

dlnd_face_generation

May 3, 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]

```
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 [1]: data_dir = 'processed_celeba_small/'
```

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

import pickle as pkl
import matplotlib.pyplot as plt
import numpy as np
import problem_unittests as tests
#import helper

%matplotlib inline
```

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 [2]: # necessary imports
import os
import numpy as np
import torch
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision import transforms

In [3]: def get_dataloader(batch_size, image_size, data_dir='processed_celeba_small/'):
    """
    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), # resize to 128x128
                                    transforms.ToTensor()])

    img_path = os.path.join(".", data_dir)
    image_dataset = datasets.ImageFolder(img_path, transform=transform)
```

```

image_loader = DataLoader(dataset=image_dataset, batch_size=batch_size,
                          shuffle=True, num_workers=0)

return image_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 [23]: # 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 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.

```

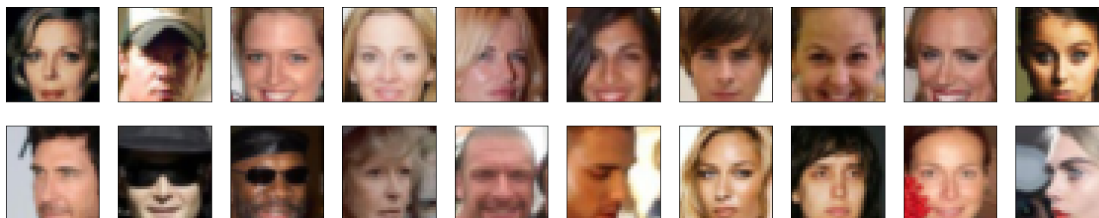
In [24]: # helper display function
        def imshow(img):
            npimg = img.numpy()
            plt.imshow(np.transpose(npimg, (1, 2, 0)))

            """
            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])

```



```
In [25]: print(f"one image shape: {images[0].numpy().shape}")
         print(f"batch image tensor shape: {images.numpy().shape}")
```

one image shape: (3, 32, 32)

batch image tensor shape: (128, 3, 32, 32)

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 [26]: # 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 [27]: """
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""

# check scaled range
# should be close to -1 to 1
img = images[0]
scaled_img = scale(img)

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

Min: tensor(-1.)

Max: tensor(0.7882)

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

In [29]: # 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)

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

In [30]: 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__()
self.conv_dim = conv_dim

# complete init function
# shape format in comments: (channels_or_filters x width x height)
# shape: 3x32x32 --> 64x16x16
self.conv1 = conv(in_channels=3, out_channels=conv_dim,
                  kernel_size=4, batch_norm=False)
# shape: 64x16x16 --> 128x8x8
self.conv2 = conv(in_channels=conv_dim, out_channels=conv_dim*2,
                  kernel_size=4)
# shape: 128x8x8 --> 256x4x4
self.conv3 = conv(in_channels=conv_dim*2, out_channels=conv_dim*4,
                  kernel_size=4)
# shape: 256x4x4 --> fully connected layer (flattenning)
self.fc = nn.Linear(in_features=conv_dim*4*4*4, out_features=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)

    return self.fc(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 [31]: 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__()
        self.z_size = z_size
        self.conv_dim = conv_dim

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

        # transpose conv layers
        # NOTE: nn.ConvTranspose2d() expects # of channels, the image width and height
        # shape: conv_dim*4 --> conv_dim*2
        self.t_conv1 = deconv(in_channels=conv_dim*4, out_channels=conv_dim*2,
                              kernel_size=4)
        # shape: conv_dim*2 --> conv_dim
        self.t_conv2 = deconv(in_channels=conv_dim*2, out_channels=conv_dim,
                              kernel_size=4)
        # shape: conv_dim --> 3
        # DON'T apply nn.BatchNorm2d() at the end, we'll apply F.tanh() later on in the
        self.t_conv3 = deconv(in_channels=conv_dim, out_channels=3,
                              kernel_size=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
        """
        # fully-connected + reshape
        out = self.fc(x)
```

```

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

        # transpose convolution / deconvolution layers + ReLU
        out = F.relu(self.t_conv1(out))
        out = F.relu(self.t_conv2(out))
        # NOTE: apply "tanh" instead of "ReLU" activation function to the output layer
        out = F.tanh(self.t_conv3(out))

    return out

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

```

Tests Passed

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 [32]: 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
        # https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/blob/master/models/network

```

```

init_gain = 0.02
if hasattr(m, 'weight') and (classname.find('Conv') != -1 or classname.find('Linear') != -1):
    nn.init.normal_(m.weight.data, 0.0, init_gain)
if hasattr(m, 'bias') and m.bias is not None:
    nn.init.constant_(m.bias.data, 0.0)

```

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 [33]: """
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 [47]: # 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 [48]: """
        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 [50]: 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)

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

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

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

    return loss

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

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

```

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

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

return loss

```

2.6 Optimizers

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

```

In [51]: 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 [66]: def save_model(D, G, d_optimizer, g_optimizer, filename="model.pt"):
         data = dict(
             D=D,
             G=G,
             d_optimizer=d_optimizer,
             g_optimizer=g_optimizer
         )
         torch.save(data, filename)

```

```

def load_model(filename="model.pt"):
    data = torch.load(filename)
    return data["D"], data["G"], data["d_optimizer"], data["g_optimizer"]

model_fn = "model.pt"

In [67]: 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
            # =====

            # =====
            #             TRAIN THE DISCRIMINATOR

```

```

# =====
d_optimizer.zero_grad()

# 1. Train with real images

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

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

# 2. Train with fake images

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

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

fake_images = G(z)

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

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

# 2. Train the generator with an adversarial loss
# =====
#             TRAIN THE GENERATOR (with an adversarial loss)
# =====
g_optimizer.zero_grad()

# 1. Train with fake images and flipped labels

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

```



```

# Compute the discriminator losses on fake images
# using flipped labels!
D_fake = D(fake_images)
g_loss = real_loss(D_fake) # use real loss to flip labels

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

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

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

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

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

# finally return losses
return losses

```

Set your number of training epochs and train your GAN!

```

In [69]: # set number of epochs
         n_epochs = 50
         print_every = 100
         train_from_scratch = False

         if train_from_scratch or not os.path.exists(model_fn):

```

```

D, G = build_network(d_conv_dim, g_conv_dim, z_size)
d_optimizer = optim.Adam(D.parameters(), lr, [beta1, beta2])
g_optimizer = optim.Adam(G.parameters(), lr, [beta1, beta2])
elif os.path.exists(model_fn):
    print(f"loading Discsrminator and Generator models and continuing training...")
    D, G, d_optimizer, g_optimizer = load_model(model_fn)

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

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

```

loading Discsrminator and Generator models and continuing training...

Epoch [1/ 50] | d_loss: 1.2324 | g_loss: 1.0669

/opt/conda/lib/python3.6/site-packages/torch/serialization.py:193: UserWarning: Couldn't retrieve "type " + obj.__name__ + ". It won't be checked "

/opt/conda/lib/python3.6/site-packages/torch/serialization.py:193: UserWarning: Couldn't retrieve "type " + obj.__name__ + ". It won't be checked "

Epoch [1/ 50] | d_loss: 1.1808 | g_loss: 1.0775

Epoch [1/ 50] | d_loss: 1.3159 | g_loss: 1.2511

Epoch [1/ 50] | d_loss: 1.1442 | g_loss: 0.8158

Epoch [1/ 50] | d_loss: 1.2618 | g_loss: 1.3176

Epoch [1/ 50] | d_loss: 1.2176 | g_loss: 1.0067

Epoch [1/ 50] | d_loss: 1.2159 | g_loss: 1.0276

Epoch [1/ 50] | d_loss: 1.2738 | g_loss: 0.9212

Epoch [2/ 50] | d_loss: 1.2193 | g_loss: 1.1882

Epoch [2/ 50] | d_loss: 1.1326 | g_loss: 1.8027

Epoch [2/ 50] | d_loss: 1.1351 | g_loss: 1.2056

Epoch [2/ 50] | d_loss: 1.1304 | g_loss: 0.9955

Epoch [2/ 50] | d_loss: 1.0791 | g_loss: 1.0959

Epoch [2/ 50] | d_loss: 1.1089 | g_loss: 0.9240

Epoch [2/ 50] | d_loss: 1.1682 | g_loss: 1.1076

Epoch [2/ 50] | d_loss: 1.0996 | g_loss: 1.1801

Epoch [3/ 50] | d_loss: 1.0843 | g_loss: 0.9002

Epoch [3/ 50] | d_loss: 1.1410 | g_loss: 1.2637

Epoch [3/ 50] | d_loss: 1.0210 | g_loss: 1.1671

Epoch [3/ 50] | d_loss: 0.9106 | g_loss: 0.9820

Epoch [3/ 50] | d_loss: 1.3612 | g_loss: 0.4516

Epoch [3/ 50] | d_loss: 1.3556 | g_loss: 0.7538

Epoch [3/ 50] | d_loss: 1.0472 | g_loss: 1.4333

Epoch [3/ 50] | d_loss: 1.0415 | g_loss: 1.0481

Epoch [4/ 50] | d_loss: 1.1436 | g_loss: 0.9245

Epoch [4/ 50] | d_loss: 1.0768 | g_loss: 1.3391

Epoch [4/	50]	d_loss: 1.1346	g_loss: 0.6289
Epoch [4/	50]	d_loss: 0.8284	g_loss: 1.6287
Epoch [4/	50]	d_loss: 1.0985	g_loss: 1.1465
Epoch [4/	50]	d_loss: 1.1434	g_loss: 1.2264
Epoch [4/	50]	d_loss: 1.2273	g_loss: 0.7100
Epoch [4/	50]	d_loss: 1.1460	g_loss: 0.8793
Epoch [5/	50]	d_loss: 1.2278	g_loss: 0.8928
Epoch [5/	50]	d_loss: 0.9753	g_loss: 1.3726
Epoch [5/	50]	d_loss: 1.2180	g_loss: 1.5551
Epoch [5/	50]	d_loss: 1.0653	g_loss: 1.0978
Epoch [5/	50]	d_loss: 1.0401	g_loss: 1.0016
Epoch [5/	50]	d_loss: 0.9379	g_loss: 1.3975
Epoch [5/	50]	d_loss: 1.0225	g_loss: 0.9424
Epoch [5/	50]	d_loss: 1.1004	g_loss: 1.9373
Epoch [6/	50]	d_loss: 1.0244	g_loss: 1.2083
Epoch [6/	50]	d_loss: 1.0005	g_loss: 1.0460
Epoch [6/	50]	d_loss: 1.2608	g_loss: 0.7280
Epoch [6/	50]	d_loss: 0.9238	g_loss: 1.7015
Epoch [6/	50]	d_loss: 1.3777	g_loss: 0.8577
Epoch [6/	50]	d_loss: 1.1707	g_loss: 0.6677
Epoch [6/	50]	d_loss: 0.9504	g_loss: 1.6898
Epoch [6/	50]	d_loss: 1.0440	g_loss: 1.2702
Epoch [7/	50]	d_loss: 0.8041	g_loss: 1.1251
Epoch [7/	50]	d_loss: 1.0693	g_loss: 1.1466
Epoch [7/	50]	d_loss: 1.0443	g_loss: 2.1551
Epoch [7/	50]	d_loss: 0.9730	g_loss: 1.1837
Epoch [7/	50]	d_loss: 0.8781	g_loss: 1.6508
Epoch [7/	50]	d_loss: 1.0123	g_loss: 1.4311
Epoch [7/	50]	d_loss: 0.9700	g_loss: 1.3984
Epoch [7/	50]	d_loss: 1.0890	g_loss: 1.7393
Epoch [8/	50]	d_loss: 0.7744	g_loss: 1.3204
Epoch [8/	50]	d_loss: 0.9652	g_loss: 0.9274
Epoch [8/	50]	d_loss: 0.7971	g_loss: 1.4221
Epoch [8/	50]	d_loss: 0.8853	g_loss: 1.2784
Epoch [8/	50]	d_loss: 0.9162	g_loss: 1.0688
Epoch [8/	50]	d_loss: 1.1878	g_loss: 0.8080
Epoch [8/	50]	d_loss: 1.0612	g_loss: 2.7006
Epoch [8/	50]	d_loss: 0.9869	g_loss: 0.9213
Epoch [9/	50]	d_loss: 0.8406	g_loss: 1.6989
Epoch [9/	50]	d_loss: 0.9739	g_loss: 1.4438
Epoch [9/	50]	d_loss: 0.9126	g_loss: 1.2621
Epoch [9/	50]	d_loss: 0.9894	g_loss: 1.3082
Epoch [9/	50]	d_loss: 0.8999	g_loss: 1.5061
Epoch [9/	50]	d_loss: 0.7926	g_loss: 0.9875
Epoch [9/	50]	d_loss: 0.7760	g_loss: 1.5169
Epoch [9/	50]	d_loss: 0.8181	g_loss: 1.3369
Epoch [10/	50]	d_loss: 1.2106	g_loss: 2.9621
Epoch [10/	50]	d_loss: 0.8162	g_loss: 1.4015

Epoch [10/	50]	d_loss: 0.8682	g_loss: 1.0773
Epoch [10/	50]	d_loss: 0.8053	g_loss: 1.0267
Epoch [10/	50]	d_loss: 0.9741	g_loss: 1.1923
Epoch [10/	50]	d_loss: 1.3334	g_loss: 1.0534
Epoch [10/	50]	d_loss: 0.8960	g_loss: 1.1519
Epoch [10/	50]	d_loss: 0.8589	g_loss: 1.1213
Epoch [11/	50]	d_loss: 0.9697	g_loss: 0.7845
Epoch [11/	50]	d_loss: 0.8127	g_loss: 1.3019
Epoch [11/	50]	d_loss: 0.8606	g_loss: 1.4834
Epoch [11/	50]	d_loss: 1.1045	g_loss: 2.2951
Epoch [11/	50]	d_loss: 1.0019	g_loss: 0.6438
Epoch [11/	50]	d_loss: 0.7534	g_loss: 0.9826
Epoch [11/	50]	d_loss: 1.0162	g_loss: 0.5876
Epoch [11/	50]	d_loss: 0.7165	g_loss: 1.3764
Epoch [12/	50]	d_loss: 1.0766	g_loss: 1.3802
Epoch [12/	50]	d_loss: 0.8088	g_loss: 1.0293
Epoch [12/	50]	d_loss: 0.9783	g_loss: 1.9757
Epoch [12/	50]	d_loss: 0.6823	g_loss: 1.6759
Epoch [12/	50]	d_loss: 0.7832	g_loss: 1.4890
Epoch [12/	50]	d_loss: 0.8494	g_loss: 0.9968
Epoch [12/	50]	d_loss: 0.8703	g_loss: 1.3604
Epoch [12/	50]	d_loss: 0.8198	g_loss: 1.8151
Epoch [13/	50]	d_loss: 0.7991	g_loss: 1.7455
Epoch [13/	50]	d_loss: 0.6485	g_loss: 1.8404
Epoch [13/	50]	d_loss: 0.8159	g_loss: 1.4574
Epoch [13/	50]	d_loss: 0.7478	g_loss: 1.6360
Epoch [13/	50]	d_loss: 0.8274	g_loss: 1.3194
Epoch [13/	50]	d_loss: 0.8662	g_loss: 1.1652
Epoch [13/	50]	d_loss: 0.7985	g_loss: 1.1285
Epoch [13/	50]	d_loss: 0.8277	g_loss: 1.2915
Epoch [14/	50]	d_loss: 0.7548	g_loss: 1.7282
Epoch [14/	50]	d_loss: 0.6645	g_loss: 1.3845
Epoch [14/	50]	d_loss: 0.8907	g_loss: 1.6110
Epoch [14/	50]	d_loss: 0.7767	g_loss: 1.5676
Epoch [14/	50]	d_loss: 1.5693	g_loss: 3.0063
Epoch [14/	50]	d_loss: 1.0544	g_loss: 0.9080
Epoch [14/	50]	d_loss: 0.7567	g_loss: 1.3324
Epoch [14/	50]	d_loss: 0.6962	g_loss: 1.3224
Epoch [15/	50]	d_loss: 0.5624	g_loss: 2.1858
Epoch [15/	50]	d_loss: 0.6503	g_loss: 1.7535
Epoch [15/	50]	d_loss: 0.8154	g_loss: 1.4144
Epoch [15/	50]	d_loss: 0.7778	g_loss: 2.5531
Epoch [15/	50]	d_loss: 0.6963	g_loss: 1.9261
Epoch [15/	50]	d_loss: 0.6841	g_loss: 1.7340
Epoch [15/	50]	d_loss: 0.8465	g_loss: 0.9679
Epoch [15/	50]	d_loss: 0.7353	g_loss: 1.4981
Epoch [16/	50]	d_loss: 0.6591	g_loss: 1.7472
Epoch [16/	50]	d_loss: 0.7561	g_loss: 2.0956

Epoch [16/	50]	d_loss: 0.8134	g_loss: 2.5165
Epoch [16/	50]	d_loss: 0.9446	g_loss: 1.0390
Epoch [16/	50]	d_loss: 0.7882	g_loss: 1.3232
Epoch [16/	50]	d_loss: 0.8168	g_loss: 1.4081
Epoch [16/	50]	d_loss: 1.2774	g_loss: 0.9153
Epoch [16/	50]	d_loss: 0.6312	g_loss: 1.4435
Epoch [17/	50]	d_loss: 0.6913	g_loss: 1.7322
Epoch [17/	50]	d_loss: 0.6642	g_loss: 1.6736
Epoch [17/	50]	d_loss: 0.9678	g_loss: 0.8337
Epoch [17/	50]	d_loss: 0.6924	g_loss: 1.7369
Epoch [17/	50]	d_loss: 0.8384	g_loss: 2.2304
Epoch [17/	50]	d_loss: 0.6754	g_loss: 1.5971
Epoch [17/	50]	d_loss: 0.7738	g_loss: 1.2040
Epoch [17/	50]	d_loss: 0.6972	g_loss: 1.6227
Epoch [18/	50]	d_loss: 0.5650	g_loss: 1.5823
Epoch [18/	50]	d_loss: 0.7027	g_loss: 1.2475
Epoch [18/	50]	d_loss: 0.7984	g_loss: 2.0269
Epoch [18/	50]	d_loss: 1.3469	g_loss: 3.2720
Epoch [18/	50]	d_loss: 0.6275	g_loss: 1.5115
Epoch [18/	50]	d_loss: 0.6410	g_loss: 1.9235
Epoch [18/	50]	d_loss: 0.6828	g_loss: 1.6294
Epoch [18/	50]	d_loss: 0.5731	g_loss: 1.5011
Epoch [19/	50]	d_loss: 0.8106	g_loss: 1.1885
Epoch [19/	50]	d_loss: 0.7869	g_loss: 1.3455
Epoch [19/	50]	d_loss: 0.7740	g_loss: 2.0877
Epoch [19/	50]	d_loss: 0.6349	g_loss: 2.0280
Epoch [19/	50]	d_loss: 0.6522	g_loss: 2.7209
Epoch [19/	50]	d_loss: 0.7710	g_loss: 2.0233
Epoch [19/	50]	d_loss: 0.6889	g_loss: 1.4346
Epoch [19/	50]	d_loss: 0.8475	g_loss: 2.0852
Epoch [20/	50]	d_loss: 0.5262	g_loss: 1.9535
Epoch [20/	50]	d_loss: 0.8950	g_loss: 1.4675
Epoch [20/	50]	d_loss: 0.7144	g_loss: 2.3525
Epoch [20/	50]	d_loss: 0.7317	g_loss: 1.9102
Epoch [20/	50]	d_loss: 0.7432	g_loss: 2.8915
Epoch [20/	50]	d_loss: 1.1559	g_loss: 0.7253
Epoch [20/	50]	d_loss: 0.6350	g_loss: 2.5842
Epoch [20/	50]	d_loss: 0.7592	g_loss: 1.9339
Epoch [21/	50]	d_loss: 0.7510	g_loss: 1.2295
Epoch [21/	50]	d_loss: 0.5715	g_loss: 1.7260
Epoch [21/	50]	d_loss: 0.7996	g_loss: 1.7341
Epoch [21/	50]	d_loss: 0.4589	g_loss: 1.8512
Epoch [21/	50]	d_loss: 0.6815	g_loss: 1.4013
Epoch [21/	50]	d_loss: 0.5821	g_loss: 1.7095
Epoch [21/	50]	d_loss: 0.5672	g_loss: 1.6510
Epoch [21/	50]	d_loss: 0.5465	g_loss: 2.3817
Epoch [22/	50]	d_loss: 0.4897	g_loss: 1.5954
Epoch [22/	50]	d_loss: 0.6961	g_loss: 1.4987

Epoch [22/	50]	d_loss: 0.6364	g_loss: 3.0035
Epoch [22/	50]	d_loss: 0.5857	g_loss: 2.2704
Epoch [22/	50]	d_loss: 0.4966	g_loss: 2.0512
Epoch [22/	50]	d_loss: 0.7436	g_loss: 1.3981
Epoch [22/	50]	d_loss: 0.7947	g_loss: 2.6824
Epoch [22/	50]	d_loss: 0.6250	g_loss: 2.1517
Epoch [23/	50]	d_loss: 0.6808	g_loss: 1.2089
Epoch [23/	50]	d_loss: 0.5744	g_loss: 1.8906
Epoch [23/	50]	d_loss: 0.5112	g_loss: 2.1828
Epoch [23/	50]	d_loss: 0.6849	g_loss: 2.8491
Epoch [23/	50]	d_loss: 0.6353	g_loss: 1.9149
Epoch [23/	50]	d_loss: 0.6771	g_loss: 2.4464
Epoch [23/	50]	d_loss: 0.4830	g_loss: 2.4065
Epoch [23/	50]	d_loss: 0.6820	g_loss: 2.5083
Epoch [24/	50]	d_loss: 0.6287	g_loss: 1.5856
Epoch [24/	50]	d_loss: 0.5463	g_loss: 2.0197
Epoch [24/	50]	d_loss: 0.4744	g_loss: 2.3401
Epoch [24/	50]	d_loss: 0.4512	g_loss: 2.0392
Epoch [24/	50]	d_loss: 0.4770	g_loss: 1.7826
Epoch [24/	50]	d_loss: 0.5249	g_loss: 1.8300
Epoch [24/	50]	d_loss: 0.6506	g_loss: 1.3057
Epoch [24/	50]	d_loss: 0.4444	g_loss: 2.7832
Epoch [25/	50]	d_loss: 0.5354	g_loss: 1.4008
Epoch [25/	50]	d_loss: 0.6888	g_loss: 1.6784
Epoch [25/	50]	d_loss: 0.8089	g_loss: 1.9652
Epoch [25/	50]	d_loss: 0.4559	g_loss: 2.5853
Epoch [25/	50]	d_loss: 0.6642	g_loss: 1.3232
Epoch [25/	50]	d_loss: 0.5443	g_loss: 2.2906
Epoch [25/	50]	d_loss: 0.6053	g_loss: 1.4698
Epoch [25/	50]	d_loss: 1.1570	g_loss: 0.4060
Epoch [26/	50]	d_loss: 0.6160	g_loss: 1.6688
Epoch [26/	50]	d_loss: 0.4241	g_loss: 2.6445
Epoch [26/	50]	d_loss: 0.4437	g_loss: 2.2551
Epoch [26/	50]	d_loss: 0.5831	g_loss: 1.3069
Epoch [26/	50]	d_loss: 0.4701	g_loss: 2.4220
Epoch [26/	50]	d_loss: 0.5203	g_loss: 2.1913
Epoch [26/	50]	d_loss: 0.4353	g_loss: 1.4870
Epoch [26/	50]	d_loss: 0.4038	g_loss: 1.5321
Epoch [27/	50]	d_loss: 0.4243	g_loss: 1.9403
Epoch [27/	50]	d_loss: 0.5839	g_loss: 3.0921
Epoch [27/	50]	d_loss: 0.7636	g_loss: 2.8041
Epoch [27/	50]	d_loss: 0.5093	g_loss: 1.7471
Epoch [27/	50]	d_loss: 0.4528	g_loss: 2.1327
Epoch [27/	50]	d_loss: 0.3062	g_loss: 2.3284
Epoch [27/	50]	d_loss: 0.5316	g_loss: 1.7665
Epoch [27/	50]	d_loss: 0.5501	g_loss: 1.6666
Epoch [28/	50]	d_loss: 0.5697	g_loss: 1.7612
Epoch [28/	50]	d_loss: 0.2519	g_loss: 2.9214

Epoch [28/	50]	d_loss: 0.3308	g_loss: 2.3427
Epoch [28/	50]	d_loss: 0.2146	g_loss: 2.8676
Epoch [28/	50]	d_loss: 0.3776	g_loss: 2.3714
Epoch [28/	50]	d_loss: 0.5107	g_loss: 1.7921
Epoch [28/	50]	d_loss: 0.5172	g_loss: 3.4863
Epoch [28/	50]	d_loss: 0.4679	g_loss: 2.0018
Epoch [29/	50]	d_loss: 0.4914	g_loss: 1.5860
Epoch [29/	50]	d_loss: 0.7497	g_loss: 3.7961
Epoch [29/	50]	d_loss: 0.3288	g_loss: 2.4110
Epoch [29/	50]	d_loss: 0.4135	g_loss: 2.3400
Epoch [29/	50]	d_loss: 0.3574	g_loss: 2.0063
Epoch [29/	50]	d_loss: 0.3933	g_loss: 1.3279
Epoch [29/	50]	d_loss: 0.3917	g_loss: 2.4578
Epoch [29/	50]	d_loss: 0.5915	g_loss: 1.1930
Epoch [30/	50]	d_loss: 0.5035	g_loss: 2.3110
Epoch [30/	50]	d_loss: 0.3537	g_loss: 2.0534
Epoch [30/	50]	d_loss: 0.3708	g_loss: 1.9393
Epoch [30/	50]	d_loss: 0.3293	g_loss: 2.5926
Epoch [30/	50]	d_loss: 0.2646	g_loss: 2.8144
Epoch [30/	50]	d_loss: 0.3428	g_loss: 2.3812
Epoch [30/	50]	d_loss: 0.7322	g_loss: 0.9720
Epoch [30/	50]	d_loss: 0.3386	g_loss: 2.5875
Epoch [31/	50]	d_loss: 0.3686	g_loss: 1.4425
Epoch [31/	50]	d_loss: 0.3771	g_loss: 2.1713
Epoch [31/	50]	d_loss: 0.4179	g_loss: 2.6138
Epoch [31/	50]	d_loss: 0.3846	g_loss: 1.7576
Epoch [31/	50]	d_loss: 0.3777	g_loss: 3.1531
Epoch [31/	50]	d_loss: 0.4283	g_loss: 1.4479
Epoch [31/	50]	d_loss: 0.4218	g_loss: 2.5448
Epoch [31/	50]	d_loss: 0.3590	g_loss: 2.5038
Epoch [32/	50]	d_loss: 0.3893	g_loss: 2.0691
Epoch [32/	50]	d_loss: 0.3292	g_loss: 2.2843
Epoch [32/	50]	d_loss: 0.3860	g_loss: 1.9483
Epoch [32/	50]	d_loss: 0.3510	g_loss: 2.4783
Epoch [32/	50]	d_loss: 0.4156	g_loss: 3.2876
Epoch [32/	50]	d_loss: 0.4273	g_loss: 2.6754
Epoch [32/	50]	d_loss: 0.3651	g_loss: 2.5138
Epoch [32/	50]	d_loss: 0.3657	g_loss: 2.8077
Epoch [33/	50]	d_loss: 0.3308	g_loss: 2.5474
Epoch [33/	50]	d_loss: 0.5034	g_loss: 2.6330
Epoch [33/	50]	d_loss: 0.4219	g_loss: 1.8813
Epoch [33/	50]	d_loss: 0.6266	g_loss: 1.3456
Epoch [33/	50]	d_loss: 0.4576	g_loss: 2.3890
Epoch [33/	50]	d_loss: 0.3416	g_loss: 2.4767
Epoch [33/	50]	d_loss: 0.6047	g_loss: 1.4936
Epoch [33/	50]	d_loss: 0.3403	g_loss: 2.4347
Epoch [34/	50]	d_loss: 0.2561	g_loss: 3.0691
Epoch [34/	50]	d_loss: 0.3508	g_loss: 2.4693

Epoch [34/	50]		d_loss: 0.4322		g_loss: 1.7840
Epoch [34/	50]		d_loss: 0.5220		g_loss: 3.0549
Epoch [34/	50]		d_loss: 0.3602		g_loss: 3.2915
Epoch [34/	50]		d_loss: 3.3908		g_loss: 0.4862
Epoch [34/	50]		d_loss: 0.3721		g_loss: 2.4601
Epoch [34/	50]		d_loss: 0.5673		g_loss: 2.5220
Epoch [35/	50]		d_loss: 0.3502		g_loss: 3.0799
Epoch [35/	50]		d_loss: 0.4436		g_loss: 2.4219
Epoch [35/	50]		d_loss: 0.3048		g_loss: 3.4359
Epoch [35/	50]		d_loss: 1.2003		g_loss: 4.1648
Epoch [35/	50]		d_loss: 0.4497		g_loss: 1.6755
Epoch [35/	50]		d_loss: 0.3165		g_loss: 2.1064
Epoch [35/	50]		d_loss: 0.2310		g_loss: 2.4927
Epoch [35/	50]		d_loss: 0.3413		g_loss: 2.5939
Epoch [36/	50]		d_loss: 0.3156		g_loss: 3.3271
Epoch [36/	50]		d_loss: 0.3181		g_loss: 3.0078
Epoch [36/	50]		d_loss: 0.2347		g_loss: 3.2193
Epoch [36/	50]		d_loss: 0.3833		g_loss: 2.1640
Epoch [36/	50]		d_loss: 0.4301		g_loss: 1.4354
Epoch [36/	50]		d_loss: 0.2382		g_loss: 3.3253
Epoch [36/	50]		d_loss: 0.3478		g_loss: 2.6118
Epoch [36/	50]		d_loss: 0.3669		g_loss: 1.9952
Epoch [37/	50]		d_loss: 0.2599		g_loss: 2.7295
Epoch [37/	50]		d_loss: 0.4593		g_loss: 3.2405
Epoch [37/	50]		d_loss: 3.1042		g_loss: 0.1306
Epoch [37/	50]		d_loss: 0.4738		g_loss: 2.8923
Epoch [37/	50]		d_loss: 0.3943		g_loss: 2.1086
Epoch [37/	50]		d_loss: 1.2083		g_loss: 3.7513
Epoch [37/	50]		d_loss: 0.3283		g_loss: 1.8915
Epoch [37/	50]		d_loss: 0.2334		g_loss: 3.0025
Epoch [38/	50]		d_loss: 0.4152		g_loss: 2.2199
Epoch [38/	50]		d_loss: 2.1713		g_loss: 8.3741
Epoch [38/	50]		d_loss: 0.5262		g_loss: 2.8488
Epoch [38/	50]		d_loss: 0.2439		g_loss: 2.9390
Epoch [38/	50]		d_loss: 0.4105		g_loss: 2.5100
Epoch [38/	50]		d_loss: 0.2617		g_loss: 2.2323
Epoch [38/	50]		d_loss: 0.3727		g_loss: 2.8768
Epoch [38/	50]		d_loss: 0.2861		g_loss: 3.4730
Epoch [39/	50]		d_loss: 0.2685		g_loss: 2.7460
Epoch [39/	50]		d_loss: 0.3795		g_loss: 2.5234
Epoch [39/	50]		d_loss: 0.3623		g_loss: 1.7983
Epoch [39/	50]		d_loss: 0.3451		g_loss: 3.2486
Epoch [39/	50]		d_loss: 0.3441		g_loss: 3.1701
Epoch [39/	50]		d_loss: 0.5246		g_loss: 3.0461
Epoch [39/	50]		d_loss: 0.4262		g_loss: 1.4459
Epoch [39/	50]		d_loss: 0.3863		g_loss: 2.0100
Epoch [40/	50]		d_loss: 0.3253		g_loss: 2.9142
Epoch [40/	50]		d_loss: 0.5756		g_loss: 1.6008

Epoch [40/	50]	d_loss: 0.2878	g_loss: 2.6065
Epoch [40/	50]	d_loss: 0.3474	g_loss: 2.4387
Epoch [40/	50]	d_loss: 0.2080	g_loss: 2.0549
Epoch [40/	50]	d_loss: 0.2771	g_loss: 2.8798
Epoch [40/	50]	d_loss: 0.2910	g_loss: 3.4469
Epoch [40/	50]	d_loss: 0.2211	g_loss: 2.8747
Epoch [41/	50]	d_loss: 0.3067	g_loss: 2.9119
Epoch [41/	50]	d_loss: 0.2445	g_loss: 3.3495
Epoch [41/	50]	d_loss: 0.3288	g_loss: 2.9128
Epoch [41/	50]	d_loss: 0.4419	g_loss: 2.2126
Epoch [41/	50]	d_loss: 0.2768	g_loss: 2.6364
Epoch [41/	50]	d_loss: 0.5445	g_loss: 1.3559
Epoch [41/	50]	d_loss: 0.5666	g_loss: 1.4743
Epoch [41/	50]	d_loss: 0.1787	g_loss: 3.0435
Epoch [42/	50]	d_loss: 0.2152	g_loss: 3.3623
Epoch [42/	50]	d_loss: 0.2240	g_loss: 2.8522
Epoch [42/	50]	d_loss: 0.3004	g_loss: 2.9482
Epoch [42/	50]	d_loss: 0.3310	g_loss: 3.4751
Epoch [42/	50]	d_loss: 0.2392	g_loss: 2.7710
Epoch [42/	50]	d_loss: 0.2875	g_loss: 2.1772
Epoch [42/	50]	d_loss: 0.3881	g_loss: 2.3759
Epoch [42/	50]	d_loss: 0.4997	g_loss: 3.3887
Epoch [43/	50]	d_loss: 0.3442	g_loss: 1.9196
Epoch [43/	50]	d_loss: 0.3393	g_loss: 2.6423
Epoch [43/	50]	d_loss: 0.7460	g_loss: 0.5955
Epoch [43/	50]	d_loss: 0.4787	g_loss: 1.5847
Epoch [43/	50]	d_loss: 0.2860	g_loss: 2.6868
Epoch [43/	50]	d_loss: 0.2742	g_loss: 2.4130
Epoch [43/	50]	d_loss: 0.2614	g_loss: 3.3856
Epoch [43/	50]	d_loss: 0.3331	g_loss: 1.8996
Epoch [44/	50]	d_loss: 0.2662	g_loss: 2.9793
Epoch [44/	50]	d_loss: 0.3630	g_loss: 1.8725
Epoch [44/	50]	d_loss: 0.2779	g_loss: 3.3407
Epoch [44/	50]	d_loss: 0.3111	g_loss: 3.2731
Epoch [44/	50]	d_loss: 0.2186	g_loss: 2.4544
Epoch [44/	50]	d_loss: 0.2947	g_loss: 2.6069
Epoch [44/	50]	d_loss: 1.0422	g_loss: 3.8348
Epoch [44/	50]	d_loss: 0.2462	g_loss: 2.7675
Epoch [45/	50]	d_loss: 0.4891	g_loss: 2.9161
Epoch [45/	50]	d_loss: 0.7197	g_loss: 1.0425
Epoch [45/	50]	d_loss: 0.2408	g_loss: 2.9955
Epoch [45/	50]	d_loss: 0.2089	g_loss: 3.9965
Epoch [45/	50]	d_loss: 0.4148	g_loss: 4.9365
Epoch [45/	50]	d_loss: 0.3415	g_loss: 1.9730
Epoch [45/	50]	d_loss: 0.4326	g_loss: 2.3499
Epoch [45/	50]	d_loss: 0.1893	g_loss: 2.9496
Epoch [46/	50]	d_loss: 0.3613	g_loss: 2.3818
Epoch [46/	50]	d_loss: 0.3341	g_loss: 4.6451

Epoch [46/	50]	d_loss: 0.3034	g_loss: 4.3403
Epoch [46/	50]	d_loss: 0.5558	g_loss: 1.5682
Epoch [46/	50]	d_loss: 0.3448	g_loss: 1.2443
Epoch [46/	50]	d_loss: 0.1611	g_loss: 3.1714
Epoch [46/	50]	d_loss: 0.2889	g_loss: 2.8153
Epoch [46/	50]	d_loss: 0.6186	g_loss: 2.3011
Epoch [47/	50]	d_loss: 0.7448	g_loss: 4.9609
Epoch [47/	50]	d_loss: 0.2679	g_loss: 2.1135
Epoch [47/	50]	d_loss: 0.2852	g_loss: 3.1823
Epoch [47/	50]	d_loss: 0.2855	g_loss: 2.1534
Epoch [47/	50]	d_loss: 0.2664	g_loss: 3.3120
Epoch [47/	50]	d_loss: 0.2070	g_loss: 2.9009
Epoch [47/	50]	d_loss: 2.1162	g_loss: 4.8998
Epoch [47/	50]	d_loss: 0.2648	g_loss: 3.4436
Epoch [48/	50]	d_loss: 0.3641	g_loss: 3.4939
Epoch [48/	50]	d_loss: 0.1906	g_loss: 3.0128
Epoch [48/	50]	d_loss: 0.4373	g_loss: 1.7099
Epoch [48/	50]	d_loss: 0.4161	g_loss: 2.5542
Epoch [48/	50]	d_loss: 0.2095	g_loss: 3.0922
Epoch [48/	50]	d_loss: 0.2915	g_loss: 3.7948
Epoch [48/	50]	d_loss: 0.1716	g_loss: 2.6087
Epoch [48/	50]	d_loss: 0.3142	g_loss: 3.6506
Epoch [49/	50]	d_loss: 0.4469	g_loss: 4.4789
Epoch [49/	50]	d_loss: 0.3641	g_loss: 3.1749
Epoch [49/	50]	d_loss: 0.1731	g_loss: 2.7895
Epoch [49/	50]	d_loss: 0.1732	g_loss: 3.1064
Epoch [49/	50]	d_loss: 0.2803	g_loss: 3.5064
Epoch [49/	50]	d_loss: 0.2772	g_loss: 3.5002
Epoch [49/	50]	d_loss: 0.4259	g_loss: 3.6595
Epoch [49/	50]	d_loss: 0.2818	g_loss: 2.1224
Epoch [50/	50]	d_loss: 0.2285	g_loss: 2.9350
Epoch [50/	50]	d_loss: 0.3803	g_loss: 2.6541
Epoch [50/	50]	d_loss: 0.5239	g_loss: 3.8655
Epoch [50/	50]	d_loss: 0.3049	g_loss: 3.8599
Epoch [50/	50]	d_loss: 0.2083	g_loss: 2.7985
Epoch [50/	50]	d_loss: 2.5682	g_loss: 0.8169
Epoch [50/	50]	d_loss: 0.2111	g_loss: 3.1086
Epoch [50/	50]	d_loss: 0.1935	g_loss: 3.8520

2.8 Training loss

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

```
In [70]: fig, ax = plt.subplots(figsize=(16,4))
         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[70]: <matplotlib.legend.Legend at 0x7f35509a9898>



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.

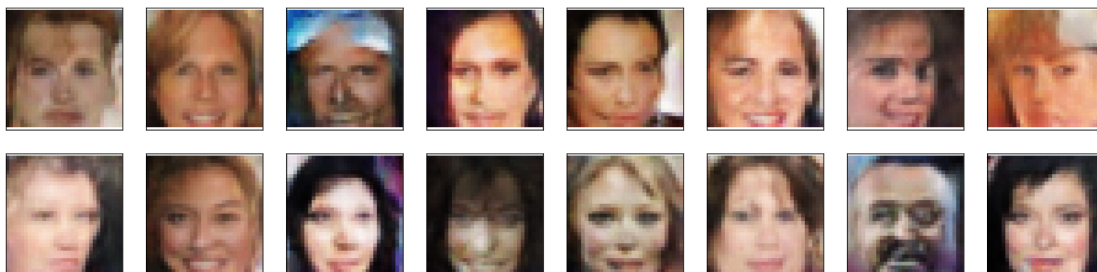
In [71]: *# helper function for viewing a list of passed in sample images*

```
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 [72]: *# Load samples from generator, taken while training*

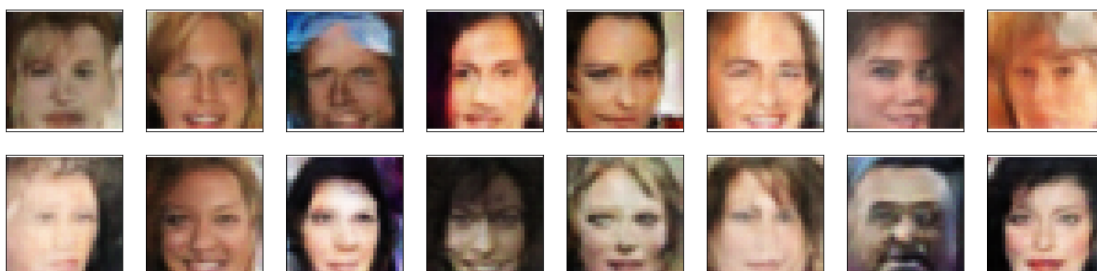
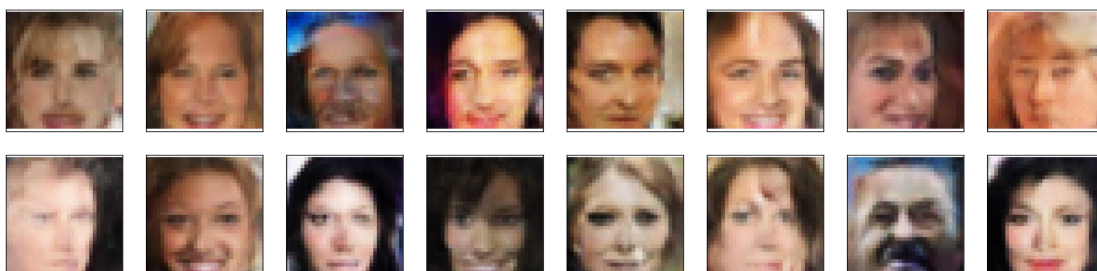
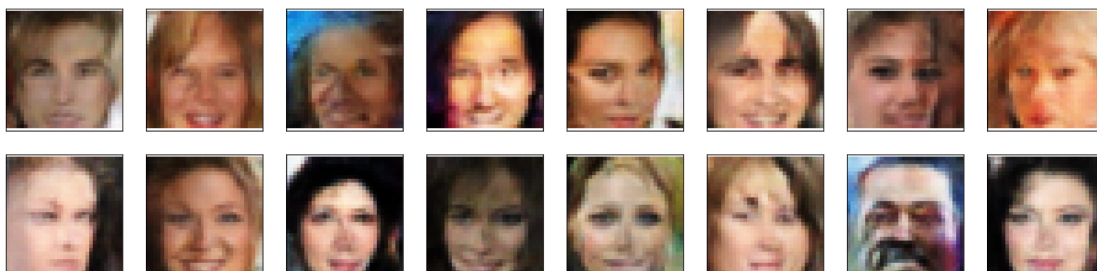
```
with open('train_samples.pkl', 'rb') as f:
    samples = pkl.load(f)
```

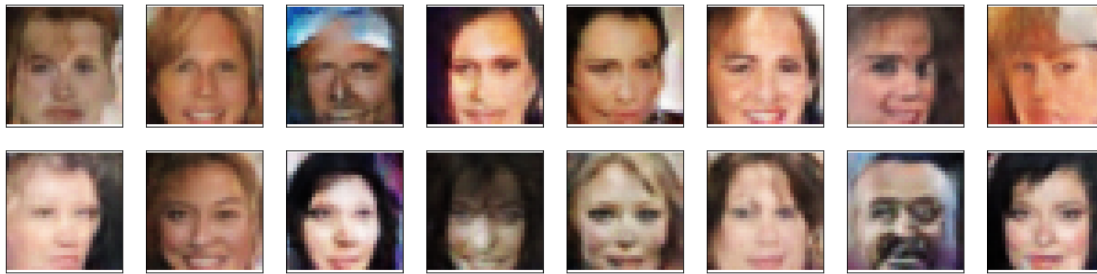
In [77]: `_ = view_samples(-1, samples)`



2.9.1 Let's check the model evolution

```
In [75]: _ = [view_samples(epoch, samples) for epoch in (1, 15, 30, 45, -1)]
```





2.9.2 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:

- the model was able to generate pictures that look very similar to human faces
- the larger model could learn more features and could produce better pictures and more details, but it would take much longer to train it
- [here](#) we may find some practical tips about training and improving GAN models

2.9.3 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.