_loss: 1.6250
Epoch [    5/   10] | d_loss: 0.9617 | g_loss: 0.8069
```

```
Epoch [    5/   10] | d_loss: 0.7481 | g_loss: 1.2254
Epoch [    5/   10] | d_loss: 1.2104 | g_loss: 2.6066
Epoch [    5/   10] | d_loss: 1.2634 | g_loss: 3.0451
Epoch [    5/   10] | d_loss: 1.3173 | g_loss: 0.4815
Epoch [    5/   10] | d_loss: 0.8571 | g_loss: 1.2050
Epoch [    5/   10] | d_loss: 1.4727 | g_loss: 1.4522
Epoch [    5/   10] | d_loss: 0.6468 | g_loss: 1.6101
Epoch [    5/   10] | d_loss: 1.3495 | g_loss: 1.7757
Epoch [    5/   10] | d_loss: 1.2707 | g_loss: 1.0924
Epoch [    5/   10] | d_loss: 0.9537 | g_loss: 1.0868
Epoch [    5/   10] | d_loss: 0.6343 | g_loss: 2.1441
Epoch [    5/   10] | d_loss: 0.7363 | g_loss: 1.4944
Epoch [    5/   10] | d_loss: 1.1470 | g_loss: 0.8106
Epoch [    5/   10] | d_loss: 0.6263 | g_loss: 1.8273
Epoch [    5/   10] | d_loss: 0.5548 | g_loss: 2.3027
Epoch [    5/   10] | d_loss: 0.9288 | g_loss: 2.1814
Epoch [    5/   10] | d_loss: 0.9313 | g_loss: 1.8208
Epoch [    5/   10] | d_loss: 1.2500 | g_loss: 1.9439
Epoch [    5/   10] | d_loss: 0.6947 | g_loss: 1.5238
Epoch [    5/   10] | d_loss: 1.0407 | g_loss: 1.5261
Epoch [    5/   10] | d_loss: 1.0193 | g_loss: 2.3485
Epoch [    5/   10] | d_loss: 0.7951 | g_loss: 1.1454
Epoch [    5/   10] | d_loss: 0.8142 | g_loss: 1.9082
Epoch [    5/   10] | d_loss: 0.8810 | g_loss: 2.6096
Epoch [    5/   10] | d_loss: 0.8751 | g_loss: 1.4037
Epoch [    5/   10] | d_loss: 0.7016 | g_loss: 1.9413
Epoch [    5/   10] | d_loss: 0.9675 | g_loss: 2.4667
Epoch [    5/   10] | d_loss: 0.7592 | g_loss: 1.3252
Epoch [    5/   10] | d_loss: 1.2981 | g_loss: 0.8734
Epoch [    5/   10] | d_loss: 0.8893 | g_loss: 1.9614
Epoch [    5/   10] | d_loss: 1.1661 | g_loss: 2.2346
Epoch [    5/   10] | d_loss: 0.7977 | g_loss: 1.3500
Epoch [    5/   10] | d_loss: 0.6829 | g_loss: 1.7819
Epoch [    5/   10] | d_loss: 1.1961 | g_loss: 1.6565
Epoch [    5/   10] | d_loss: 1.4171 | g_loss: 0.8677
Epoch [    5/   10] | d_loss: 0.8070 | g_loss: 1.1225
Epoch [    5/   10] | d_loss: 0.6209 | g_loss: 1.8520
Epoch [    5/   10] | d_loss: 0.9298 | g_loss: 1.7088
Epoch [    5/   10] | d_loss: 1.3736 | g_loss: 1.7714
Epoch [    5/   10] | d_loss: 1.0659 | g_loss: 1.4655
Epoch [    5/   10] | d_loss: 0.6740 | g_loss: 1.4516
Epoch [    5/   10] | d_loss: 0.8683 | g_loss: 1.6674
Epoch [    5/   10] | d_loss: 1.5871 | g_loss: 1.8583
Epoch [    5/   10] | d_loss: 0.7465 | g_loss: 2.1066
Epoch [    5/   10] | d_loss: 1.2062 | g_loss: 0.7780
Epoch [    6/   10] | d_loss: 1.4250 | g_loss: 2.7882
Epoch [    6/   10] | d_loss: 0.8720 | g_loss: 2.6640
Epoch [    6/   10] | d_loss: 0.8521 | g_loss: 1.1320
```

```
Epoch [    6/   10] | d_loss: 1.7684 | g_loss: 0.5917
Epoch [    6/   10] | d_loss: 1.2578 | g_loss: 0.8816
Epoch [    6/   10] | d_loss: 1.1706 | g_loss: 0.8676
Epoch [    6/   10] | d_loss: 0.9345 | g_loss: 1.7406
Epoch [    6/   10] | d_loss: 0.5859 | g_loss: 1.6846
Epoch [    6/   10] | d_loss: 1.6689 | g_loss: 0.9139
Epoch [    6/   10] | d_loss: 0.8892 | g_loss: 1.7129
Epoch [    6/   10] | d_loss: 1.1140 | g_loss: 0.9444
Epoch [    6/   10] | d_loss: 0.8697 | g_loss: 1.6381
Epoch [    6/   10] | d_loss: 0.8479 | g_loss: 1.9758
Epoch [    6/   10] | d_loss: 0.5680 | g_loss: 2.3358
Epoch [    6/   10] | d_loss: 1.2702 | g_loss: 1.5182
Epoch [    6/   10] | d_loss: 1.0473 | g_loss: 1.3541
Epoch [    6/   10] | d_loss: 0.7378 | g_loss: 1.1823
Epoch [    6/   10] | d_loss: 0.8861 | g_loss: 1.7672
Epoch [    6/   10] | d_loss: 1.4007 | g_loss: 2.3100
Epoch [    6/   10] | d_loss: 1.2109 | g_loss: 1.1629
Epoch [    6/   10] | d_loss: 1.0677 | g_loss: 1.2983
Epoch [    6/   10] | d_loss: 0.9102 | g_loss: 2.3185
Epoch [    6/   10] | d_loss: 1.5296 | g_loss: 0.8756
Epoch [    6/   10] | d_loss: 0.7984 | g_loss: 2.0042
Epoch [    6/   10] | d_loss: 1.3222 | g_loss: 0.4767
Epoch [    6/   10] | d_loss: 1.0952 | g_loss: 1.8266
Epoch [    6/   10] | d_loss: 0.8410 | g_loss: 1.8611
Epoch [    6/   10] | d_loss: 1.3753 | g_loss: 1.4048
Epoch [    6/   10] | d_loss: 1.4978 | g_loss: 0.7934
Epoch [    6/   10] | d_loss: 1.1181 | g_loss: 1.8696
Epoch [    6/   10] | d_loss: 1.0634 | g_loss: 2.5611
Epoch [    6/   10] | d_loss: 0.6618 | g_loss: 2.7094
Epoch [    6/   10] | d_loss: 0.8346 | g_loss: 1.0999
Epoch [    6/   10] | d_loss: 0.8691 | g_loss: 3.3966
Epoch [    6/   10] | d_loss: 0.8672 | g_loss: 1.5879
Epoch [    6/   10] | d_loss: 0.9509 | g_loss: 1.4326
Epoch [    6/   10] | d_loss: 0.9790 | g_loss: 2.4175
Epoch [    6/   10] | d_loss: 1.1414 | g_loss: 1.1430
Epoch [    6/   10] | d_loss: 1.0889 | g_loss: 1.3757
Epoch [    6/   10] | d_loss: 0.7713 | g_loss: 2.1558
Epoch [    6/   10] | d_loss: 1.0888 | g_loss: 0.7501
Epoch [    6/   10] | d_loss: 0.9294 | g_loss: 1.4335
Epoch [    6/   10] | d_loss: 0.9770 | g_loss: 1.5831
Epoch [    6/   10] | d_loss: 0.7698 | g_loss: 1.2390
Epoch [    6/   10] | d_loss: 1.2482 | g_loss: 0.9524
Epoch [    6/   10] | d_loss: 1.3267 | g_loss: 1.6403
Epoch [    6/   10] | d_loss: 0.6662 | g_loss: 1.4787
Epoch [    6/   10] | d_loss: 1.4592 | g_loss: 3.4445
Epoch [    6/   10] | d_loss: 0.9540 | g_loss: 1.4721
Epoch [    6/   10] | d_loss: 1.1247 | g_loss: 1.7337
Epoch [    6/   10] | d_loss: 0.7378 | g_loss: 1.2276
```

```
Epoch [    6/   10] | d_loss: 1.1816 | g_loss: 2.7881
Epoch [    6/   10] | d_loss: 0.8952 | g_loss: 1.3970
Epoch [    6/   10] | d_loss: 0.5914 | g_loss: 1.3719
Epoch [    6/   10] | d_loss: 0.6932 | g_loss: 1.5376
Epoch [    6/   10] | d_loss: 0.8033 | g_loss: 1.3559
Epoch [    6/   10] | d_loss: 0.6146 | g_loss: 1.4037
Epoch [    6/   10] | d_loss: 0.7671 | g_loss: 1.8319
Epoch [    6/   10] | d_loss: 1.3831 | g_loss: 1.1665
Epoch [    6/   10] | d_loss: 1.2877 | g_loss: 2.7399
Epoch [    6/   10] | d_loss: 1.0341 | g_loss: 1.0020
Epoch [    6/   10] | d_loss: 1.5644 | g_loss: 2.4344
Epoch [    6/   10] | d_loss: 1.0648 | g_loss: 1.4189
Epoch [    6/   10] | d_loss: 0.5959 | g_loss: 1.3869
Epoch [    6/   10] | d_loss: 1.0660 | g_loss: 1.5331
Epoch [    6/   10] | d_loss: 1.1338 | g_loss: 0.6853
Epoch [    6/   10] | d_loss: 0.9550 | g_loss: 1.2980
Epoch [    6/   10] | d_loss: 0.8976 | g_loss: 1.2840
Epoch [    6/   10] | d_loss: 1.0090 | g_loss: 0.9045
Epoch [    6/   10] | d_loss: 0.5004 | g_loss: 2.7501
Epoch [    6/   10] | d_loss: 1.1630 | g_loss: 1.3988
Epoch [    6/   10] | d_loss: 0.7619 | g_loss: 2.0980
Epoch [    6/   10] | d_loss: 1.2471 | g_loss: 1.1085
Epoch [    6/   10] | d_loss: 0.9637 | g_loss: 1.0797
Epoch [    6/   10] | d_loss: 0.9734 | g_loss: 1.0593
Epoch [    6/   10] | d_loss: 1.1911 | g_loss: 0.9331
Epoch [    6/   10] | d_loss: 1.1212 | g_loss: 1.2914
Epoch [    6/   10] | d_loss: 0.7436 | g_loss: 1.5432
Epoch [    6/   10] | d_loss: 0.9698 | g_loss: 1.7825
Epoch [    6/   10] | d_loss: 0.9088 | g_loss: 0.7057
Epoch [    6/   10] | d_loss: 0.7578 | g_loss: 1.9996
Epoch [    6/   10] | d_loss: 0.7167 | g_loss: 1.6305
Epoch [    6/   10] | d_loss: 0.7271 | g_loss: 2.4807
Epoch [    6/   10] | d_loss: 0.8660 | g_loss: 1.7858
Epoch [    6/   10] | d_loss: 1.6879 | g_loss: 1.6712
Epoch [    6/   10] | d_loss: 1.3331 | g_loss: 2.4035
Epoch [    6/   10] | d_loss: 0.8148 | g_loss: 0.9077
Epoch [    6/   10] | d_loss: 1.0730 | g_loss: 1.9865
Epoch [    6/   10] | d_loss: 1.1134 | g_loss: 2.2211
Epoch [    6/   10] | d_loss: 0.7743 | g_loss: 1.3932
Epoch [    7/   10] | d_loss: 0.9832 | g_loss: 1.2389
Epoch [    7/   10] | d_loss: 1.3259 | g_loss: 1.3239
Epoch [    7/   10] | d_loss: 0.8246 | g_loss: 1.1267
Epoch [    7/   10] | d_loss: 0.5303 | g_loss: 2.1189
Epoch [    7/   10] | d_loss: 0.6017 | g_loss: 1.4290
Epoch [    7/   10] | d_loss: 0.9810 | g_loss: 0.6529
Epoch [    7/   10] | d_loss: 0.9017 | g_loss: 1.6018
Epoch [    7/   10] | d_loss: 0.9937 | g_loss: 1.1704
Epoch [    7/   10] | d_loss: 0.7347 | g_loss: 1.5510
```

```
Epoch [    7/   10] | d_loss: 1.2292 | g_loss: 1.6607
Epoch [    7/   10] | d_loss: 0.8850 | g_loss: 0.9888
Epoch [    7/   10] | d_loss: 1.1096 | g_loss: 1.0860
Epoch [    7/   10] | d_loss: 0.8570 | g_loss: 1.4345
Epoch [    7/   10] | d_loss: 0.5851 | g_loss: 3.0706
Epoch [    7/   10] | d_loss: 1.3173 | g_loss: 1.3844
Epoch [    7/   10] | d_loss: 1.0054 | g_loss: 1.4896
Epoch [    7/   10] | d_loss: 0.6089 | g_loss: 2.2562
Epoch [    7/   10] | d_loss: 1.1886 | g_loss: 2.3746
Epoch [    7/   10] | d_loss: 0.9191 | g_loss: 2.2482
Epoch [    7/   10] | d_loss: 1.3633 | g_loss: 2.7447
Epoch [    7/   10] | d_loss: 1.0784 | g_loss: 1.1613
Epoch [    7/   10] | d_loss: 0.5889 | g_loss: 2.4379
Epoch [    7/   10] | d_loss: 0.8633 | g_loss: 1.0314
Epoch [    7/   10] | d_loss: 0.9013 | g_loss: 1.9624
Epoch [    7/   10] | d_loss: 0.9309 | g_loss: 0.7054
Epoch [    7/   10] | d_loss: 1.0756 | g_loss: 1.7590
Epoch [    7/   10] | d_loss: 1.2922 | g_loss: 2.1340
Epoch [    7/   10] | d_loss: 1.5976 | g_loss: 1.0648
Epoch [    7/   10] | d_loss: 0.8383 | g_loss: 2.4233
Epoch [    7/   10] | d_loss: 0.7406 | g_loss: 1.5595
Epoch [    7/   10] | d_loss: 1.6125 | g_loss: 1.3186
Epoch [    7/   10] | d_loss: 1.0481 | g_loss: 1.7933
Epoch [    7/   10] | d_loss: 0.5137 | g_loss: 2.9293
Epoch [    7/   10] | d_loss: 0.8022 | g_loss: 1.3955
Epoch [    7/   10] | d_loss: 0.6655 | g_loss: 1.6284
Epoch [    7/   10] | d_loss: 1.2457 | g_loss: 0.8281
Epoch [    7/   10] | d_loss: 1.4008 | g_loss: 3.0965
Epoch [    7/   10] | d_loss: 0.7079 | g_loss: 1.2935
Epoch [    7/   10] | d_loss: 0.8799 | g_loss: 1.2394
Epoch [    7/   10] | d_loss: 0.8600 | g_loss: 2.4117
Epoch [    7/   10] | d_loss: 1.2369 | g_loss: 1.6762
Epoch [    7/   10] | d_loss: 0.7410 | g_loss: 2.6945
Epoch [    7/   10] | d_loss: 0.8873 | g_loss: 2.2021
Epoch [    7/   10] | d_loss: 1.2878 | g_loss: 1.2701
Epoch [    7/   10] | d_loss: 0.7389 | g_loss: 2.1464
Epoch [    7/   10] | d_loss: 0.8881 | g_loss: 1.8859
Epoch [    7/   10] | d_loss: 0.6638 | g_loss: 2.1203
Epoch [    7/   10] | d_loss: 0.9131 | g_loss: 1.7972
Epoch [    7/   10] | d_loss: 0.8611 | g_loss: 1.4104
Epoch [    7/   10] | d_loss: 0.8724 | g_loss: 1.7404
Epoch [    7/   10] | d_loss: 0.8577 | g_loss: 1.3647
Epoch [    7/   10] | d_loss: 1.1319 | g_loss: 1.1247
Epoch [    7/   10] | d_loss: 0.9180 | g_loss: 2.5746
Epoch [    7/   10] | d_loss: 0.6490 | g_loss: 2.1792
Epoch [    7/   10] | d_loss: 0.9730 | g_loss: 2.3709
Epoch [    7/   10] | d_loss: 1.1422 | g_loss: 1.7629
Epoch [    7/   10] | d_loss: 1.0092 | g_loss: 1.3433
```

```
Epoch [    7/   10] | d_loss: 1.2869 | g_loss: 1.4783
Epoch [    7/   10] | d_loss: 0.7989 | g_loss: 1.5123
Epoch [    7/   10] | d_loss: 1.0054 | g_loss: 2.4025
Epoch [    7/   10] | d_loss: 0.9099 | g_loss: 1.2777
Epoch [    7/   10] | d_loss: 0.8845 | g_loss: 1.8346
Epoch [    7/   10] | d_loss: 1.1272 | g_loss: 2.0638
Epoch [    7/   10] | d_loss: 0.9431 | g_loss: 1.5864
Epoch [    7/   10] | d_loss: 1.0458 | g_loss: 3.8013
Epoch [    7/   10] | d_loss: 1.0771 | g_loss: 1.3579
Epoch [    7/   10] | d_loss: 1.7283 | g_loss: 1.9654
Epoch [    7/   10] | d_loss: 0.9469 | g_loss: 2.1977
Epoch [    7/   10] | d_loss: 0.8162 | g_loss: 2.6007
Epoch [    7/   10] | d_loss: 1.1781 | g_loss: 1.3534
Epoch [    7/   10] | d_loss: 0.9052 | g_loss: 1.4187
Epoch [    7/   10] | d_loss: 1.8337 | g_loss: 0.7132
Epoch [    7/   10] | d_loss: 0.9595 | g_loss: 1.4336
Epoch [    7/   10] | d_loss: 0.7851 | g_loss: 2.0970
Epoch [    7/   10] | d_loss: 0.6954 | g_loss: 2.3723
Epoch [    7/   10] | d_loss: 1.0225 | g_loss: 2.5177
Epoch [    7/   10] | d_loss: 0.6899 | g_loss: 1.1928
Epoch [    7/   10] | d_loss: 1.1289 | g_loss: 0.8644
Epoch [    7/   10] | d_loss: 0.7559 | g_loss: 1.4595
Epoch [    7/   10] | d_loss: 0.8719 | g_loss: 1.1070
Epoch [    7/   10] | d_loss: 0.9614 | g_loss: 2.7570
Epoch [    7/   10] | d_loss: 0.8925 | g_loss: 2.1880
Epoch [    7/   10] | d_loss: 0.7354 | g_loss: 1.1059
Epoch [    7/   10] | d_loss: 0.7620 | g_loss: 1.0141
Epoch [    7/   10] | d_loss: 0.8771 | g_loss: 1.7345
Epoch [    7/   10] | d_loss: 0.6950 | g_loss: 2.1624
Epoch [    7/   10] | d_loss: 1.0692 | g_loss: 1.9985
Epoch [    7/   10] | d_loss: 0.6484 | g_loss: 1.7117
Epoch [    7/   10] | d_loss: 1.0777 | g_loss: 1.7878
Epoch [    7/   10] | d_loss: 0.9225 | g_loss: 2.1554
Epoch [    8/   10] | d_loss: 0.7918 | g_loss: 2.1960
Epoch [    8/   10] | d_loss: 1.0478 | g_loss: 1.9607
Epoch [    8/   10] | d_loss: 1.1279 | g_loss: 2.5284
Epoch [    8/   10] | d_loss: 0.7861 | g_loss: 1.8285
Epoch [    8/   10] | d_loss: 1.0051 | g_loss: 0.9922
Epoch [    8/   10] | d_loss: 0.5010 | g_loss: 2.0198
Epoch [    8/   10] | d_loss: 1.5435 | g_loss: 1.2359
Epoch [    8/   10] | d_loss: 0.7558 | g_loss: 2.1014
Epoch [    8/   10] | d_loss: 1.0910 | g_loss: 1.1916
Epoch [    8/   10] | d_loss: 1.1795 | g_loss: 1.1372
Epoch [    8/   10] | d_loss: 0.7634 | g_loss: 1.4697
Epoch [    8/   10] | d_loss: 0.6715 | g_loss: 0.8878
Epoch [    8/   10] | d_loss: 0.6140 | g_loss: 2.0526
Epoch [    8/   10] | d_loss: 1.4654 | g_loss: 0.8190
Epoch [    8/   10] | d_loss: 0.7186 | g_loss: 2.0784
```

```
Epoch [    8/   10] | d_loss: 1.2983 | g_loss: 1.9815
Epoch [    8/   10] | d_loss: 1.3141 | g_loss: 2.0868
Epoch [    8/   10] | d_loss: 0.9343 | g_loss: 1.0632
Epoch [    8/   10] | d_loss: 1.0339 | g_loss: 3.0128
Epoch [    8/   10] | d_loss: 0.7546 | g_loss: 2.3286
Epoch [    8/   10] | d_loss: 0.7908 | g_loss: 1.8858
Epoch [    8/   10] | d_loss: 1.1077 | g_loss: 1.1247
Epoch [    8/   10] | d_loss: 0.7954 | g_loss: 2.3514
Epoch [    8/   10] | d_loss: 1.4524 | g_loss: 0.9789
Epoch [    8/   10] | d_loss: 0.8086 | g_loss: 1.3136
Epoch [    8/   10] | d_loss: 0.6421 | g_loss: 2.8472
Epoch [    8/   10] | d_loss: 1.0943 | g_loss: 1.7574
Epoch [    8/   10] | d_loss: 0.8612 | g_loss: 2.7814
Epoch [    8/   10] | d_loss: 1.1405 | g_loss: 1.9476
Epoch [    8/   10] | d_loss: 0.8273 | g_loss: 1.6878
Epoch [    8/   10] | d_loss: 0.8735 | g_loss: 2.8736
Epoch [    8/   10] | d_loss: 0.4872 | g_loss: 2.3515
Epoch [    8/   10] | d_loss: 0.6194 | g_loss: 2.6196
Epoch [    8/   10] | d_loss: 0.6052 | g_loss: 2.1412
Epoch [    8/   10] | d_loss: 1.2355 | g_loss: 1.2913
Epoch [    8/   10] | d_loss: 0.6816 | g_loss: 1.4983
Epoch [    8/   10] | d_loss: 0.8024 | g_loss: 2.5858
Epoch [    8/   10] | d_loss: 1.0933 | g_loss: 3.6184
Epoch [    8/   10] | d_loss: 1.0110 | g_loss: 3.1034
Epoch [    8/   10] | d_loss: 0.7571 | g_loss: 1.9247
Epoch [    8/   10] | d_loss: 0.8880 | g_loss: 2.6252
Epoch [    8/   10] | d_loss: 1.0039 | g_loss: 1.5612
Epoch [    8/   10] | d_loss: 1.0183 | g_loss: 1.9759
Epoch [    8/   10] | d_loss: 0.9224 | g_loss: 2.8940
Epoch [    8/   10] | d_loss: 0.6854 | g_loss: 2.2664
Epoch [    8/   10] | d_loss: 0.7670 | g_loss: 2.2917
Epoch [    8/   10] | d_loss: 0.5354 | g_loss: 1.3654
Epoch [    8/   10] | d_loss: 0.9835 | g_loss: 1.3671
Epoch [    8/   10] | d_loss: 0.6889 | g_loss: 2.5114
Epoch [    8/   10] | d_loss: 1.2708 | g_loss: 2.4359
Epoch [    8/   10] | d_loss: 0.9924 | g_loss: 1.7111
Epoch [    8/   10] | d_loss: 0.8324 | g_loss: 2.5917
Epoch [    8/   10] | d_loss: 0.5919 | g_loss: 2.8174
Epoch [    8/   10] | d_loss: 0.6137 | g_loss: 1.8088
Epoch [    8/   10] | d_loss: 0.7334 | g_loss: 1.4561
Epoch [    8/   10] | d_loss: 1.0133 | g_loss: 1.2435
Epoch [    8/   10] | d_loss: 1.3144 | g_loss: 2.8424
Epoch [    8/   10] | d_loss: 1.1627 | g_loss: 1.3109
Epoch [    8/   10] | d_loss: 0.9726 | g_loss: 1.8123
Epoch [    8/   10] | d_loss: 1.3089 | g_loss: 3.5324
Epoch [    8/   10] | d_loss: 1.0461 | g_loss: 0.8428
Epoch [    8/   10] | d_loss: 0.8196 | g_loss: 1.9199
Epoch [    8/   10] | d_loss: 0.9176 | g_loss: 1.1626
```

```
Epoch [    8/   10] | d_loss: 0.5924 | g_loss: 1.6474
Epoch [    8/   10] | d_loss: 0.8787 | g_loss: 0.9900
Epoch [    8/   10] | d_loss: 0.8035 | g_loss: 1.2505
Epoch [    8/   10] | d_loss: 1.0451 | g_loss: 2.1737
Epoch [    8/   10] | d_loss: 0.9598 | g_loss: 0.9274
Epoch [    8/   10] | d_loss: 0.6745 | g_loss: 2.0649
Epoch [    8/   10] | d_loss: 0.6283 | g_loss: 1.6901
Epoch [    8/   10] | d_loss: 1.4618 | g_loss: 2.9967
Epoch [    8/   10] | d_loss: 1.0357 | g_loss: 3.2163
Epoch [    8/   10] | d_loss: 0.9631 | g_loss: 2.5392
Epoch [    8/   10] | d_loss: 1.2619 | g_loss: 2.4341
Epoch [    8/   10] | d_loss: 1.0154 | g_loss: 0.7984
Epoch [    8/   10] | d_loss: 0.7701 | g_loss: 1.4395
Epoch [    8/   10] | d_loss: 1.1962 | g_loss: 2.8623
Epoch [    8/   10] | d_loss: 0.7117 | g_loss: 2.2822
Epoch [    8/   10] | d_loss: 1.3575 | g_loss: 0.8509
Epoch [    8/   10] | d_loss: 1.1333 | g_loss: 1.6819
Epoch [    8/   10] | d_loss: 0.9081 | g_loss: 1.5477
Epoch [    8/   10] | d_loss: 0.6987 | g_loss: 1.3661
Epoch [    8/   10] | d_loss: 0.9038 | g_loss: 2.4889
Epoch [    8/   10] | d_loss: 1.1444 | g_loss: 1.6110
Epoch [    8/   10] | d_loss: 0.9373 | g_loss: 1.8816
Epoch [    8/   10] | d_loss: 0.9983 | g_loss: 2.2010
Epoch [    8/   10] | d_loss: 1.6550 | g_loss: 1.3623
Epoch [    8/   10] | d_loss: 1.0981 | g_loss: 3.7121
Epoch [    8/   10] | d_loss: 0.7028 | g_loss: 1.6321
Epoch [    8/   10] | d_loss: 0.7153 | g_loss: 2.4977
Epoch [    9/   10] | d_loss: 0.7606 | g_loss: 1.9253
Epoch [    9/   10] | d_loss: 0.5026 | g_loss: 2.3842
Epoch [    9/   10] | d_loss: 0.8090 | g_loss: 0.9323
Epoch [    9/   10] | d_loss: 0.5941 | g_loss: 2.0908
Epoch [    9/   10] | d_loss: 0.6804 | g_loss: 1.5060
Epoch [    9/   10] | d_loss: 0.8574 | g_loss: 1.7809
Epoch [    9/   10] | d_loss: 1.7079 | g_loss: 2.9482
Epoch [    9/   10] | d_loss: 1.0217 | g_loss: 1.7747
Epoch [    9/   10] | d_loss: 1.0875 | g_loss: 3.0244
Epoch [    9/   10] | d_loss: 0.8915 | g_loss: 2.8352
Epoch [    9/   10] | d_loss: 0.6787 | g_loss: 2.0770
Epoch [    9/   10] | d_loss: 0.9768 | g_loss: 2.9597
Epoch [    9/   10] | d_loss: 0.8458 | g_loss: 1.1887
Epoch [    9/   10] | d_loss: 1.0741 | g_loss: 2.1457
Epoch [    9/   10] | d_loss: 0.6579 | g_loss: 2.5003
Epoch [    9/   10] | d_loss: 0.4908 | g_loss: 3.3861
Epoch [    9/   10] | d_loss: 0.4882 | g_loss: 2.0469
Epoch [    9/   10] | d_loss: 0.5985 | g_loss: 0.9357
Epoch [    9/   10] | d_loss: 0.7202 | g_loss: 1.6967
Epoch [    9/   10] | d_loss: 0.7003 | g_loss: 1.7794
Epoch [    9/   10] | d_loss: 0.6105 | g_loss: 3.1953
```

```
Epoch [    9/   10] | d_loss: 0.9430 | g_loss: 1.4835
Epoch [    9/   10] | d_loss: 0.6438 | g_loss: 3.0813
Epoch [    9/   10] | d_loss: 1.2055 | g_loss: 1.8407
Epoch [    9/   10] | d_loss: 1.1846 | g_loss: 1.6896
Epoch [    9/   10] | d_loss: 1.3750 | g_loss: 3.1959
Epoch [    9/   10] | d_loss: 0.6124 | g_loss: 3.1329
Epoch [    9/   10] | d_loss: 1.1733 | g_loss: 0.9527
Epoch [    9/   10] | d_loss: 0.9037 | g_loss: 1.9663
Epoch [    9/   10] | d_loss: 2.0347 | g_loss: 1.1053
Epoch [    9/   10] | d_loss: 0.6775 | g_loss: 2.2646
Epoch [    9/   10] | d_loss: 0.8172 | g_loss: 1.7777
Epoch [    9/   10] | d_loss: 1.0211 | g_loss: 2.5591
Epoch [    9/   10] | d_loss: 0.5846 | g_loss: 2.8619
Epoch [    9/   10] | d_loss: 0.8351 | g_loss: 1.6022
Epoch [    9/   10] | d_loss: 0.7741 | g_loss: 2.6813
Epoch [    9/   10] | d_loss: 1.0193 | g_loss: 1.8472
Epoch [    9/   10] | d_loss: 0.6882 | g_loss: 2.4203
Epoch [    9/   10] | d_loss: 1.1589 | g_loss: 1.9986
Epoch [    9/   10] | d_loss: 1.4498 | g_loss: 1.6908
Epoch [    9/   10] | d_loss: 0.9927 | g_loss: 1.5666
Epoch [    9/   10] | d_loss: 1.7342 | g_loss: 2.6329
Epoch [    9/   10] | d_loss: 0.6813 | g_loss: 2.9540
Epoch [    9/   10] | d_loss: 0.9815 | g_loss: 2.3408
Epoch [    9/   10] | d_loss: 0.4962 | g_loss: 2.5424
Epoch [    9/   10] | d_loss: 0.8681 | g_loss: 2.0602
Epoch [    9/   10] | d_loss: 0.7733 | g_loss: 1.8563
Epoch [    9/   10] | d_loss: 0.5494 | g_loss: 1.9303
Epoch [    9/   10] | d_loss: 0.7201 | g_loss: 1.8798
Epoch [    9/   10] | d_loss: 0.7505 | g_loss: 1.5502
Epoch [    9/   10] | d_loss: 0.9644 | g_loss: 3.0160
Epoch [    9/   10] | d_loss: 0.7080 | g_loss: 2.9744
Epoch [    9/   10] | d_loss: 0.8232 | g_loss: 1.9664
Epoch [    9/   10] | d_loss: 0.8325 | g_loss: 2.3467
Epoch [    9/   10] | d_loss: 0.8721 | g_loss: 1.2859
Epoch [    9/   10] | d_loss: 0.9628 | g_loss: 1.7637
Epoch [    9/   10] | d_loss: 0.9137 | g_loss: 1.2927
Epoch [    9/   10] | d_loss: 1.1996 | g_loss: 1.0110
Epoch [    9/   10] | d_loss: 1.0410 | g_loss: 1.3981
Epoch [    9/   10] | d_loss: 0.9457 | g_loss: 1.3803
Epoch [    9/   10] | d_loss: 0.8360 | g_loss: 3.8459
Epoch [    9/   10] | d_loss: 0.7127 | g_loss: 2.0064
Epoch [    9/   10] | d_loss: 0.6244 | g_loss: 1.6841
Epoch [    9/   10] | d_loss: 0.6709 | g_loss: 1.9345
Epoch [    9/   10] | d_loss: 1.0170 | g_loss: 1.6227
Epoch [    9/   10] | d_loss: 1.5996 | g_loss: 0.9937
Epoch [    9/   10] | d_loss: 0.8264 | g_loss: 1.8016
Epoch [    9/   10] | d_loss: 1.4382 | g_loss: 0.6305
Epoch [    9/   10] | d_loss: 1.1940 | g_loss: 1.3615
```

```
Epoch [    9/   10] | d_loss: 0.8549 | g_loss: 1.7473
Epoch [    9/   10] | d_loss: 1.0550 | g_loss: 1.6313
Epoch [    9/   10] | d_loss: 0.8641 | g_loss: 2.6841
Epoch [    9/   10] | d_loss: 0.7676 | g_loss: 1.4875
Epoch [    9/   10] | d_loss: 0.8102 | g_loss: 1.9931
Epoch [    9/   10] | d_loss: 1.0009 | g_loss: 1.1767
Epoch [    9/   10] | d_loss: 0.5766 | g_loss: 1.3928
Epoch [    9/   10] | d_loss: 0.8951 | g_loss: 1.9705
Epoch [    9/   10] | d_loss: 0.5066 | g_loss: 1.5311
Epoch [    9/   10] | d_loss: 0.5723 | g_loss: 2.0845
Epoch [    9/   10] | d_loss: 0.7193 | g_loss: 1.4589
Epoch [    9/   10] | d_loss: 0.6093 | g_loss: 1.8789
Epoch [    9/   10] | d_loss: 1.4843 | g_loss: 1.5149
Epoch [    9/   10] | d_loss: 1.0875 | g_loss: 1.5127
Epoch [    9/   10] | d_loss: 0.7980 | g_loss: 1.4533
Epoch [    9/   10] | d_loss: 0.6097 | g_loss: 1.3206
Epoch [    9/   10] | d_loss: 1.2009 | g_loss: 1.8586
Epoch [    9/   10] | d_loss: 0.8661 | g_loss: 2.1951
Epoch [    9/   10] | d_loss: 1.0333 | g_loss: 2.0485
Epoch [    9/   10] | d_loss: 0.8903 | g_loss: 0.7065
Epoch [    9/   10] | d_loss: 0.7936 | g_loss: 1.5751
Epoch [   10/   10] | d_loss: 0.9406 | g_loss: 2.7488
Epoch [   10/   10] | d_loss: 0.9576 | g_loss: 1.7243
Epoch [   10/   10] | d_loss: 0.9135 | g_loss: 1.6217
Epoch [   10/   10] | d_loss: 0.7289 | g_loss: 1.9270
Epoch [   10/   10] | d_loss: 0.5733 | g_loss: 1.5898
Epoch [   10/   10] | d_loss: 0.9050 | g_loss: 1.9918
Epoch [   10/   10] | d_loss: 1.1455 | g_loss: 2.1894
Epoch [   10/   10] | d_loss: 1.3563 | g_loss: 1.5630
Epoch [   10/   10] | d_loss: 0.8272 | g_loss: 2.6491
Epoch [   10/   10] | d_loss: 0.6754 | g_loss: 1.8610
Epoch [   10/   10] | d_loss: 0.8615 | g_loss: 2.4417
Epoch [   10/   10] | d_loss: 0.9485 | g_loss: 1.0264
Epoch [   10/   10] | d_loss: 0.6942 | g_loss: 1.1883
Epoch [   10/   10] | d_loss: 0.6531 | g_loss: 2.1462
Epoch [   10/   10] | d_loss: 0.6973 | g_loss: 2.7918
Epoch [   10/   10] | d_loss: 1.1941 | g_loss: 3.5123
Epoch [   10/   10] | d_loss: 0.6670 | g_loss: 1.7770
Epoch [   10/   10] | d_loss: 0.8500 | g_loss: 3.0427
Epoch [   10/   10] | d_loss: 1.1184 | g_loss: 2.6594
Epoch [   10/   10] | d_loss: 0.7401 | g_loss: 2.3154
Epoch [   10/   10] | d_loss: 0.8726 | g_loss: 1.0813
Epoch [   10/   10] | d_loss: 0.9633 | g_loss: 1.9611
Epoch [   10/   10] | d_loss: 1.0688 | g_loss: 3.4147
Epoch [   10/   10] | d_loss: 1.5055 | g_loss: 2.7259
Epoch [   10/   10] | d_loss: 0.7492 | g_loss: 1.8062
Epoch [   10/   10] | d_loss: 0.9444 | g_loss: 3.0584
Epoch [   10/   10] | d_loss: 0.9860 | g_loss: 2.2958
```

```
Epoch [   10/   10] | d_loss: 0.9785 | g_loss: 1.2940
Epoch [   10/   10] | d_loss: 0.9268 | g_loss: 2.0435
Epoch [   10/   10] | d_loss: 0.9004 | g_loss: 3.9172
Epoch [   10/   10] | d_loss: 0.9093 | g_loss: 1.2837
Epoch [   10/   10] | d_loss: 0.5568 | g_loss: 2.2053
Epoch [   10/   10] | d_loss: 0.8664 | g_loss: 1.8986
Epoch [   10/   10] | d_loss: 0.9719 | g_loss: 1.8358
Epoch [   10/   10] | d_loss: 0.8021 | g_loss: 1.3541
Epoch [   10/   10] | d_loss: 0.9600 | g_loss: 1.9109
Epoch [   10/   10] | d_loss: 1.0693 | g_loss: 1.6508
Epoch [   10/   10] | d_loss: 0.5700 | g_loss: 1.9503
Epoch [   10/   10] | d_loss: 0.7307 | g_loss: 1.7985
Epoch [   10/   10] | d_loss: 0.6426 | g_loss: 2.3221
Epoch [   10/   10] | d_loss: 0.5455 | g_loss: 1.6468
Epoch [   10/   10] | d_loss: 0.8292 | g_loss: 2.6325
Epoch [   10/   10] | d_loss: 0.6854 | g_loss: 1.3032
Epoch [   10/   10] | d_loss: 0.7969 | g_loss: 1.4057
Epoch [   10/   10] | d_loss: 0.9978 | g_loss: 2.0570
Epoch [   10/   10] | d_loss: 0.4787 | g_loss: 2.6910
Epoch [   10/   10] | d_loss: 1.2138 | g_loss: 1.0740
Epoch [   10/   10] | d_loss: 0.7269 | g_loss: 1.6942
Epoch [   10/   10] | d_loss: 0.7723 | g_loss: 1.9335
Epoch [   10/   10] | d_loss: 0.7599 | g_loss: 2.2494
Epoch [   10/   10] | d_loss: 0.9096 | g_loss: 2.2536
Epoch [   10/   10] | d_loss: 0.8299 | g_loss: 4.0770
Epoch [   10/   10] | d_loss: 0.8464 | g_loss: 1.4656
Epoch [   10/   10] | d_loss: 1.6105 | g_loss: 2.0340
Epoch [   10/   10] | d_loss: 0.8878 | g_loss: 3.7235
Epoch [   10/   10] | d_loss: 0.8360 | g_loss: 1.1480
Epoch [   10/   10] | d_loss: 0.6850 | g_loss: 1.8707
Epoch [   10/   10] | d_loss: 0.8421 | g_loss: 1.8218
Epoch [   10/   10] | d_loss: 0.8911 | g_loss: 3.8742
Epoch [   10/   10] | d_loss: 0.7242 | g_loss: 2.0084
Epoch [   10/   10] | d_loss: 0.7232 | g_loss: 2.7728
Epoch [   10/   10] | d_loss: 0.8939 | g_loss: 2.3327
Epoch [   10/   10] | d_loss: 0.5396 | g_loss: 1.8506
Epoch [   10/   10] | d_loss: 0.8033 | g_loss: 1.7955
Epoch [   10/   10] | d_loss: 0.7645 | g_loss: 3.2570
Epoch [   10/   10] | d_loss: 0.5677 | g_loss: 2.8280
Epoch [   10/   10] | d_loss: 0.7155 | g_loss: 1.2804
Epoch [   10/   10] | d_loss: 0.5471 | g_loss: 2.6488
Epoch [   10/   10] | d_loss: 1.4045 | g_loss: 3.0037
Epoch [   10/   10] | d_loss: 1.0053 | g_loss: 2.3222
Epoch [   10/   10] | d_loss: 0.6380 | g_loss: 1.2978
Epoch [   10/   10] | d_loss: 1.1008 | g_loss: 1.7858
Epoch [   10/   10] | d_loss: 0.8840 | g_loss: 1.3930
Epoch [   10/   10] | d_loss: 0.7378 | g_loss: 1.6578
Epoch [   10/   10] | d_loss: 1.0001 | g_loss: 1.4578
```

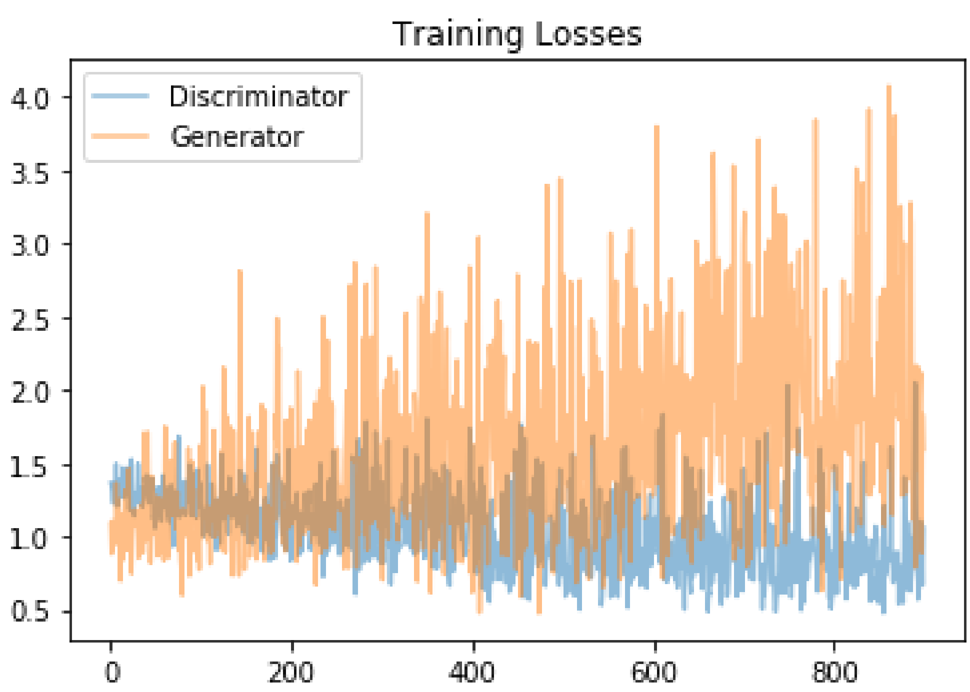
```
Epoch [   10/   10] | d_loss: 0.6269 | g_loss: 3.2813
Epoch [   10/   10] | d_loss: 0.7211 | g_loss: 3.1330
Epoch [   10/   10] | d_loss: 0.7783 | g_loss: 1.2105
Epoch [   10/   10] | d_loss: 0.7884 | g_loss: 1.1184
Epoch [   10/   10] | d_loss: 1.4769 | g_loss: 1.6394
Epoch [   10/   10] | d_loss: 2.0494 | g_loss: 0.7895
Epoch [   10/   10] | d_loss: 1.2950 | g_loss: 2.1676
Epoch [   10/   10] | d_loss: 1.0184 | g_loss: 1.4635
Epoch [   10/   10] | d_loss: 0.5696 | g_loss: 2.1593
Epoch [   10/   10] | d_loss: 0.6495 | g_loss: 1.5507
Epoch [   10/   10] | d_loss: 0.8057 | g_loss: 1.9277
Epoch [   10/   10] | d_loss: 1.1067 | g_loss: 2.1201
Epoch [   10/   10] | d_loss: 0.9305 | g_loss: 0.8883
Epoch [   10/   10] | d_loss: 0.6657 | g_loss: 1.8341
Epoch [   10/   10] | d_loss: 1.0595 | g_loss: 1.5959
```

2.9 Training loss

Plot the training losses for the generator and discriminator, recorded after each epoch.

In [51]:
```
fig, ax = plt.subplots()
losses = np.array(losses)
plt.plot(losses.T[0], label='Discriminator', alpha=0.5)
plt.plot(losses.T[1], label='Generator', alpha=0.5)
plt.title("Training Losses")
plt.legend()
```

Out[51]: <matplotlib.legend.Legend at 0x7fe66439ccc0>



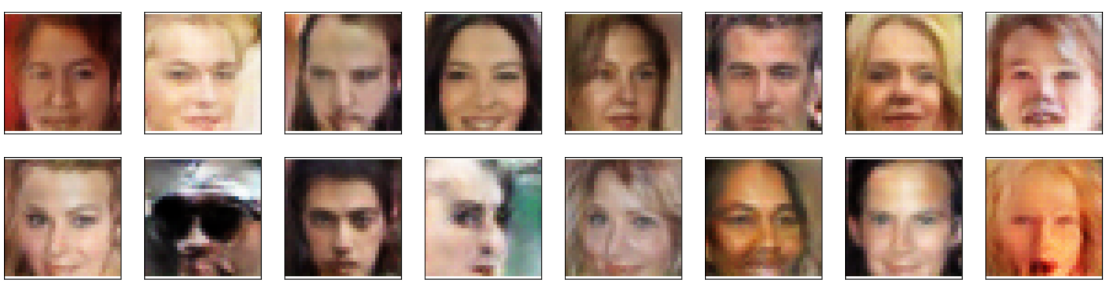
2.10 Generator samples from training

View samples of images from the generator, and answer a question about the strengths and weaknesses of your trained models.

```
In [52]: # helper function for viewing a list of passed in sample images
         def view_samples(epoch, samples):
             fig, axes = plt.subplots(figsize=(16,4), nrows=2, ncols=8, sharey=True, sharex=True
             for ax, img in zip(axes.flatten(), samples[epoch]):
                 img = img.detach().cpu().numpy()
                 img = np.transpose(img, (1, 2, 0))
                 img = ((img + 1)*255 / (2)).astype(np.uint8)
                 ax.xaxis.set_visible(False)
                 ax.yaxis.set_visible(False)
                 im = ax.imshow(img.reshape((32,32,3)))
```

```
In [53]: # Load samples from generator, taken while training
         with open('train_samples.pkl', 'rb') as f:
             samples = pkl.load(f)
```

```
In [54]: _ = view_samples(-1, samples)
```



2.10.1 Question: What do you notice about your generated samples and how might you improve this model?

When you answer this question, consider the following factors: * The dataset is biased; it is made of "celebrity" faces that are mostly white * Model size; larger models have the opportunity to learn more features in a data feature space * Optimization strategy; optimizers and number of epochs affect your final result

Answer: - more variety of faces might help to train neural network better and generate a new type of faces. -Model size matters we have to ensure that our models reconginze and generate faces correctly. Deep models allow to catch some more characteristrics of the faces. -I used suggested beta1 0.5 and it generated more types of faces than beta1 0.1 which was suggested by the paper. -Adam is the best choice for GAN's as well as other architectures. -Number of epochs is a critical component of GAN's. Especially spread betwenn batch size of a generator and a descriminator.

2.10.2 Submitting This Project

When submitting this project, make sure to run all the cells before saving the notebook. Save the notebook file as "dlnd_face_generation.ipynb" and save it as a HTML file under "File" -> "Download as". Include the "problem_unittests.py" files in your submission.