dlnd_face_generation

October 21, 2019

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 [3]: # can comment out after executing
        # unzip processed_celeba_small.zip
```

```
Archive:  processed_celeba_small.zip
replace processed_celeba_small/.DS_Store? [y]es, [n]o, [A]ll, [N]one, [r]ename: ^C
```

```
In [4]: 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 [5]: # necessary imports
        import torch
        from torchvision import datasets
        from torchvision import transforms
```

```
In [6]: 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
    # resize and normalize the images
    transform = transforms.Compose([transforms.Resize(image_size), # resize to 128x128
                                    transforms.ToTensor()])

    # define datasets using ImageFolder
    dataset = datasets.ImageFolder(data_dir, transform)
    data_loader = torch.utils.data.DataLoader(dataset=dataset,
                                              batch_size=batch_size,
                                              shuffle=True)
    return data_loader
```

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 [7]: # 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.

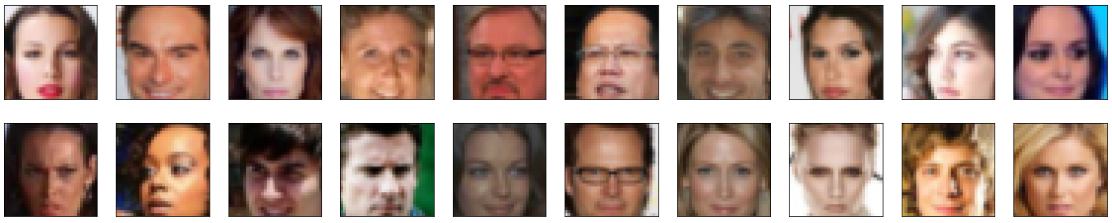
Note: You'll need to convert the Tensor images into a NumPy type and transpose the dimensions to correctly display an image, suggested `imshow` code is below, but it may not be perfect.

```
In [8]: # 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 [9]: # 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 [10]: """
         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(-0.9059)
Max:  tensor(0.9765)
```

---

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 [11]:
```
import torch.nn as nn
import torch.nn.functional as F


# 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_nor
            # 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
            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
```

2.2 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):

    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

        # 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 = self.t_conv3(out)
                out = F.tanh(out)

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

```
Tests Passed
```

2.3 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]: 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.
             # TODO: Apply initial weights to convolutional and linear layers
```

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

2.4 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)
  )
)
```

2.4.1 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 [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!
```

2.5 Discriminator and Generator Losses

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

2.5.1 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.

2.5.2 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 [19]: def real_loss(D_out, smooth=False):
             '''Calculates how close discriminator outputs are to being real.
                param, D_out: discriminator logits
                return: real loss'''

             batch_size = D_out.size(0)
             # label smoothing
             if smooth:
                 # smooth, real labels = 0.9
                 labels = torch.ones(batch_size)*0.9
             else:
                 labels = torch.ones(batch_size) # real labels = 1
             # 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
             if train_on_gpu:
                 labels = labels.cuda()
             criterion = nn.BCEWithLogitsLoss()
             # calculate loss
             loss = criterion(D_out.squeeze(), labels)
             return loss
```

2.6 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 # default value

         # 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])
```

2.7 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()
        if train_on_gpu:
            real_images = real_images.cuda()

        # Compute the discriminator losses on real images
        D_real = D(real_images)
        d_real_loss = real_loss(D_real)

        # 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
        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()
            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
            # 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
```

```
In [22]: !wget -O workspace_utils.py "https://s3.amazonaws.com/video.udacity-data.com/topher/201
```

```
--2019-10-20 23:15:03--  https://s3.amazonaws.com/video.udacity-data.com/topher/2018/May/5b0dea9
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.216.128.93
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.128.93|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1540 (1.5K) []
Saving to: workspace_utils.py

workspace_utils.py  100%[===================>]   1.50K  --.-KB/s    in 0s

2019-10-20 23:15:03 (80.6 MB/s) - workspace_utils.py saved [1540/1540]
```

Set your number of training epochs and train your GAN!

```
In [23]: # set number of epochs
         n_epochs = 50


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

```
Epoch [    1/   50] | d_loss: 1.3446 | g_loss: 0.9245
Epoch [    1/   50] | d_loss: 0.3096 | g_loss: 2.6385
Epoch [    1/   50] | d_loss: 0.1478 | g_loss: 3.5545
Epoch [    1/   50] | d_loss: 0.7554 | g_loss: 5.6508
Epoch [    1/   50] | d_loss: 0.2155 | g_loss: 2.9880
Epoch [    1/   50] | d_loss: 0.5548 | g_loss: 1.2638
Epoch [    1/   50] | d_loss: 0.4151 | g_loss: 2.9133
Epoch [    1/   50] | d_loss: 0.5534 | g_loss: 1.7035
Epoch [    1/   50] | d_loss: 1.2644 | g_loss: 2.8673
Epoch [    1/   50] | d_loss: 0.7585 | g_loss: 1.1694
Epoch [    1/   50] | d_loss: 0.8014 | g_loss: 1.8938
Epoch [    1/   50] | d_loss: 0.7611 | g_loss: 1.4161
Epoch [    1/   50] | d_loss: 0.8109 | g_loss: 1.3326
Epoch [    1/   50] | d_loss: 0.7948 | g_loss: 2.1428
Epoch [    1/   50] | d_loss: 0.9324 | g_loss: 1.9028
Epoch [    2/   50] | d_loss: 1.1706 | g_loss: 1.0291
Epoch [    2/   50] | d_loss: 1.0468 | g_loss: 1.5196
Epoch [    2/   50] | d_loss: 1.0373 | g_loss: 0.7841
Epoch [    2/   50] | d_loss: 1.0122 | g_loss: 1.3761
```

```
Epoch [    2/   50] | d_loss: 0.8239 | g_loss: 2.0370
Epoch [    2/   50] | d_loss: 0.9344 | g_loss: 1.5110
Epoch [    2/   50] | d_loss: 1.1295 | g_loss: 1.8082
Epoch [    2/   50] | d_loss: 1.0916 | g_loss: 1.4686
Epoch [    2/   50] | d_loss: 0.8811 | g_loss: 1.1866
Epoch [    2/   50] | d_loss: 1.0330 | g_loss: 1.4687
Epoch [    2/   50] | d_loss: 1.3800 | g_loss: 0.5921
Epoch [    2/   50] | d_loss: 1.2932 | g_loss: 0.8139
Epoch [    2/   50] | d_loss: 1.0731 | g_loss: 1.3637
Epoch [    2/   50] | d_loss: 1.0839 | g_loss: 1.0569
Epoch [    2/   50] | d_loss: 1.1845 | g_loss: 0.9358
Epoch [    3/   50] | d_loss: 1.0716 | g_loss: 1.0026
Epoch [    3/   50] | d_loss: 0.9876 | g_loss: 1.2051
Epoch [    3/   50] | d_loss: 1.1975 | g_loss: 1.5371
Epoch [    3/   50] | d_loss: 1.0197 | g_loss: 1.0177
Epoch [    3/   50] | d_loss: 1.0254 | g_loss: 1.3110
Epoch [    3/   50] | d_loss: 1.1548 | g_loss: 1.0091
Epoch [    3/   50] | d_loss: 1.0436 | g_loss: 1.0931
Epoch [    3/   50] | d_loss: 0.8004 | g_loss: 1.5116
Epoch [    3/   50] | d_loss: 1.2550 | g_loss: 0.6349
Epoch [    3/   50] | d_loss: 1.3380 | g_loss: 1.1746
Epoch [    3/   50] | d_loss: 1.0972 | g_loss: 1.0874
Epoch [    3/   50] | d_loss: 1.1086 | g_loss: 0.9883
Epoch [    3/   50] | d_loss: 1.0783 | g_loss: 1.1157
Epoch [    3/   50] | d_loss: 1.2612 | g_loss: 1.1136
Epoch [    3/   50] | d_loss: 1.2342 | g_loss: 0.9537
Epoch [    4/   50] | d_loss: 0.9720 | g_loss: 1.1956
Epoch [    4/   50] | d_loss: 1.0621 | g_loss: 1.4705
Epoch [    4/   50] | d_loss: 1.1287 | g_loss: 0.8959
Epoch [    4/   50] | d_loss: 1.0090 | g_loss: 1.4079
Epoch [    4/   50] | d_loss: 1.1154 | g_loss: 1.1937
Epoch [    4/   50] | d_loss: 1.2202 | g_loss: 0.8115
Epoch [    4/   50] | d_loss: 1.2025 | g_loss: 0.8074
Epoch [    4/   50] | d_loss: 1.0725 | g_loss: 1.1357
Epoch [    4/   50] | d_loss: 1.0324 | g_loss: 1.0239
Epoch [    4/   50] | d_loss: 1.1722 | g_loss: 0.9456
Epoch [    4/   50] | d_loss: 1.1108 | g_loss: 1.0740
Epoch [    4/   50] | d_loss: 1.0750 | g_loss: 0.8328
Epoch [    4/   50] | d_loss: 1.3062 | g_loss: 1.2430
Epoch [    4/   50] | d_loss: 1.1457 | g_loss: 1.3389
Epoch [    4/   50] | d_loss: 1.1507 | g_loss: 1.2240
Epoch [    5/   50] | d_loss: 1.8221 | g_loss: 0.1429
Epoch [    5/   50] | d_loss: 1.1156 | g_loss: 0.9387
Epoch [    5/   50] | d_loss: 1.1870 | g_loss: 1.0450
Epoch [    5/   50] | d_loss: 1.0910 | g_loss: 1.2518
Epoch [    5/   50] | d_loss: 0.9965 | g_loss: 0.7526
Epoch [    5/   50] | d_loss: 1.0793 | g_loss: 1.4291
Epoch [    5/   50] | d_loss: 1.1854 | g_loss: 0.8234
```

```
Epoch [    5/   50] | d_loss: 1.1255 | g_loss: 1.1285
Epoch [    5/   50] | d_loss: 1.2218 | g_loss: 1.1555
Epoch [    5/   50] | d_loss: 1.2071 | g_loss: 0.8233
Epoch [    5/   50] | d_loss: 1.3357 | g_loss: 0.2207
Epoch [    5/   50] | d_loss: 1.3071 | g_loss: 0.8517
Epoch [    5/   50] | d_loss: 1.1222 | g_loss: 1.6764
Epoch [    5/   50] | d_loss: 1.1924 | g_loss: 0.8472
Epoch [    5/   50] | d_loss: 1.1711 | g_loss: 1.1286
Epoch [    6/   50] | d_loss: 1.0763 | g_loss: 0.8482
Epoch [    6/   50] | d_loss: 1.1314 | g_loss: 1.0904
Epoch [    6/   50] | d_loss: 1.1636 | g_loss: 0.9054
Epoch [    6/   50] | d_loss: 0.9425 | g_loss: 1.0717
Epoch [    6/   50] | d_loss: 1.1104 | g_loss: 1.2307
Epoch [    6/   50] | d_loss: 1.2048 | g_loss: 0.7197
Epoch [    6/   50] | d_loss: 1.2391 | g_loss: 0.7581
Epoch [    6/   50] | d_loss: 1.0915 | g_loss: 0.8789
Epoch [    6/   50] | d_loss: 1.0512 | g_loss: 1.1797
Epoch [    6/   50] | d_loss: 1.1094 | g_loss: 1.2266
Epoch [    6/   50] | d_loss: 1.1955 | g_loss: 1.2829
Epoch [    6/   50] | d_loss: 1.0193 | g_loss: 1.1763
Epoch [    6/   50] | d_loss: 1.2685 | g_loss: 1.6457
Epoch [    6/   50] | d_loss: 1.1821 | g_loss: 0.7910
Epoch [    6/   50] | d_loss: 1.0649 | g_loss: 0.9461
Epoch [    7/   50] | d_loss: 1.0948 | g_loss: 1.0801
Epoch [    7/   50] | d_loss: 1.2112 | g_loss: 0.6985
Epoch [    7/   50] | d_loss: 1.2210 | g_loss: 0.7179
Epoch [    7/   50] | d_loss: 1.0192 | g_loss: 1.1999
Epoch [    7/   50] | d_loss: 1.0384 | g_loss: 1.1291
Epoch [    7/   50] | d_loss: 1.0226 | g_loss: 1.6758
Epoch [    7/   50] | d_loss: 0.8890 | g_loss: 1.4756
Epoch [    7/   50] | d_loss: 1.0905 | g_loss: 0.9877
Epoch [    7/   50] | d_loss: 1.2626 | g_loss: 1.8627
Epoch [    7/   50] | d_loss: 1.0694 | g_loss: 1.3889
Epoch [    7/   50] | d_loss: 1.1902 | g_loss: 0.7723
Epoch [    7/   50] | d_loss: 1.1455 | g_loss: 1.0882
Epoch [    7/   50] | d_loss: 1.0803 | g_loss: 1.2768
Epoch [    7/   50] | d_loss: 0.9944 | g_loss: 1.2158
Epoch [    7/   50] | d_loss: 0.8461 | g_loss: 1.2450
Epoch [    8/   50] | d_loss: 0.9837 | g_loss: 0.8726
Epoch [    8/   50] | d_loss: 1.1727 | g_loss: 1.2932
Epoch [    8/   50] | d_loss: 1.1064 | g_loss: 0.9354
Epoch [    8/   50] | d_loss: 1.0382 | g_loss: 1.3351
Epoch [    8/   50] | d_loss: 0.8735 | g_loss: 1.6159
Epoch [    8/   50] | d_loss: 1.0315 | g_loss: 1.4734
Epoch [    8/   50] | d_loss: 0.9868 | g_loss: 0.9603
Epoch [    8/   50] | d_loss: 1.2413 | g_loss: 1.1984
Epoch [    8/   50] | d_loss: 0.9710 | g_loss: 1.0120
Epoch [    8/   50] | d_loss: 0.8310 | g_loss: 1.9930
```

```
Epoch [    8/   50] | d_loss: 1.6018 | g_loss: 2.5891
Epoch [    8/   50] | d_loss: 1.0338 | g_loss: 1.1241
Epoch [    8/   50] | d_loss: 1.0342 | g_loss: 1.2729
Epoch [    8/   50] | d_loss: 1.0540 | g_loss: 1.9045
Epoch [    8/   50] | d_loss: 1.0161 | g_loss: 1.0140
Epoch [    9/   50] | d_loss: 1.2217 | g_loss: 0.8749
Epoch [    9/   50] | d_loss: 1.1257 | g_loss: 1.0442
Epoch [    9/   50] | d_loss: 1.5186 | g_loss: 1.6966
Epoch [    9/   50] | d_loss: 0.7912 | g_loss: 1.4057
Epoch [    9/   50] | d_loss: 1.0497 | g_loss: 1.3051
Epoch [    9/   50] | d_loss: 0.9498 | g_loss: 1.0157
Epoch [    9/   50] | d_loss: 0.9308 | g_loss: 1.5562
Epoch [    9/   50] | d_loss: 1.0935 | g_loss: 1.0211
Epoch [    9/   50] | d_loss: 0.9235 | g_loss: 1.4694
Epoch [    9/   50] | d_loss: 0.9347 | g_loss: 0.9189
Epoch [    9/   50] | d_loss: 1.1124 | g_loss: 1.5252
Epoch [    9/   50] | d_loss: 1.0243 | g_loss: 1.3956
Epoch [    9/   50] | d_loss: 1.0184 | g_loss: 1.3204
Epoch [    9/   50] | d_loss: 1.0439 | g_loss: 0.8872
Epoch [    9/   50] | d_loss: 0.9209 | g_loss: 1.3485
Epoch [   10/   50] | d_loss: 0.9910 | g_loss: 1.1593
Epoch [   10/   50] | d_loss: 0.9002 | g_loss: 1.3702
Epoch [   10/   50] | d_loss: 1.3364 | g_loss: 1.0056
Epoch [   10/   50] | d_loss: 1.0291 | g_loss: 0.8381
Epoch [   10/   50] | d_loss: 1.0782 | g_loss: 0.9477
Epoch [   10/   50] | d_loss: 0.9132 | g_loss: 1.4616
Epoch [   10/   50] | d_loss: 1.0648 | g_loss: 1.3817
Epoch [   10/   50] | d_loss: 0.9240 | g_loss: 1.3726
Epoch [   10/   50] | d_loss: 0.9124 | g_loss: 1.3909
Epoch [   10/   50] | d_loss: 1.0357 | g_loss: 1.5223
Epoch [   10/   50] | d_loss: 0.9162 | g_loss: 1.3060
Epoch [   10/   50] | d_loss: 0.9976 | g_loss: 1.2208
Epoch [   10/   50] | d_loss: 0.9584 | g_loss: 1.2510
Epoch [   10/   50] | d_loss: 1.2362 | g_loss: 0.9482
Epoch [   10/   50] | d_loss: 1.2287 | g_loss: 2.1136
Epoch [   11/   50] | d_loss: 1.0519 | g_loss: 0.9681
Epoch [   11/   50] | d_loss: 1.0027 | g_loss: 1.8933
Epoch [   11/   50] | d_loss: 1.0211 | g_loss: 1.6761
Epoch [   11/   50] | d_loss: 1.0775 | g_loss: 0.8052
Epoch [   11/   50] | d_loss: 1.1362 | g_loss: 0.5176
Epoch [   11/   50] | d_loss: 0.9520 | g_loss: 1.7607
Epoch [   11/   50] | d_loss: 1.2710 | g_loss: 2.2540
Epoch [   11/   50] | d_loss: 0.9954 | g_loss: 1.5134
Epoch [   11/   50] | d_loss: 0.8469 | g_loss: 1.2573
Epoch [   11/   50] | d_loss: 1.1951 | g_loss: 0.7374
Epoch [   11/   50] | d_loss: 0.9964 | g_loss: 1.1977
Epoch [   11/   50] | d_loss: 0.9518 | g_loss: 1.3640
Epoch [   11/   50] | d_loss: 0.9541 | g_loss: 1.4144
```

```
Epoch [   11/   50] | d_loss: 0.9351 | g_loss: 1.2696
Epoch [   11/   50] | d_loss: 0.8268 | g_loss: 1.0655
Epoch [   12/   50] | d_loss: 0.9355 | g_loss: 1.4873
Epoch [   12/   50] | d_loss: 1.0744 | g_loss: 0.7582
Epoch [   12/   50] | d_loss: 1.2724 | g_loss: 0.4194
Epoch [   12/   50] | d_loss: 1.2414 | g_loss: 0.6018
Epoch [   12/   50] | d_loss: 0.9023 | g_loss: 1.2723
Epoch [   12/   50] | d_loss: 0.8067 | g_loss: 1.1583
Epoch [   12/   50] | d_loss: 1.2465 | g_loss: 1.9021
Epoch [   12/   50] | d_loss: 0.9211 | g_loss: 1.0537
Epoch [   12/   50] | d_loss: 1.4436 | g_loss: 2.4161
Epoch [   12/   50] | d_loss: 0.6850 | g_loss: 2.0776
Epoch [   12/   50] | d_loss: 0.9305 | g_loss: 1.4833
Epoch [   12/   50] | d_loss: 0.8056 | g_loss: 1.4836
Epoch [   12/   50] | d_loss: 1.1286 | g_loss: 1.5613
Epoch [   12/   50] | d_loss: 1.0290 | g_loss: 2.3192
Epoch [   12/   50] | d_loss: 1.0396 | g_loss: 1.1783
Epoch [   13/   50] | d_loss: 1.0448 | g_loss: 1.0667
Epoch [   13/   50] | d_loss: 1.0340 | g_loss: 0.6677
Epoch [   13/   50] | d_loss: 0.9787 | g_loss: 0.6250
Epoch [   13/   50] | d_loss: 0.8699 | g_loss: 1.3340
Epoch [   13/   50] | d_loss: 0.9362 | g_loss: 1.0099
Epoch [   13/   50] | d_loss: 0.9562 | g_loss: 0.9675
Epoch [   13/   50] | d_loss: 1.5443 | g_loss: 2.7933
Epoch [   13/   50] | d_loss: 0.8043 | g_loss: 1.6451
Epoch [   13/   50] | d_loss: 0.8244 | g_loss: 1.7062
Epoch [   13/   50] | d_loss: 0.7636 | g_loss: 1.5978
Epoch [   13/   50] | d_loss: 0.6829 | g_loss: 1.8293
Epoch [   13/   50] | d_loss: 0.8656 | g_loss: 1.2596
Epoch [   13/   50] | d_loss: 0.9259 | g_loss: 1.0981
Epoch [   13/   50] | d_loss: 0.7099 | g_loss: 1.7898
Epoch [   13/   50] | d_loss: 0.8672 | g_loss: 1.5283
Epoch [   14/   50] | d_loss: 0.7196 | g_loss: 1.4840
Epoch [   14/   50] | d_loss: 0.9958 | g_loss: 1.3398
Epoch [   14/   50] | d_loss: 0.9432 | g_loss: 0.9755
Epoch [   14/   50] | d_loss: 0.8006 | g_loss: 1.2770
Epoch [   14/   50] | d_loss: 1.0243 | g_loss: 1.0651
Epoch [   14/   50] | d_loss: 0.9704 | g_loss: 1.0326
Epoch [   14/   50] | d_loss: 0.8596 | g_loss: 1.4316
Epoch [   14/   50] | d_loss: 1.4026 | g_loss: 1.2531
Epoch [   14/   50] | d_loss: 1.0853 | g_loss: 1.1237
Epoch [   14/   50] | d_loss: 0.8474 | g_loss: 1.5624
Epoch [   14/   50] | d_loss: 0.6671 | g_loss: 1.6265
Epoch [   14/   50] | d_loss: 0.7057 | g_loss: 1.4870
Epoch [   14/   50] | d_loss: 0.9130 | g_loss: 1.8392
Epoch [   14/   50] | d_loss: 0.7744 | g_loss: 1.4321
Epoch [   14/   50] | d_loss: 0.9457 | g_loss: 1.1697
Epoch [   15/   50] | d_loss: 0.8476 | g_loss: 1.2590
```

```
Epoch [   15/   50] | d_loss: 0.8114 | g_loss: 1.2166
Epoch [   15/   50] | d_loss: 0.8366 | g_loss: 1.1027
Epoch [   15/   50] | d_loss: 1.0222 | g_loss: 1.0029
Epoch [   15/   50] | d_loss: 0.5244 | g_loss: 1.9583
Epoch [   15/   50] | d_loss: 0.8496 | g_loss: 1.9591
Epoch [   15/   50] | d_loss: 0.8859 | g_loss: 1.6459
Epoch [   15/   50] | d_loss: 0.9292 | g_loss: 0.5564
Epoch [   15/   50] | d_loss: 1.0562 | g_loss: 1.7434
Epoch [   15/   50] | d_loss: 0.7857 | g_loss: 1.3603
Epoch [   15/   50] | d_loss: 0.8634 | g_loss: 1.3846
Epoch [   15/   50] | d_loss: 0.8419 | g_loss: 1.2582
Epoch [   15/   50] | d_loss: 0.7114 | g_loss: 1.5543
Epoch [   15/   50] | d_loss: 0.8040 | g_loss: 1.5054
Epoch [   15/   50] | d_loss: 0.7783 | g_loss: 0.9655
Epoch [   16/   50] | d_loss: 0.7624 | g_loss: 1.3505
Epoch [   16/   50] | d_loss: 0.8461 | g_loss: 0.9678
Epoch [   16/   50] | d_loss: 0.9058 | g_loss: 0.8859
Epoch [   16/   50] | d_loss: 0.7599 | g_loss: 1.5587
Epoch [   16/   50] | d_loss: 0.8070 | g_loss: 2.1423
Epoch [   16/   50] | d_loss: 0.7887 | g_loss: 1.7081
Epoch [   16/   50] | d_loss: 0.9276 | g_loss: 1.1761
Epoch [   16/   50] | d_loss: 0.8735 | g_loss: 0.6972
Epoch [   16/   50] | d_loss: 1.0206 | g_loss: 0.4140
Epoch [   16/   50] | d_loss: 0.7130 | g_loss: 1.5358
Epoch [   16/   50] | d_loss: 1.0639 | g_loss: 2.1668
Epoch [   16/   50] | d_loss: 0.6842 | g_loss: 2.3227
Epoch [   16/   50] | d_loss: 0.8217 | g_loss: 1.2064
Epoch [   16/   50] | d_loss: 0.8470 | g_loss: 2.2761
Epoch [   16/   50] | d_loss: 0.7275 | g_loss: 1.7191
Epoch [   17/   50] | d_loss: 0.7827 | g_loss: 2.1540
Epoch [   17/   50] | d_loss: 0.9600 | g_loss: 2.2218
Epoch [   17/   50] | d_loss: 0.7238 | g_loss: 1.2975
Epoch [   17/   50] | d_loss: 0.8413 | g_loss: 1.7452
Epoch [   17/   50] | d_loss: 1.1612 | g_loss: 1.0342
Epoch [   17/   50] | d_loss: 0.7267 | g_loss: 1.4758
Epoch [   17/   50] | d_loss: 1.0466 | g_loss: 1.2135
Epoch [   17/   50] | d_loss: 0.7040 | g_loss: 2.1584
Epoch [   17/   50] | d_loss: 0.7794 | g_loss: 1.9421
Epoch [   17/   50] | d_loss: 0.7252 | g_loss: 1.7523
Epoch [   17/   50] | d_loss: 0.7172 | g_loss: 1.3682
Epoch [   17/   50] | d_loss: 0.7003 | g_loss: 1.2449
Epoch [   17/   50] | d_loss: 0.8035 | g_loss: 1.9694
Epoch [   17/   50] | d_loss: 0.6594 | g_loss: 1.6394
Epoch [   17/   50] | d_loss: 0.7845 | g_loss: 1.0948
Epoch [   18/   50] | d_loss: 0.7432 | g_loss: 2.4154
Epoch [   18/   50] | d_loss: 0.9202 | g_loss: 0.9002
Epoch [   18/   50] | d_loss: 0.6949 | g_loss: 1.3540
Epoch [   18/   50] | d_loss: 0.7635 | g_loss: 1.7188
```

```
Epoch [   18/   50] | d_loss: 0.7864 | g_loss: 1.7587
Epoch [   18/   50] | d_loss: 0.6757 | g_loss: 1.6647
Epoch [   18/   50] | d_loss: 0.7288 | g_loss: 1.1264
Epoch [   18/   50] | d_loss: 0.8725 | g_loss: 2.1042
Epoch [   18/   50] | d_loss: 0.6515 | g_loss: 1.8632
Epoch [   18/   50] | d_loss: 0.6030 | g_loss: 2.1910
Epoch [   18/   50] | d_loss: 0.5888 | g_loss: 1.7016
Epoch [   18/   50] | d_loss: 0.9717 | g_loss: 0.9925
Epoch [   18/   50] | d_loss: 0.8256 | g_loss: 1.0548
Epoch [   18/   50] | d_loss: 0.8606 | g_loss: 1.5166
Epoch [   18/   50] | d_loss: 0.7187 | g_loss: 1.8095
Epoch [   19/   50] | d_loss: 0.7338 | g_loss: 2.3624
Epoch [   19/   50] | d_loss: 0.7532 | g_loss: 1.3861
Epoch [   19/   50] | d_loss: 0.6633 | g_loss: 1.5478
Epoch [   19/   50] | d_loss: 0.9608 | g_loss: 2.3775
Epoch [   19/   50] | d_loss: 0.6095 | g_loss: 1.9958
Epoch [   19/   50] | d_loss: 0.7651 | g_loss: 1.7058
Epoch [   19/   50] | d_loss: 0.5348 | g_loss: 1.6422
Epoch [   19/   50] | d_loss: 0.6704 | g_loss: 1.6911
Epoch [   19/   50] | d_loss: 0.9998 | g_loss: 1.0246
Epoch [   19/   50] | d_loss: 0.8120 | g_loss: 1.6223
Epoch [   19/   50] | d_loss: 0.6286 | g_loss: 1.0965
Epoch [   19/   50] | d_loss: 0.7631 | g_loss: 1.7019
Epoch [   19/   50] | d_loss: 0.6972 | g_loss: 1.8290
Epoch [   19/   50] | d_loss: 0.7417 | g_loss: 3.1043
Epoch [   19/   50] | d_loss: 0.6083 | g_loss: 2.2821
Epoch [   20/   50] | d_loss: 0.7041 | g_loss: 1.8331
Epoch [   20/   50] | d_loss: 0.5530 | g_loss: 2.3032
Epoch [   20/   50] | d_loss: 0.6338 | g_loss: 1.1891
Epoch [   20/   50] | d_loss: 0.7034 | g_loss: 1.6068
Epoch [   20/   50] | d_loss: 1.0149 | g_loss: 3.0278
Epoch [   20/   50] | d_loss: 1.1543 | g_loss: 0.5786
Epoch [   20/   50] | d_loss: 0.8178 | g_loss: 1.0562
Epoch [   20/   50] | d_loss: 0.6806 | g_loss: 1.4214
Epoch [   20/   50] | d_loss: 0.8165 | g_loss: 0.9584
Epoch [   20/   50] | d_loss: 0.6957 | g_loss: 2.2914
Epoch [   20/   50] | d_loss: 0.7369 | g_loss: 1.7588
Epoch [   20/   50] | d_loss: 0.6666 | g_loss: 1.3794
Epoch [   20/   50] | d_loss: 0.6609 | g_loss: 1.6531
Epoch [   20/   50] | d_loss: 0.7733 | g_loss: 2.0140
Epoch [   20/   50] | d_loss: 1.8930 | g_loss: 4.3270
Epoch [   21/   50] | d_loss: 1.1985 | g_loss: 1.7382
Epoch [   21/   50] | d_loss: 0.7512 | g_loss: 2.1882
Epoch [   21/   50] | d_loss: 0.5489 | g_loss: 1.5183
Epoch [   21/   50] | d_loss: 0.5527 | g_loss: 2.1346
Epoch [   21/   50] | d_loss: 0.6578 | g_loss: 2.0876
Epoch [   21/   50] | d_loss: 0.9144 | g_loss: 0.4678
Epoch [   21/   50] | d_loss: 0.5202 | g_loss: 1.5029
```

```
Epoch [   21/   50] | d_loss: 0.6482 | g_loss: 2.4241
Epoch [   21/   50] | d_loss: 0.5818 | g_loss: 1.6515
Epoch [   21/   50] | d_loss: 0.5734 | g_loss: 1.7746
Epoch [   21/   50] | d_loss: 0.6780 | g_loss: 2.1041
Epoch [   21/   50] | d_loss: 0.9311 | g_loss: 1.2098
Epoch [   21/   50] | d_loss: 0.7011 | g_loss: 1.1870
Epoch [   21/   50] | d_loss: 1.1001 | g_loss: 2.4864
Epoch [   21/   50] | d_loss: 0.7388 | g_loss: 1.5494
Epoch [   22/   50] | d_loss: 0.7455 | g_loss: 1.5190
Epoch [   22/   50] | d_loss: 0.7338 | g_loss: 0.9879
Epoch [   22/   50] | d_loss: 0.5789 | g_loss: 2.2055
Epoch [   22/   50] | d_loss: 0.5829 | g_loss: 1.9507
Epoch [   22/   50] | d_loss: 0.5638 | g_loss: 1.5079
Epoch [   22/   50] | d_loss: 0.5480 | g_loss: 2.1013
Epoch [   22/   50] | d_loss: 0.6593 | g_loss: 2.1379
Epoch [   22/   50] | d_loss: 0.6952 | g_loss: 1.5132
Epoch [   22/   50] | d_loss: 0.6260 | g_loss: 2.3574
Epoch [   22/   50] | d_loss: 0.5553 | g_loss: 1.7481
Epoch [   22/   50] | d_loss: 0.5941 | g_loss: 1.3686
Epoch [   22/   50] | d_loss: 0.5797 | g_loss: 1.9520
Epoch [   22/   50] | d_loss: 0.6953 | g_loss: 1.3717
Epoch [   22/   50] | d_loss: 0.6353 | g_loss: 2.0818
Epoch [   22/   50] | d_loss: 0.5904 | g_loss: 1.4562
Epoch [   23/   50] | d_loss: 0.6340 | g_loss: 1.5040
Epoch [   23/   50] | d_loss: 0.6032 | g_loss: 2.1435
Epoch [   23/   50] | d_loss: 0.6315 | g_loss: 1.4587
Epoch [   23/   50] | d_loss: 0.6215 | g_loss: 2.3779
Epoch [   23/   50] | d_loss: 0.5993 | g_loss: 1.8776
Epoch [   23/   50] | d_loss: 0.5410 | g_loss: 1.8228
Epoch [   23/   50] | d_loss: 1.1121 | g_loss: 0.5171
Epoch [   23/   50] | d_loss: 0.5234 | g_loss: 2.1928
Epoch [   23/   50] | d_loss: 0.5759 | g_loss: 2.6739
Epoch [   23/   50] | d_loss: 1.5702 | g_loss: 2.8672
Epoch [   23/   50] | d_loss: 0.6758 | g_loss: 2.5281
Epoch [   23/   50] | d_loss: 0.5359 | g_loss: 2.1323
Epoch [   23/   50] | d_loss: 0.6156 | g_loss: 1.8908
Epoch [   23/   50] | d_loss: 0.5868 | g_loss: 2.3069
Epoch [   23/   50] | d_loss: 0.8221 | g_loss: 1.8957
Epoch [   24/   50] | d_loss: 0.5603 | g_loss: 1.4898
Epoch [   24/   50] | d_loss: 0.7779 | g_loss: 2.2414
Epoch [   24/   50] | d_loss: 0.6757 | g_loss: 1.7196
Epoch [   24/   50] | d_loss: 0.6028 | g_loss: 1.8335
Epoch [   24/   50] | d_loss: 0.4502 | g_loss: 2.2414
Epoch [   24/   50] | d_loss: 0.5923 | g_loss: 1.7928
Epoch [   24/   50] | d_loss: 0.6984 | g_loss: 2.6817
Epoch [   24/   50] | d_loss: 0.6473 | g_loss: 2.0765
Epoch [   24/   50] | d_loss: 0.6551 | g_loss: 1.4988
Epoch [   24/   50] | d_loss: 0.7849 | g_loss: 2.8884
```

```
Epoch [   24/   50] | d_loss: 0.4430 | g_loss: 1.9593
Epoch [   24/   50] | d_loss: 0.4998 | g_loss: 1.8461
Epoch [   24/   50] | d_loss: 0.4862 | g_loss: 2.1091
Epoch [   24/   50] | d_loss: 0.5817 | g_loss: 1.3948
Epoch [   24/   50] | d_loss: 0.5110 | g_loss: 1.6597
Epoch [   25/   50] | d_loss: 0.5015 | g_loss: 2.1753
Epoch [   25/   50] | d_loss: 0.6089 | g_loss: 1.4722
Epoch [   25/   50] | d_loss: 0.7645 | g_loss: 3.2464
Epoch [   25/   50] | d_loss: 0.5057 | g_loss: 1.4301
Epoch [   25/   50] | d_loss: 0.5663 | g_loss: 1.8509
Epoch [   25/   50] | d_loss: 0.6776 | g_loss: 2.0813
Epoch [   25/   50] | d_loss: 0.6391 | g_loss: 1.2474
Epoch [   25/   50] | d_loss: 0.4684 | g_loss: 2.8115
Epoch [   25/   50] | d_loss: 0.4729 | g_loss: 1.6456
Epoch [   25/   50] | d_loss: 0.7470 | g_loss: 0.8511
Epoch [   25/   50] | d_loss: 0.6506 | g_loss: 1.5226
Epoch [   25/   50] | d_loss: 0.5539 | g_loss: 1.6268
Epoch [   25/   50] | d_loss: 0.4263 | g_loss: 1.9698
Epoch [   25/   50] | d_loss: 0.4876 | g_loss: 1.7870
Epoch [   25/   50] | d_loss: 0.4927 | g_loss: 2.6229
Epoch [   26/   50] | d_loss: 0.7366 | g_loss: 1.1757
Epoch [   26/   50] | d_loss: 0.5956 | g_loss: 2.2691
Epoch [   26/   50] | d_loss: 0.5779 | g_loss: 1.5510
Epoch [   26/   50] | d_loss: 0.4748 | g_loss: 1.9969
Epoch [   26/   50] | d_loss: 0.4868 | g_loss: 2.7850
Epoch [   26/   50] | d_loss: 0.5022 | g_loss: 1.6712
Epoch [   26/   50] | d_loss: 0.4981 | g_loss: 2.1500
Epoch [   26/   50] | d_loss: 0.5924 | g_loss: 2.2162
Epoch [   26/   50] | d_loss: 0.5852 | g_loss: 1.4941
Epoch [   26/   50] | d_loss: 0.4371 | g_loss: 2.4703
Epoch [   26/   50] | d_loss: 0.5219 | g_loss: 1.7050
Epoch [   26/   50] | d_loss: 0.4825 | g_loss: 2.9628
Epoch [   26/   50] | d_loss: 0.6503 | g_loss: 1.9053
Epoch [   26/   50] | d_loss: 0.3813 | g_loss: 1.7859
Epoch [   26/   50] | d_loss: 0.4136 | g_loss: 2.4345
Epoch [   27/   50] | d_loss: 0.5124 | g_loss: 1.8794
Epoch [   27/   50] | d_loss: 0.7375 | g_loss: 1.3400
Epoch [   27/   50] | d_loss: 0.3677 | g_loss: 1.9778
Epoch [   27/   50] | d_loss: 0.6212 | g_loss: 2.4251
Epoch [   27/   50] | d_loss: 2.5847 | g_loss: 3.2442
Epoch [   27/   50] | d_loss: 0.4619 | g_loss: 2.1391
Epoch [   27/   50] | d_loss: 0.6738 | g_loss: 2.3850
Epoch [   27/   50] | d_loss: 0.5990 | g_loss: 2.6904
Epoch [   27/   50] | d_loss: 0.4547 | g_loss: 1.6322
Epoch [   27/   50] | d_loss: 0.4951 | g_loss: 1.8428
Epoch [   27/   50] | d_loss: 0.4729 | g_loss: 1.4945
Epoch [   27/   50] | d_loss: 0.5329 | g_loss: 1.7344
Epoch [   27/   50] | d_loss: 0.6863 | g_loss: 2.4486
```

```
Epoch [   27/   50] | d_loss: 0.5170 | g_loss: 1.9354
Epoch [   27/   50] | d_loss: 0.4199 | g_loss: 2.6748
Epoch [   28/   50] | d_loss: 0.4241 | g_loss: 2.0807
Epoch [   28/   50] | d_loss: 0.5166 | g_loss: 1.8212
Epoch [   28/   50] | d_loss: 0.3974 | g_loss: 2.6596
Epoch [   28/   50] | d_loss: 0.3789 | g_loss: 2.4336
Epoch [   28/   50] | d_loss: 0.8011 | g_loss: 1.1928
Epoch [   28/   50] | d_loss: 0.5180 | g_loss: 2.2866
Epoch [   28/   50] | d_loss: 0.3140 | g_loss: 2.4729
Epoch [   28/   50] | d_loss: 0.4610 | g_loss: 1.6520
Epoch [   28/   50] | d_loss: 0.4623 | g_loss: 1.6773
Epoch [   28/   50] | d_loss: 1.0146 | g_loss: 1.5894
Epoch [   28/   50] | d_loss: 0.7221 | g_loss: 1.6866
Epoch [   28/   50] | d_loss: 0.4418 | g_loss: 2.1090
Epoch [   28/   50] | d_loss: 0.4900 | g_loss: 1.7670
Epoch [   28/   50] | d_loss: 0.5640 | g_loss: 1.9697
Epoch [   28/   50] | d_loss: 0.5512 | g_loss: 1.1715
Epoch [   29/   50] | d_loss: 0.4837 | g_loss: 1.4617
Epoch [   29/   50] | d_loss: 0.5379 | g_loss: 2.1331
Epoch [   29/   50] | d_loss: 0.4884 | g_loss: 1.9601
Epoch [   29/   50] | d_loss: 0.4766 | g_loss: 1.9507
Epoch [   29/   50] | d_loss: 0.4696 | g_loss: 1.7838
Epoch [   29/   50] | d_loss: 0.3988 | g_loss: 1.9968
Epoch [   29/   50] | d_loss: 0.4607 | g_loss: 2.0488
Epoch [   29/   50] | d_loss: 0.4880 | g_loss: 1.7214
Epoch [   29/   50] | d_loss: 0.5908 | g_loss: 1.6874
Epoch [   29/   50] | d_loss: 0.4883 | g_loss: 1.9520
Epoch [   29/   50] | d_loss: 0.5442 | g_loss: 1.8712
Epoch [   29/   50] | d_loss: 0.4445 | g_loss: 2.2581
Epoch [   29/   50] | d_loss: 0.5803 | g_loss: 1.3018
Epoch [   29/   50] | d_loss: 0.3452 | g_loss: 2.2382
Epoch [   29/   50] | d_loss: 0.6335 | g_loss: 2.8415
Epoch [   30/   50] | d_loss: 0.6712 | g_loss: 3.1168
Epoch [   30/   50] | d_loss: 0.4435 | g_loss: 1.9093
Epoch [   30/   50] | d_loss: 0.4571 | g_loss: 1.9165
Epoch [   30/   50] | d_loss: 0.5506 | g_loss: 2.2521
Epoch [   30/   50] | d_loss: 0.5136 | g_loss: 2.7811
Epoch [   30/   50] | d_loss: 0.6919 | g_loss: 1.1893
Epoch [   30/   50] | d_loss: 0.4984 | g_loss: 2.5181
Epoch [   30/   50] | d_loss: 0.3596 | g_loss: 2.6248
Epoch [   30/   50] | d_loss: 0.5343 | g_loss: 1.9343
Epoch [   30/   50] | d_loss: 0.4143 | g_loss: 1.9912
Epoch [   30/   50] | d_loss: 0.3959 | g_loss: 1.9500
Epoch [   30/   50] | d_loss: 0.5166 | g_loss: 2.6380
Epoch [   30/   50] | d_loss: 0.5745 | g_loss: 2.4307
Epoch [   30/   50] | d_loss: 0.5131 | g_loss: 2.4442
Epoch [   30/   50] | d_loss: 0.6002 | g_loss: 1.1375
Epoch [   31/   50] | d_loss: 0.4125 | g_loss: 1.4660
```

```
Epoch [   31/   50] | d_loss: 0.5588 | g_loss: 1.4256
Epoch [   31/   50] | d_loss: 0.4545 | g_loss: 2.0015
Epoch [   31/   50] | d_loss: 0.3344 | g_loss: 2.1067
Epoch [   31/   50] | d_loss: 2.7513 | g_loss: 4.8374
Epoch [   31/   50] | d_loss: 0.4581 | g_loss: 2.7008
Epoch [   31/   50] | d_loss: 2.6790 | g_loss: 6.3894
Epoch [   31/   50] | d_loss: 0.3024 | g_loss: 2.3246
Epoch [   31/   50] | d_loss: 0.3948 | g_loss: 2.3664
Epoch [   31/   50] | d_loss: 0.4623 | g_loss: 1.4637
Epoch [   31/   50] | d_loss: 0.3738 | g_loss: 2.7555
Epoch [   31/   50] | d_loss: 0.4655 | g_loss: 2.7150
Epoch [   31/   50] | d_loss: 0.3559 | g_loss: 1.9432
Epoch [   31/   50] | d_loss: 0.5274 | g_loss: 3.6357
Epoch [   31/   50] | d_loss: 0.5752 | g_loss: 1.6075
Epoch [   32/   50] | d_loss: 0.4391 | g_loss: 2.1624
Epoch [   32/   50] | d_loss: 0.3184 | g_loss: 3.1163
Epoch [   32/   50] | d_loss: 1.6849 | g_loss: 0.5474
Epoch [   32/   50] | d_loss: 0.4531 | g_loss: 2.2047
Epoch [   32/   50] | d_loss: 0.4252 | g_loss: 2.9831
Epoch [   32/   50] | d_loss: 0.4262 | g_loss: 2.6824
Epoch [   32/   50] | d_loss: 0.5692 | g_loss: 1.8020
Epoch [   32/   50] | d_loss: 1.0392 | g_loss: 1.0822
Epoch [   32/   50] | d_loss: 0.4501 | g_loss: 2.1534
Epoch [   32/   50] | d_loss: 0.3196 | g_loss: 2.5005
Epoch [   32/   50] | d_loss: 0.2943 | g_loss: 2.8284
Epoch [   32/   50] | d_loss: 0.7891 | g_loss: 3.0736
Epoch [   32/   50] | d_loss: 1.3112 | g_loss: 4.4308
Epoch [   32/   50] | d_loss: 0.4812 | g_loss: 2.1733
Epoch [   32/   50] | d_loss: 0.5736 | g_loss: 3.2994
Epoch [   33/   50] | d_loss: 0.5895 | g_loss: 3.2268
Epoch [   33/   50] | d_loss: 0.5541 | g_loss: 2.3951
Epoch [   33/   50] | d_loss: 0.4552 | g_loss: 2.4648
Epoch [   33/   50] | d_loss: 0.5037 | g_loss: 2.2838
Epoch [   33/   50] | d_loss: 0.5766 | g_loss: 2.3715
Epoch [   33/   50] | d_loss: 0.4411 | g_loss: 3.3700
Epoch [   33/   50] | d_loss: 2.1807 | g_loss: 0.6177
Epoch [   33/   50] | d_loss: 0.3384 | g_loss: 2.5687
Epoch [   33/   50] | d_loss: 0.3703 | g_loss: 2.2938
Epoch [   33/   50] | d_loss: 0.3174 | g_loss: 2.5228
Epoch [   33/   50] | d_loss: 0.6322 | g_loss: 1.7384
Epoch [   33/   50] | d_loss: 0.4058 | g_loss: 1.8873
Epoch [   33/   50] | d_loss: 0.4699 | g_loss: 2.9755
Epoch [   33/   50] | d_loss: 0.4752 | g_loss: 3.7565
Epoch [   33/   50] | d_loss: 0.4807 | g_loss: 3.1408
Epoch [   34/   50] | d_loss: 0.3837 | g_loss: 2.1114
Epoch [   34/   50] | d_loss: 0.4291 | g_loss: 2.0840
Epoch [   34/   50] | d_loss: 0.3574 | g_loss: 1.5629
Epoch [   34/   50] | d_loss: 0.3833 | g_loss: 3.1009
```

```
Epoch [   34/   50] | d_loss: 0.3899 | g_loss: 2.2530
Epoch [   34/   50] | d_loss: 0.4409 | g_loss: 2.3852
Epoch [   34/   50] | d_loss: 0.4968 | g_loss: 1.7893
Epoch [   34/   50] | d_loss: 0.2678 | g_loss: 2.2880
Epoch [   34/   50] | d_loss: 0.3850 | g_loss: 2.3265
Epoch [   34/   50] | d_loss: 0.6164 | g_loss: 0.9944
Epoch [   34/   50] | d_loss: 0.4167 | g_loss: 2.4448
Epoch [   34/   50] | d_loss: 0.2804 | g_loss: 2.6379
Epoch [   34/   50] | d_loss: 0.3733 | g_loss: 2.0159
Epoch [   34/   50] | d_loss: 0.3444 | g_loss: 2.7605
Epoch [   34/   50] | d_loss: 0.4302 | g_loss: 1.6521
Epoch [   35/   50] | d_loss: 0.5628 | g_loss: 2.2592
Epoch [   35/   50] | d_loss: 0.4290 | g_loss: 2.2935
Epoch [   35/   50] | d_loss: 0.4227 | g_loss: 2.5553
Epoch [   35/   50] | d_loss: 0.4330 | g_loss: 2.5666
Epoch [   35/   50] | d_loss: 0.4083 | g_loss: 2.1948
Epoch [   35/   50] | d_loss: 0.3196 | g_loss: 2.1328
Epoch [   35/   50] | d_loss: 0.3627 | g_loss: 2.4798
Epoch [   35/   50] | d_loss: 0.5532 | g_loss: 1.5085
Epoch [   35/   50] | d_loss: 0.3816 | g_loss: 2.0413
Epoch [   35/   50] | d_loss: 0.5653 | g_loss: 3.0567
Epoch [   35/   50] | d_loss: 0.3436 | g_loss: 2.7931
Epoch [   35/   50] | d_loss: 0.3896 | g_loss: 1.6011
Epoch [   35/   50] | d_loss: 1.0828 | g_loss: 0.8422
Epoch [   35/   50] | d_loss: 0.3766 | g_loss: 2.2178
Epoch [   35/   50] | d_loss: 0.3745 | g_loss: 3.1315
Epoch [   36/   50] | d_loss: 0.7060 | g_loss: 1.0968
Epoch [   36/   50] | d_loss: 0.4314 | g_loss: 2.7210
Epoch [   36/   50] | d_loss: 0.3679 | g_loss: 2.9359
Epoch [   36/   50] | d_loss: 0.6980 | g_loss: 1.2955
Epoch [   36/   50] | d_loss: 0.4282 | g_loss: 2.3554
Epoch [   36/   50] | d_loss: 0.3020 | g_loss: 2.8878
Epoch [   36/   50] | d_loss: 0.3821 | g_loss: 2.2073
Epoch [   36/   50] | d_loss: 0.4614 | g_loss: 2.6164
Epoch [   36/   50] | d_loss: 0.3076 | g_loss: 2.5280
Epoch [   36/   50] | d_loss: 0.4122 | g_loss: 2.0460
Epoch [   36/   50] | d_loss: 0.2432 | g_loss: 2.4737
Epoch [   36/   50] | d_loss: 0.5150 | g_loss: 1.9398
Epoch [   36/   50] | d_loss: 0.3173 | g_loss: 1.9213
Epoch [   36/   50] | d_loss: 0.4437 | g_loss: 1.3583
Epoch [   36/   50] | d_loss: 0.3118 | g_loss: 1.9455
Epoch [   37/   50] | d_loss: 0.5492 | g_loss: 2.1508
Epoch [   37/   50] | d_loss: 0.4572 | g_loss: 3.1781
Epoch [   37/   50] | d_loss: 0.3628 | g_loss: 2.3532
Epoch [   37/   50] | d_loss: 0.2975 | g_loss: 2.3871
Epoch [   37/   50] | d_loss: 0.5855 | g_loss: 1.4419
Epoch [   37/   50] | d_loss: 0.5155 | g_loss: 1.4315
Epoch [   37/   50] | d_loss: 0.4294 | g_loss: 1.7589
```

```
Epoch [   37/   50] | d_loss: 0.4443 | g_loss: 1.8998
Epoch [   37/   50] | d_loss: 0.2733 | g_loss: 2.5894
Epoch [   37/   50] | d_loss: 2.6563 | g_loss: 3.4400
Epoch [   37/   50] | d_loss: 0.4693 | g_loss: 2.0074
Epoch [   37/   50] | d_loss: 0.4834 | g_loss: 2.9523
Epoch [   37/   50] | d_loss: 0.3295 | g_loss: 2.3225
Epoch [   37/   50] | d_loss: 0.4662 | g_loss: 3.2738
Epoch [   37/   50] | d_loss: 0.4816 | g_loss: 2.8456
Epoch [   38/   50] | d_loss: 0.4807 | g_loss: 1.6947
Epoch [   38/   50] | d_loss: 0.3686 | g_loss: 2.8784
Epoch [   38/   50] | d_loss: 0.4695 | g_loss: 2.8519
Epoch [   38/   50] | d_loss: 0.3550 | g_loss: 2.0726
Epoch [   38/   50] | d_loss: 0.2768 | g_loss: 2.7825
Epoch [   38/   50] | d_loss: 0.4803 | g_loss: 2.2317
Epoch [   38/   50] | d_loss: 0.2981 | g_loss: 2.9273
Epoch [   38/   50] | d_loss: 0.2989 | g_loss: 2.8583
Epoch [   38/   50] | d_loss: 0.3950 | g_loss: 3.3048
Epoch [   38/   50] | d_loss: 0.3314 | g_loss: 2.1915
Epoch [   38/   50] | d_loss: 0.3620 | g_loss: 2.5193
Epoch [   38/   50] | d_loss: 0.2585 | g_loss: 2.6471
Epoch [   38/   50] | d_loss: 0.5086 | g_loss: 2.3363
Epoch [   38/   50] | d_loss: 0.4206 | g_loss: 1.9465
Epoch [   38/   50] | d_loss: 0.2467 | g_loss: 3.0886
Epoch [   39/   50] | d_loss: 0.4180 | g_loss: 2.6764
Epoch [   39/   50] | d_loss: 2.1320 | g_loss: 2.5583
Epoch [   39/   50] | d_loss: 0.3298 | g_loss: 2.6360
Epoch [   39/   50] | d_loss: 0.3311 | g_loss: 2.5646
Epoch [   39/   50] | d_loss: 0.4017 | g_loss: 3.2488
Epoch [   39/   50] | d_loss: 0.4437 | g_loss: 2.0347
Epoch [   39/   50] | d_loss: 0.5310 | g_loss: 2.5498
Epoch [   39/   50] | d_loss: 0.3617 | g_loss: 2.7693
Epoch [   39/   50] | d_loss: 0.4416 | g_loss: 1.9068
Epoch [   39/   50] | d_loss: 0.3530 | g_loss: 3.1487
Epoch [   39/   50] | d_loss: 0.4210 | g_loss: 3.7051
Epoch [   39/   50] | d_loss: 0.4025 | g_loss: 2.7150
Epoch [   39/   50] | d_loss: 0.3376 | g_loss: 2.9203
Epoch [   39/   50] | d_loss: 0.3005 | g_loss: 2.7140
Epoch [   39/   50] | d_loss: 0.3175 | g_loss: 2.5122
Epoch [   40/   50] | d_loss: 0.3640 | g_loss: 2.7860
Epoch [   40/   50] | d_loss: 0.3196 | g_loss: 3.3983
Epoch [   40/   50] | d_loss: 0.3783 | g_loss: 2.7023
Epoch [   40/   50] | d_loss: 0.3709 | g_loss: 2.0600
Epoch [   40/   50] | d_loss: 0.3961 | g_loss: 2.2779
Epoch [   40/   50] | d_loss: 0.3076 | g_loss: 2.4943
Epoch [   40/   50] | d_loss: 0.3794 | g_loss: 3.0407
Epoch [   40/   50] | d_loss: 0.4719 | g_loss: 2.8877
Epoch [   40/   50] | d_loss: 0.2268 | g_loss: 3.1052
Epoch [   40/   50] | d_loss: 0.2745 | g_loss: 3.0576
```

```
Epoch [   40/   50] | d_loss: 2.5918 | g_loss: 4.7562
Epoch [   40/   50] | d_loss: 0.3132 | g_loss: 2.6720
Epoch [   40/   50] | d_loss: 0.2412 | g_loss: 2.5509
Epoch [   40/   50] | d_loss: 0.4161 | g_loss: 4.2414
Epoch [   40/   50] | d_loss: 0.2693 | g_loss: 2.5473
Epoch [   41/   50] | d_loss: 0.3115 | g_loss: 2.9367
Epoch [   41/   50] | d_loss: 0.3501 | g_loss: 2.8849
Epoch [   41/   50] | d_loss: 0.3180 | g_loss: 2.7838
Epoch [   41/   50] | d_loss: 0.4043 | g_loss: 2.1274
Epoch [   41/   50] | d_loss: 0.2475 | g_loss: 3.0554
Epoch [   41/   50] | d_loss: 0.3697 | g_loss: 2.7137
Epoch [   41/   50] | d_loss: 0.3857 | g_loss: 2.4296
Epoch [   41/   50] | d_loss: 1.8861 | g_loss: 4.6267
Epoch [   41/   50] | d_loss: 0.3059 | g_loss: 2.5347
Epoch [   41/   50] | d_loss: 0.3473 | g_loss: 1.9622
Epoch [   41/   50] | d_loss: 0.2487 | g_loss: 2.3517
Epoch [   41/   50] | d_loss: 0.3172 | g_loss: 2.7798
Epoch [   41/   50] | d_loss: 0.3376 | g_loss: 3.5367
Epoch [   41/   50] | d_loss: 0.4672 | g_loss: 3.1994
Epoch [   41/   50] | d_loss: 0.3159 | g_loss: 2.6462
Epoch [   42/   50] | d_loss: 0.3061 | g_loss: 2.5944
Epoch [   42/   50] | d_loss: 0.1920 | g_loss: 3.4565
Epoch [   42/   50] | d_loss: 0.5018 | g_loss: 1.8128
Epoch [   42/   50] | d_loss: 0.2906 | g_loss: 2.4902
Epoch [   42/   50] | d_loss: 0.3861 | g_loss: 2.0945
Epoch [   42/   50] | d_loss: 0.2451 | g_loss: 3.3947
Epoch [   42/   50] | d_loss: 0.2939 | g_loss: 3.6915
Epoch [   42/   50] | d_loss: 0.2776 | g_loss: 2.6119
Epoch [   42/   50] | d_loss: 0.3171 | g_loss: 3.3116
Epoch [   42/   50] | d_loss: 0.3652 | g_loss: 3.8118
Epoch [   42/   50] | d_loss: 1.0151 | g_loss: 1.0376
Epoch [   42/   50] | d_loss: 0.5026 | g_loss: 3.6243
Epoch [   42/   50] | d_loss: 0.3300 | g_loss: 2.1062
Epoch [   42/   50] | d_loss: 0.2794 | g_loss: 2.2395
Epoch [   42/   50] | d_loss: 0.2870 | g_loss: 3.5209
Epoch [   43/   50] | d_loss: 0.2723 | g_loss: 2.7091
Epoch [   43/   50] | d_loss: 0.4186 | g_loss: 3.1062
Epoch [   43/   50] | d_loss: 0.3898 | g_loss: 3.6012
Epoch [   43/   50] | d_loss: 0.9771 | g_loss: 2.3805
Epoch [   43/   50] | d_loss: 0.4419 | g_loss: 2.4847
Epoch [   43/   50] | d_loss: 0.3213 | g_loss: 2.9525
Epoch [   43/   50] | d_loss: 0.2679 | g_loss: 3.0070
Epoch [   43/   50] | d_loss: 0.2087 | g_loss: 2.3523
Epoch [   43/   50] | d_loss: 0.2766 | g_loss: 3.1586
Epoch [   43/   50] | d_loss: 0.2469 | g_loss: 2.5507
Epoch [   43/   50] | d_loss: 0.8329 | g_loss: 2.0757
Epoch [   43/   50] | d_loss: 0.2748 | g_loss: 2.6200
Epoch [   43/   50] | d_loss: 0.4695 | g_loss: 1.3729
```

```
Epoch [   43/   50] | d_loss: 0.3228 | g_loss: 2.5803
Epoch [   43/   50] | d_loss: 0.8863 | g_loss: 4.4429
Epoch [   44/   50] | d_loss: 1.2493 | g_loss: 0.3800
Epoch [   44/   50] | d_loss: 0.2886 | g_loss: 2.5054
Epoch [   44/   50] | d_loss: 0.2499 | g_loss: 2.3276
Epoch [   44/   50] | d_loss: 0.4236 | g_loss: 2.0569
Epoch [   44/   50] | d_loss: 0.2469 | g_loss: 3.4900
Epoch [   44/   50] | d_loss: 0.2455 | g_loss: 3.0870
Epoch [   44/   50] | d_loss: 0.3469 | g_loss: 2.2592
Epoch [   44/   50] | d_loss: 0.3790 | g_loss: 3.7552
Epoch [   44/   50] | d_loss: 0.4331 | g_loss: 1.9839
Epoch [   44/   50] | d_loss: 0.4599 | g_loss: 1.8852
Epoch [   44/   50] | d_loss: 0.2256 | g_loss: 3.0487
Epoch [   44/   50] | d_loss: 0.3866 | g_loss: 2.3253
Epoch [   44/   50] | d_loss: 0.8974 | g_loss: 4.5047
Epoch [   44/   50] | d_loss: 2.6188 | g_loss: 4.5586
Epoch [   44/   50] | d_loss: 0.1825 | g_loss: 3.8491
Epoch [   45/   50] | d_loss: 0.5131 | g_loss: 3.9875
Epoch [   45/   50] | d_loss: 0.3110 | g_loss: 2.7088
Epoch [   45/   50] | d_loss: 0.2181 | g_loss: 2.6469
Epoch [   45/   50] | d_loss: 0.2624 | g_loss: 3.6378
Epoch [   45/   50] | d_loss: 0.2520 | g_loss: 3.3221
Epoch [   45/   50] | d_loss: 0.2963 | g_loss: 2.3476
Epoch [   45/   50] | d_loss: 0.3000 | g_loss: 3.0327
Epoch [   45/   50] | d_loss: 0.2441 | g_loss: 3.3622
Epoch [   45/   50] | d_loss: 0.1427 | g_loss: 3.5280
Epoch [   45/   50] | d_loss: 0.3482 | g_loss: 3.6149
Epoch [   45/   50] | d_loss: 0.3585 | g_loss: 2.4051
Epoch [   45/   50] | d_loss: 0.3520 | g_loss: 2.2860
Epoch [   45/   50] | d_loss: 0.2261 | g_loss: 3.5747
Epoch [   45/   50] | d_loss: 0.3061 | g_loss: 2.9986
Epoch [   45/   50] | d_loss: 0.3123 | g_loss: 3.1742
Epoch [   46/   50] | d_loss: 0.2339 | g_loss: 2.5305
Epoch [   46/   50] | d_loss: 1.9281 | g_loss: 2.5681
Epoch [   46/   50] | d_loss: 0.2763 | g_loss: 2.4456
Epoch [   46/   50] | d_loss: 0.1779 | g_loss: 2.6705
Epoch [   46/   50] | d_loss: 0.2422 | g_loss: 2.8528
Epoch [   46/   50] | d_loss: 0.3010 | g_loss: 2.7703
Epoch [   46/   50] | d_loss: 0.2012 | g_loss: 2.8797
Epoch [   46/   50] | d_loss: 0.3786 | g_loss: 2.2215
Epoch [   46/   50] | d_loss: 0.1780 | g_loss: 3.4589
Epoch [   46/   50] | d_loss: 0.3775 | g_loss: 2.4861
Epoch [   46/   50] | d_loss: 0.1677 | g_loss: 3.6561
Epoch [   46/   50] | d_loss: 0.2586 | g_loss: 2.5478
Epoch [   46/   50] | d_loss: 0.2895 | g_loss: 2.3085
Epoch [   46/   50] | d_loss: 0.2359 | g_loss: 3.1056
Epoch [   46/   50] | d_loss: 1.3912 | g_loss: 1.1903
Epoch [   47/   50] | d_loss: 1.1339 | g_loss: 2.6682
```

```
Epoch [   47/   50] | d_loss: 0.2586 | g_loss: 2.3702
Epoch [   47/   50] | d_loss: 0.3036 | g_loss: 2.6435
Epoch [   47/   50] | d_loss: 0.2386 | g_loss: 2.8971
Epoch [   47/   50] | d_loss: 1.2621 | g_loss: 1.0425
Epoch [   47/   50] | d_loss: 0.2771 | g_loss: 3.0256
Epoch [   47/   50] | d_loss: 0.2422 | g_loss: 2.8423
Epoch [   47/   50] | d_loss: 0.1538 | g_loss: 3.4082
Epoch [   47/   50] | d_loss: 0.3663 | g_loss: 2.4398
Epoch [   47/   50] | d_loss: 0.3657 | g_loss: 3.8114
Epoch [   47/   50] | d_loss: 0.2171 | g_loss: 3.1648
Epoch [   47/   50] | d_loss: 0.2099 | g_loss: 2.4892
Epoch [   47/   50] | d_loss: 0.4297 | g_loss: 1.2318
Epoch [   47/   50] | d_loss: 0.2288 | g_loss: 2.7974
Epoch [   47/   50] | d_loss: 0.1981 | g_loss: 2.3195
Epoch [   48/   50] | d_loss: 0.2667 | g_loss: 3.5016
Epoch [   48/   50] | d_loss: 0.3745 | g_loss: 3.5354
Epoch [   48/   50] | d_loss: 0.2690 | g_loss: 3.5101
Epoch [   48/   50] | d_loss: 2.8861 | g_loss: 3.4793
Epoch [   48/   50] | d_loss: 0.4031 | g_loss: 2.6316
Epoch [   48/   50] | d_loss: 0.3916 | g_loss: 4.2037
Epoch [   48/   50] | d_loss: 0.2568 | g_loss: 2.4616
Epoch [   48/   50] | d_loss: 0.2730 | g_loss: 2.4233
Epoch [   48/   50] | d_loss: 0.3744 | g_loss: 2.9821
Epoch [   48/   50] | d_loss: 0.4192 | g_loss: 2.1025
Epoch [   48/   50] | d_loss: 0.2500 | g_loss: 2.7349
Epoch [   48/   50] | d_loss: 0.2827 | g_loss: 1.5410
Epoch [   48/   50] | d_loss: 0.3189 | g_loss: 3.8579
Epoch [   48/   50] | d_loss: 0.1766 | g_loss: 2.1546
Epoch [   48/   50] | d_loss: 0.2676 | g_loss: 2.5487
Epoch [   49/   50] | d_loss: 0.4389 | g_loss: 2.9484
Epoch [   49/   50] | d_loss: 0.2807 | g_loss: 2.3586
Epoch [   49/   50] | d_loss: 0.2837 | g_loss: 1.8900
Epoch [   49/   50] | d_loss: 0.1993 | g_loss: 3.1729
Epoch [   49/   50] | d_loss: 0.2338 | g_loss: 2.7091
Epoch [   49/   50] | d_loss: 0.1890 | g_loss: 3.1132
Epoch [   49/   50] | d_loss: 0.3993 | g_loss: 4.0564
Epoch [   49/   50] | d_loss: 0.2990 | g_loss: 3.0883
Epoch [   49/   50] | d_loss: 0.2551 | g_loss: 2.4357
Epoch [   49/   50] | d_loss: 0.3539 | g_loss: 4.0896
Epoch [   49/   50] | d_loss: 0.1699 | g_loss: 3.0317
Epoch [   49/   50] | d_loss: 0.1770 | g_loss: 2.9690
Epoch [   49/   50] | d_loss: 0.2537 | g_loss: 3.2846
Epoch [   49/   50] | d_loss: 0.1434 | g_loss: 3.1556
Epoch [   49/   50] | d_loss: 0.6354 | g_loss: 4.4643
Epoch [   50/   50] | d_loss: 0.6708 | g_loss: 1.3984
Epoch [   50/   50] | d_loss: 0.2242 | g_loss: 2.5698
Epoch [   50/   50] | d_loss: 0.2302 | g_loss: 2.7185
Epoch [   50/   50] | d_loss: 0.2522 | g_loss: 2.2385
```

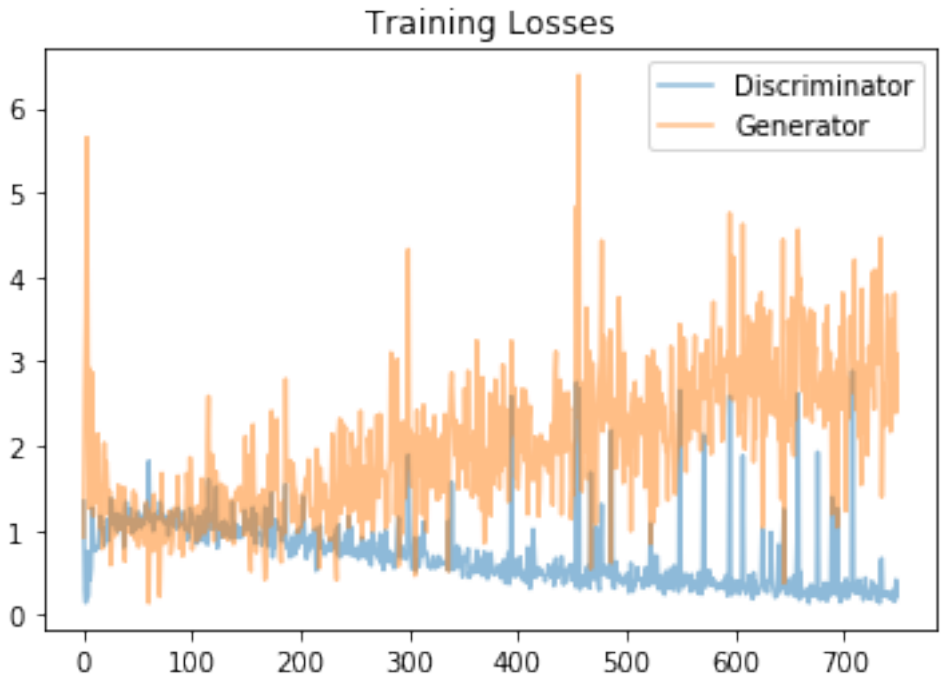
```
Epoch [   50/   50] | d_loss: 0.2228 | g_loss: 3.1890
Epoch [   50/   50] | d_loss: 0.2784 | g_loss: 3.7862
Epoch [   50/   50] | d_loss: 0.2140 | g_loss: 3.4892
Epoch [   50/   50] | d_loss: 0.1990 | g_loss: 3.2475
Epoch [   50/   50] | d_loss: 0.2538 | g_loss: 2.1699
Epoch [   50/   50] | d_loss: 0.1954 | g_loss: 3.4823
Epoch [   50/   50] | d_loss: 0.2905 | g_loss: 3.1636
Epoch [   50/   50] | d_loss: 0.1638 | g_loss: 2.4223
Epoch [   50/   50] | d_loss: 0.1614 | g_loss: 3.8063
Epoch [   50/   50] | d_loss: 0.4174 | g_loss: 2.3920
Epoch [   50/   50] | d_loss: 0.2178 | g_loss: 3.0886
```

2.8 Training loss

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

In [24]:
```
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[24]:
```
<matplotlib.legend.Legend at 0x7fb500aebbe0>
```



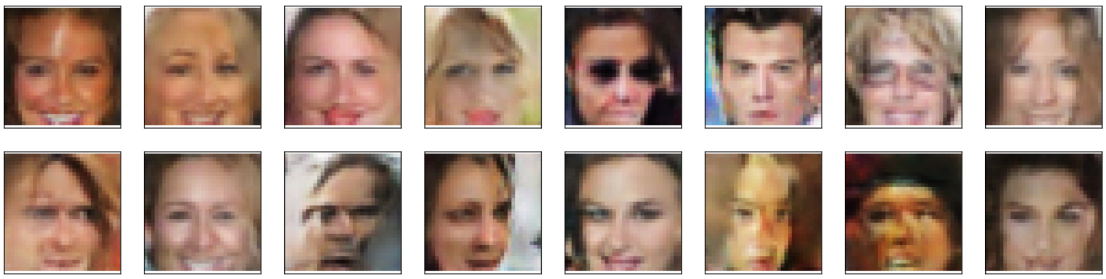
2.9 Generator samples from training

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

```
In [25]: # 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 [26]: # Load samples from generator, taken while training
         with open('train_samples.pkl', 'rb') as f:
             samples = pkl.load(f)
```

```
In [27]: _ = view_samples(-1, samples)
```



2.9.1 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:

1. The generated images are of low resolution. To improve that, a deeper model with more CNN layers should be used.
2. Some of the faces are disfigured as no of epochs were low 50 as its computationally expensive. 100-200 epochs would have given a better score.
3. Dataset is also missing a lot of non-white faces, faces with hats. Also, there might not be a lot of faces not facing forward. I think adding that data might train model better.

4. Different learning rate for discriminator and generator could have been tried https://arxiv.org/pdf/1704.00028.pdf article used WGAN Adam (= .0001, 1 = .5, 2 = .9)

5. Image size could be increased to 128.

6. Other optimizer than Adam optimizer could be tried to get better performance

7. The original article changed momentum beta 1 from 0.9 to 0.5 but we can try tuning that.

8. Adding random noise to the labels in the discriminator might train the model better. https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/

9. Dropout 0.5 during training might be helpful

2.9.2 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.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: