

# dlnd\_face\_generation

December 26, 2019

## 1 Face Generation

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

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

### 1.0.1 Get the Data

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

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

### 1.0.2 Pre-processed Data

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

If you are working locally, you can download this data [by clicking here](#)

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

```
In [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]



```
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 torch
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
    image_augmented = transforms.Compose([
        transforms.Resize(image_size),
        transforms.CenterCrop(image_size),
        transforms.ToTensor()
    ])
    image_data = datasets.ImageFolder(data_dir, transform=image_augmented)
    data_loader = torch.utils.data.DataLoader(image_data, batch_size=batch_size, shuffle=True)
    return data_loader
```

## 1.2 Create a DataLoader

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

```
In [4]: # Define function hyperparameters
        batch_size = 128
        img_size = 32

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

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

Next, you can view some images! You should 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 [5]: # helper display function
        def imshow(img):
            npimg = img.numpy()
            plt.imshow(np.transpose(npimg, (1, 2, 0)))

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

        # obtain one batch of training images
        dataiter = iter(celeba_train_loader)
        images, _ = dataiter.next() # _ for no labels

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



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

```
In [6]: # 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

    return x*(max-min) + min

In [7]: """
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(-0.9843)

Max: tensor(0.7176)

---

## 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 [8]: import torch.nn as nn
        import torch.nn.functional as F

In [9]: def conv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True):
        """
        Creates a convolutional layer, with optional batch normalization.
        """

        layers=[]
        conv_layer = nn.Conv2d(in_channels, out_channels,
                                kernel_size, stride, padding, bias=False)

        # append conv layer
        layers.append(conv_layer)

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

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

class Discriminator(nn.Module):

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

        # complete init function
        self.conv_dim = conv_dim
        # 32x32x3 tensor images
        self.conv1 = conv(3, conv_dim, 4, batch_norm=False)
        # 16x16
        self.conv2 = conv(conv_dim, conv_dim*2, 4)
        # 8x8
        self.conv3 = conv(conv_dim*2, conv_dim*4, 4)
        # 4x4
        self.conv4 = conv(conv_dim*4, conv_dim*8, 4)
        # final layer
        self.fc = nn.Linear(conv_dim*8*2*2, 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)
        out = F.leaky_relu(self.conv4(out), 0.2)
        # flattening
        out = out.view(-1, self.conv_dim*8*2*2)
        out = self.fc(out)
        return out

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

```

Tests Passed

## 2.2 Generator

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

### Exercise: Complete the Generator class

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

```

In [10]: def deconv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True):

    layers=[]
    transpose_conv_layer = nn.ConvTranspose2d(in_channels, out_channels,
                                                kernel_size, stride, padding, bias=False)

    # append transpose convolutional layer
    layers.append(transpose_conv_layer)

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

    return nn.Sequential(*layers)

```



```

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.deconv1 = deconv(conv_dim*8, conv_dim*4, 4)
        self.deconv2 = deconv(conv_dim*4, conv_dim*2, 4)
        self.deconv3 = deconv(conv_dim*2, conv_dim, 4)
        self.deconv4 = 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)
        out = F.relu(self.deconv1(out))
        out = F.relu(self.deconv2(out))
        out = F.relu(self.deconv3(out))
        out = self.deconv4(out)
        out = torch.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 [11]: def weights_init_normal(m):
        """
        Applies initial weights to certain layers in a model .
        The weights are taken from a normal distribution
        with mean = 0, std dev = 0.02.
        :param m: A module or layer in a network
        """
        # classname will be something like:
        # `Conv`, `BatchNorm2d`, `Linear`, etc.
        classname = m.__class__.__name__

        # TODO: Apply initial weights to convolutional and linear layers
        if hasattr(m, 'weight') and (classname.find('Conv') != -1 or classname.find('Linear') != -1):
            m.weight.data.normal_(0.0, 0.02)

            # The bias terms, if they exist, set to 0
            if hasattr(m, 'bias') and m.bias is not None:
                m.bias.data.zero_()
```

## 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 [12]: """
        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 [13]: # Define model hyperparams
         d_conv_dim = 64
         g_conv_dim = 64
         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, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
  (conv2): 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)
  )
  (conv3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(512, 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)
  (deconv1): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (deconv2): 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)
  )
  (deconv3): 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)
)
(deconv4): Sequential(
  (0): ConvTranspose2d(64, 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 [14]: """
        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 [23]: def real_loss(D_out, smooth=False):
    '''Calculates how close discriminator outputs are to being real.
        param, D_out: discriminator logits
        return: real loss'''
    batch_size = D_out.size(0)

    labels = torch.ones(batch_size) * 0.9

    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) # fake labels = 0
    if train_on_gpu:
        labels = labels.cuda()
    criterion = nn.BCEWithLogitsLoss()
    # calculate loss
    loss = criterion(D_out.squeeze(), labels)
    return loss
```

## 2.6 Optimizers

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

```
In [24]: import torch.optim as optim
    lr = 0.002

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

---

## 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 [25]: 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()
    # 1. Train the discriminator on real and fake images
    D_real = D(real_images)
    d_real_loss = real_loss(D_real, smooth=True)
    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)
# back-prop and optimizer
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 [26]: # 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: 0.5014 | g_loss: 13.6551
Epoch [ 1/ 10] | d_loss: 0.4647 | g_loss: 5.0613
Epoch [ 1/ 10] | d_loss: 0.3592 | g_loss: 5.7975
Epoch [ 1/ 10] | d_loss: 0.3900 | g_loss: 4.0574
Epoch [ 1/ 10] | d_loss: 0.3686 | g_loss: 6.7400
Epoch [ 1/ 10] | d_loss: 0.3832 | g_loss: 5.5605
Epoch [ 1/ 10] | d_loss: 0.4391 | g_loss: 4.6485
Epoch [ 1/ 10] | d_loss: 0.3566 | g_loss: 5.2976
Epoch [ 1/ 10] | d_loss: 0.3774 | g_loss: 6.4476
Epoch [ 1/ 10] | d_loss: 0.4255 | g_loss: 6.9599
Epoch [ 1/ 10] | d_loss: 0.3650 | g_loss: 7.0081
Epoch [ 1/ 10] | d_loss: 0.5673 | g_loss: 6.4221
Epoch [ 1/ 10] | d_loss: 0.3561 | g_loss: 6.7370
Epoch [ 1/ 10] | d_loss: 0.3646 | g_loss: 5.1145
Epoch [ 1/ 10] | d_loss: 0.3696 | g_loss: 4.2255
Epoch [ 2/ 10] | d_loss: 0.3977 | g_loss: 4.3879
Epoch [ 2/ 10] | d_loss: 0.3836 | g_loss: 4.9103
Epoch [ 2/ 10] | d_loss: 0.4148 | g_loss: 6.7445
Epoch [ 2/ 10] | d_loss: 0.3381 | g_loss: 5.4217
Epoch [ 2/ 10] | d_loss: 0.3777 | g_loss: 4.8902
Epoch [ 2/ 10] | d_loss: 0.3377 | g_loss: 7.0793
Epoch [ 2/ 10] | d_loss: 0.3323 | g_loss: 6.9637
Epoch [ 2/ 10] | d_loss: 0.3480 | g_loss: 7.3131

```



Epoch [	2/	10]	d_loss: 0.5865	g_loss: 4.6038
Epoch [	2/	10]	d_loss: 0.3815	g_loss: 6.0910
Epoch [	2/	10]	d_loss: 0.3419	g_loss: 7.7945
Epoch [	2/	10]	d_loss: 0.3422	g_loss: 6.8624
Epoch [	2/	10]	d_loss: 0.3376	g_loss: 5.9148
Epoch [	2/	10]	d_loss: 0.3709	g_loss: 5.5215
Epoch [	2/	10]	d_loss: 0.3627	g_loss: 7.9334
Epoch [	3/	10]	d_loss: 0.3641	g_loss: 7.2183
Epoch [	3/	10]	d_loss: 0.5227	g_loss: 5.8275
Epoch [	3/	10]	d_loss: 0.5205	g_loss: 6.2369
Epoch [	3/	10]	d_loss: 0.3653	g_loss: 6.1588
Epoch [	3/	10]	d_loss: 0.3639	g_loss: 7.8339
Epoch [	3/	10]	d_loss: 0.3517	g_loss: 5.7372
Epoch [	3/	10]	d_loss: 0.3842	g_loss: 4.4416
Epoch [	3/	10]	d_loss: 0.4140	g_loss: 12.9866
Epoch [	3/	10]	d_loss: 0.4034	g_loss: 8.0780
Epoch [	3/	10]	d_loss: 0.3587	g_loss: 7.9184
Epoch [	3/	10]	d_loss: 0.4730	g_loss: 9.9130
Epoch [	3/	10]	d_loss: 0.3730	g_loss: 6.7014
Epoch [	3/	10]	d_loss: 0.5300	g_loss: 8.4493
Epoch [	3/	10]	d_loss: 0.4888	g_loss: 5.1640
Epoch [	3/	10]	d_loss: 0.3441	g_loss: 6.3226
Epoch [	4/	10]	d_loss: 0.3378	g_loss: 5.9256
Epoch [	4/	10]	d_loss: 0.3410	g_loss: 5.4120
Epoch [	4/	10]	d_loss: 0.3513	g_loss: 4.8642
Epoch [	4/	10]	d_loss: 0.3661	g_loss: 5.6082
Epoch [	4/	10]	d_loss: 0.4185	g_loss: 7.3009
Epoch [	4/	10]	d_loss: 0.3480	g_loss: 6.1236
Epoch [	4/	10]	d_loss: 0.3479	g_loss: 6.1885
Epoch [	4/	10]	d_loss: 0.3439	g_loss: 6.0541
Epoch [	4/	10]	d_loss: 0.3511	g_loss: 6.4951
Epoch [	4/	10]	d_loss: 0.3746	g_loss: 6.6667
Epoch [	4/	10]	d_loss: 0.3357	g_loss: 9.0843
Epoch [	4/	10]	d_loss: 0.4307	g_loss: 4.8539
Epoch [	4/	10]	d_loss: 0.3783	g_loss: 7.7679
Epoch [	4/	10]	d_loss: 0.4369	g_loss: 3.0695
Epoch [	4/	10]	d_loss: 0.4706	g_loss: 8.7203
Epoch [	5/	10]	d_loss: 0.4136	g_loss: 6.4479
Epoch [	5/	10]	d_loss: 0.4035	g_loss: 8.7515
Epoch [	5/	10]	d_loss: 0.3759	g_loss: 5.0955
Epoch [	5/	10]	d_loss: 0.3472	g_loss: 6.2742
Epoch [	5/	10]	d_loss: 0.3626	g_loss: 6.9638
Epoch [	5/	10]	d_loss: 0.3462	g_loss: 8.6712
Epoch [	5/	10]	d_loss: 0.4270	g_loss: 8.4513
Epoch [	5/	10]	d_loss: 0.3541	g_loss: 8.5147
Epoch [	5/	10]	d_loss: 0.4174	g_loss: 5.5249
Epoch [	5/	10]	d_loss: 0.3499	g_loss: 6.9191
Epoch [	5/	10]	d_loss: 0.5801	g_loss: 2.1266

Epoch [	5/	10]	d_loss: 0.3926	g_loss: 3.5037
Epoch [	5/	10]	d_loss: 0.3494	g_loss: 9.8656
Epoch [	5/	10]	d_loss: 0.4341	g_loss: 6.0768
Epoch [	5/	10]	d_loss: 0.4511	g_loss: 5.8550
Epoch [	6/	10]	d_loss: 0.4180	g_loss: 6.5934
Epoch [	6/	10]	d_loss: 0.3497	g_loss: 7.1227
Epoch [	6/	10]	d_loss: 0.4599	g_loss: 6.2087
Epoch [	6/	10]	d_loss: 0.3538	g_loss: 7.4734
Epoch [	6/	10]	d_loss: 0.3996	g_loss: 5.0330
Epoch [	6/	10]	d_loss: 0.3487	g_loss: 8.5916
Epoch [	6/	10]	d_loss: 0.3538	g_loss: 4.8129
Epoch [	6/	10]	d_loss: 0.3570	g_loss: 5.2478
Epoch [	6/	10]	d_loss: 0.4131	g_loss: 5.2242
Epoch [	6/	10]	d_loss: 0.3557	g_loss: 6.1360
Epoch [	6/	10]	d_loss: 0.3564	g_loss: 7.3601
Epoch [	6/	10]	d_loss: 0.4346	g_loss: 8.5344
Epoch [	6/	10]	d_loss: 0.3388	g_loss: 8.7430
Epoch [	6/	10]	d_loss: 0.3433	g_loss: 7.1437
Epoch [	6/	10]	d_loss: 0.3632	g_loss: 5.7459
Epoch [	7/	10]	d_loss: 0.3658	g_loss: 6.0144
Epoch [	7/	10]	d_loss: 0.3563	g_loss: 7.3313
Epoch [	7/	10]	d_loss: 0.3382	g_loss: 6.8388
Epoch [	7/	10]	d_loss: 0.3494	g_loss: 9.5919
Epoch [	7/	10]	d_loss: 0.3730	g_loss: 8.7194
Epoch [	7/	10]	d_loss: 0.3420	g_loss: 7.0730
Epoch [	7/	10]	d_loss: 0.3454	g_loss: 7.6719
Epoch [	7/	10]	d_loss: 0.6060	g_loss: 2.6335
Epoch [	7/	10]	d_loss: 0.3717	g_loss: 5.8375
Epoch [	7/	10]	d_loss: 0.3412	g_loss: 5.9860
Epoch [	7/	10]	d_loss: 0.3415	g_loss: 7.0850
Epoch [	7/	10]	d_loss: 0.3595	g_loss: 6.2994
Epoch [	7/	10]	d_loss: 0.3548	g_loss: 9.0388
Epoch [	7/	10]	d_loss: 0.3561	g_loss: 7.3431
Epoch [	7/	10]	d_loss: 0.3441	g_loss: 6.4853
Epoch [	8/	10]	d_loss: 0.3438	g_loss: 7.0936
Epoch [	8/	10]	d_loss: 0.3407	g_loss: 4.6086
Epoch [	8/	10]	d_loss: 0.3743	g_loss: 7.2942
Epoch [	8/	10]	d_loss: 0.3414	g_loss: 7.1283
Epoch [	8/	10]	d_loss: 0.3409	g_loss: 8.5029
Epoch [	8/	10]	d_loss: 0.3572	g_loss: 7.4163
Epoch [	8/	10]	d_loss: 0.4147	g_loss: 5.2285
Epoch [	8/	10]	d_loss: 0.3639	g_loss: 6.5076
Epoch [	8/	10]	d_loss: 0.3591	g_loss: 5.6451
Epoch [	8/	10]	d_loss: 0.3697	g_loss: 4.1899
Epoch [	8/	10]	d_loss: 0.3452	g_loss: 5.8120
Epoch [	8/	10]	d_loss: 0.3533	g_loss: 4.9759
Epoch [	8/	10]	d_loss: 0.3401	g_loss: 9.4403
Epoch [	8/	10]	d_loss: 0.3433	g_loss: 6.9106

```

Epoch [ 8/ 10] | d_loss: 0.3578 | g_loss: 7.5262
Epoch [ 9/ 10] | d_loss: 0.3662 | g_loss: 4.8227
Epoch [ 9/ 10] | d_loss: 0.3371 | g_loss: 7.2087
Epoch [ 9/ 10] | d_loss: 0.3613 | g_loss: 5.6620
Epoch [ 9/ 10] | d_loss: 0.3316 | g_loss: 5.9149
Epoch [ 9/ 10] | d_loss: 0.3393 | g_loss: 7.8292
Epoch [ 9/ 10] | d_loss: 0.4120 | g_loss: 5.3510
Epoch [ 9/ 10] | d_loss: 0.3720 | g_loss: 8.9643
Epoch [ 9/ 10] | d_loss: 0.3435 | g_loss: 7.1582
Epoch [ 9/ 10] | d_loss: 0.3868 | g_loss: 7.1928
Epoch [ 9/ 10] | d_loss: 0.3684 | g_loss: 5.5161
Epoch [ 9/ 10] | d_loss: 0.3634 | g_loss: 6.9893
Epoch [ 9/ 10] | d_loss: 0.4316 | g_loss: 5.6805
Epoch [ 9/ 10] | d_loss: 0.3396 | g_loss: 5.6590
Epoch [ 9/ 10] | d_loss: 0.3778 | g_loss: 5.5321
Epoch [ 9/ 10] | d_loss: 0.3597 | g_loss: 5.7376
Epoch [ 10/ 10] | d_loss: 0.3714 | g_loss: 5.2620
Epoch [ 10/ 10] | d_loss: 0.3610 | g_loss: 6.3998
Epoch [ 10/ 10] | d_loss: 0.3520 | g_loss: 8.4549
Epoch [ 10/ 10] | d_loss: 0.3461 | g_loss: 5.3475
Epoch [ 10/ 10] | d_loss: 0.3356 | g_loss: 8.1556
Epoch [ 10/ 10] | d_loss: 0.3610 | g_loss: 11.3547
Epoch [ 10/ 10] | d_loss: 0.3859 | g_loss: 4.6891
Epoch [ 10/ 10] | d_loss: 0.3511 | g_loss: 8.0025
Epoch [ 10/ 10] | d_loss: 0.3651 | g_loss: 8.3323
Epoch [ 10/ 10] | d_loss: 0.3385 | g_loss: 8.0880
Epoch [ 10/ 10] | d_loss: 0.3599 | g_loss: 5.4038
Epoch [ 10/ 10] | d_loss: 0.3517 | g_loss: 5.4750
Epoch [ 10/ 10] | d_loss: 0.3428 | g_loss: 4.7900
Epoch [ 10/ 10] | d_loss: 0.3462 | g_loss: 8.2563
Epoch [ 10/ 10] | d_loss: 0.3864 | g_loss: 4.4532

```

## 2.8 Training loss

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

```

In [27]: 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[27]: <matplotlib.legend.Legend at 0x7f8fbe50b780>

```



## 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 [28]: # 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 [29]: # Load samples from generator, taken while training
with open('train_samples.pkl', 'rb') as f:
    samples = pkl.load(f)

In [30]: _ = 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:** (Write your answer in this cell) The images I generated are quite blurred, there are some features like sharpness which was not really learned well the model and need some more fine tuning. I tried to replicate the model from here: [https://pytorch.org/tutorials/beginner/dcgan\\_faces\\_tutorial.html](https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html). And tuned the hyperparameters accordingly. Also, for the loss part I took the reference from here [https://github.com/udacity/deep-learning-v2-pytorch/blob/master/gan-mnist/MNIST\\_GAN\\_Solution.ipynb](https://github.com/udacity/deep-learning-v2-pytorch/blob/master/gan-mnist/MNIST_GAN_Solution.ipynb). the image size differs though it's 32 for our case. The dataset basically has celebrity faces that are mostly white. It would be great if you add up some more images with different shades of color of the skin which would have helped the model to generate more variety of images. Also, for this model we can work upon g\_loss part.

### 2.9.2 Submitting This Project

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

In [ ]: