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.

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.

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 [1]:

```
# can comment out after executing
# !unzip processed_celeba_small.zip
```

In [1]:

```
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
```

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)]#RGB_Images) each.

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](https://pytorch.org/docs/stable/torchvision/datasets.html#imagefolder) wrapper, with a root directory `processed_celeba_small/` and data transformation passed in.

In [2]:

```
# necessary imports
import os, torch
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision import transforms
```

In [3]:

```
def get_dataloader(batch_size, image_size, data_dir='processed_celeba_small/', num_workers=0):
    """
    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
    # resize and normalize the images
    transform = transforms.Compose([transforms.Resize(image_size), # resize to 32x32
                                    transforms.ToTensor()])

    # get training and test directories
    train_path = './' + data_dir

    # define datasets using ImageFolder
    train_dataset = datasets.ImageFolder(train_path, transform)

    # create and return DataLoaders
    train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True, num_worke
rs=num_workers)

    return train_loader
```

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 [4]:

```
# Define function hyperparameters
batch_size = 128
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 [5]:

```
# 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 [6]:

```
# current range
img = images[0]

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

```
Min:  tensor(1.00000e-02 *
       5.4902)
Max:  tensor(0.9961)
```

In [7]:

```
# 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 [9]:

```
"""
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(-0.9294)
Max:  tensor(0.9843)
```

Define the Model

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

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 [10]:

```
import torch.nn as nn
import torch.nn.functional as F
```

In [11]:

```
# 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, out_channels,
                           kernel_size, stride, padding, bias=False)

    # append conv layer
    layers.append(conv_layer)

    if batch_norm:
        # append batchnorm layer
        layers.append(nn.BatchNorm2d(out_channels))

    # using Sequential container
    return nn.Sequential(*layers)
```

In [12]:

```
class Discriminator(nn.Module):

    def __init__(self, conv_dim):
        """
        Initialize the Discriminator Module
        :param conv_dim: The depth of the first convolutional layer
        """
        super(Discriminator, self).__init__()

        # complete init function
        self.conv_dim = conv_dim

        # 32x32 input
```

```
        self.conv1 = conv(3, conv_dim, 4, batch_norm=False) # first layer, no batch_norm
        # 16x16 out
        self.conv2 = conv(conv_dim, conv_dim*2, 4)
        # 8x8 out
        self.conv3 = conv(conv_dim*2, conv_dim*4, 4)
        # 4x4 out

        # final, fully-connected layer
        self.fc = nn.Linear(conv_dim*4*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
        # all hidden layers + leaky relu activation
        out = F.leaky_relu(self.conv1(x), 0.2)
        out = F.leaky_relu(self.conv2(out), 0.2)
        out = F.leaky_relu(self.conv3(out), 0.2)

        # flatten
        out = out.view(-1, self.conv_dim*4*4*4)

        # final output layer
        out = self.fc(out)

        return out


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

```
Tests Passed
```

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 [13]:

```
# helper deconv function
def deconv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True):
    """Creates a transposed-convolutional layer, with optional batch normalization.
    """
    # create a sequence of transpose + optional batch norm layers
    layers = []
    transpose_conv_layer = nn.ConvTranspose2d(in_channels, out_channels,
                                              kernel_size, stride, padding, bias=False)
    # append transpose convolutional layer
    layers.append(transpose_conv_layer)

    if batch_norm:
        # append batchnorm layer
        layers.append(nn.BatchNorm2d(out_channels))

    return nn.Sequential(*layers)
```

In [14]:

```
class Generator(nn.Module):
```

```python
    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 layer
        """
        super(Generator, self).__init__()

        # complete init function
        self.conv_dim = conv_dim

        # first, fully-connected layer
        self.fc = nn.Linear(z_size, conv_dim*4*4*4)

        # transpose conv layers
        self.t_conv1 = deconv(conv_dim*4, conv_dim*2, 4)
        self.t_conv2 = deconv(conv_dim*2, conv_dim, 4)
        self.t_conv3 = deconv(conv_dim, 3, 4, batch_norm=False)

    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
        # fully-connected + reshape
        out = self.fc(x)
        out = out.view(-1, self.conv_dim*4, 4, 4) # (batch_size, depth, 4, 4)

        # hidden transpose conv layers + relu
        out = F.relu(self.t_conv1(out))
        out = F.relu(self.t_conv2(out))

        # last layer + tanh activation
        out = F.tanh(self.t_conv3(out))

        return out

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

```
Tests Passed
```

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 [15]:

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

        if hasattr(m, 'bias') and m.bias is not None:
            m.bias.data.zero_()
```

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 [16]:

```
"""
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 [17]:

```
# 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): Sequential(
    (0): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (fc): Linear(in_features=2048, out_features=1, bias=True)
)

Generator(
  (fc): Linear(in_features=100, out_features=2048, bias=True)
  (t_conv1): Sequential(
```

```
    (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (t_conv2): Sequential(
    (0): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (t_conv3): Sequential(
    (0): ConvTranspose2d(32, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
)
```

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 [18]:

```
"""
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!
```

Discriminator and Generator Losses

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

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.

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 [19]:

```
def real_loss(D_out):
    '''Calculates how close discriminator outputs are to being real.
```

```
    '''Calculates how close discriminator outputs are to being real.
       param, D_out: discriminator logits
       return: real loss'''
    batch_size = D_out.size(0)

    # smoothing real labels with 0.9
    labels = torch.ones(batch_size)*0.9

    # move labels to GPU if available
    if train_on_gpu:
        labels = labels.cuda()

    # binary cross entropy with logits loss
    criterion = nn.BCEWithLogitsLoss()

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

    return loss

def fake_loss(D_out):
    '''Calculates how close discriminator outputs are to being fake.
       param, D_out: discriminator logits
       return: fake loss'''
    batch_size = D_out.size(0)
    labels = torch.zeros(batch_size) # fake labels = 0

    # move labels to GPU if available
    if train_on_gpu:
        labels = labels.cuda()

    # binary cross entropy with logits loss
    criterion = nn.BCEWithLogitsLoss()

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

    return loss
```

Optimizers

Exercise: Define optimizers for your Discriminator (D) and Generator (G)

Define optimizers for your models with appropriate hyperparameters.

In [20]:

```
import torch.optim as optim

# params
lr = 0.0002
beta1=0.5
beta2=0.999

# Create optimizers for the discriminator D and generator G
d_optimizer = optim.Adam(D.parameters(), lr, [beta1, beta2])
g_optimizer = optim.Adam(G.parameters(), lr, [beta1, beta2])
```

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 [21]:

```
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:
        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
            d_optimizer.zero_grad()

            # Train with real images

            # Compute the discriminator losses on real images
            if train_on_gpu:
                real_images = real_images.cuda()

            D_real = D(real_images)
            d_real_loss = real_loss(D_real)

            # Train with fake images
            # Generate fake images
            z = np.random.uniform(-1, 1, size=(batch_size, z_size))
            z = torch.from_numpy(z).float()

            # move x to GPU, if available
            if train_on_gpu:
                z = z.cuda()

            fake_images = G(z)

            # Compute the discriminator losses on fake images
            D_fake = D(fake_images)
            d_fake_loss = fake_loss(D_fake)

            # add up loss and perform backprop
            d_loss = d_real_loss + d_fake_loss
            d_loss.backward()
```

```
            d_optimizer.step()

            # 2. Train the generator with an adversarial loss
            g_optimizer.zero_grad()

            # Train with fake images and flipped labels
            # Generate fake images
            z = np.random.uniform(-1, 1, size=(batch_size, z_size))
            z = torch.from_numpy(z).float()
            if train_on_gpu:
                z = z.cuda()
            fake_images = G(z)

            # Compute the discriminator losses on fake images
            # using flipped labels!
            D_fake = D(fake_images)

            g_loss = real_loss(D_fake) # use real loss to flip labels

            # perform backprop
            g_loss.backward()
            g_optimizer.step()


            # ===============================================
            #              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 [22]:

```
from workspace_utils import keep_awake, active_session

# set number of epochs
n_epochs = 50

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

```
Epoch [    1/   50] | d_loss: 1.4017 | g_loss: 0.7764
Epoch [    1/   50] | d_loss: 0.5049 | g_loss: 2.4111
Epoch [    1/   50] | d_loss: 0.5126 | g_loss: 2.9460
Epoch [    1/   50] | d_loss: 0.4862 | g_loss: 2.6548
Epoch [    1/   50] | d_loss: 0.4857 | g_loss: 2.5725
Epoch [    1/   50] | d_loss: 0.5393 | g_loss: 2.4223
```

```
Epoch [    1/   50] | d_loss: 0.6165 | g_loss: 1.6804
Epoch [    1/   50] | d_loss: 0.9311 | g_loss: 2.6842
Epoch [    1/   50] | d_loss: 1.0746 | g_loss: 0.8896
Epoch [    1/   50] | d_loss: 0.9227 | g_loss: 1.3343
Epoch [    1/   50] | d_loss: 1.1347 | g_loss: 0.9376
Epoch [    1/   50] | d_loss: 0.9915 | g_loss: 1.0153
Epoch [    1/   50] | d_loss: 1.0169 | g_loss: 1.6662
Epoch [    1/   50] | d_loss: 1.3813 | g_loss: 0.7744
Epoch [    1/   50] | d_loss: 1.1672 | g_loss: 1.1276
Epoch [    2/   50] | d_loss: 1.0695 | g_loss: 1.0539
Epoch [    2/   50] | d_loss: 1.2064 | g_loss: 1.2204
Epoch [    2/   50] | d_loss: 1.2466 | g_loss: 0.9732
Epoch [    2/   50] | d_loss: 1.1882 | g_loss: 1.0286
Epoch [    2/   50] | d_loss: 1.2419 | g_loss: 1.1679
Epoch [    2/   50] | d_loss: 1.1565 | g_loss: 0.8812
Epoch [    2/   50] | d_loss: 1.1828 | g_loss: 2.0894
Epoch [    2/   50] | d_loss: 1.3176 | g_loss: 1.5671
Epoch [    2/   50] | d_loss: 1.2756 | g_loss: 0.9795
Epoch [    2/   50] | d_loss: 1.0930 | g_loss: 1.3317
Epoch [    2/   50] | d_loss: 1.2508 | g_loss: 0.8991
Epoch [    2/   50] | d_loss: 1.2409 | g_loss: 1.3255
Epoch [    2/   50] | d_loss: 1.0355 | g_loss: 1.2861
Epoch [    2/   50] | d_loss: 1.5139 | g_loss: 1.1248
Epoch [    2/   50] | d_loss: 1.1082 | g_loss: 1.2990
Epoch [    3/   50] | d_loss: 1.1259 | g_loss: 1.3233
Epoch [    3/   50] | d_loss: 1.1391 | g_loss: 1.1912
Epoch [    3/   50] | d_loss: 1.4674 | g_loss: 1.1588
Epoch [    3/   50] | d_loss: 1.1596 | g_loss: 1.0354
Epoch [    3/   50] | d_loss: 1.0275 | g_loss: 1.1809
Epoch [    3/   50] | d_loss: 1.1506 | g_loss: 0.9659
Epoch [    3/   50] | d_loss: 1.2326 | g_loss: 1.0411
Epoch [    3/   50] | d_loss: 1.1441 | g_loss: 1.4689
Epoch [    3/   50] | d_loss: 1.1179 | g_loss: 1.0715
Epoch [    3/   50] | d_loss: 1.2257 | g_loss: 0.9395
Epoch [    3/   50] | d_loss: 1.1778 | g_loss: 1.0012
Epoch [    3/   50] | d_loss: 1.1541 | g_loss: 1.3424
Epoch [    3/   50] | d_loss: 1.1426 | g_loss: 1.0488
Epoch [    3/   50] | d_loss: 1.1361 | g_loss: 1.1076
Epoch [    3/   50] | d_loss: 1.2047 | g_loss: 0.9338
Epoch [    4/   50] | d_loss: 1.1084 | g_loss: 1.4626
Epoch [    4/   50] | d_loss: 1.1194 | g_loss: 1.2046
Epoch [    4/   50] | d_loss: 1.1247 | g_loss: 0.9912
Epoch [    4/   50] | d_loss: 1.1917 | g_loss: 0.8218
Epoch [    4/   50] | d_loss: 1.1152 | g_loss: 1.3717
Epoch [    4/   50] | d_loss: 1.1747 | g_loss: 0.7575
Epoch [    4/   50] | d_loss: 1.1670 | g_loss: 0.9794
Epoch [    4/   50] | d_loss: 1.2418 | g_loss: 0.8602
Epoch [    4/   50] | d_loss: 1.2401 | g_loss: 0.9079
Epoch [    4/   50] | d_loss: 1.2116 | g_loss: 1.0196
Epoch [    4/   50] | d_loss: 1.2142 | g_loss: 1.2089
Epoch [    4/   50] | d_loss: 1.2383 | g_loss: 1.4155
Epoch [    4/   50] | d_loss: 1.2597 | g_loss: 1.0170
Epoch [    4/   50] | d_loss: 1.3493 | g_loss: 0.4862
Epoch [    4/   50] | d_loss: 1.0537 | g_loss: 1.0453
Epoch [    5/   50] | d_loss: 1.1332 | g_loss: 1.0505
Epoch [    5/   50] | d_loss: 1.1308 | g_loss: 1.1888
Epoch [    5/   50] | d_loss: 1.1404 | g_loss: 1.3274
Epoch [    5/   50] | d_loss: 1.2382 | g_loss: 1.2966
Epoch [    5/   50] | d_loss: 1.0068 | g_loss: 1.4494
Epoch [    5/   50] | d_loss: 1.0479 | g_loss: 1.3503
Epoch [    5/   50] | d_loss: 1.1962 | g_loss: 0.8288
Epoch [    5/   50] | d_loss: 1.1063 | g_loss: 1.2201
Epoch [    5/   50] | d_loss: 1.0739 | g_loss: 0.9970
Epoch [    5/   50] | d_loss: 1.7288 | g_loss: 2.0772
Epoch [    5/   50] | d_loss: 1.0982 | g_loss: 1.0353
Epoch [    5/   50] | d_loss: 0.8966 | g_loss: 1.6282
Epoch [    5/   50] | d_loss: 1.1107 | g_loss: 1.5597
Epoch [    5/   50] | d_loss: 1.1005 | g_loss: 1.3109
Epoch [    5/   50] | d_loss: 1.2294 | g_loss: 0.8478
Epoch [    6/   50] | d_loss: 1.1933 | g_loss: 1.2190
Epoch [    6/   50] | d_loss: 1.2259 | g_loss: 0.9910
Epoch [    6/   50] | d_loss: 1.0688 | g_loss: 1.0351
Epoch [    6/   50] | d_loss: 1.0892 | g_loss: 1.3472
Epoch [    6/   50] | d_loss: 1.1398 | g_loss: 1.3598
Epoch [    6/   50] | d_loss: 1.1152 | g_loss: 1.3271
Epoch [    6/   50] | d_loss: 1.2374 | g_loss: 0.7343
Epoch [    6/   50] | d_loss: 1.1232 | g_loss: 1.0390
```

```
Epoch [    6/   50] | d_loss: 1.1424 | g_loss: 1.1291
Epoch [    6/   50] | d_loss: 1.1164 | g_loss: 0.7909
Epoch [    6/   50] | d_loss: 0.9987 | g_loss: 1.3762
Epoch [    6/   50] | d_loss: 1.1303 | g_loss: 1.0092
Epoch [    6/   50] | d_loss: 0.9730 | g_loss: 1.4587
Epoch [    6/   50] | d_loss: 1.0669 | g_loss: 1.2146
Epoch [    6/   50] | d_loss: 1.0157 | g_loss: 1.0110
Epoch [    7/   50] | d_loss: 1.0667 | g_loss: 1.0733
Epoch [    7/   50] | d_loss: 0.9777 | g_loss: 1.1130
Epoch [    7/   50] | d_loss: 1.2548 | g_loss: 1.1802
Epoch [    7/   50] | d_loss: 1.0216 | g_loss: 1.0474
Epoch [    7/   50] | d_loss: 1.1277 | g_loss: 0.9659
Epoch [    7/   50] | d_loss: 1.0570 | g_loss: 1.0758
Epoch [    7/   50] | d_loss: 1.0779 | g_loss: 1.0216
Epoch [    7/   50] | d_loss: 0.9554 | g_loss: 1.3894
Epoch [    7/   50] | d_loss: 1.1692 | g_loss: 0.5540
Epoch [    7/   50] | d_loss: 1.0697 | g_loss: 1.5030
Epoch [    7/   50] | d_loss: 1.0544 | g_loss: 1.0014
Epoch [    7/   50] | d_loss: 1.0732 | g_loss: 1.0681
Epoch [    7/   50] | d_loss: 1.0351 | g_loss: 1.2737
Epoch [    7/   50] | d_loss: 1.0600 | g_loss: 0.7525
Epoch [    7/   50] | d_loss: 1.0697 | g_loss: 1.2118
Epoch [    8/   50] | d_loss: 1.1216 | g_loss: 1.8475
Epoch [    8/   50] | d_loss: 1.1372 | g_loss: 0.9529
Epoch [    8/   50] | d_loss: 1.7518 | g_loss: 0.5833
Epoch [    8/   50] | d_loss: 1.0656 | g_loss: 0.9271
Epoch [    8/   50] | d_loss: 1.0960 | g_loss: 1.1599
Epoch [    8/   50] | d_loss: 1.1718 | g_loss: 0.7753
Epoch [    8/   50] | d_loss: 0.9309 | g_loss: 1.3186
Epoch [    8/   50] | d_loss: 0.9357 | g_loss: 1.3183
Epoch [    8/   50] | d_loss: 1.0287 | g_loss: 1.2789
Epoch [    8/   50] | d_loss: 1.0211 | g_loss: 1.0868
Epoch [    8/   50] | d_loss: 1.1682 | g_loss: 0.9804
Epoch [    8/   50] | d_loss: 0.9966 | g_loss: 0.9446
Epoch [    8/   50] | d_loss: 1.0983 | g_loss: 1.1568
Epoch [    8/   50] | d_loss: 0.9973 | g_loss: 1.2472
Epoch [    8/   50] | d_loss: 1.1528 | g_loss: 1.2565
Epoch [    9/   50] | d_loss: 1.0143 | g_loss: 1.0932
Epoch [    9/   50] | d_loss: 1.0414 | g_loss: 1.1671
Epoch [    9/   50] | d_loss: 1.0874 | g_loss: 1.2816
Epoch [    9/   50] | d_loss: 1.0582 | g_loss: 1.0761
Epoch [    9/   50] | d_loss: 0.9886 | g_loss: 1.6358
Epoch [    9/   50] | d_loss: 1.0205 | g_loss: 2.2109
Epoch [    9/   50] | d_loss: 1.2045 | g_loss: 0.8330
Epoch [    9/   50] | d_loss: 1.1318 | g_loss: 1.0369
Epoch [    9/   50] | d_loss: 0.9771 | g_loss: 1.1848
Epoch [    9/   50] | d_loss: 1.2230 | g_loss: 1.9665
Epoch [    9/   50] | d_loss: 1.1299 | g_loss: 1.0781
Epoch [    9/   50] | d_loss: 0.9486 | g_loss: 1.4255
Epoch [    9/   50] | d_loss: 1.0164 | g_loss: 0.8856
Epoch [    9/   50] | d_loss: 1.0872 | g_loss: 1.2887
Epoch [    9/   50] | d_loss: 1.0428 | g_loss: 1.3446
Epoch [   10/   50] | d_loss: 1.2792 | g_loss: 0.7245
Epoch [   10/   50] | d_loss: 0.9842 | g_loss: 1.3240
Epoch [   10/   50] | d_loss: 0.9989 | g_loss: 1.1025
Epoch [   10/   50] | d_loss: 1.0644 | g_loss: 0.8704
Epoch [   10/   50] | d_loss: 0.8452 | g_loss: 1.5605
Epoch [   10/   50] | d_loss: 0.9137 | g_loss: 1.1540
Epoch [   10/   50] | d_loss: 1.0543 | g_loss: 1.5645
Epoch [   10/   50] | d_loss: 0.9593 | g_loss: 1.1773
Epoch [   10/   50] | d_loss: 1.0757 | g_loss: 1.4780
Epoch [   10/   50] | d_loss: 0.9325 | g_loss: 1.2383
Epoch [   10/   50] | d_loss: 1.0690 | g_loss: 1.4893
Epoch [   10/   50] | d_loss: 1.0041 | g_loss: 1.2306
Epoch [   10/   50] | d_loss: 1.2382 | g_loss: 1.6401
Epoch [   10/   50] | d_loss: 0.9639 | g_loss: 1.4021
Epoch [   10/   50] | d_loss: 1.0239 | g_loss: 0.9715
Epoch [   11/   50] | d_loss: 1.2179 | g_loss: 0.8700
Epoch [   11/   50] | d_loss: 1.0685 | g_loss: 1.6097
Epoch [   11/   50] | d_loss: 1.1488 | g_loss: 1.5823
Epoch [   11/   50] | d_loss: 1.0855 | g_loss: 1.2329
Epoch [   11/   50] | d_loss: 1.2978 | g_loss: 0.7889
Epoch [   11/   50] | d_loss: 1.1694 | g_loss: 1.2423
Epoch [   11/   50] | d_loss: 1.0491 | g_loss: 1.2579
Epoch [   11/   50] | d_loss: 1.0029 | g_loss: 0.9775
Epoch [   11/   50] | d_loss: 1.0851 | g_loss: 1.4270
Epoch [   11/   50] | d_loss: 1.0583 | g_loss: 1.1691
```

```
Epoch [   11/   50] | d_loss: 1.0239 | g_loss: 1.1147
Epoch [   11/   50] | d_loss: 1.0865 | g_loss: 1.2633
Epoch [   11/   50] | d_loss: 1.0452 | g_loss: 1.3558
Epoch [   11/   50] | d_loss: 1.0786 | g_loss: 1.3766
Epoch [   11/   50] | d_loss: 0.9551 | g_loss: 1.3139
Epoch [   12/   50] | d_loss: 1.0059 | g_loss: 1.3304
Epoch [   12/   50] | d_loss: 1.1111 | g_loss: 0.8166
Epoch [   12/   50] | d_loss: 1.0386 | g_loss: 0.9052
Epoch [   12/   50] | d_loss: 1.1011 | g_loss: 1.4359
Epoch [   12/   50] | d_loss: 1.0248 | g_loss: 1.4785
Epoch [   12/   50] | d_loss: 1.1080 | g_loss: 1.3595
Epoch [   12/   50] | d_loss: 1.0271 | g_loss: 2.1707
Epoch [   12/   50] | d_loss: 0.9165 | g_loss: 1.3252
Epoch [   12/   50] | d_loss: 0.9287 | g_loss: 1.1333
Epoch [   12/   50] | d_loss: 0.9071 | g_loss: 0.9656
Epoch [   12/   50] | d_loss: 1.0198 | g_loss: 1.2374
Epoch [   12/   50] | d_loss: 1.0346 | g_loss: 1.4328
Epoch [   12/   50] | d_loss: 1.4221 | g_loss: 0.7540
Epoch [   12/   50] | d_loss: 0.9824 | g_loss: 1.5229
Epoch [   12/   50] | d_loss: 1.0852 | g_loss: 1.0034
Epoch [   13/   50] | d_loss: 0.9682 | g_loss: 1.0249
Epoch [   13/   50] | d_loss: 1.0738 | g_loss: 1.4825
Epoch [   13/   50] | d_loss: 1.0403 | g_loss: 0.9736
Epoch [   13/   50] | d_loss: 1.0294 | g_loss: 1.6968
Epoch [   13/   50] | d_loss: 0.9406 | g_loss: 1.2916
Epoch [   13/   50] | d_loss: 1.2267 | g_loss: 0.7127
Epoch [   13/   50] | d_loss: 0.9353 | g_loss: 1.3298
Epoch [   13/   50] | d_loss: 1.0780 | g_loss: 1.3147
Epoch [   13/   50] | d_loss: 1.2211 | g_loss: 1.6009
Epoch [   13/   50] | d_loss: 1.0218 | g_loss: 1.4578
Epoch [   13/   50] | d_loss: 1.0974 | g_loss: 2.1420
Epoch [   13/   50] | d_loss: 0.9545 | g_loss: 1.5608
Epoch [   13/   50] | d_loss: 1.0942 | g_loss: 1.0520
Epoch [   13/   50] | d_loss: 0.9125 | g_loss: 0.8115
Epoch [   13/   50] | d_loss: 1.0176 | g_loss: 1.2982
Epoch [   14/   50] | d_loss: 1.4380 | g_loss: 2.5060
Epoch [   14/   50] | d_loss: 0.9358 | g_loss: 1.2237
Epoch [   14/   50] | d_loss: 1.0114 | g_loss: 1.3035
Epoch [   14/   50] | d_loss: 1.6646 | g_loss: 2.3564
Epoch [   14/   50] | d_loss: 0.9715 | g_loss: 1.3911
Epoch [   14/   50] | d_loss: 0.9647 | g_loss: 0.8797
Epoch [   14/   50] | d_loss: 1.2859 | g_loss: 0.4100
Epoch [   14/   50] | d_loss: 0.9383 | g_loss: 1.3753
Epoch [   14/   50] | d_loss: 0.9409 | g_loss: 1.2529
Epoch [   14/   50] | d_loss: 1.1390 | g_loss: 2.1416
Epoch [   14/   50] | d_loss: 0.9890 | g_loss: 1.1349
Epoch [   14/   50] | d_loss: 0.9435 | g_loss: 1.1391
Epoch [   14/   50] | d_loss: 0.9862 | g_loss: 1.9472
Epoch [   14/   50] | d_loss: 0.8665 | g_loss: 1.1939
Epoch [   14/   50] | d_loss: 0.9959 | g_loss: 1.8707
Epoch [   15/   50] | d_loss: 0.9621 | g_loss: 1.7312
Epoch [   15/   50] | d_loss: 0.9307 | g_loss: 0.9925
Epoch [   15/   50] | d_loss: 0.8639 | g_loss: 1.9943
Epoch [   15/   50] | d_loss: 0.9198 | g_loss: 1.3626
Epoch [   15/   50] | d_loss: 0.9364 | g_loss: 1.9383
Epoch [   15/   50] | d_loss: 1.0751 | g_loss: 1.0041
Epoch [   15/   50] | d_loss: 1.0132 | g_loss: 1.1206
Epoch [   15/   50] | d_loss: 1.2584 | g_loss: 0.8897
Epoch [   15/   50] | d_loss: 1.0514 | g_loss: 1.5436
Epoch [   15/   50] | d_loss: 1.0929 | g_loss: 1.3069
Epoch [   15/   50] | d_loss: 0.8534 | g_loss: 1.2582
Epoch [   15/   50] | d_loss: 0.9822 | g_loss: 0.9641
Epoch [   15/   50] | d_loss: 0.8491 | g_loss: 1.5715
Epoch [   15/   50] | d_loss: 1.0551 | g_loss: 1.5236
Epoch [   15/   50] | d_loss: 0.9724 | g_loss: 1.4228
Epoch [   16/   50] | d_loss: 0.9185 | g_loss: 1.2036
Epoch [   16/   50] | d_loss: 0.8493 | g_loss: 1.7120
Epoch [   16/   50] | d_loss: 1.0019 | g_loss: 1.5241
Epoch [   16/   50] | d_loss: 0.8892 | g_loss: 1.2706
Epoch [   16/   50] | d_loss: 0.8576 | g_loss: 2.1180
Epoch [   16/   50] | d_loss: 0.9765 | g_loss: 1.0879
Epoch [   16/   50] | d_loss: 1.1340 | g_loss: 0.7975
Epoch [   16/   50] | d_loss: 0.9379 | g_loss: 1.3617
Epoch [   16/   50] | d_loss: 0.9821 | g_loss: 1.0577
Epoch [   16/   50] | d_loss: 0.8699 | g_loss: 1.3425
Epoch [   16/   50] | d_loss: 0.8671 | g_loss: 1.1927
Epoch [   16/   50] | d_loss: 0.9212 | g_loss: 0.9036
```

```
Epoch [   16/   50] | d_loss: 0.8527 | g_loss: 1.7005
Epoch [   16/   50] | d_loss: 1.1170 | g_loss: 1.4142
Epoch [   16/   50] | d_loss: 0.9150 | g_loss: 2.0495
Epoch [   17/   50] | d_loss: 0.8466 | g_loss: 1.8015
Epoch [   17/   50] | d_loss: 1.0538 | g_loss: 1.2642
Epoch [   17/   50] | d_loss: 0.8609 | g_loss: 1.5033
Epoch [   17/   50] | d_loss: 0.9689 | g_loss: 1.1680
Epoch [   17/   50] | d_loss: 1.0572 | g_loss: 0.5953
Epoch [   17/   50] | d_loss: 0.8677 | g_loss: 1.2555
Epoch [   17/   50] | d_loss: 0.8176 | g_loss: 1.3330
Epoch [   17/   50] | d_loss: 0.8472 | g_loss: 1.4200
Epoch [   17/   50] | d_loss: 1.0421 | g_loss: 1.8535
Epoch [   17/   50] | d_loss: 0.7338 | g_loss: 2.1656
Epoch [   17/   50] | d_loss: 0.8567 | g_loss: 1.3439
Epoch [   17/   50] | d_loss: 0.8826 | g_loss: 1.7266
Epoch [   17/   50] | d_loss: 1.0018 | g_loss: 1.2964
Epoch [   17/   50] | d_loss: 0.7778 | g_loss: 1.7199
Epoch [   17/   50] | d_loss: 0.7774 | g_loss: 1.4088
Epoch [   18/   50] | d_loss: 0.8322 | g_loss: 1.4298
Epoch [   18/   50] | d_loss: 0.8842 | g_loss: 1.7010
Epoch [   18/   50] | d_loss: 0.7473 | g_loss: 1.5784
Epoch [   18/   50] | d_loss: 0.7206 | g_loss: 1.3732
Epoch [   18/   50] | d_loss: 0.7627 | g_loss: 1.8478
Epoch [   18/   50] | d_loss: 0.8768 | g_loss: 1.0986
Epoch [   18/   50] | d_loss: 0.7459 | g_loss: 1.2763
Epoch [   18/   50] | d_loss: 0.8341 | g_loss: 1.6603
Epoch [   18/   50] | d_loss: 0.8042 | g_loss: 1.6962
Epoch [   18/   50] | d_loss: 0.9488 | g_loss: 1.7019
Epoch [   18/   50] | d_loss: 0.7483 | g_loss: 1.8310
Epoch [   18/   50] | d_loss: 0.9225 | g_loss: 1.8461
Epoch [   18/   50] | d_loss: 0.7878 | g_loss: 1.6959
Epoch [   18/   50] | d_loss: 2.1277 | g_loss: 0.6834
Epoch [   18/   50] | d_loss: 0.9100 | g_loss: 1.1633
Epoch [   19/   50] | d_loss: 0.6934 | g_loss: 1.7444
Epoch [   19/   50] | d_loss: 0.8343 | g_loss: 1.1235
Epoch [   19/   50] | d_loss: 0.9019 | g_loss: 1.8050
Epoch [   19/   50] | d_loss: 0.8799 | g_loss: 1.6662
Epoch [   19/   50] | d_loss: 0.7717 | g_loss: 1.7707
Epoch [   19/   50] | d_loss: 0.8494 | g_loss: 2.1788
Epoch [   19/   50] | d_loss: 0.8166 | g_loss: 1.5822
Epoch [   19/   50] | d_loss: 0.7486 | g_loss: 1.2908
Epoch [   19/   50] | d_loss: 1.0159 | g_loss: 1.8541
Epoch [   19/   50] | d_loss: 0.8698 | g_loss: 1.6137
Epoch [   19/   50] | d_loss: 0.8749 | g_loss: 1.7306
Epoch [   19/   50] | d_loss: 0.7798 | g_loss: 1.3884
Epoch [   19/   50] | d_loss: 0.7870 | g_loss: 1.9231
Epoch [   19/   50] | d_loss: 0.7576 | g_loss: 1.3905
Epoch [   19/   50] | d_loss: 0.9672 | g_loss: 2.2782
Epoch [   20/   50] | d_loss: 0.8767 | g_loss: 1.4274
Epoch [   20/   50] | d_loss: 0.8322 | g_loss: 1.2624
Epoch [   20/   50] | d_loss: 0.6156 | g_loss: 2.6605
Epoch [   20/   50] | d_loss: 1.1690 | g_loss: 2.9659
Epoch [   20/   50] | d_loss: 0.8887 | g_loss: 2.0871
Epoch [   20/   50] | d_loss: 0.8731 | g_loss: 1.9802
Epoch [   20/   50] | d_loss: 0.9295 | g_loss: 0.9880
Epoch [   20/   50] | d_loss: 0.8274 | g_loss: 2.0073
Epoch [   20/   50] | d_loss: 0.8096 | g_loss: 1.0764
Epoch [   20/   50] | d_loss: 0.7858 | g_loss: 1.0312
Epoch [   20/   50] | d_loss: 0.7569 | g_loss: 1.2673
Epoch [   20/   50] | d_loss: 0.7028 | g_loss: 2.1658
Epoch [   20/   50] | d_loss: 0.8085 | g_loss: 1.3011
Epoch [   20/   50] | d_loss: 0.9018 | g_loss: 1.2305
Epoch [   20/   50] | d_loss: 0.7548 | g_loss: 1.2539
Epoch [   21/   50] | d_loss: 0.8261 | g_loss: 1.4732
Epoch [   21/   50] | d_loss: 0.7577 | g_loss: 1.6516
Epoch [   21/   50] | d_loss: 0.7396 | g_loss: 1.9998
Epoch [   21/   50] | d_loss: 0.7746 | g_loss: 1.4535
Epoch [   21/   50] | d_loss: 0.8287 | g_loss: 1.2130
Epoch [   21/   50] | d_loss: 0.7144 | g_loss: 2.0344
Epoch [   21/   50] | d_loss: 0.8328 | g_loss: 1.5716
Epoch [   21/   50] | d_loss: 0.7481 | g_loss: 1.8329
Epoch [   21/   50] | d_loss: 0.8679 | g_loss: 1.1395
Epoch [   21/   50] | d_loss: 0.7214 | g_loss: 2.1340
Epoch [   21/   50] | d_loss: 1.3679 | g_loss: 2.3286
Epoch [   21/   50] | d_loss: 0.8000 | g_loss: 1.5879
Epoch [   21/   50] | d_loss: 0.7762 | g_loss: 1.8969
Epoch [   21/   50] | d_loss: 0.7351 | g_loss: 1.6515
```

```
Epoch [   21/   50] | d_loss: 0.7466 | g_loss: 1.4900
Epoch [   22/   50] | d_loss: 0.7427 | g_loss: 1.6793
Epoch [   22/   50] | d_loss: 0.7626 | g_loss: 1.7536
Epoch [   22/   50] | d_loss: 0.7708 | g_loss: 1.8561
Epoch [   22/   50] | d_loss: 0.8377 | g_loss: 1.1521
Epoch [   22/   50] | d_loss: 0.7834 | g_loss: 2.1342
Epoch [   22/   50] | d_loss: 0.8345 | g_loss: 1.7704
Epoch [   22/   50] | d_loss: 1.0465 | g_loss: 2.2474
Epoch [   22/   50] | d_loss: 0.8238 | g_loss: 1.8242
Epoch [   22/   50] | d_loss: 0.6753 | g_loss: 1.6183
Epoch [   22/   50] | d_loss: 0.7523 | g_loss: 1.9291
Epoch [   22/   50] | d_loss: 0.7290 | g_loss: 1.5809
Epoch [   22/   50] | d_loss: 0.7173 | g_loss: 2.0359
Epoch [   22/   50] | d_loss: 0.7647 | g_loss: 1.6646
Epoch [   22/   50] | d_loss: 0.6858 | g_loss: 1.7307
Epoch [   22/   50] | d_loss: 0.7321 | g_loss: 1.0087
Epoch [   23/   50] | d_loss: 0.7209 | g_loss: 1.4217
Epoch [   23/   50] | d_loss: 0.8144 | g_loss: 1.4477
Epoch [   23/   50] | d_loss: 0.7913 | g_loss: 1.2732
Epoch [   23/   50] | d_loss: 0.7198 | g_loss: 1.8160
Epoch [   23/   50] | d_loss: 0.8165 | g_loss: 1.9423
Epoch [   23/   50] | d_loss: 0.7563 | g_loss: 1.6457
Epoch [   23/   50] | d_loss: 0.7804 | g_loss: 1.6885
Epoch [   23/   50] | d_loss: 1.4537 | g_loss: 2.9147
Epoch [   23/   50] | d_loss: 0.8327 | g_loss: 1.6257
Epoch [   23/   50] | d_loss: 0.8847 | g_loss: 1.2552
Epoch [   23/   50] | d_loss: 0.7466 | g_loss: 1.3427
Epoch [   23/   50] | d_loss: 0.7227 | g_loss: 1.3252
Epoch [   23/   50] | d_loss: 0.7751 | g_loss: 1.9311
Epoch [   23/   50] | d_loss: 0.8762 | g_loss: 1.4697
Epoch [   23/   50] | d_loss: 0.6955 | g_loss: 1.5108
Epoch [   24/   50] | d_loss: 0.7228 | g_loss: 2.4381
Epoch [   24/   50] | d_loss: 0.8026 | g_loss: 1.4984
Epoch [   24/   50] | d_loss: 0.7187 | g_loss: 2.1321
Epoch [   24/   50] | d_loss: 0.9604 | g_loss: 2.6909
Epoch [   24/   50] | d_loss: 0.8199 | g_loss: 1.4289
Epoch [   24/   50] | d_loss: 0.5551 | g_loss: 2.2802
Epoch [   24/   50] | d_loss: 0.8159 | g_loss: 1.5906
Epoch [   24/   50] | d_loss: 0.7387 | g_loss: 1.3637
Epoch [   24/   50] | d_loss: 0.8168 | g_loss: 1.6366
Epoch [   24/   50] | d_loss: 1.1573 | g_loss: 2.7768
Epoch [   24/   50] | d_loss: 0.7414 | g_loss: 2.0225
Epoch [   24/   50] | d_loss: 0.7527 | g_loss: 1.6936
Epoch [   24/   50] | d_loss: 0.8473 | g_loss: 0.9549
Epoch [   24/   50] | d_loss: 0.7520 | g_loss: 1.8262
Epoch [   24/   50] | d_loss: 0.9355 | g_loss: 1.3578
Epoch [   25/   50] | d_loss: 0.8402 | g_loss: 1.8687
Epoch [   25/   50] | d_loss: 0.5680 | g_loss: 2.1319
Epoch [   25/   50] | d_loss: 0.6606 | g_loss: 1.7954
Epoch [   25/   50] | d_loss: 0.6630 | g_loss: 2.0036
Epoch [   25/   50] | d_loss: 0.8117 | g_loss: 2.0502
Epoch [   25/   50] | d_loss: 0.6581 | g_loss: 1.7634
Epoch [   25/   50] | d_loss: 0.7529 | g_loss: 1.5529
Epoch [   25/   50] | d_loss: 1.2450 | g_loss: 0.5116
Epoch [   25/   50] | d_loss: 0.6636 | g_loss: 1.6034
Epoch [   25/   50] | d_loss: 0.8052 | g_loss: 1.3867
Epoch [   25/   50] | d_loss: 2.6577 | g_loss: 0.5409
Epoch [   25/   50] | d_loss: 0.6046 | g_loss: 1.6364
Epoch [   25/   50] | d_loss: 0.6195 | g_loss: 1.8290
Epoch [   25/   50] | d_loss: 0.7034 | g_loss: 1.4489
Epoch [   25/   50] | d_loss: 0.6397 | g_loss: 2.4628
Epoch [   26/   50] | d_loss: 0.7419 | g_loss: 2.3805
Epoch [   26/   50] | d_loss: 1.6484 | g_loss: 1.0762
Epoch [   26/   50] | d_loss: 0.5708 | g_loss: 2.4646
Epoch [   26/   50] | d_loss: 0.7550 | g_loss: 2.1373
Epoch [   26/   50] | d_loss: 0.7473 | g_loss: 1.4021
Epoch [   26/   50] | d_loss: 0.8549 | g_loss: 1.8228
Epoch [   26/   50] | d_loss: 0.8013 | g_loss: 2.8779
Epoch [   26/   50] | d_loss: 0.7538 | g_loss: 1.9934
Epoch [   26/   50] | d_loss: 0.7619 | g_loss: 1.6403
Epoch [   26/   50] | d_loss: 0.7066 | g_loss: 1.6229
Epoch [   26/   50] | d_loss: 2.1542 | g_loss: 2.4406
Epoch [   26/   50] | d_loss: 0.6846 | g_loss: 1.9039
Epoch [   26/   50] | d_loss: 0.7892 | g_loss: 1.7129
Epoch [   26/   50] | d_loss: 0.7514 | g_loss: 1.2716
Epoch [   26/   50] | d_loss: 0.7020 | g_loss: 1.4563
Epoch [   27/   50] | d_loss: 0.7013 | g_loss: 1.5407
```

```
Epoch [   27/   50] | d_loss: 0.5953 | g_loss: 3.0721
Epoch [   27/   50] | d_loss: 0.6802 | g_loss: 1.6339
Epoch [   27/   50] | d_loss: 1.1737 | g_loss: 1.3882
Epoch [   27/   50] | d_loss: 0.7521 | g_loss: 1.9349
Epoch [   27/   50] | d_loss: 0.7004 | g_loss: 2.0645
Epoch [   27/   50] | d_loss: 0.6860 | g_loss: 1.7766
Epoch [   27/   50] | d_loss: 0.7612 | g_loss: 1.7339
Epoch [   27/   50] | d_loss: 0.7071 | g_loss: 1.5748
Epoch [   27/   50] | d_loss: 0.6707 | g_loss: 1.6045
Epoch [   27/   50] | d_loss: 0.6394 | g_loss: 2.5464
Epoch [   27/   50] | d_loss: 0.7046 | g_loss: 1.8503
Epoch [   27/   50] | d_loss: 0.6954 | g_loss: 2.1335
Epoch [   27/   50] | d_loss: 0.7914 | g_loss: 2.6767
Epoch [   27/   50] | d_loss: 0.5524 | g_loss: 2.1796
Epoch [   28/   50] | d_loss: 0.6035 | g_loss: 2.1807
Epoch [   28/   50] | d_loss: 0.7567 | g_loss: 1.4959
Epoch [   28/   50] | d_loss: 0.7180 | g_loss: 1.2287
Epoch [   28/   50] | d_loss: 0.6206 | g_loss: 1.9733
Epoch [   28/   50] | d_loss: 0.7122 | g_loss: 2.5398
Epoch [   28/   50] | d_loss: 0.7209 | g_loss: 1.7040
Epoch [   28/   50] | d_loss: 0.6978 | g_loss: 1.9724
Epoch [   28/   50] | d_loss: 0.7108 | g_loss: 1.3120
Epoch [   28/   50] | d_loss: 0.6150 | g_loss: 1.7519
Epoch [   28/   50] | d_loss: 0.8772 | g_loss: 2.2688
Epoch [   28/   50] | d_loss: 0.6488 | g_loss: 2.0574
Epoch [   28/   50] | d_loss: 0.6264 | g_loss: 2.0973
Epoch [   28/   50] | d_loss: 0.6545 | g_loss: 1.8657
Epoch [   28/   50] | d_loss: 0.6977 | g_loss: 2.4461
Epoch [   28/   50] | d_loss: 0.6678 | g_loss: 1.5275
Epoch [   29/   50] | d_loss: 0.6581 | g_loss: 1.7286
Epoch [   29/   50] | d_loss: 0.9767 | g_loss: 1.2012
Epoch [   29/   50] | d_loss: 0.8242 | g_loss: 1.8711
Epoch [   29/   50] | d_loss: 0.6855 | g_loss: 2.2215
Epoch [   29/   50] | d_loss: 0.6889 | g_loss: 2.0590
Epoch [   29/   50] | d_loss: 0.6906 | g_loss: 1.5275
Epoch [   29/   50] | d_loss: 0.6402 | g_loss: 2.3739
Epoch [   29/   50] | d_loss: 0.6045 | g_loss: 1.8806
Epoch [   29/   50] | d_loss: 0.7220 | g_loss: 2.0914
Epoch [   29/   50] | d_loss: 0.8404 | g_loss: 0.9200
Epoch [   29/   50] | d_loss: 0.6887 | g_loss: 2.0687
Epoch [   29/   50] | d_loss: 0.6638 | g_loss: 2.2294
Epoch [   29/   50] | d_loss: 0.6195 | g_loss: 2.1870
Epoch [   29/   50] | d_loss: 1.2523 | g_loss: 2.8802
Epoch [   29/   50] | d_loss: 0.6168 | g_loss: 1.4134
Epoch [   30/   50] | d_loss: 0.7302 | g_loss: 1.6510
Epoch [   30/   50] | d_loss: 0.7215 | g_loss: 1.5831
Epoch [   30/   50] | d_loss: 0.5616 | g_loss: 1.9297
Epoch [   30/   50] | d_loss: 0.5622 | g_loss: 2.1312
Epoch [   30/   50] | d_loss: 0.5669 | g_loss: 1.8583
Epoch [   30/   50] | d_loss: 0.9904 | g_loss: 0.9772
Epoch [   30/   50] | d_loss: 0.7390 | g_loss: 2.1908
Epoch [   30/   50] | d_loss: 0.6517 | g_loss: 1.6705
Epoch [   30/   50] | d_loss: 1.0237 | g_loss: 1.9305
Epoch [   30/   50] | d_loss: 0.5928 | g_loss: 2.1368
Epoch [   30/   50] | d_loss: 0.6199 | g_loss: 2.2122
Epoch [   30/   50] | d_loss: 0.5850 | g_loss: 2.7180
Epoch [   30/   50] | d_loss: 0.6510 | g_loss: 1.9008
Epoch [   30/   50] | d_loss: 0.6969 | g_loss: 1.5290
Epoch [   30/   50] | d_loss: 0.6134 | g_loss: 1.8459
Epoch [   31/   50] | d_loss: 0.5974 | g_loss: 1.7216
Epoch [   31/   50] | d_loss: 0.6341 | g_loss: 1.9907
Epoch [   31/   50] | d_loss: 0.6099 | g_loss: 1.9271
Epoch [   31/   50] | d_loss: 0.6129 | g_loss: 2.0216
Epoch [   31/   50] | d_loss: 0.5991 | g_loss: 1.9337
Epoch [   31/   50] | d_loss: 0.5585 | g_loss: 1.9553
Epoch [   31/   50] | d_loss: 0.7846 | g_loss: 1.2955
Epoch [   31/   50] | d_loss: 0.6353 | g_loss: 1.6243
Epoch [   31/   50] | d_loss: 0.6330 | g_loss: 2.5873
Epoch [   31/   50] | d_loss: 0.6109 | g_loss: 1.6089
Epoch [   31/   50] | d_loss: 0.6616 | g_loss: 2.2466
Epoch [   31/   50] | d_loss: 0.6301 | g_loss: 2.7044
Epoch [   31/   50] | d_loss: 0.6137 | g_loss: 2.4189
Epoch [   31/   50] | d_loss: 0.5105 | g_loss: 2.5655
Epoch [   31/   50] | d_loss: 0.6271 | g_loss: 2.0810
Epoch [   32/   50] | d_loss: 0.7248 | g_loss: 1.8151
Epoch [   32/   50] | d_loss: 0.6630 | g_loss: 2.3213
Epoch [   32/   50] | d_loss: 0.7955 | g_loss: 1.1989
```

```
Epoch [   32/   50] | d_loss: 0.6508 | g_loss: 1.8599
Epoch [   32/   50] | d_loss: 1.5613 | g_loss: 0.5189
Epoch [   32/   50] | d_loss: 0.6296 | g_loss: 2.8091
Epoch [   32/   50] | d_loss: 0.5835 | g_loss: 1.8432
Epoch [   32/   50] | d_loss: 0.7221 | g_loss: 1.3847
Epoch [   32/   50] | d_loss: 0.6472 | g_loss: 3.0226
Epoch [   32/   50] | d_loss: 0.7887 | g_loss: 2.7835
Epoch [   32/   50] | d_loss: 2.6977 | g_loss: 1.0871
Epoch [   32/   50] | d_loss: 0.6082 | g_loss: 2.1554
Epoch [   32/   50] | d_loss: 0.5455 | g_loss: 2.8749
Epoch [   32/   50] | d_loss: 0.5093 | g_loss: 1.8969
Epoch [   32/   50] | d_loss: 0.5867 | g_loss: 2.0972
Epoch [   33/   50] | d_loss: 0.6174 | g_loss: 2.3340
Epoch [   33/   50] | d_loss: 0.5989 | g_loss: 2.3392
Epoch [   33/   50] | d_loss: 0.7176 | g_loss: 1.6300
Epoch [   33/   50] | d_loss: 0.7019 | g_loss: 2.1026
Epoch [   33/   50] | d_loss: 0.6797 | g_loss: 2.0564
Epoch [   33/   50] | d_loss: 3.1152 | g_loss: 0.4512
Epoch [   33/   50] | d_loss: 0.6553 | g_loss: 1.7715
Epoch [   33/   50] | d_loss: 0.5875 | g_loss: 2.4740
Epoch [   33/   50] | d_loss: 0.8499 | g_loss: 2.2775
Epoch [   33/   50] | d_loss: 0.6328 | g_loss: 1.9425
Epoch [   33/   50] | d_loss: 0.6423 | g_loss: 2.6230
Epoch [   33/   50] | d_loss: 0.8137 | g_loss: 1.6264
Epoch [   33/   50] | d_loss: 0.6041 | g_loss: 2.4448
Epoch [   33/   50] | d_loss: 0.5208 | g_loss: 2.5418
Epoch [   33/   50] | d_loss: 0.5152 | g_loss: 2.1192
Epoch [   34/   50] | d_loss: 0.6553 | g_loss: 2.5315
Epoch [   34/   50] | d_loss: 0.7979 | g_loss: 2.1117
Epoch [   34/   50] | d_loss: 0.5733 | g_loss: 2.3880
Epoch [   34/   50] | d_loss: 0.7619 | g_loss: 2.9044
Epoch [   34/   50] | d_loss: 0.7428 | g_loss: 1.6053
Epoch [   34/   50] | d_loss: 0.6492 | g_loss: 2.0597
Epoch [   34/   50] | d_loss: 0.6619 | g_loss: 1.9165
Epoch [   34/   50] | d_loss: 0.6011 | g_loss: 2.3023
Epoch [   34/   50] | d_loss: 0.5773 | g_loss: 1.8471
Epoch [   34/   50] | d_loss: 0.6124 | g_loss: 1.6027
Epoch [   34/   50] | d_loss: 1.0234 | g_loss: 0.8647
Epoch [   34/   50] | d_loss: 0.5639 | g_loss: 2.2671
Epoch [   34/   50] | d_loss: 0.7076 | g_loss: 3.4603
Epoch [   34/   50] | d_loss: 0.5554 | g_loss: 2.1699
Epoch [   34/   50] | d_loss: 0.5929 | g_loss: 2.2328
Epoch [   35/   50] | d_loss: 0.5445 | g_loss: 2.7977
Epoch [   35/   50] | d_loss: 0.7875 | g_loss: 2.0613
Epoch [   35/   50] | d_loss: 0.5789 | g_loss: 2.2298
Epoch [   35/   50] | d_loss: 0.5901 | g_loss: 2.1732
Epoch [   35/   50] | d_loss: 0.6062 | g_loss: 2.3507
Epoch [   35/   50] | d_loss: 0.5096 | g_loss: 2.2867
Epoch [   35/   50] | d_loss: 0.6062 | g_loss: 2.4638
Epoch [   35/   50] | d_loss: 0.7029 | g_loss: 2.7275
Epoch [   35/   50] | d_loss: 1.4477 | g_loss: 1.0869
Epoch [   35/   50] | d_loss: 0.5446 | g_loss: 2.6444
Epoch [   35/   50] | d_loss: 0.6373 | g_loss: 2.7342
Epoch [   35/   50] | d_loss: 0.5809 | g_loss: 2.0282
Epoch [   35/   50] | d_loss: 0.6292 | g_loss: 2.1238
Epoch [   35/   50] | d_loss: 0.5798 | g_loss: 2.1546
Epoch [   35/   50] | d_loss: 0.5962 | g_loss: 1.9759
Epoch [   36/   50] | d_loss: 0.6268 | g_loss: 1.7149
Epoch [   36/   50] | d_loss: 0.5280 | g_loss: 2.9174
Epoch [   36/   50] | d_loss: 0.4827 | g_loss: 3.0521
Epoch [   36/   50] | d_loss: 0.5493 | g_loss: 2.3120
Epoch [   36/   50] | d_loss: 0.7072 | g_loss: 1.9641
Epoch [   36/   50] | d_loss: 0.6031 | g_loss: 2.4420
Epoch [   36/   50] | d_loss: 0.5222 | g_loss: 2.5112
Epoch [   36/   50] | d_loss: 0.5455 | g_loss: 2.3540
Epoch [   36/   50] | d_loss: 0.5776 | g_loss: 2.2090
Epoch [   36/   50] | d_loss: 0.6614 | g_loss: 2.4425
Epoch [   36/   50] | d_loss: 0.6470 | g_loss: 2.4864
Epoch [   36/   50] | d_loss: 1.0137 | g_loss: 0.9081
Epoch [   36/   50] | d_loss: 0.5700 | g_loss: 1.7656
Epoch [   36/   50] | d_loss: 0.5820 | g_loss: 1.9743
Epoch [   36/   50] | d_loss: 0.5618 | g_loss: 2.3135
Epoch [   37/   50] | d_loss: 0.5908 | g_loss: 1.6255
Epoch [   37/   50] | d_loss: 0.5402 | g_loss: 2.2013
Epoch [   37/   50] | d_loss: 0.5720 | g_loss: 2.4093
Epoch [   37/   50] | d_loss: 0.6389 | g_loss: 2.0931
Epoch [   37/   50] | d_loss: 0.5063 | g_loss: 2.8580
```

```
Epoch [   37/   50] | d_loss: 0.6463 | g_loss: 2.5440
Epoch [   37/   50] | d_loss: 0.6871 | g_loss: 2.0068
Epoch [   37/   50] | d_loss: 0.6381 | g_loss: 1.9925
Epoch [   37/   50] | d_loss: 0.5621 | g_loss: 2.0818
Epoch [   37/   50] | d_loss: 0.5364 | g_loss: 2.8157
Epoch [   37/   50] | d_loss: 1.8194 | g_loss: 0.4452
Epoch [   37/   50] | d_loss: 0.6636 | g_loss: 2.2076
Epoch [   37/   50] | d_loss: 0.6279 | g_loss: 1.9460
Epoch [   37/   50] | d_loss: 0.5372 | g_loss: 2.2121
Epoch [   37/   50] | d_loss: 1.3863 | g_loss: 1.7113
Epoch [   38/   50] | d_loss: 1.3112 | g_loss: 4.1562
Epoch [   38/   50] | d_loss: 0.6039 | g_loss: 1.9227
Epoch [   38/   50] | d_loss: 0.5336 | g_loss: 2.7656
Epoch [   38/   50] | d_loss: 0.7062 | g_loss: 1.8508
Epoch [   38/   50] | d_loss: 0.5547 | g_loss: 2.2605
Epoch [   38/   50] | d_loss: 0.5509 | g_loss: 2.6092
Epoch [   38/   50] | d_loss: 0.5686 | g_loss: 2.2067
Epoch [   38/   50] | d_loss: 0.7279 | g_loss: 2.0921
Epoch [   38/   50] | d_loss: 0.6886 | g_loss: 1.0559
Epoch [   38/   50] | d_loss: 0.6666 | g_loss: 1.8081
Epoch [   38/   50] | d_loss: 0.4781 | g_loss: 3.4146
Epoch [   38/   50] | d_loss: 0.8442 | g_loss: 1.3241
Epoch [   38/   50] | d_loss: 0.6286 | g_loss: 2.1500
Epoch [   38/   50] | d_loss: 0.4795 | g_loss: 2.5837
Epoch [   38/   50] | d_loss: 0.6131 | g_loss: 1.8537
Epoch [   39/   50] | d_loss: 0.5156 | g_loss: 2.0873
Epoch [   39/   50] | d_loss: 0.6629 | g_loss: 2.6813
Epoch [   39/   50] | d_loss: 0.4925 | g_loss: 2.4045
Epoch [   39/   50] | d_loss: 0.4847 | g_loss: 2.1893
Epoch [   39/   50] | d_loss: 0.5632 | g_loss: 1.3187
Epoch [   39/   50] | d_loss: 0.4962 | g_loss: 2.5545
Epoch [   39/   50] | d_loss: 0.5618 | g_loss: 2.6062
Epoch [   39/   50] | d_loss: 0.5862 | g_loss: 2.1264
Epoch [   39/   50] | d_loss: 0.5271 | g_loss: 2.6060
Epoch [   39/   50] | d_loss: 0.5756 | g_loss: 2.0906
Epoch [   39/   50] | d_loss: 0.6756 | g_loss: 1.3022
Epoch [   39/   50] | d_loss: 0.5774 | g_loss: 1.8162
Epoch [   39/   50] | d_loss: 0.5863 | g_loss: 1.8815
Epoch [   39/   50] | d_loss: 0.7201 | g_loss: 3.0063
Epoch [   39/   50] | d_loss: 0.6530 | g_loss: 2.0948
Epoch [   40/   50] | d_loss: 0.5927 | g_loss: 2.9169
Epoch [   40/   50] | d_loss: 0.5253 | g_loss: 2.7086
Epoch [   40/   50] | d_loss: 0.8209 | g_loss: 2.7351
Epoch [   40/   50] | d_loss: 0.4925 | g_loss: 2.5514
Epoch [   40/   50] | d_loss: 0.6021 | g_loss: 3.0800
Epoch [   40/   50] | d_loss: 0.9058 | g_loss: 2.4338
Epoch [   40/   50] | d_loss: 0.5884 | g_loss: 2.1550
Epoch [   40/   50] | d_loss: 0.7430 | g_loss: 2.9052
Epoch [   40/   50] | d_loss: 0.5158 | g_loss: 2.3572
Epoch [   40/   50] | d_loss: 0.4844 | g_loss: 2.4799
Epoch [   40/   50] | d_loss: 0.5233 | g_loss: 2.2882
Epoch [   40/   50] | d_loss: 1.2771 | g_loss: 0.8060
Epoch [   40/   50] | d_loss: 0.4958 | g_loss: 2.7701
Epoch [   40/   50] | d_loss: 0.5196 | g_loss: 2.0733
Epoch [   40/   50] | d_loss: 1.6987 | g_loss: 1.0891
Epoch [   41/   50] | d_loss: 0.8508 | g_loss: 1.8465
Epoch [   41/   50] | d_loss: 0.7019 | g_loss: 3.1725
Epoch [   41/   50] | d_loss: 0.5539 | g_loss: 2.5301
Epoch [   41/   50] | d_loss: 0.5057 | g_loss: 2.2120
Epoch [   41/   50] | d_loss: 0.8142 | g_loss: 1.3575
Epoch [   41/   50] | d_loss: 0.8702 | g_loss: 1.2726
Epoch [   41/   50] | d_loss: 0.5538 | g_loss: 2.9199
Epoch [   41/   50] | d_loss: 0.6387 | g_loss: 2.0454
Epoch [   41/   50] | d_loss: 0.4557 | g_loss: 3.2501
Epoch [   41/   50] | d_loss: 0.5137 | g_loss: 1.9699
Epoch [   41/   50] | d_loss: 0.5653 | g_loss: 1.8323
Epoch [   41/   50] | d_loss: 0.5952 | g_loss: 2.9854
Epoch [   41/   50] | d_loss: 0.5202 | g_loss: 2.4865
Epoch [   41/   50] | d_loss: 0.5457 | g_loss: 2.9368
Epoch [   41/   50] | d_loss: 0.6386 | g_loss: 2.0123
Epoch [   42/   50] | d_loss: 0.4558 | g_loss: 2.6490
Epoch [   42/   50] | d_loss: 0.5374 | g_loss: 2.6722
Epoch [   42/   50] | d_loss: 0.6179 | g_loss: 2.3538
Epoch [   42/   50] | d_loss: 1.5855 | g_loss: 0.9444
Epoch [   42/   50] | d_loss: 0.6435 | g_loss: 2.6582
Epoch [   42/   50] | d_loss: 0.6864 | g_loss: 1.9784
Epoch [   42/   50] | d_loss: 0.5280 | g_loss: 2.3455
```

```
Epoch [   42/   50] | d_loss: 0.4877 | g_loss: 2.7216
Epoch [   42/   50] | d_loss: 0.5556 | g_loss: 2.6101
Epoch [   42/   50] | d_loss: 0.4611 | g_loss: 2.5203
Epoch [   42/   50] | d_loss: 0.7452 | g_loss: 2.0703
Epoch [   42/   50] | d_loss: 0.5663 | g_loss: 2.0295
Epoch [   42/   50] | d_loss: 0.8777 | g_loss: 1.5580
Epoch [   42/   50] | d_loss: 0.5974 | g_loss: 2.2120
Epoch [   42/   50] | d_loss: 0.5546 | g_loss: 2.6218
Epoch [   43/   50] | d_loss: 0.4674 | g_loss: 2.8116
Epoch [   43/   50] | d_loss: 0.4797 | g_loss: 2.4619
Epoch [   43/   50] | d_loss: 0.6061 | g_loss: 2.0729
Epoch [   43/   50] | d_loss: 0.6982 | g_loss: 2.7130
Epoch [   43/   50] | d_loss: 0.4938 | g_loss: 2.3620
Epoch [   43/   50] | d_loss: 0.5589 | g_loss: 2.3743
Epoch [   43/   50] | d_loss: 0.5374 | g_loss: 2.6289
Epoch [   43/   50] | d_loss: 1.4039 | g_loss: 1.0562
Epoch [   43/   50] | d_loss: 0.5606 | g_loss: 2.3901
Epoch [   43/   50] | d_loss: 0.4826 | g_loss: 2.6973
Epoch [   43/   50] | d_loss: 0.5291 | g_loss: 2.5675
Epoch [   43/   50] | d_loss: 0.4457 | g_loss: 3.2118
Epoch [   43/   50] | d_loss: 0.5125 | g_loss: 2.4147
Epoch [   43/   50] | d_loss: 0.5548 | g_loss: 3.1130
Epoch [   43/   50] | d_loss: 1.3459 | g_loss: 3.5996
Epoch [   44/   50] | d_loss: 0.7039 | g_loss: 2.1132
Epoch [   44/   50] | d_loss: 0.5329 | g_loss: 2.7696
Epoch [   44/   50] | d_loss: 0.5442 | g_loss: 2.2038
Epoch [   44/   50] | d_loss: 0.6025 | g_loss: 1.9156
Epoch [   44/   50] | d_loss: 0.5501 | g_loss: 2.1635
Epoch [   44/   50] | d_loss: 0.6082 | g_loss: 2.2494
Epoch [   44/   50] | d_loss: 0.6271 | g_loss: 2.6277
Epoch [   44/   50] | d_loss: 0.5499 | g_loss: 2.0160
Epoch [   44/   50] | d_loss: 0.4892 | g_loss: 2.8039
Epoch [   44/   50] | d_loss: 0.5807 | g_loss: 2.0382
Epoch [   44/   50] | d_loss: 0.5710 | g_loss: 2.0656
Epoch [   44/   50] | d_loss: 0.5907 | g_loss: 2.6030
Epoch [   44/   50] | d_loss: 0.5352 | g_loss: 2.3020
Epoch [   44/   50] | d_loss: 0.4817 | g_loss: 3.5934
Epoch [   44/   50] | d_loss: 0.6696 | g_loss: 1.9223
Epoch [   45/   50] | d_loss: 0.5486 | g_loss: 2.0731
Epoch [   45/   50] | d_loss: 0.5003 | g_loss: 2.5706
Epoch [   45/   50] | d_loss: 0.5018 | g_loss: 2.3340
Epoch [   45/   50] | d_loss: 0.5619 | g_loss: 2.2965
Epoch [   45/   50] | d_loss: 0.4846 | g_loss: 2.6043
Epoch [   45/   50] | d_loss: 0.7211 | g_loss: 3.0058
Epoch [   45/   50] | d_loss: 1.5625 | g_loss: 4.7523
Epoch [   45/   50] | d_loss: 0.5801 | g_loss: 2.2277
Epoch [   45/   50] | d_loss: 0.7921 | g_loss: 2.6978
Epoch [   45/   50] | d_loss: 0.5280 | g_loss: 1.8236
Epoch [   45/   50] | d_loss: 0.4940 | g_loss: 2.7836
Epoch [   45/   50] | d_loss: 0.5790 | g_loss: 3.6038
Epoch [   45/   50] | d_loss: 0.9194 | g_loss: 1.6867
Epoch [   45/   50] | d_loss: 0.5498 | g_loss: 2.0473
Epoch [   45/   50] | d_loss: 0.8630 | g_loss: 1.8657
Epoch [   46/   50] | d_loss: 1.0116 | g_loss: 3.4661
Epoch [   46/   50] | d_loss: 0.5416 | g_loss: 2.0050
Epoch [   46/   50] | d_loss: 0.4767 | g_loss: 2.7397
Epoch [   46/   50] | d_loss: 0.6261 | g_loss: 1.9201
Epoch [   46/   50] | d_loss: 0.5333 | g_loss: 2.7643
Epoch [   46/   50] | d_loss: 0.5673 | g_loss: 1.9768
Epoch [   46/   50] | d_loss: 0.4820 | g_loss: 2.4752
Epoch [   46/   50] | d_loss: 0.6853 | g_loss: 2.3048
Epoch [   46/   50] | d_loss: 0.6243 | g_loss: 2.6388
Epoch [   46/   50] | d_loss: 0.5831 | g_loss: 2.9036
Epoch [   46/   50] | d_loss: 0.6073 | g_loss: 1.5764
Epoch [   46/   50] | d_loss: 0.6492 | g_loss: 2.4943
Epoch [   46/   50] | d_loss: 0.4522 | g_loss: 3.1171
Epoch [   46/   50] | d_loss: 0.4930 | g_loss: 3.0116
Epoch [   46/   50] | d_loss: 0.5708 | g_loss: 1.9994
Epoch [   47/   50] | d_loss: 0.4693 | g_loss: 2.6240
Epoch [   47/   50] | d_loss: 1.1346 | g_loss: 2.9496
Epoch [   47/   50] | d_loss: 0.4876 | g_loss: 2.8961
Epoch [   47/   50] | d_loss: 0.4662 | g_loss: 2.6609
Epoch [   47/   50] | d_loss: 0.5348 | g_loss: 2.4534
Epoch [   47/   50] | d_loss: 0.6346 | g_loss: 3.3539
Epoch [   47/   50] | d_loss: 3.8906 | g_loss: 5.0499
Epoch [   47/   50] | d_loss: 0.9410 | g_loss: 1.4759
Epoch [   47/   50] | d_loss: 0.5292 | g_loss: 2.2661
```

```
Epoch [   47/   50] | d_loss: 0.5644 | g_loss: 2.4493
Epoch [   47/   50] | d_loss: 0.4803 | g_loss: 2.9036
Epoch [   47/   50] | d_loss: 0.4508 | g_loss: 2.5443
Epoch [   47/   50] | d_loss: 0.5938 | g_loss: 2.0221
Epoch [   47/   50] | d_loss: 0.6236 | g_loss: 2.4987
Epoch [   47/   50] | d_loss: 0.5005 | g_loss: 2.1490
Epoch [   48/   50] | d_loss: 0.5776 | g_loss: 2.3303
Epoch [   48/   50] | d_loss: 0.4486 | g_loss: 2.9367
Epoch [   48/   50] | d_loss: 0.5239 | g_loss: 2.2221
Epoch [   48/   50] | d_loss: 1.0477 | g_loss: 3.5116
Epoch [   48/   50] | d_loss: 0.5184 | g_loss: 2.7675
Epoch [   48/   50] | d_loss: 0.4808 | g_loss: 2.2757
Epoch [   48/   50] | d_loss: 0.9243 | g_loss: 1.7239
Epoch [   48/   50] | d_loss: 0.5577 | g_loss: 2.9151
Epoch [   48/   50] | d_loss: 0.6136 | g_loss: 1.2942
Epoch [   48/   50] | d_loss: 0.6145 | g_loss: 2.6787
Epoch [   48/   50] | d_loss: 0.5059 | g_loss: 2.6801
Epoch [   48/   50] | d_loss: 0.5366 | g_loss: 3.0236
Epoch [   48/   50] | d_loss: 0.5224 | g_loss: 3.1950
Epoch [   48/   50] | d_loss: 0.4808 | g_loss: 2.6787
Epoch [   48/   50] | d_loss: 0.4480 | g_loss: 2.7469
Epoch [   49/   50] | d_loss: 0.4806 | g_loss: 2.3284
Epoch [   49/   50] | d_loss: 0.5027 | g_loss: 2.8486
Epoch [   49/   50] | d_loss: 0.5239 | g_loss: 3.0470
Epoch [   49/   50] | d_loss: 0.5272 | g_loss: 2.5118
Epoch [   49/   50] | d_loss: 0.7646 | g_loss: 3.3687
Epoch [   49/   50] | d_loss: 0.4909 | g_loss: 2.6438
Epoch [   49/   50] | d_loss: 0.5809 | g_loss: 2.9975
Epoch [   49/   50] | d_loss: 0.4635 | g_loss: 2.4063
Epoch [   49/   50] | d_loss: 0.5077 | g_loss: 2.4278
Epoch [   49/   50] | d_loss: 0.4456 | g_loss: 2.6277
Epoch [   49/   50] | d_loss: 0.6316 | g_loss: 2.4556
Epoch [   49/   50] | d_loss: 0.4727 | g_loss: 2.6864
Epoch [   49/   50] | d_loss: 0.5344 | g_loss: 2.0002
Epoch [   49/   50] | d_loss: 0.5129 | g_loss: 2.3199
Epoch [   49/   50] | d_loss: 0.6103 | g_loss: 3.0086
Epoch [   50/   50] | d_loss: 0.5161 | g_loss: 2.7894
Epoch [   50/   50] | d_loss: 0.5195 | g_loss: 2.6385
Epoch [   50/   50] | d_loss: 0.4379 | g_loss: 3.1565
Epoch [   50/   50] | d_loss: 0.5396 | g_loss: 2.3471
Epoch [   50/   50] | d_loss: 0.5236 | g_loss: 2.9670
Epoch [   50/   50] | d_loss: 0.4903 | g_loss: 2.9336
Epoch [   50/   50] | d_loss: 0.5266 | g_loss: 2.2509
Epoch [   50/   50] | d_loss: 0.4987 | g_loss: 2.7863
Epoch [   50/   50] | d_loss: 0.5989 | g_loss: 2.1002
Epoch [   50/   50] | d_loss: 0.4675 | g_loss: 2.4372
Epoch [   50/   50] | d_loss: 0.7940 | g_loss: 3.9158
Epoch [   50/   50] | d_loss: 0.5226 | g_loss: 2.3544
Epoch [   50/   50] | d_loss: 0.5343 | g_loss: 1.9957
Epoch [   50/   50] | d_loss: 0.5947 | g_loss: 1.9145
Epoch [   50/   50] | d_loss: 0.5338 | g_loss: 2.9399
```

Training loss

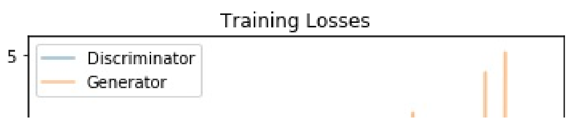
Plot the training losses for the generator and discriminator, recorded after each epoch.

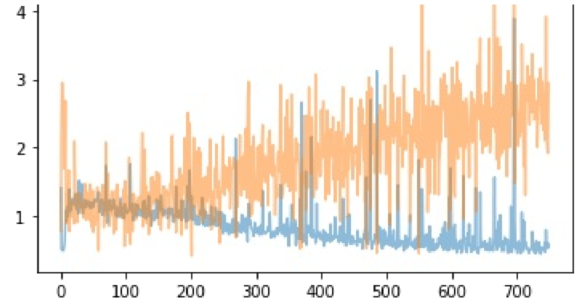
In [23]:

```
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[23]:

```
<matplotlib.legend.Legend at 0x7f4dad16fb70>
```





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 [24]:

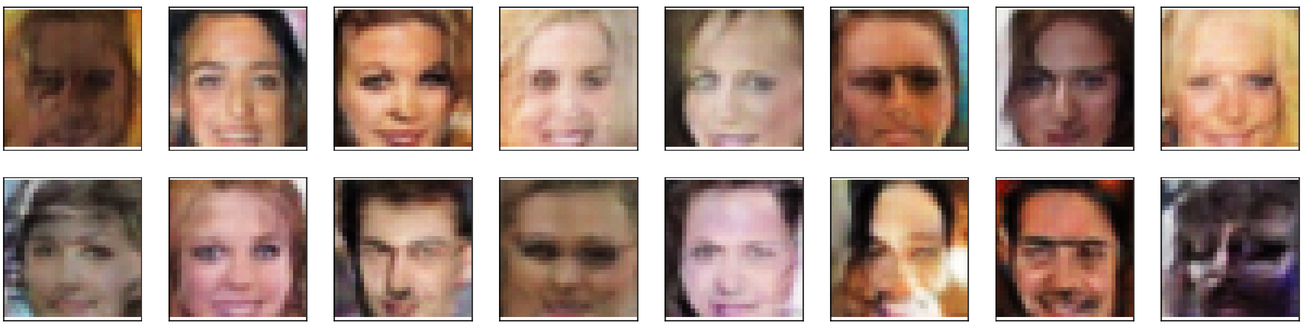
```
# helper function for viewing a list of passed in sample images
def view_samples(epoch, samples):
    fig, axes = plt.subplots(figsize=(16,4), nrows=2, ncols=8, sharey=True, sharex=True)
    for ax, img in zip(axes.flatten(), samples[epoch]):
        img = img.detach().cpu().numpy()
        img = 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 [25]:

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

In [26]:

```
_ = view_samples(-1, samples)
```



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: (Write your answer in this cell)

The dataset is biased towards generating more white ethnics due to the non-stratified random sampling of true celebrity population distribution in the US, i.e. white are the majority. One way to generate a more evenly distributed images from different ethinic is to incorporate stratified samplings from different ethnics, eg. 500 images from each ethnic of interest.

To increase the accuracy of generator further, one approach may be to increase the depths of RNN in order to capture more detailed features on the celebrities look (more fasionable than a typical person, who may put on wide-range of hair, head and/or face accessories/make-up with different designs), i.e. more conv and/or deconv layers in both discrimator and generator models. There is a fake image with a man wearing dark glasses at view_samples (4, 2).

The current optimizer option, ADAM, should work well. However, i would be interested to try NADAM (Dozat, 2016), in view of faster and stable convergence and lower trainig loss shown when training MNIST dataset.

I feel that the epoch of 50 is sufficient as the training loss by the dicriminator doesn't really go down any longer with the current epoch of 50.

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.