

dlnd_face_generation

May 20, 2020

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

```
Archive:  processed_celeba_small.zip
  creating: processed_celeba_small/
  inflating: processed_celeba_small/.DS_Store
  creating: __MACOSX/
  creating: __MACOSX/processed_celeba_small/
  inflating: __MACOSX/processed_celeba_small/..DS_Store
  creating: processed_celeba_small/celeba/
```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
inflating: processed_celeba_small/celeba/New Folder With Items/057301.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057302.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057303.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057304.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057305.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057306.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057307.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057308.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057309.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057310.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057311.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057312.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057313.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057314.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057315.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057316.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057317.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057318.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057319.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057320.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057321.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057322.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057323.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057324.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057325.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057326.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057327.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057328.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057329.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057330.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/057331.jpg
```

```
In [2]: 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 [3]: # necessary imports
```

```
import torch
from torchvision import datasets
from torchvision import transforms
```

```
In [4]: def get_dataloader(batch_size, image_size, data_dir='processed_celeba_small/'):
        """
```

```
    Batch the neural network data using DataLoader
    :param batch_size: The size of each batch; the number of images in a batch
    :param img_size: The square size of the image data (x, y)
    :param data_dir: Directory where image data is located
    :return: DataLoader with batched data
    """
```

```
    # TODO: Implement function and return a dataloader
```

```
    transform = transforms.Compose([transforms.Resize(image_size),
                                    transforms.ToTensor()])
```

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

```
    return torch.utils.data.DataLoader(image_dataset, batch_size = batch_size, shuffle=True)
```

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 [5]: # Define function hyperparameters
        batch_size = 20
        img_size = 32

        """
        DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
        """

        # Call your function and get a dataloader
        celeba_train_loader = get_dataloader(batch_size, img_size)
```

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

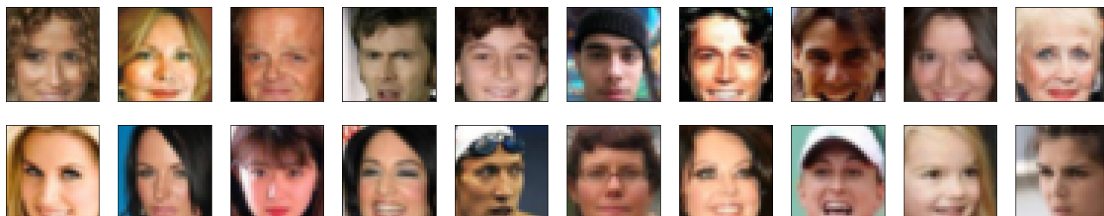
Note: You'll need to convert the Tensor images into a NumPy type and transpose the dimensions to correctly display an image, suggested `imshow` code is below, but it may not be perfect.

```
In [6]: # 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 [8]: # 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(-0.9765)

Max: tensor(0.8039)

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 [10]: import torch.nn as nn
         import torch.nn.functional as F

In [12]: # helper conv function
         def conv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True):
             """Creates a convolutional layer, with optional batch normalization.
             """
             layers = []
             conv_layer = nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                                     kernel_size=kernel_size, stride=stride, padding=padding, bias=True)

             layers.append(conv_layer)

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

In [13]: 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__()
             super(Discriminator, self).__init__()

             # complete init function
             self.conv_dim = conv_dim

             # Define all convolutional layers
             # Should accept an RGB image as input and output a single value
             self.conv1 = conv(3, conv_dim, 4, batch_norm=False) # x, y = 64 depth = 3
             self.conv2 = conv(conv_dim, conv_dim * 2, 4) # x, y = 32 depth = 64
             self.conv3 = conv(conv_dim * 2, conv_dim * 4, 4) # x, y = 16 depth = 128

             self.fc = nn.Linear(conv_dim*4*4*4, 1)
             self.out = nn.Sigmoid()
             self.dropout = nn.Dropout(0.5)

         def forward(self, x):
             """
             Forward propagation of the neural network
             :param x: The input to the neural network
             :return: Discriminator logits; the output of the neural network
             """
             # define feedforward behavior
             x = F.leaky_relu(self.conv1(x), 0.2)

```

```

        # x = self.dropout(x)
        x = F.leaky_relu(self.conv2(x), 0.2)
        # x = self.dropout(x)
        x = F.leaky_relu(self.conv3(x), 0.2)
        # x = self.dropout(x)

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

        x = self.fc(x)
        x = self.dropout(x)

        # x = self.out(x)

        return x

    """
    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 [14]: def deconv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True):
        layers = []
        layers.append(nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride, padding))
        if batch_norm:
            layers.append(nn.BatchNorm2d(out_channels))
        return nn.Sequential(*layers)

In [15]: class Generator(nn.Module):

        def __init__(self, z_size, conv_dim):
            """
            Initialize the Generator Module
            :param z_size: The length of the input latent vector, z

```



```

        :param conv_dim: The depth of the inputs to the *last* transpose convolutional
        """
        super(Generator, self).__init__()

        # complete init function
        self.conv_dim = conv_dim

        self.fc = nn.Linear(z_size, conv_dim*8*2*2)

        self.t_conv1 = deconv(conv_dim*8, conv_dim*4, 4)
        self.t_conv2 = deconv(conv_dim*4, conv_dim*2, 4)
        self.t_conv3 = deconv(conv_dim*2, conv_dim, 4)
        self.t_conv4 = 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
        out = self.fc(x)
        out = out.view(-1, self.conv_dim*8, 2, 2) # (batch_size, depth, 4, 4)

        out = F.relu(self.t_conv1(out))
        out = F.relu(self.t_conv2(out))
        out = F.relu(self.t_conv3(out))

        # last layer: tanh activation instead of relu
        out = self.t_conv4(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 [16]: from torch.nn import init
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
    classname = m.__class__.__name__
    isConvolution = classname.find('Conv') != -1
    isLinear = classname.find('Linear') != -1
    if (hasattr(m, 'weight') and isConvolution or isLinear):
        init.normal_(m.weight.data, 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 [17]: """
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 [18]: *# 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)
    (out): Sigmoid()
    (dropout): Dropout(p=0.5)
)

```

Generator(

```

    (fc): Linear(in_features=100, out_features=1024, bias=True)
    (t_conv1): Sequential(
      (0): ConvTranspose2d(256, 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)
    )
    (t_conv2): 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_conv3): 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_conv4): 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 [19]: """
          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 [20]: def real_loss(D_out):
    '''Calculates how close discriminator outputs are to being real.
        param, D_out: discriminator logits
        return: real loss'''
    batch_size = D_out.size(0)
    labels = torch.ones(batch_size)
    if train_on_gpu:
        labels = labels.cuda()

    criterion = nn.BCEWithLogitsLoss()
    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)
    if train_on_gpu:
        labels = labels.cuda()

    criterion = nn.BCEWithLogitsLoss()
    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 [22]: import torch.optim as optim
    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 [23]: 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
```

```

# =====

d_optimizer.zero_grad()
if train_on_gpu:
    real_images = real_images.cuda()

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

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)
D_fake = D(fake_images)
d_fake_loss = fake_loss(D_fake)

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()
if train_on_gpu:
    z = z.cuda()

fake_images = G(z)
D_fake = D(fake_images)
g_loss = real_loss(D_fake)

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: {:.4f} | g_loss: {:.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:
        pickle.dump(samples, f)

    # finally return losses
    return losses

```

Set your number of training epochs and train your GAN!

```

In [24]: # set number of epochs
         n_epochs = 10

         """
         DON'T MODIFY ANYTHING IN THIS CELL
         """

         # call training function
         losses = train(D, G, n_epochs=n_epochs)

```

Epoch [1/	10]	d_loss: 1.5693	g_loss: 0.7069
Epoch [1/	10]	d_loss: 0.9577	g_loss: 1.5777
Epoch [1/	10]	d_loss: 0.8110	g_loss: 2.1763
Epoch [1/	10]	d_loss: 0.6020	g_loss: 2.5129
Epoch [1/	10]	d_loss: 0.7518	g_loss: 2.9524
Epoch [1/	10]	d_loss: 0.6178	g_loss: 2.5002
Epoch [1/	10]	d_loss: 0.7631	g_loss: 2.5883
Epoch [1/	10]	d_loss: 0.5278	g_loss: 2.9507
Epoch [1/	10]	d_loss: 0.9082	g_loss: 3.6454
Epoch [1/	10]	d_loss: 0.8622	g_loss: 1.8018
Epoch [1/	10]	d_loss: 0.8999	g_loss: 2.2114
Epoch [1/	10]	d_loss: 0.6699	g_loss: 4.2632
Epoch [1/	10]	d_loss: 0.8782	g_loss: 1.8316
Epoch [1/	10]	d_loss: 1.5005	g_loss: 1.4879
Epoch [1/	10]	d_loss: 0.8208	g_loss: 2.9958
Epoch [1/	10]	d_loss: 1.0077	g_loss: 1.6996
Epoch [1/	10]	d_loss: 0.9046	g_loss: 1.7947
Epoch [1/	10]	d_loss: 0.9639	g_loss: 1.9659
Epoch [1/	10]	d_loss: 1.0313	g_loss: 1.2126

Epoch [1/	10]	d_loss: 1.0947	g_loss: 2.5700
Epoch [1/	10]	d_loss: 1.1763	g_loss: 1.7608
Epoch [1/	10]	d_loss: 1.1178	g_loss: 1.2219
Epoch [1/	10]	d_loss: 0.9839	g_loss: 1.5222
Epoch [1/	10]	d_loss: 0.9390	g_loss: 1.4556
Epoch [1/	10]	d_loss: 1.1845	g_loss: 1.8472
Epoch [1/	10]	d_loss: 1.9044	g_loss: 1.3453
Epoch [1/	10]	d_loss: 0.9985	g_loss: 2.4406
Epoch [1/	10]	d_loss: 1.0220	g_loss: 0.9968
Epoch [1/	10]	d_loss: 1.2119	g_loss: 1.2250
Epoch [1/	10]	d_loss: 1.0040	g_loss: 2.1867
Epoch [1/	10]	d_loss: 1.1383	g_loss: 1.6209
Epoch [1/	10]	d_loss: 1.3808	g_loss: 0.8505
Epoch [1/	10]	d_loss: 1.0967	g_loss: 2.1239
Epoch [1/	10]	d_loss: 1.1063	g_loss: 1.7644
Epoch [1/	10]	d_loss: 1.2885	g_loss: 1.1232
Epoch [1/	10]	d_loss: 1.1312	g_loss: 0.9974
Epoch [1/	10]	d_loss: 1.5967	g_loss: 2.6834
Epoch [1/	10]	d_loss: 1.0418	g_loss: 0.9777
Epoch [1/	10]	d_loss: 1.0795	g_loss: 1.8018
Epoch [1/	10]	d_loss: 1.2053	g_loss: 1.2837
Epoch [1/	10]	d_loss: 1.2351	g_loss: 1.4183
Epoch [1/	10]	d_loss: 1.0878	g_loss: 1.5046
Epoch [1/	10]	d_loss: 1.5055	g_loss: 1.8770
Epoch [1/	10]	d_loss: 1.1637	g_loss: 1.0266
Epoch [1/	10]	d_loss: 1.1324	g_loss: 1.3520
Epoch [1/	10]	d_loss: 1.2126	g_loss: 0.7946
Epoch [1/	10]	d_loss: 1.5498	g_loss: 0.6890
Epoch [1/	10]	d_loss: 0.9943	g_loss: 1.3944
Epoch [1/	10]	d_loss: 1.2253	g_loss: 0.8888
Epoch [1/	10]	d_loss: 1.1412	g_loss: 0.7613
Epoch [1/	10]	d_loss: 1.5445	g_loss: 0.5804
Epoch [1/	10]	d_loss: 1.2485	g_loss: 1.0301
Epoch [1/	10]	d_loss: 1.2063	g_loss: 1.3231
Epoch [1/	10]	d_loss: 1.3252	g_loss: 1.1165
Epoch [1/	10]	d_loss: 1.0144	g_loss: 0.8594
Epoch [1/	10]	d_loss: 1.1666	g_loss: 0.7928
Epoch [1/	10]	d_loss: 1.2398	g_loss: 1.2771
Epoch [1/	10]	d_loss: 1.4143	g_loss: 1.3454
Epoch [1/	10]	d_loss: 1.2641	g_loss: 1.4376
Epoch [1/	10]	d_loss: 0.9634	g_loss: 1.0589
Epoch [1/	10]	d_loss: 1.5205	g_loss: 1.4394
Epoch [1/	10]	d_loss: 1.3339	g_loss: 1.0795
Epoch [1/	10]	d_loss: 1.0454	g_loss: 0.9339
Epoch [1/	10]	d_loss: 1.1655	g_loss: 1.7573
Epoch [1/	10]	d_loss: 1.7106	g_loss: 0.7959
Epoch [1/	10]	d_loss: 1.2785	g_loss: 1.6472
Epoch [1/	10]	d_loss: 0.8678	g_loss: 1.5482

Epoch [1/	10]	d_loss: 1.0462	g_loss: 0.9639
Epoch [1/	10]	d_loss: 1.2079	g_loss: 1.3272
Epoch [1/	10]	d_loss: 1.1503	g_loss: 1.2413
Epoch [1/	10]	d_loss: 1.5833	g_loss: 0.8799
Epoch [1/	10]	d_loss: 1.0413	g_loss: 0.9011
Epoch [1/	10]	d_loss: 1.3444	g_loss: 1.1904
Epoch [1/	10]	d_loss: 1.1965	g_loss: 0.8173
Epoch [1/	10]	d_loss: 1.3008	g_loss: 0.9674
Epoch [1/	10]	d_loss: 0.9506	g_loss: 1.1132
Epoch [1/	10]	d_loss: 0.9883	g_loss: 0.8242
Epoch [1/	10]	d_loss: 1.1252	g_loss: 1.3003
Epoch [1/	10]	d_loss: 1.1280	g_loss: 0.8829
Epoch [1/	10]	d_loss: 1.2691	g_loss: 0.9984
Epoch [1/	10]	d_loss: 1.0796	g_loss: 1.3408
Epoch [1/	10]	d_loss: 1.3459	g_loss: 0.8232
Epoch [1/	10]	d_loss: 0.9181	g_loss: 1.6613
Epoch [1/	10]	d_loss: 1.2368	g_loss: 1.0843
Epoch [1/	10]	d_loss: 0.9581	g_loss: 1.1737
Epoch [1/	10]	d_loss: 1.1173	g_loss: 0.9917
Epoch [1/	10]	d_loss: 1.0260	g_loss: 0.9274
Epoch [1/	10]	d_loss: 1.1076	g_loss: 1.7054
Epoch [1/	10]	d_loss: 1.0439	g_loss: 1.5041
Epoch [1/	10]	d_loss: 1.1223	g_loss: 1.0392
Epoch [2/	10]	d_loss: 0.9650	g_loss: 1.2182
Epoch [2/	10]	d_loss: 1.0721	g_loss: 0.9264
Epoch [2/	10]	d_loss: 1.0595	g_loss: 0.9616
Epoch [2/	10]	d_loss: 1.3546	g_loss: 1.1965
Epoch [2/	10]	d_loss: 1.2630	g_loss: 0.9799
Epoch [2/	10]	d_loss: 1.2665	g_loss: 1.0201
Epoch [2/	10]	d_loss: 1.1820	g_loss: 0.8690
Epoch [2/	10]	d_loss: 1.1798	g_loss: 1.0846
Epoch [2/	10]	d_loss: 1.2482	g_loss: 0.8309
Epoch [2/	10]	d_loss: 1.0998	g_loss: 1.0316
Epoch [2/	10]	d_loss: 0.9432	g_loss: 1.4167
Epoch [2/	10]	d_loss: 1.1936	g_loss: 1.1133
Epoch [2/	10]	d_loss: 1.1188	g_loss: 0.9788
Epoch [2/	10]	d_loss: 1.2942	g_loss: 0.9904
Epoch [2/	10]	d_loss: 1.1154	g_loss: 1.1959
Epoch [2/	10]	d_loss: 1.3601	g_loss: 0.8323
Epoch [2/	10]	d_loss: 1.1873	g_loss: 0.6792
Epoch [2/	10]	d_loss: 1.0010	g_loss: 0.9897
Epoch [2/	10]	d_loss: 1.1017	g_loss: 0.9026
Epoch [2/	10]	d_loss: 1.6533	g_loss: 0.6716
Epoch [2/	10]	d_loss: 1.1510	g_loss: 1.8388
Epoch [2/	10]	d_loss: 1.1351	g_loss: 1.6205
Epoch [2/	10]	d_loss: 1.1485	g_loss: 0.9803
Epoch [2/	10]	d_loss: 1.2235	g_loss: 0.8937
Epoch [2/	10]	d_loss: 1.2383	g_loss: 1.5005

Epoch [2/	10]	d_loss: 1.0954	g_loss: 1.0564
Epoch [2/	10]	d_loss: 1.1457	g_loss: 0.8949
Epoch [2/	10]	d_loss: 1.0750	g_loss: 1.0283
Epoch [2/	10]	d_loss: 0.9330	g_loss: 1.2401
Epoch [2/	10]	d_loss: 1.1496	g_loss: 1.5108
Epoch [2/	10]	d_loss: 1.3060	g_loss: 1.2368
Epoch [2/	10]	d_loss: 0.9098	g_loss: 1.8673
Epoch [2/	10]	d_loss: 1.0256	g_loss: 1.2057
Epoch [2/	10]	d_loss: 1.5742	g_loss: 0.9137
Epoch [2/	10]	d_loss: 0.9518	g_loss: 0.8638
Epoch [2/	10]	d_loss: 0.9696	g_loss: 1.2311
Epoch [2/	10]	d_loss: 1.0988	g_loss: 0.9067
Epoch [2/	10]	d_loss: 1.4376	g_loss: 0.7763
Epoch [2/	10]	d_loss: 1.2092	g_loss: 1.2240
Epoch [2/	10]	d_loss: 1.0236	g_loss: 1.1293
Epoch [2/	10]	d_loss: 1.1303	g_loss: 1.1429
Epoch [2/	10]	d_loss: 0.9739	g_loss: 1.1151
Epoch [2/	10]	d_loss: 1.3662	g_loss: 0.9861
Epoch [2/	10]	d_loss: 1.1232	g_loss: 0.9144
Epoch [2/	10]	d_loss: 1.0965	g_loss: 1.4587
Epoch [2/	10]	d_loss: 1.5030	g_loss: 0.9147
Epoch [2/	10]	d_loss: 1.4187	g_loss: 1.3438
Epoch [2/	10]	d_loss: 1.0528	g_loss: 0.8191
Epoch [2/	10]	d_loss: 1.1661	g_loss: 0.8501
Epoch [2/	10]	d_loss: 1.2702	g_loss: 1.3788
Epoch [2/	10]	d_loss: 1.4108	g_loss: 1.0840
Epoch [2/	10]	d_loss: 1.1174	g_loss: 1.1790
Epoch [2/	10]	d_loss: 1.0673	g_loss: 1.9673
Epoch [2/	10]	d_loss: 1.3395	g_loss: 0.8230
Epoch [2/	10]	d_loss: 1.0942	g_loss: 1.0681
Epoch [2/	10]	d_loss: 1.3211	g_loss: 1.0941
Epoch [2/	10]	d_loss: 1.2480	g_loss: 1.1751
Epoch [2/	10]	d_loss: 1.1157	g_loss: 1.3314
Epoch [2/	10]	d_loss: 1.0875	g_loss: 1.2261
Epoch [2/	10]	d_loss: 1.2598	g_loss: 0.9041
Epoch [2/	10]	d_loss: 1.0878	g_loss: 1.0588
Epoch [2/	10]	d_loss: 1.1948	g_loss: 1.1728
Epoch [2/	10]	d_loss: 1.2117	g_loss: 1.2401
Epoch [2/	10]	d_loss: 1.1773	g_loss: 1.0874
Epoch [2/	10]	d_loss: 1.2043	g_loss: 1.1219
Epoch [2/	10]	d_loss: 1.0621	g_loss: 1.2827
Epoch [2/	10]	d_loss: 1.1312	g_loss: 0.9363
Epoch [2/	10]	d_loss: 1.2011	g_loss: 0.9984
Epoch [2/	10]	d_loss: 1.0619	g_loss: 1.2185
Epoch [2/	10]	d_loss: 1.2379	g_loss: 1.5300
Epoch [2/	10]	d_loss: 1.1691	g_loss: 0.9933
Epoch [2/	10]	d_loss: 1.1069	g_loss: 0.7306
Epoch [2/	10]	d_loss: 1.0616	g_loss: 1.1844

Epoch [2/	10]	d_loss: 1.1615	g_loss: 1.3539
Epoch [2/	10]	d_loss: 1.2490	g_loss: 1.2400
Epoch [2/	10]	d_loss: 1.2206	g_loss: 1.1303
Epoch [2/	10]	d_loss: 1.2769	g_loss: 1.1672
Epoch [2/	10]	d_loss: 1.1854	g_loss: 1.3129
Epoch [2/	10]	d_loss: 1.1349	g_loss: 1.3197
Epoch [2/	10]	d_loss: 0.9643	g_loss: 1.5418
Epoch [2/	10]	d_loss: 1.3382	g_loss: 1.1375
Epoch [2/	10]	d_loss: 1.1310	g_loss: 1.1004
Epoch [2/	10]	d_loss: 1.3037	g_loss: 1.2476
Epoch [2/	10]	d_loss: 1.1943	g_loss: 0.9409
Epoch [2/	10]	d_loss: 1.2467	g_loss: 1.2039
Epoch [2/	10]	d_loss: 1.0156	g_loss: 0.9185
Epoch [2/	10]	d_loss: 1.1058	g_loss: 1.1119
Epoch [2/	10]	d_loss: 1.0070	g_loss: 1.1517
Epoch [2/	10]	d_loss: 1.3853	g_loss: 0.9467
Epoch [2/	10]	d_loss: 1.1610	g_loss: 1.7116
Epoch [3/	10]	d_loss: 1.3203	g_loss: 0.9454
Epoch [3/	10]	d_loss: 1.3036	g_loss: 1.0935
Epoch [3/	10]	d_loss: 1.1128	g_loss: 1.4200
Epoch [3/	10]	d_loss: 1.0721	g_loss: 0.9531
Epoch [3/	10]	d_loss: 1.1374	g_loss: 0.9357
Epoch [3/	10]	d_loss: 1.2775	g_loss: 1.0084
Epoch [3/	10]	d_loss: 1.0121	g_loss: 0.9983
Epoch [3/	10]	d_loss: 1.1084	g_loss: 1.5347
Epoch [3/	10]	d_loss: 1.0721	g_loss: 1.3159
Epoch [3/	10]	d_loss: 1.0227	g_loss: 1.0403
Epoch [3/	10]	d_loss: 1.2449	g_loss: 1.0863
Epoch [3/	10]	d_loss: 1.1420	g_loss: 1.2903
Epoch [3/	10]	d_loss: 1.2328	g_loss: 0.9671
Epoch [3/	10]	d_loss: 1.4292	g_loss: 0.8428
Epoch [3/	10]	d_loss: 1.2728	g_loss: 1.0697
Epoch [3/	10]	d_loss: 0.7576	g_loss: 1.1501
Epoch [3/	10]	d_loss: 1.3534	g_loss: 0.7315
Epoch [3/	10]	d_loss: 1.1397	g_loss: 0.9944
Epoch [3/	10]	d_loss: 1.2991	g_loss: 0.6208
Epoch [3/	10]	d_loss: 1.2280	g_loss: 0.7647
Epoch [3/	10]	d_loss: 1.1225	g_loss: 1.1797
Epoch [3/	10]	d_loss: 1.3931	g_loss: 1.0551
Epoch [3/	10]	d_loss: 1.1572	g_loss: 1.1307
Epoch [3/	10]	d_loss: 1.3261	g_loss: 0.8560
Epoch [3/	10]	d_loss: 1.3191	g_loss: 0.7414
Epoch [3/	10]	d_loss: 1.2849	g_loss: 0.9327
Epoch [3/	10]	d_loss: 1.2053	g_loss: 1.4024
Epoch [3/	10]	d_loss: 1.1424	g_loss: 0.9656
Epoch [3/	10]	d_loss: 1.1867	g_loss: 1.0507
Epoch [3/	10]	d_loss: 1.3855	g_loss: 1.0057
Epoch [3/	10]	d_loss: 0.9179	g_loss: 0.9481

Epoch [3/	10]	d_loss: 1.4198	g_loss: 1.4739
Epoch [3/	10]	d_loss: 1.0958	g_loss: 0.8688
Epoch [3/	10]	d_loss: 1.1818	g_loss: 1.2651
Epoch [3/	10]	d_loss: 1.7152	g_loss: 1.0790
Epoch [3/	10]	d_loss: 1.1192	g_loss: 1.0069
Epoch [3/	10]	d_loss: 1.2029	g_loss: 1.1034
Epoch [3/	10]	d_loss: 1.3149	g_loss: 0.7904
Epoch [3/	10]	d_loss: 1.3537	g_loss: 1.1113
Epoch [3/	10]	d_loss: 1.3980	g_loss: 0.8150
Epoch [3/	10]	d_loss: 1.1357	g_loss: 1.2283
Epoch [3/	10]	d_loss: 0.9478	g_loss: 0.9794
Epoch [3/	10]	d_loss: 1.2530	g_loss: 0.8881
Epoch [3/	10]	d_loss: 1.2402	g_loss: 1.0092
Epoch [3/	10]	d_loss: 1.0826	g_loss: 0.9212
Epoch [3/	10]	d_loss: 1.0977	g_loss: 0.8103
Epoch [3/	10]	d_loss: 1.0596	g_loss: 1.1749
Epoch [3/	10]	d_loss: 1.0708	g_loss: 0.9377
Epoch [3/	10]	d_loss: 1.2823	g_loss: 0.9733
Epoch [3/	10]	d_loss: 1.2453	g_loss: 1.1175
Epoch [3/	10]	d_loss: 1.1949	g_loss: 0.8738
Epoch [3/	10]	d_loss: 1.1094	g_loss: 1.0042
Epoch [3/	10]	d_loss: 1.4673	g_loss: 0.7635
Epoch [3/	10]	d_loss: 1.5505	g_loss: 1.2650
Epoch [3/	10]	d_loss: 1.1572	g_loss: 0.9331
Epoch [3/	10]	d_loss: 1.0699	g_loss: 1.7845
Epoch [3/	10]	d_loss: 1.2729	g_loss: 1.1388
Epoch [3/	10]	d_loss: 1.0853	g_loss: 1.2645
Epoch [3/	10]	d_loss: 1.2476	g_loss: 1.5634
Epoch [3/	10]	d_loss: 1.1531	g_loss: 0.9229
Epoch [3/	10]	d_loss: 1.2755	g_loss: 0.9513
Epoch [3/	10]	d_loss: 1.1869	g_loss: 1.0744
Epoch [3/	10]	d_loss: 0.9354	g_loss: 1.0778
Epoch [3/	10]	d_loss: 1.4333	g_loss: 1.2571
Epoch [3/	10]	d_loss: 1.2949	g_loss: 0.9955
Epoch [3/	10]	d_loss: 1.2288	g_loss: 0.7946
Epoch [3/	10]	d_loss: 1.0496	g_loss: 1.4860
Epoch [3/	10]	d_loss: 1.2199	g_loss: 1.0725
Epoch [3/	10]	d_loss: 0.9871	g_loss: 1.2845
Epoch [3/	10]	d_loss: 1.1300	g_loss: 0.9692
Epoch [3/	10]	d_loss: 1.1175	g_loss: 0.8632
Epoch [3/	10]	d_loss: 1.3231	g_loss: 0.8765
Epoch [3/	10]	d_loss: 1.1350	g_loss: 1.0917
Epoch [3/	10]	d_loss: 1.2316	g_loss: 1.2527
Epoch [3/	10]	d_loss: 1.2113	g_loss: 0.6046
Epoch [3/	10]	d_loss: 1.2960	g_loss: 1.2076
Epoch [3/	10]	d_loss: 1.0996	g_loss: 0.8856
Epoch [3/	10]	d_loss: 1.2783	g_loss: 0.9941
Epoch [3/	10]	d_loss: 1.1836	g_loss: 0.9686

Epoch [3/	10]	d_loss: 1.3035	g_loss: 1.5222
Epoch [3/	10]	d_loss: 1.1098	g_loss: 0.8437
Epoch [3/	10]	d_loss: 1.3423	g_loss: 1.0712
Epoch [3/	10]	d_loss: 1.1453	g_loss: 0.9117
Epoch [3/	10]	d_loss: 1.1683	g_loss: 0.9353
Epoch [3/	10]	d_loss: 1.0987	g_loss: 0.9625
Epoch [3/	10]	d_loss: 1.2255	g_loss: 0.7001
Epoch [3/	10]	d_loss: 1.2042	g_loss: 0.7409
Epoch [3/	10]	d_loss: 1.2043	g_loss: 0.6676
Epoch [3/	10]	d_loss: 1.1042	g_loss: 0.9042
Epoch [3/	10]	d_loss: 1.2145	g_loss: 0.6924
Epoch [4/	10]	d_loss: 1.2343	g_loss: 1.0067
Epoch [4/	10]	d_loss: 1.1062	g_loss: 1.1159
Epoch [4/	10]	d_loss: 1.3310	g_loss: 0.9343
Epoch [4/	10]	d_loss: 1.0094	g_loss: 0.9992
Epoch [4/	10]	d_loss: 1.1446	g_loss: 0.9613
Epoch [4/	10]	d_loss: 1.2678	g_loss: 1.6457
Epoch [4/	10]	d_loss: 1.1971	g_loss: 0.8057
Epoch [4/	10]	d_loss: 1.1681	g_loss: 0.9114
Epoch [4/	10]	d_loss: 1.2793	g_loss: 0.9259
Epoch [4/	10]	d_loss: 1.1556	g_loss: 0.8536
Epoch [4/	10]	d_loss: 1.0807	g_loss: 1.1111
Epoch [4/	10]	d_loss: 1.4349	g_loss: 0.7261
Epoch [4/	10]	d_loss: 1.3615	g_loss: 1.0124
Epoch [4/	10]	d_loss: 1.3015	g_loss: 0.7785
Epoch [4/	10]	d_loss: 1.1054	g_loss: 1.1502
Epoch [4/	10]	d_loss: 1.3758	g_loss: 0.4477
Epoch [4/	10]	d_loss: 1.0676	g_loss: 0.7636
Epoch [4/	10]	d_loss: 1.3390	g_loss: 0.9365
Epoch [4/	10]	d_loss: 1.3464	g_loss: 0.9001
Epoch [4/	10]	d_loss: 1.3932	g_loss: 0.8701
Epoch [4/	10]	d_loss: 1.3393	g_loss: 1.4436
Epoch [4/	10]	d_loss: 1.2308	g_loss: 1.1390
Epoch [4/	10]	d_loss: 1.2795	g_loss: 0.9749
Epoch [4/	10]	d_loss: 1.0020	g_loss: 1.1288
Epoch [4/	10]	d_loss: 1.2652	g_loss: 1.0767
Epoch [4/	10]	d_loss: 1.2471	g_loss: 1.3781
Epoch [4/	10]	d_loss: 1.3216	g_loss: 0.7591
Epoch [4/	10]	d_loss: 0.9523	g_loss: 1.2771
Epoch [4/	10]	d_loss: 1.2164	g_loss: 0.6282
Epoch [4/	10]	d_loss: 1.1894	g_loss: 0.8164
Epoch [4/	10]	d_loss: 1.0255	g_loss: 0.9176
Epoch [4/	10]	d_loss: 1.1029	g_loss: 1.7125
Epoch [4/	10]	d_loss: 1.2301	g_loss: 0.7953
Epoch [4/	10]	d_loss: 1.0889	g_loss: 1.3598
Epoch [4/	10]	d_loss: 1.2862	g_loss: 0.9320
Epoch [4/	10]	d_loss: 1.3119	g_loss: 1.0733
Epoch [4/	10]	d_loss: 1.0215	g_loss: 1.7325

Epoch [4/	10]	d_loss: 1.1004	g_loss: 0.7817
Epoch [4/	10]	d_loss: 1.3521	g_loss: 1.1286
Epoch [4/	10]	d_loss: 1.1964	g_loss: 0.6815
Epoch [4/	10]	d_loss: 1.1311	g_loss: 1.1641
Epoch [4/	10]	d_loss: 1.2559	g_loss: 0.8190
Epoch [4/	10]	d_loss: 1.0516	g_loss: 0.7905
Epoch [4/	10]	d_loss: 1.2724	g_loss: 1.0544
Epoch [4/	10]	d_loss: 1.2829	g_loss: 0.8881
Epoch [4/	10]	d_loss: 1.2467	g_loss: 1.1500
Epoch [4/	10]	d_loss: 1.0614	g_loss: 0.7874
Epoch [4/	10]	d_loss: 1.0734	g_loss: 1.0114
Epoch [4/	10]	d_loss: 1.1221	g_loss: 1.2324
Epoch [4/	10]	d_loss: 1.3202	g_loss: 0.9868
Epoch [4/	10]	d_loss: 1.3550	g_loss: 0.7363
Epoch [4/	10]	d_loss: 1.2667	g_loss: 0.7922
Epoch [4/	10]	d_loss: 1.3508	g_loss: 0.9103
Epoch [4/	10]	d_loss: 1.2793	g_loss: 1.2573
Epoch [4/	10]	d_loss: 1.1545	g_loss: 1.2429
Epoch [4/	10]	d_loss: 1.2266	g_loss: 0.6235
Epoch [4/	10]	d_loss: 1.1924	g_loss: 1.2313
Epoch [4/	10]	d_loss: 1.1642	g_loss: 0.9211
Epoch [4/	10]	d_loss: 1.3707	g_loss: 1.2028
Epoch [4/	10]	d_loss: 1.2129	g_loss: 0.7635
Epoch [4/	10]	d_loss: 1.2241	g_loss: 0.8544
Epoch [4/	10]	d_loss: 1.2586	g_loss: 0.9832
Epoch [4/	10]	d_loss: 1.4519	g_loss: 0.9701
Epoch [4/	10]	d_loss: 1.2911	g_loss: 0.9363
Epoch [4/	10]	d_loss: 1.1306	g_loss: 0.8508
Epoch [4/	10]	d_loss: 1.1731	g_loss: 0.8744
Epoch [4/	10]	d_loss: 1.1267	g_loss: 1.1147
Epoch [4/	10]	d_loss: 1.3385	g_loss: 1.1254
Epoch [4/	10]	d_loss: 1.2196	g_loss: 1.1041
Epoch [4/	10]	d_loss: 1.2774	g_loss: 1.0054
Epoch [4/	10]	d_loss: 1.1716	g_loss: 1.0168
Epoch [4/	10]	d_loss: 1.2466	g_loss: 0.9136
Epoch [4/	10]	d_loss: 1.0670	g_loss: 0.9165
Epoch [4/	10]	d_loss: 1.3566	g_loss: 1.0136
Epoch [4/	10]	d_loss: 1.4449	g_loss: 0.9760
Epoch [4/	10]	d_loss: 1.1711	g_loss: 0.9633
Epoch [4/	10]	d_loss: 1.2380	g_loss: 0.4902
Epoch [4/	10]	d_loss: 1.7472	g_loss: 0.7112
Epoch [4/	10]	d_loss: 1.3348	g_loss: 1.1778
Epoch [4/	10]	d_loss: 1.3437	g_loss: 1.1702
Epoch [4/	10]	d_loss: 1.4462	g_loss: 0.8856
Epoch [4/	10]	d_loss: 1.1282	g_loss: 0.9104
Epoch [4/	10]	d_loss: 1.4113	g_loss: 1.0151
Epoch [4/	10]	d_loss: 1.1542	g_loss: 1.0908
Epoch [4/	10]	d_loss: 1.5600	g_loss: 1.0218

Epoch [4/	10]	d_loss: 1.2192	g_loss: 0.8732
Epoch [4/	10]	d_loss: 1.0964	g_loss: 1.0201
Epoch [4/	10]	d_loss: 1.1839	g_loss: 0.7633
Epoch [4/	10]	d_loss: 1.0593	g_loss: 0.8247
Epoch [4/	10]	d_loss: 1.1024	g_loss: 0.8021
Epoch [5/	10]	d_loss: 1.2780	g_loss: 0.8461
Epoch [5/	10]	d_loss: 1.1190	g_loss: 1.1712
Epoch [5/	10]	d_loss: 1.2292	g_loss: 1.3754
Epoch [5/	10]	d_loss: 1.2225	g_loss: 0.9516
Epoch [5/	10]	d_loss: 1.2444	g_loss: 0.9639
Epoch [5/	10]	d_loss: 1.1898	g_loss: 1.1622
Epoch [5/	10]	d_loss: 1.4950	g_loss: 1.4111
Epoch [5/	10]	d_loss: 1.3474	g_loss: 1.3241
Epoch [5/	10]	d_loss: 1.3576	g_loss: 1.2033
Epoch [5/	10]	d_loss: 1.2718	g_loss: 1.3563
Epoch [5/	10]	d_loss: 1.2688	g_loss: 0.9416
Epoch [5/	10]	d_loss: 1.1041	g_loss: 1.4097
Epoch [5/	10]	d_loss: 1.1343	g_loss: 1.1417
Epoch [5/	10]	d_loss: 1.3555	g_loss: 1.3803
Epoch [5/	10]	d_loss: 1.1777	g_loss: 1.2319
Epoch [5/	10]	d_loss: 1.2997	g_loss: 1.6132
Epoch [5/	10]	d_loss: 1.1972	g_loss: 0.8212
Epoch [5/	10]	d_loss: 1.1172	g_loss: 1.0947
Epoch [5/	10]	d_loss: 1.1951	g_loss: 1.1931
Epoch [5/	10]	d_loss: 1.2055	g_loss: 1.2370
Epoch [5/	10]	d_loss: 1.3684	g_loss: 0.8865
Epoch [5/	10]	d_loss: 1.2492	g_loss: 1.2092
Epoch [5/	10]	d_loss: 0.9017	g_loss: 1.1830
Epoch [5/	10]	d_loss: 1.0169	g_loss: 1.1501
Epoch [5/	10]	d_loss: 1.3054	g_loss: 1.0301
Epoch [5/	10]	d_loss: 0.9068	g_loss: 1.1125
Epoch [5/	10]	d_loss: 1.2209	g_loss: 0.9824
Epoch [5/	10]	d_loss: 1.0398	g_loss: 1.1711
Epoch [5/	10]	d_loss: 1.5375	g_loss: 0.8667
Epoch [5/	10]	d_loss: 1.1747	g_loss: 1.0114
Epoch [5/	10]	d_loss: 1.0734	g_loss: 1.3111
Epoch [5/	10]	d_loss: 1.0630	g_loss: 1.2934
Epoch [5/	10]	d_loss: 1.3160	g_loss: 1.3786
Epoch [5/	10]	d_loss: 1.1447	g_loss: 1.1151
Epoch [5/	10]	d_loss: 1.1103	g_loss: 1.0679
Epoch [5/	10]	d_loss: 1.1587	g_loss: 1.0381
Epoch [5/	10]	d_loss: 1.1469	g_loss: 0.7251
Epoch [5/	10]	d_loss: 1.4251	g_loss: 1.0912
Epoch [5/	10]	d_loss: 1.0648	g_loss: 0.9549
Epoch [5/	10]	d_loss: 1.2340	g_loss: 0.9908
Epoch [5/	10]	d_loss: 1.0687	g_loss: 0.6913
Epoch [5/	10]	d_loss: 1.1242	g_loss: 1.4828
Epoch [5/	10]	d_loss: 1.1940	g_loss: 0.9102

Epoch [5/	10]	d_loss: 1.2044	g_loss: 1.7050
Epoch [5/	10]	d_loss: 1.0963	g_loss: 1.1869
Epoch [5/	10]	d_loss: 1.3886	g_loss: 0.8966
Epoch [5/	10]	d_loss: 1.1812	g_loss: 1.2467
Epoch [5/	10]	d_loss: 1.3734	g_loss: 0.7717
Epoch [5/	10]	d_loss: 1.1294	g_loss: 0.9604
Epoch [5/	10]	d_loss: 1.2745	g_loss: 1.6540
Epoch [5/	10]	d_loss: 1.1713	g_loss: 1.4014
Epoch [5/	10]	d_loss: 1.0579	g_loss: 1.1714
Epoch [5/	10]	d_loss: 1.0669	g_loss: 1.2437
Epoch [5/	10]	d_loss: 1.1266	g_loss: 0.9542
Epoch [5/	10]	d_loss: 1.1390	g_loss: 1.0824
Epoch [5/	10]	d_loss: 1.2144	g_loss: 0.7487
Epoch [5/	10]	d_loss: 1.0016	g_loss: 1.0708
Epoch [5/	10]	d_loss: 1.1342	g_loss: 1.4561
Epoch [5/	10]	d_loss: 1.0600	g_loss: 0.9074
Epoch [5/	10]	d_loss: 0.9524	g_loss: 1.3128
Epoch [5/	10]	d_loss: 1.1922	g_loss: 0.8743
Epoch [5/	10]	d_loss: 1.0718	g_loss: 1.0761
Epoch [5/	10]	d_loss: 1.1504	g_loss: 0.9016
Epoch [5/	10]	d_loss: 1.0552	g_loss: 1.4507
Epoch [5/	10]	d_loss: 1.0720	g_loss: 1.0660
Epoch [5/	10]	d_loss: 1.0583	g_loss: 0.8015
Epoch [5/	10]	d_loss: 1.0710	g_loss: 0.8109
Epoch [5/	10]	d_loss: 1.1911	g_loss: 0.9369
Epoch [5/	10]	d_loss: 1.2684	g_loss: 1.5220
Epoch [5/	10]	d_loss: 1.1584	g_loss: 0.9735
Epoch [5/	10]	d_loss: 1.2662	g_loss: 0.7804
Epoch [5/	10]	d_loss: 1.1530	g_loss: 0.9882
Epoch [5/	10]	d_loss: 1.0608	g_loss: 1.0826
Epoch [5/	10]	d_loss: 1.3986	g_loss: 1.3119
Epoch [5/	10]	d_loss: 1.0951	g_loss: 1.3141
Epoch [5/	10]	d_loss: 1.1527	g_loss: 0.8788
Epoch [5/	10]	d_loss: 1.1599	g_loss: 1.4348
Epoch [5/	10]	d_loss: 1.3204	g_loss: 0.8936
Epoch [5/	10]	d_loss: 1.1235	g_loss: 1.2673
Epoch [5/	10]	d_loss: 0.9697	g_loss: 0.8546
Epoch [5/	10]	d_loss: 1.2407	g_loss: 1.5102
Epoch [5/	10]	d_loss: 1.2299	g_loss: 0.7595
Epoch [5/	10]	d_loss: 1.1372	g_loss: 0.7321
Epoch [5/	10]	d_loss: 1.2809	g_loss: 1.0314
Epoch [5/	10]	d_loss: 1.2398	g_loss: 0.7542
Epoch [5/	10]	d_loss: 1.0853	g_loss: 1.4341
Epoch [5/	10]	d_loss: 1.3530	g_loss: 1.3174
Epoch [5/	10]	d_loss: 1.1615	g_loss: 1.1928
Epoch [5/	10]	d_loss: 1.1498	g_loss: 1.1443
Epoch [5/	10]	d_loss: 1.1605	g_loss: 0.6326
Epoch [6/	10]	d_loss: 1.0730	g_loss: 0.7920

Epoch [6/	10]	d_loss: 1.1257	g_loss: 0.9763
Epoch [6/	10]	d_loss: 1.2516	g_loss: 0.8755
Epoch [6/	10]	d_loss: 1.4945	g_loss: 0.7936
Epoch [6/	10]	d_loss: 1.0316	g_loss: 1.3888
Epoch [6/	10]	d_loss: 1.2350	g_loss: 0.8010
Epoch [6/	10]	d_loss: 1.1707	g_loss: 1.2022
Epoch [6/	10]	d_loss: 1.1160	g_loss: 1.0591
Epoch [6/	10]	d_loss: 1.3801	g_loss: 1.1074
Epoch [6/	10]	d_loss: 1.0718	g_loss: 1.5475
Epoch [6/	10]	d_loss: 1.3083	g_loss: 0.9275
Epoch [6/	10]	d_loss: 1.1705	g_loss: 0.7466
Epoch [6/	10]	d_loss: 1.2022	g_loss: 0.8474
Epoch [6/	10]	d_loss: 1.3600	g_loss: 1.2856
Epoch [6/	10]	d_loss: 1.2514	g_loss: 0.8830
Epoch [6/	10]	d_loss: 1.3162	g_loss: 1.3155
Epoch [6/	10]	d_loss: 1.1565	g_loss: 0.8522
Epoch [6/	10]	d_loss: 1.2938	g_loss: 1.2253
Epoch [6/	10]	d_loss: 1.3792	g_loss: 1.0761
Epoch [6/	10]	d_loss: 1.0206	g_loss: 0.8917
Epoch [6/	10]	d_loss: 1.2340	g_loss: 1.4505
Epoch [6/	10]	d_loss: 1.2090	g_loss: 1.0486
Epoch [6/	10]	d_loss: 1.0347	g_loss: 1.5621
Epoch [6/	10]	d_loss: 1.1174	g_loss: 1.0788
Epoch [6/	10]	d_loss: 1.4424	g_loss: 1.7971
Epoch [6/	10]	d_loss: 1.2050	g_loss: 0.8605
Epoch [6/	10]	d_loss: 1.2195	g_loss: 0.7046
Epoch [6/	10]	d_loss: 1.1627	g_loss: 0.9664
Epoch [6/	10]	d_loss: 1.4239	g_loss: 1.4702
Epoch [6/	10]	d_loss: 1.2098	g_loss: 1.0585
Epoch [6/	10]	d_loss: 1.4377	g_loss: 0.8329
Epoch [6/	10]	d_loss: 1.0519	g_loss: 0.8360
Epoch [6/	10]	d_loss: 1.4773	g_loss: 0.9169
Epoch [6/	10]	d_loss: 1.1291	g_loss: 1.3760
Epoch [6/	10]	d_loss: 1.0259	g_loss: 1.1081
Epoch [6/	10]	d_loss: 1.1477	g_loss: 1.1595
Epoch [6/	10]	d_loss: 1.0161	g_loss: 1.0142
Epoch [6/	10]	d_loss: 1.3734	g_loss: 0.9519
Epoch [6/	10]	d_loss: 1.1338	g_loss: 1.0909
Epoch [6/	10]	d_loss: 1.0651	g_loss: 1.3779
Epoch [6/	10]	d_loss: 1.1669	g_loss: 0.7601
Epoch [6/	10]	d_loss: 1.0271	g_loss: 1.4555
Epoch [6/	10]	d_loss: 1.1511	g_loss: 1.4673
Epoch [6/	10]	d_loss: 1.2126	g_loss: 0.7242
Epoch [6/	10]	d_loss: 1.0899	g_loss: 0.8547
Epoch [6/	10]	d_loss: 1.3143	g_loss: 1.7385
Epoch [6/	10]	d_loss: 1.2054	g_loss: 1.2151
Epoch [6/	10]	d_loss: 1.0734	g_loss: 1.2346
Epoch [6/	10]	d_loss: 1.0866	g_loss: 0.9258

Epoch [6/	10]	d_loss: 1.3402	g_loss: 0.6708
Epoch [6/	10]	d_loss: 0.8979	g_loss: 1.4917
Epoch [6/	10]	d_loss: 1.2270	g_loss: 1.2945
Epoch [6/	10]	d_loss: 1.0414	g_loss: 1.1855
Epoch [6/	10]	d_loss: 1.1789	g_loss: 1.4890
Epoch [6/	10]	d_loss: 1.3590	g_loss: 1.3618
Epoch [6/	10]	d_loss: 1.0249	g_loss: 1.2544
Epoch [6/	10]	d_loss: 1.1767	g_loss: 1.1517
Epoch [6/	10]	d_loss: 1.2099	g_loss: 1.3047
Epoch [6/	10]	d_loss: 1.1252	g_loss: 1.1571
Epoch [6/	10]	d_loss: 1.0195	g_loss: 1.0036
Epoch [6/	10]	d_loss: 1.0854	g_loss: 0.8077
Epoch [6/	10]	d_loss: 1.0348	g_loss: 1.2070
Epoch [6/	10]	d_loss: 1.1041	g_loss: 1.4380
Epoch [6/	10]	d_loss: 1.2261	g_loss: 0.8078
Epoch [6/	10]	d_loss: 1.0966	g_loss: 1.6665
Epoch [6/	10]	d_loss: 1.2512	g_loss: 1.0825
Epoch [6/	10]	d_loss: 1.3201	g_loss: 0.9760
Epoch [6/	10]	d_loss: 1.4058	g_loss: 1.0107
Epoch [6/	10]	d_loss: 1.1979	g_loss: 1.1943
Epoch [6/	10]	d_loss: 1.1593	g_loss: 0.9311
Epoch [6/	10]	d_loss: 1.0206	g_loss: 1.3763
Epoch [6/	10]	d_loss: 1.2415	g_loss: 1.6824
Epoch [6/	10]	d_loss: 1.2525	g_loss: 1.1130
Epoch [6/	10]	d_loss: 1.3053	g_loss: 1.1167
Epoch [6/	10]	d_loss: 0.9156	g_loss: 1.5156
Epoch [6/	10]	d_loss: 1.0841	g_loss: 1.2665
Epoch [6/	10]	d_loss: 1.1584	g_loss: 1.4293
Epoch [6/	10]	d_loss: 1.0209	g_loss: 0.7985
Epoch [6/	10]	d_loss: 1.1819	g_loss: 0.9709
Epoch [6/	10]	d_loss: 1.1650	g_loss: 0.7932
Epoch [6/	10]	d_loss: 1.2317	g_loss: 1.0626
Epoch [6/	10]	d_loss: 1.1244	g_loss: 1.0203
Epoch [6/	10]	d_loss: 0.8547	g_loss: 0.9335
Epoch [6/	10]	d_loss: 1.2060	g_loss: 0.9006
Epoch [6/	10]	d_loss: 1.1602	g_loss: 0.7894
Epoch [6/	10]	d_loss: 1.0187	g_loss: 1.2619
Epoch [6/	10]	d_loss: 1.1494	g_loss: 1.3033
Epoch [6/	10]	d_loss: 1.1901	g_loss: 0.8781
Epoch [6/	10]	d_loss: 1.2263	g_loss: 1.1578
Epoch [6/	10]	d_loss: 1.1821	g_loss: 0.9573
Epoch [7/	10]	d_loss: 1.1732	g_loss: 1.1476
Epoch [7/	10]	d_loss: 1.1467	g_loss: 1.3974
Epoch [7/	10]	d_loss: 1.2581	g_loss: 1.2248
Epoch [7/	10]	d_loss: 1.1671	g_loss: 1.0263
Epoch [7/	10]	d_loss: 1.1581	g_loss: 1.0820
Epoch [7/	10]	d_loss: 0.9903	g_loss: 1.1350
Epoch [7/	10]	d_loss: 1.1334	g_loss: 1.0559

Epoch [7/	10]	d_loss: 1.2480	g_loss: 0.8779
Epoch [7/	10]	d_loss: 1.0574	g_loss: 1.4727
Epoch [7/	10]	d_loss: 1.2593	g_loss: 0.4865
Epoch [7/	10]	d_loss: 1.0608	g_loss: 1.0052
Epoch [7/	10]	d_loss: 1.1402	g_loss: 0.8119
Epoch [7/	10]	d_loss: 1.1310	g_loss: 0.9325
Epoch [7/	10]	d_loss: 1.4502	g_loss: 1.0067
Epoch [7/	10]	d_loss: 1.2507	g_loss: 1.3967
Epoch [7/	10]	d_loss: 1.1069	g_loss: 1.0546
Epoch [7/	10]	d_loss: 1.0229	g_loss: 1.1447
Epoch [7/	10]	d_loss: 1.2403	g_loss: 0.9139
Epoch [7/	10]	d_loss: 1.1453	g_loss: 1.0105
Epoch [7/	10]	d_loss: 1.2846	g_loss: 1.1612
Epoch [7/	10]	d_loss: 1.4018	g_loss: 0.9182
Epoch [7/	10]	d_loss: 1.1194	g_loss: 1.0345
Epoch [7/	10]	d_loss: 0.9523	g_loss: 1.3987
Epoch [7/	10]	d_loss: 0.9792	g_loss: 1.0411
Epoch [7/	10]	d_loss: 1.2931	g_loss: 1.2631
Epoch [7/	10]	d_loss: 1.3245	g_loss: 1.6532
Epoch [7/	10]	d_loss: 1.1319	g_loss: 0.8025
Epoch [7/	10]	d_loss: 1.1348	g_loss: 0.7219
Epoch [7/	10]	d_loss: 1.1169	g_loss: 1.1769
Epoch [7/	10]	d_loss: 1.1521	g_loss: 0.9162
Epoch [7/	10]	d_loss: 1.0857	g_loss: 1.1034
Epoch [7/	10]	d_loss: 1.3879	g_loss: 1.6485
Epoch [7/	10]	d_loss: 1.1457	g_loss: 1.4805
Epoch [7/	10]	d_loss: 1.1870	g_loss: 1.0695
Epoch [7/	10]	d_loss: 1.1226	g_loss: 0.9823
Epoch [7/	10]	d_loss: 1.0719	g_loss: 0.6887
Epoch [7/	10]	d_loss: 1.0205	g_loss: 1.1127
Epoch [7/	10]	d_loss: 1.0800	g_loss: 0.8851
Epoch [7/	10]	d_loss: 1.3375	g_loss: 0.9541
Epoch [7/	10]	d_loss: 1.1829	g_loss: 0.8048
Epoch [7/	10]	d_loss: 1.1203	g_loss: 1.0170
Epoch [7/	10]	d_loss: 1.0425	g_loss: 0.9496
Epoch [7/	10]	d_loss: 1.1684	g_loss: 0.9316
Epoch [7/	10]	d_loss: 1.2154	g_loss: 0.6846
Epoch [7/	10]	d_loss: 1.0681	g_loss: 0.7338
Epoch [7/	10]	d_loss: 1.1037	g_loss: 0.8119
Epoch [7/	10]	d_loss: 1.1239	g_loss: 0.8091
Epoch [7/	10]	d_loss: 1.0324	g_loss: 0.7603
Epoch [7/	10]	d_loss: 1.1805	g_loss: 1.0989
Epoch [7/	10]	d_loss: 1.2365	g_loss: 1.8341
Epoch [7/	10]	d_loss: 1.2167	g_loss: 1.0004
Epoch [7/	10]	d_loss: 1.1257	g_loss: 1.0913
Epoch [7/	10]	d_loss: 1.2365	g_loss: 0.7964
Epoch [7/	10]	d_loss: 0.8825	g_loss: 1.3034
Epoch [7/	10]	d_loss: 1.1268	g_loss: 1.3533

Epoch [7/	10]	d_loss: 1.0871	g_loss: 1.7459
Epoch [7/	10]	d_loss: 1.2827	g_loss: 0.6072
Epoch [7/	10]	d_loss: 1.0606	g_loss: 1.4661
Epoch [7/	10]	d_loss: 0.9369	g_loss: 1.1055
Epoch [7/	10]	d_loss: 1.1915	g_loss: 1.3185
Epoch [7/	10]	d_loss: 1.0632	g_loss: 1.1974
Epoch [7/	10]	d_loss: 0.9697	g_loss: 1.2723
Epoch [7/	10]	d_loss: 1.2271	g_loss: 0.9845
Epoch [7/	10]	d_loss: 1.1516	g_loss: 1.2700
Epoch [7/	10]	d_loss: 1.0057	g_loss: 0.8996
Epoch [7/	10]	d_loss: 1.2225	g_loss: 0.9985
Epoch [7/	10]	d_loss: 1.2667	g_loss: 0.8148
Epoch [7/	10]	d_loss: 1.2963	g_loss: 0.7734
Epoch [7/	10]	d_loss: 1.3521	g_loss: 1.0544
Epoch [7/	10]	d_loss: 1.0204	g_loss: 0.7964
Epoch [7/	10]	d_loss: 1.1991	g_loss: 0.8047
Epoch [7/	10]	d_loss: 1.0735	g_loss: 1.3074
Epoch [7/	10]	d_loss: 1.3092	g_loss: 0.8839
Epoch [7/	10]	d_loss: 0.9760	g_loss: 1.2557
Epoch [7/	10]	d_loss: 1.3489	g_loss: 0.7484
Epoch [7/	10]	d_loss: 1.4011	g_loss: 0.7732
Epoch [7/	10]	d_loss: 1.2028	g_loss: 0.7975
Epoch [7/	10]	d_loss: 1.3241	g_loss: 1.2836
Epoch [7/	10]	d_loss: 1.4079	g_loss: 1.1309
Epoch [7/	10]	d_loss: 1.2155	g_loss: 0.6150
Epoch [7/	10]	d_loss: 1.2187	g_loss: 1.0792
Epoch [7/	10]	d_loss: 1.1328	g_loss: 0.9109
Epoch [7/	10]	d_loss: 1.0502	g_loss: 0.9357
Epoch [7/	10]	d_loss: 1.4128	g_loss: 0.9568
Epoch [7/	10]	d_loss: 1.3917	g_loss: 1.1317
Epoch [7/	10]	d_loss: 1.2977	g_loss: 0.8917
Epoch [7/	10]	d_loss: 1.1795	g_loss: 1.0096
Epoch [7/	10]	d_loss: 0.7407	g_loss: 1.5613
Epoch [7/	10]	d_loss: 0.8930	g_loss: 1.2415
Epoch [7/	10]	d_loss: 1.2775	g_loss: 1.0762
Epoch [8/	10]	d_loss: 1.0588	g_loss: 1.1374
Epoch [8/	10]	d_loss: 1.1387	g_loss: 0.9783
Epoch [8/	10]	d_loss: 1.2720	g_loss: 0.7530
Epoch [8/	10]	d_loss: 1.1936	g_loss: 0.8412
Epoch [8/	10]	d_loss: 1.2290	g_loss: 1.0897
Epoch [8/	10]	d_loss: 0.9382	g_loss: 1.0752
Epoch [8/	10]	d_loss: 1.0202	g_loss: 1.2163
Epoch [8/	10]	d_loss: 1.0541	g_loss: 1.4688
Epoch [8/	10]	d_loss: 1.2581	g_loss: 0.9305
Epoch [8/	10]	d_loss: 1.1674	g_loss: 1.0575
Epoch [8/	10]	d_loss: 1.0671	g_loss: 0.8297
Epoch [8/	10]	d_loss: 1.2365	g_loss: 1.1652
Epoch [8/	10]	d_loss: 1.2273	g_loss: 1.2269

Epoch [8/	10]	d_loss: 1.2088	g_loss: 1.0195
Epoch [8/	10]	d_loss: 1.1266	g_loss: 1.0847
Epoch [8/	10]	d_loss: 1.0584	g_loss: 0.8271
Epoch [8/	10]	d_loss: 1.1116	g_loss: 0.8198
Epoch [8/	10]	d_loss: 1.1296	g_loss: 1.0380
Epoch [8/	10]	d_loss: 0.9076	g_loss: 1.4494
Epoch [8/	10]	d_loss: 1.0829	g_loss: 1.1713
Epoch [8/	10]	d_loss: 1.2202	g_loss: 0.5480
Epoch [8/	10]	d_loss: 1.1540	g_loss: 0.9943
Epoch [8/	10]	d_loss: 1.1924	g_loss: 0.9677
Epoch [8/	10]	d_loss: 1.3760	g_loss: 1.1132
Epoch [8/	10]	d_loss: 1.1201	g_loss: 0.6309
Epoch [8/	10]	d_loss: 1.2072	g_loss: 1.4210
Epoch [8/	10]	d_loss: 1.2332	g_loss: 0.9034
Epoch [8/	10]	d_loss: 1.1732	g_loss: 0.8845
Epoch [8/	10]	d_loss: 1.1728	g_loss: 0.9919
Epoch [8/	10]	d_loss: 1.2468	g_loss: 1.6959
Epoch [8/	10]	d_loss: 1.0915	g_loss: 1.1403
Epoch [8/	10]	d_loss: 1.2112	g_loss: 0.9397
Epoch [8/	10]	d_loss: 1.0963	g_loss: 1.3658
Epoch [8/	10]	d_loss: 1.3036	g_loss: 0.9281
Epoch [8/	10]	d_loss: 1.6919	g_loss: 1.3913
Epoch [8/	10]	d_loss: 1.1412	g_loss: 1.1280
Epoch [8/	10]	d_loss: 1.2199	g_loss: 1.4446
Epoch [8/	10]	d_loss: 1.1433	g_loss: 0.8593
Epoch [8/	10]	d_loss: 1.2468	g_loss: 1.2717
Epoch [8/	10]	d_loss: 0.9914	g_loss: 1.2214
Epoch [8/	10]	d_loss: 1.0930	g_loss: 1.1249
Epoch [8/	10]	d_loss: 1.0204	g_loss: 1.2932
Epoch [8/	10]	d_loss: 1.1369	g_loss: 0.9349
Epoch [8/	10]	d_loss: 1.2581	g_loss: 1.5584
Epoch [8/	10]	d_loss: 1.0565	g_loss: 0.8908
Epoch [8/	10]	d_loss: 1.1945	g_loss: 1.5893
Epoch [8/	10]	d_loss: 1.1119	g_loss: 0.9716
Epoch [8/	10]	d_loss: 1.2107	g_loss: 0.7331
Epoch [8/	10]	d_loss: 1.2976	g_loss: 1.1239
Epoch [8/	10]	d_loss: 1.3422	g_loss: 0.8898
Epoch [8/	10]	d_loss: 1.1628	g_loss: 1.2847
Epoch [8/	10]	d_loss: 1.1218	g_loss: 0.8711
Epoch [8/	10]	d_loss: 1.2757	g_loss: 1.0264
Epoch [8/	10]	d_loss: 0.9958	g_loss: 0.9850
Epoch [8/	10]	d_loss: 1.6715	g_loss: 1.1247
Epoch [8/	10]	d_loss: 1.1611	g_loss: 1.3082
Epoch [8/	10]	d_loss: 1.0148	g_loss: 1.0907
Epoch [8/	10]	d_loss: 1.1705	g_loss: 1.1221
Epoch [8/	10]	d_loss: 1.4214	g_loss: 1.0586
Epoch [8/	10]	d_loss: 1.1075	g_loss: 1.0872
Epoch [8/	10]	d_loss: 1.1222	g_loss: 1.2672

Epoch [8/	10]	d_loss: 1.3660	g_loss: 1.3399
Epoch [8/	10]	d_loss: 1.3433	g_loss: 0.7457
Epoch [8/	10]	d_loss: 1.3367	g_loss: 0.8157
Epoch [8/	10]	d_loss: 1.0755	g_loss: 1.3992
Epoch [8/	10]	d_loss: 1.1997	g_loss: 1.1722
Epoch [8/	10]	d_loss: 1.0807	g_loss: 1.2354
Epoch [8/	10]	d_loss: 1.1888	g_loss: 1.1411
Epoch [8/	10]	d_loss: 1.2648	g_loss: 0.8938
Epoch [8/	10]	d_loss: 1.1414	g_loss: 1.2299
Epoch [8/	10]	d_loss: 1.2130	g_loss: 1.3529
Epoch [8/	10]	d_loss: 1.0020	g_loss: 0.9441
Epoch [8/	10]	d_loss: 1.1434	g_loss: 1.3303
Epoch [8/	10]	d_loss: 0.8761	g_loss: 1.6646
Epoch [8/	10]	d_loss: 1.1792	g_loss: 1.2781
Epoch [8/	10]	d_loss: 1.0083	g_loss: 1.0547
Epoch [8/	10]	d_loss: 0.9787	g_loss: 1.0068
Epoch [8/	10]	d_loss: 1.1732	g_loss: 1.1954
Epoch [8/	10]	d_loss: 1.0308	g_loss: 0.9213
Epoch [8/	10]	d_loss: 1.0904	g_loss: 1.6630
Epoch [8/	10]	d_loss: 1.3179	g_loss: 1.1435
Epoch [8/	10]	d_loss: 1.3343	g_loss: 1.2108
Epoch [8/	10]	d_loss: 1.4113	g_loss: 1.4578
Epoch [8/	10]	d_loss: 1.2262	g_loss: 1.6021
Epoch [8/	10]	d_loss: 1.2276	g_loss: 1.0768
Epoch [8/	10]	d_loss: 1.0666	g_loss: 0.8270
Epoch [8/	10]	d_loss: 1.2629	g_loss: 0.5701
Epoch [8/	10]	d_loss: 1.1216	g_loss: 1.1852
Epoch [8/	10]	d_loss: 0.9711	g_loss: 1.1063
Epoch [8/	10]	d_loss: 1.1267	g_loss: 1.3758
Epoch [9/	10]	d_loss: 1.1260	g_loss: 0.9212
Epoch [9/	10]	d_loss: 1.1196	g_loss: 0.7461
Epoch [9/	10]	d_loss: 1.1379	g_loss: 1.6393
Epoch [9/	10]	d_loss: 1.2989	g_loss: 0.9597
Epoch [9/	10]	d_loss: 1.2436	g_loss: 0.7449
Epoch [9/	10]	d_loss: 1.0316	g_loss: 0.7954
Epoch [9/	10]	d_loss: 1.2471	g_loss: 1.6627
Epoch [9/	10]	d_loss: 1.4474	g_loss: 1.0979
Epoch [9/	10]	d_loss: 1.2534	g_loss: 1.3412
Epoch [9/	10]	d_loss: 0.9886	g_loss: 1.0867
Epoch [9/	10]	d_loss: 0.9754	g_loss: 0.9422
Epoch [9/	10]	d_loss: 1.0368	g_loss: 1.1628
Epoch [9/	10]	d_loss: 0.9254	g_loss: 0.9083
Epoch [9/	10]	d_loss: 1.3285	g_loss: 1.1071
Epoch [9/	10]	d_loss: 1.3965	g_loss: 1.3819
Epoch [9/	10]	d_loss: 1.6170	g_loss: 1.8180
Epoch [9/	10]	d_loss: 1.1543	g_loss: 1.0563
Epoch [9/	10]	d_loss: 1.1336	g_loss: 1.0205
Epoch [9/	10]	d_loss: 0.8851	g_loss: 1.2574

Epoch [9/	10]	d_loss: 1.1946	g_loss: 1.1199
Epoch [9/	10]	d_loss: 1.0916	g_loss: 1.1318
Epoch [9/	10]	d_loss: 0.9087	g_loss: 1.2817
Epoch [9/	10]	d_loss: 1.3317	g_loss: 1.6255
Epoch [9/	10]	d_loss: 0.9462	g_loss: 0.9335
Epoch [9/	10]	d_loss: 0.9327	g_loss: 1.1728
Epoch [9/	10]	d_loss: 1.1094	g_loss: 1.2416
Epoch [9/	10]	d_loss: 1.1224	g_loss: 0.7621
Epoch [9/	10]	d_loss: 1.1844	g_loss: 1.2253
Epoch [9/	10]	d_loss: 1.2070	g_loss: 0.8545
Epoch [9/	10]	d_loss: 1.2108	g_loss: 0.8870
Epoch [9/	10]	d_loss: 0.9820	g_loss: 1.3911
Epoch [9/	10]	d_loss: 1.1338	g_loss: 1.0617
Epoch [9/	10]	d_loss: 1.4392	g_loss: 1.3271
Epoch [9/	10]	d_loss: 1.1557	g_loss: 1.2597
Epoch [9/	10]	d_loss: 1.3667	g_loss: 1.0845
Epoch [9/	10]	d_loss: 1.4531	g_loss: 1.1860
Epoch [9/	10]	d_loss: 0.9619	g_loss: 1.5559
Epoch [9/	10]	d_loss: 1.5667	g_loss: 0.7787
Epoch [9/	10]	d_loss: 1.3582	g_loss: 1.1465
Epoch [9/	10]	d_loss: 1.0388	g_loss: 0.7552
Epoch [9/	10]	d_loss: 0.9390	g_loss: 1.2183
Epoch [9/	10]	d_loss: 1.1430	g_loss: 0.8835
Epoch [9/	10]	d_loss: 1.2905	g_loss: 0.9816
Epoch [9/	10]	d_loss: 1.3038	g_loss: 1.1086
Epoch [9/	10]	d_loss: 1.0786	g_loss: 1.1820
Epoch [9/	10]	d_loss: 1.2424	g_loss: 1.5541
Epoch [9/	10]	d_loss: 1.1242	g_loss: 0.9068
Epoch [9/	10]	d_loss: 1.1815	g_loss: 1.1209
Epoch [9/	10]	d_loss: 1.2463	g_loss: 0.9412
Epoch [9/	10]	d_loss: 1.1966	g_loss: 0.7705
Epoch [9/	10]	d_loss: 1.3233	g_loss: 0.5576
Epoch [9/	10]	d_loss: 1.0374	g_loss: 1.4858
Epoch [9/	10]	d_loss: 1.0187	g_loss: 0.8979
Epoch [9/	10]	d_loss: 1.2809	g_loss: 0.9093
Epoch [9/	10]	d_loss: 1.2153	g_loss: 1.2703
Epoch [9/	10]	d_loss: 1.2060	g_loss: 0.9411
Epoch [9/	10]	d_loss: 1.0072	g_loss: 1.1701
Epoch [9/	10]	d_loss: 1.2657	g_loss: 0.8600
Epoch [9/	10]	d_loss: 1.2149	g_loss: 1.1889
Epoch [9/	10]	d_loss: 1.0752	g_loss: 1.3029
Epoch [9/	10]	d_loss: 1.0969	g_loss: 1.2048
Epoch [9/	10]	d_loss: 0.9761	g_loss: 0.9362
Epoch [9/	10]	d_loss: 1.0159	g_loss: 0.9008
Epoch [9/	10]	d_loss: 1.3988	g_loss: 1.4159
Epoch [9/	10]	d_loss: 1.3332	g_loss: 1.3442
Epoch [9/	10]	d_loss: 1.1792	g_loss: 1.0996
Epoch [9/	10]	d_loss: 1.2178	g_loss: 0.6899

Epoch [9/	10]	d_loss: 1.1697	g_loss: 0.8757
Epoch [9/	10]	d_loss: 1.1003	g_loss: 1.4365
Epoch [9/	10]	d_loss: 1.0625	g_loss: 1.1884
Epoch [9/	10]	d_loss: 1.2848	g_loss: 1.1496
Epoch [9/	10]	d_loss: 1.4706	g_loss: 0.9106
Epoch [9/	10]	d_loss: 1.1511	g_loss: 1.2746
Epoch [9/	10]	d_loss: 1.0118	g_loss: 0.8611
Epoch [9/	10]	d_loss: 1.0485	g_loss: 0.8746
Epoch [9/	10]	d_loss: 1.0517	g_loss: 1.6487
Epoch [9/	10]	d_loss: 1.1845	g_loss: 0.7870
Epoch [9/	10]	d_loss: 1.4051	g_loss: 0.8852
Epoch [9/	10]	d_loss: 1.1488	g_loss: 1.0786
Epoch [9/	10]	d_loss: 1.0463	g_loss: 1.5692
Epoch [9/	10]	d_loss: 0.7221	g_loss: 1.7836
Epoch [9/	10]	d_loss: 1.1432	g_loss: 1.2960
Epoch [9/	10]	d_loss: 0.9031	g_loss: 1.4865
Epoch [9/	10]	d_loss: 0.9979	g_loss: 0.9289
Epoch [9/	10]	d_loss: 0.9905	g_loss: 0.8457
Epoch [9/	10]	d_loss: 0.9161	g_loss: 0.7628
Epoch [9/	10]	d_loss: 1.1245	g_loss: 0.8388
Epoch [9/	10]	d_loss: 1.5137	g_loss: 0.8044
Epoch [9/	10]	d_loss: 1.1483	g_loss: 1.3070
Epoch [9/	10]	d_loss: 1.0240	g_loss: 0.9581
Epoch [10/	10]	d_loss: 1.1211	g_loss: 0.7031
Epoch [10/	10]	d_loss: 1.0918	g_loss: 1.0118
Epoch [10/	10]	d_loss: 1.1350	g_loss: 1.0654
Epoch [10/	10]	d_loss: 1.0748	g_loss: 1.0344
Epoch [10/	10]	d_loss: 1.1558	g_loss: 0.9447
Epoch [10/	10]	d_loss: 1.0147	g_loss: 1.4055
Epoch [10/	10]	d_loss: 1.0622	g_loss: 1.5723
Epoch [10/	10]	d_loss: 1.1207	g_loss: 0.7668
Epoch [10/	10]	d_loss: 1.1877	g_loss: 1.4527
Epoch [10/	10]	d_loss: 1.2907	g_loss: 0.6938
Epoch [10/	10]	d_loss: 1.1124	g_loss: 1.4505
Epoch [10/	10]	d_loss: 1.2964	g_loss: 1.1539
Epoch [10/	10]	d_loss: 1.2800	g_loss: 2.3967
Epoch [10/	10]	d_loss: 1.2107	g_loss: 1.6137
Epoch [10/	10]	d_loss: 1.3732	g_loss: 1.0494
Epoch [10/	10]	d_loss: 0.9777	g_loss: 1.2503
Epoch [10/	10]	d_loss: 1.2230	g_loss: 1.0973
Epoch [10/	10]	d_loss: 1.1681	g_loss: 1.3028
Epoch [10/	10]	d_loss: 1.0728	g_loss: 1.0746
Epoch [10/	10]	d_loss: 1.0250	g_loss: 1.0968
Epoch [10/	10]	d_loss: 0.9255	g_loss: 1.1128
Epoch [10/	10]	d_loss: 1.1929	g_loss: 1.2623
Epoch [10/	10]	d_loss: 1.0738	g_loss: 1.4790
Epoch [10/	10]	d_loss: 0.8729	g_loss: 1.4110
Epoch [10/	10]	d_loss: 1.0142	g_loss: 1.0078

Epoch [10/	10]	d_loss: 1.4523	g_loss: 1.0255
Epoch [10/	10]	d_loss: 0.9146	g_loss: 1.4146
Epoch [10/	10]	d_loss: 0.9928	g_loss: 1.6429
Epoch [10/	10]	d_loss: 1.1591	g_loss: 1.2095
Epoch [10/	10]	d_loss: 1.0504	g_loss: 1.1178
Epoch [10/	10]	d_loss: 1.0218	g_loss: 0.8491
Epoch [10/	10]	d_loss: 1.3361	g_loss: 1.5222
Epoch [10/	10]	d_loss: 1.0141	g_loss: 1.0109
Epoch [10/	10]	d_loss: 1.1789	g_loss: 1.0317
Epoch [10/	10]	d_loss: 1.0315	g_loss: 1.2423
Epoch [10/	10]	d_loss: 1.2235	g_loss: 1.0753
Epoch [10/	10]	d_loss: 1.2236	g_loss: 0.6334
Epoch [10/	10]	d_loss: 1.0833	g_loss: 0.9629
Epoch [10/	10]	d_loss: 1.4009	g_loss: 0.6974
Epoch [10/	10]	d_loss: 1.0887	g_loss: 0.8523
Epoch [10/	10]	d_loss: 1.1777	g_loss: 0.8788
Epoch [10/	10]	d_loss: 1.2534	g_loss: 1.5423
Epoch [10/	10]	d_loss: 1.2129	g_loss: 0.9189
Epoch [10/	10]	d_loss: 1.0320	g_loss: 1.6259
Epoch [10/	10]	d_loss: 0.9720	g_loss: 1.5389
Epoch [10/	10]	d_loss: 1.1698	g_loss: 0.9646
Epoch [10/	10]	d_loss: 0.9819	g_loss: 0.8668
Epoch [10/	10]	d_loss: 1.1160	g_loss: 1.1563
Epoch [10/	10]	d_loss: 1.2639	g_loss: 1.1172
Epoch [10/	10]	d_loss: 1.0162	g_loss: 1.4084
Epoch [10/	10]	d_loss: 0.9475	g_loss: 0.7042
Epoch [10/	10]	d_loss: 0.9999	g_loss: 1.2390
Epoch [10/	10]	d_loss: 1.4821	g_loss: 0.6858
Epoch [10/	10]	d_loss: 1.4446	g_loss: 1.2766
Epoch [10/	10]	d_loss: 1.0114	g_loss: 1.1300
Epoch [10/	10]	d_loss: 1.0708	g_loss: 1.2104
Epoch [10/	10]	d_loss: 1.1379	g_loss: 1.3402
Epoch [10/	10]	d_loss: 1.1215	g_loss: 1.1378
Epoch [10/	10]	d_loss: 1.1427	g_loss: 0.7702
Epoch [10/	10]	d_loss: 1.0224	g_loss: 0.8705
Epoch [10/	10]	d_loss: 1.1849	g_loss: 1.0534
Epoch [10/	10]	d_loss: 1.0431	g_loss: 1.1079
Epoch [10/	10]	d_loss: 1.0241	g_loss: 0.7960
Epoch [10/	10]	d_loss: 1.0076	g_loss: 1.3401
Epoch [10/	10]	d_loss: 1.1447	g_loss: 0.9860
Epoch [10/	10]	d_loss: 1.3642	g_loss: 1.3758
Epoch [10/	10]	d_loss: 1.1873	g_loss: 0.9691
Epoch [10/	10]	d_loss: 0.9878	g_loss: 0.8479
Epoch [10/	10]	d_loss: 1.0264	g_loss: 1.0763
Epoch [10/	10]	d_loss: 1.1675	g_loss: 1.2406
Epoch [10/	10]	d_loss: 1.2042	g_loss: 1.2399
Epoch [10/	10]	d_loss: 1.0871	g_loss: 0.8837
Epoch [10/	10]	d_loss: 1.0756	g_loss: 1.1792

```
Epoch [ 10/ 10] | d_loss: 1.0222 | g_loss: 1.1058
Epoch [ 10/ 10] | d_loss: 1.2228 | g_loss: 0.8177
Epoch [ 10/ 10] | d_loss: 1.0770 | g_loss: 1.6376
Epoch [ 10/ 10] | d_loss: 1.2654 | g_loss: 0.9747
Epoch [ 10/ 10] | d_loss: 1.3402 | g_loss: 0.9576
Epoch [ 10/ 10] | d_loss: 1.1150 | g_loss: 1.1297
Epoch [ 10/ 10] | d_loss: 1.2699 | g_loss: 0.7373
Epoch [ 10/ 10] | d_loss: 0.9681 | g_loss: 0.8001
Epoch [ 10/ 10] | d_loss: 0.8841 | g_loss: 0.8127
Epoch [ 10/ 10] | d_loss: 0.9330 | g_loss: 1.5926
Epoch [ 10/ 10] | d_loss: 0.9612 | g_loss: 1.8434
Epoch [ 10/ 10] | d_loss: 0.9433 | g_loss: 0.9956
Epoch [ 10/ 10] | d_loss: 1.1682 | g_loss: 1.1468
Epoch [ 10/ 10] | d_loss: 1.2314 | g_loss: 1.2911
Epoch [ 10/ 10] | d_loss: 0.9458 | g_loss: 2.3047
Epoch [ 10/ 10] | d_loss: 1.0663 | g_loss: 1.1775
Epoch [ 10/ 10] | d_loss: 1.2016 | g_loss: 0.8836
```

2.8 Training loss

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

```
In [25]: 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[25]: <matplotlib.legend.Legend at 0x7f424ae91f60>
```



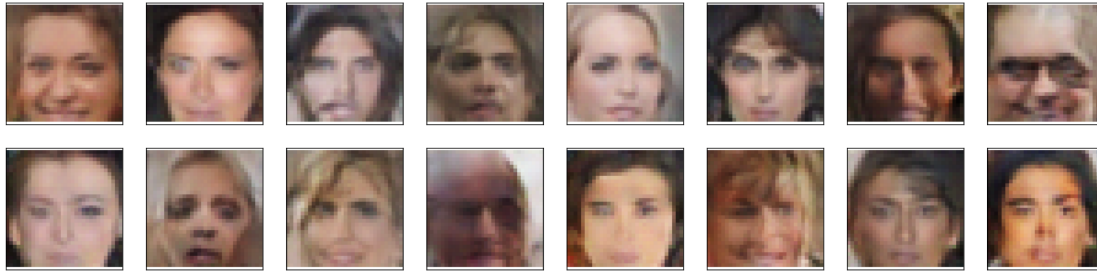
2.9 Generator samples from training

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

```
In [26]: # 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 [27]: # Load samples from generator, taken while training
with open('train_samples.pkl', 'rb') as f:
    samples = pkl.load(f)

In [28]: _ = 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: As noted above, the biased dataset makes it harder for the network to innovate. The faces are mostly white, and most of them follow Hollywood's standards of beauty that often don't reflect what the real people look like; 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 I think increasing the Conv dimension and reducing the epochs can produce the best result. even increasing the image size to 64 can produce more accurate image with little clarity

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.