

dlnd_face_generation

December 27, 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]

```
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 = 64
        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.

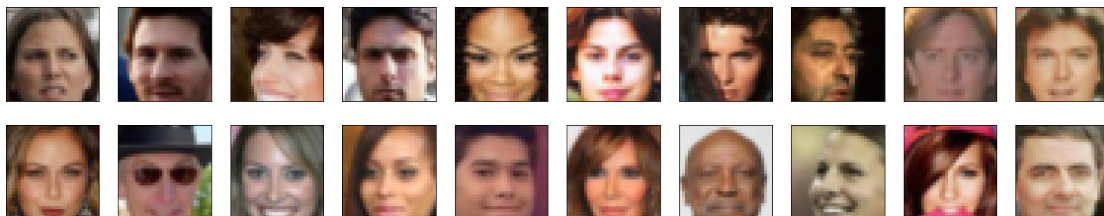
Note: You'll need to convert the Tensor images into a NumPy type and transpose the dimensions to correctly display an image, suggested `imshow` code is below, but it may not be perfect.

```
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(0.8431)
```

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 [15]: 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 [16]: import torch.optim as optim
    lr = 0.0004

    # Create optimizers for the discriminator D and generator G
    d_optimizer = optim.Adam(D.parameters(), lr, betas=(0.5, 0.999))
    g_optimizer = optim.Adam(G.parameters(), lr, betas=(0.5, 0.999))
```

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 [17]: 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 [18]: # set number of epochs
         n_epochs = 10

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

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

Epoch [ 1/ 10] | d_loss: 1.3948 | g_loss: 1.6494
Epoch [ 1/ 10] | d_loss: 1.0768 | g_loss: 2.7041
Epoch [ 1/ 10] | d_loss: 0.8074 | g_loss: 3.9108
Epoch [ 1/ 10] | d_loss: 0.9593 | g_loss: 1.6300
Epoch [ 1/ 10] | d_loss: 1.0442 | g_loss: 2.2061
Epoch [ 1/ 10] | d_loss: 1.3640 | g_loss: 2.2033
Epoch [ 1/ 10] | d_loss: 0.8451 | g_loss: 2.4473
Epoch [ 1/ 10] | d_loss: 0.8740 | g_loss: 2.5314
Epoch [ 1/ 10] | d_loss: 0.7426 | g_loss: 2.8875
Epoch [ 1/ 10] | d_loss: 1.1927 | g_loss: 3.8103
Epoch [ 1/ 10] | d_loss: 0.9192 | g_loss: 1.9919
Epoch [ 1/ 10] | d_loss: 0.8791 | g_loss: 3.0437
Epoch [ 1/ 10] | d_loss: 0.9925 | g_loss: 3.1782
Epoch [ 1/ 10] | d_loss: 1.3780 | g_loss: 1.6726
Epoch [ 1/ 10] | d_loss: 0.9660 | g_loss: 2.8086
Epoch [ 1/ 10] | d_loss: 1.1457 | g_loss: 4.3101
Epoch [ 1/ 10] | d_loss: 0.8480 | g_loss: 1.7683
Epoch [ 1/ 10] | d_loss: 0.8514 | g_loss: 2.0956
Epoch [ 1/ 10] | d_loss: 0.8712 | g_loss: 1.8510
Epoch [ 1/ 10] | d_loss: 0.9490 | g_loss: 2.0630
Epoch [ 1/ 10] | d_loss: 0.9331 | g_loss: 1.5721
Epoch [ 1/ 10] | d_loss: 0.9466 | g_loss: 1.7812
Epoch [ 1/ 10] | d_loss: 1.2732 | g_loss: 0.6861

```


Epoch [1/	10]	d_loss: 1.1117	g_loss: 1.4596
Epoch [1/	10]	d_loss: 0.9917	g_loss: 2.4055
Epoch [1/	10]	d_loss: 0.8477	g_loss: 2.8037
Epoch [1/	10]	d_loss: 1.1514	g_loss: 1.4696
Epoch [1/	10]	d_loss: 0.7607	g_loss: 2.4062
Epoch [1/	10]	d_loss: 0.9860	g_loss: 1.4720
Epoch [2/	10]	d_loss: 0.9669	g_loss: 2.2584
Epoch [2/	10]	d_loss: 0.9106	g_loss: 2.3000
Epoch [2/	10]	d_loss: 0.7053	g_loss: 2.3735
Epoch [2/	10]	d_loss: 0.8838	g_loss: 1.8600
Epoch [2/	10]	d_loss: 0.8234	g_loss: 2.3046
Epoch [2/	10]	d_loss: 0.9461	g_loss: 1.9375
Epoch [2/	10]	d_loss: 0.9064	g_loss: 1.8177
Epoch [2/	10]	d_loss: 0.8106	g_loss: 1.5522
Epoch [2/	10]	d_loss: 1.0916	g_loss: 3.0596
Epoch [2/	10]	d_loss: 0.7988	g_loss: 2.6337
Epoch [2/	10]	d_loss: 0.9647	g_loss: 1.4853
Epoch [2/	10]	d_loss: 0.9660	g_loss: 2.8543
Epoch [2/	10]	d_loss: 0.9871	g_loss: 3.1553
Epoch [2/	10]	d_loss: 0.8160	g_loss: 1.7869
Epoch [2/	10]	d_loss: 0.8990	g_loss: 1.5750
Epoch [2/	10]	d_loss: 1.1078	g_loss: 2.1997
Epoch [2/	10]	d_loss: 0.9941	g_loss: 1.4521
Epoch [2/	10]	d_loss: 0.7309	g_loss: 1.9579
Epoch [2/	10]	d_loss: 0.9214	g_loss: 1.8790
Epoch [2/	10]	d_loss: 0.9634	g_loss: 2.0536
Epoch [2/	10]	d_loss: 1.2036	g_loss: 3.5399
Epoch [2/	10]	d_loss: 0.7909	g_loss: 1.7505
Epoch [2/	10]	d_loss: 1.0389	g_loss: 1.8640
Epoch [2/	10]	d_loss: 1.0347	g_loss: 1.6714
Epoch [2/	10]	d_loss: 0.9076	g_loss: 1.7286
Epoch [2/	10]	d_loss: 0.7832	g_loss: 1.5593
Epoch [2/	10]	d_loss: 1.2808	g_loss: 0.9170
Epoch [2/	10]	d_loss: 0.8530	g_loss: 1.6360
Epoch [2/	10]	d_loss: 0.9175	g_loss: 2.5730
Epoch [3/	10]	d_loss: 1.0855	g_loss: 1.7512
Epoch [3/	10]	d_loss: 0.9380	g_loss: 0.9926
Epoch [3/	10]	d_loss: 0.8705	g_loss: 1.9934
Epoch [3/	10]	d_loss: 1.0196	g_loss: 1.1735
Epoch [3/	10]	d_loss: 0.9492	g_loss: 1.7710
Epoch [3/	10]	d_loss: 0.9908	g_loss: 2.8512
Epoch [3/	10]	d_loss: 0.8624	g_loss: 1.1071
Epoch [3/	10]	d_loss: 1.0704	g_loss: 1.5064
Epoch [3/	10]	d_loss: 1.2492	g_loss: 2.2555
Epoch [3/	10]	d_loss: 0.8687	g_loss: 1.8331
Epoch [3/	10]	d_loss: 0.7744	g_loss: 1.9813
Epoch [3/	10]	d_loss: 1.0694	g_loss: 2.3241
Epoch [3/	10]	d_loss: 0.9501	g_loss: 2.1507

Epoch [3/	10]	d_loss: 0.9079	g_loss: 1.4542
Epoch [3/	10]	d_loss: 0.8396	g_loss: 2.3526
Epoch [3/	10]	d_loss: 0.8649	g_loss: 2.1764
Epoch [3/	10]	d_loss: 0.9440	g_loss: 1.6923
Epoch [3/	10]	d_loss: 1.0764	g_loss: 1.5971
Epoch [3/	10]	d_loss: 0.7246	g_loss: 2.2434
Epoch [3/	10]	d_loss: 0.9138	g_loss: 1.2827
Epoch [3/	10]	d_loss: 1.0121	g_loss: 1.1322
Epoch [3/	10]	d_loss: 0.9316	g_loss: 1.9888
Epoch [3/	10]	d_loss: 0.7985	g_loss: 2.5791
Epoch [3/	10]	d_loss: 1.1003	g_loss: 2.0816
Epoch [3/	10]	d_loss: 1.1529	g_loss: 2.5114
Epoch [3/	10]	d_loss: 1.0519	g_loss: 1.3126
Epoch [3/	10]	d_loss: 0.8314	g_loss: 2.5690
Epoch [3/	10]	d_loss: 0.9327	g_loss: 2.0180
Epoch [3/	10]	d_loss: 0.9093	g_loss: 1.3668
Epoch [4/	10]	d_loss: 1.2971	g_loss: 2.2221
Epoch [4/	10]	d_loss: 1.1847	g_loss: 1.2460
Epoch [4/	10]	d_loss: 0.9958	g_loss: 2.3482
Epoch [4/	10]	d_loss: 1.1738	g_loss: 1.7269
Epoch [4/	10]	d_loss: 1.2064	g_loss: 3.2529
Epoch [4/	10]	d_loss: 0.9774	g_loss: 2.3558
Epoch [4/	10]	d_loss: 0.8532	g_loss: 2.2594
Epoch [4/	10]	d_loss: 0.9286	g_loss: 2.0116
Epoch [4/	10]	d_loss: 1.2068	g_loss: 1.3575
Epoch [4/	10]	d_loss: 0.9215	g_loss: 2.2745
Epoch [4/	10]	d_loss: 0.8721	g_loss: 2.2082
Epoch [4/	10]	d_loss: 1.0474	g_loss: 0.8848
Epoch [4/	10]	d_loss: 0.8365	g_loss: 2.1016
Epoch [4/	10]	d_loss: 1.0137	g_loss: 1.8597
Epoch [4/	10]	d_loss: 0.8830	g_loss: 1.5967
Epoch [4/	10]	d_loss: 0.9776	g_loss: 2.7065
Epoch [4/	10]	d_loss: 1.0181	g_loss: 1.2732
Epoch [4/	10]	d_loss: 1.0634	g_loss: 1.5133
Epoch [4/	10]	d_loss: 1.2239	g_loss: 1.1029
Epoch [4/	10]	d_loss: 1.0078	g_loss: 1.5219
Epoch [4/	10]	d_loss: 1.1529	g_loss: 2.7849
Epoch [4/	10]	d_loss: 0.7448	g_loss: 1.7276
Epoch [4/	10]	d_loss: 0.9330	g_loss: 1.3363
Epoch [4/	10]	d_loss: 1.0060	g_loss: 2.4845
Epoch [4/	10]	d_loss: 0.9087	g_loss: 1.9061
Epoch [4/	10]	d_loss: 1.0134	g_loss: 2.1096
Epoch [4/	10]	d_loss: 0.9457	g_loss: 1.8093
Epoch [4/	10]	d_loss: 0.8518	g_loss: 2.0812
Epoch [4/	10]	d_loss: 0.9108	g_loss: 1.7118
Epoch [5/	10]	d_loss: 1.3612	g_loss: 2.7395
Epoch [5/	10]	d_loss: 0.8720	g_loss: 1.6380
Epoch [5/	10]	d_loss: 0.9843	g_loss: 3.0217

Epoch [5/	10]	d_loss: 0.9514	g_loss: 1.8627
Epoch [5/	10]	d_loss: 0.8744	g_loss: 2.0548
Epoch [5/	10]	d_loss: 0.9176	g_loss: 1.6670
Epoch [5/	10]	d_loss: 1.0573	g_loss: 2.0779
Epoch [5/	10]	d_loss: 0.9782	g_loss: 1.8873
Epoch [5/	10]	d_loss: 1.0436	g_loss: 1.4012
Epoch [5/	10]	d_loss: 0.9123	g_loss: 1.1069
Epoch [5/	10]	d_loss: 1.1061	g_loss: 1.8227
Epoch [5/	10]	d_loss: 1.0998	g_loss: 2.0037
Epoch [5/	10]	d_loss: 0.8761	g_loss: 1.9303
Epoch [5/	10]	d_loss: 0.8165	g_loss: 1.6443
Epoch [5/	10]	d_loss: 1.0906	g_loss: 0.9819
Epoch [5/	10]	d_loss: 1.2677	g_loss: 0.9867
Epoch [5/	10]	d_loss: 0.7901	g_loss: 2.0496
Epoch [5/	10]	d_loss: 0.8999	g_loss: 1.5565
Epoch [5/	10]	d_loss: 0.8353	g_loss: 1.6795
Epoch [5/	10]	d_loss: 0.7695	g_loss: 1.7405
Epoch [5/	10]	d_loss: 0.9060	g_loss: 2.1673
Epoch [5/	10]	d_loss: 0.8655	g_loss: 1.2555
Epoch [5/	10]	d_loss: 1.3007	g_loss: 0.8010
Epoch [5/	10]	d_loss: 1.0979	g_loss: 2.1454
Epoch [5/	10]	d_loss: 1.1796	g_loss: 1.0809
Epoch [5/	10]	d_loss: 0.8930	g_loss: 1.1205
Epoch [5/	10]	d_loss: 0.8538	g_loss: 1.8881
Epoch [5/	10]	d_loss: 0.9830	g_loss: 2.2318
Epoch [5/	10]	d_loss: 0.8389	g_loss: 2.2827
Epoch [6/	10]	d_loss: 0.9754	g_loss: 1.1168
Epoch [6/	10]	d_loss: 0.9733	g_loss: 1.2540
Epoch [6/	10]	d_loss: 1.2579	g_loss: 2.7404
Epoch [6/	10]	d_loss: 1.1114	g_loss: 2.5328
Epoch [6/	10]	d_loss: 0.8285	g_loss: 2.1188
Epoch [6/	10]	d_loss: 0.9117	g_loss: 2.0313
Epoch [6/	10]	d_loss: 0.7756	g_loss: 2.6785
Epoch [6/	10]	d_loss: 0.9044	g_loss: 2.0113
Epoch [6/	10]	d_loss: 1.0321	g_loss: 1.3933
Epoch [6/	10]	d_loss: 0.7453	g_loss: 1.6269
Epoch [6/	10]	d_loss: 1.2410	g_loss: 1.2410
Epoch [6/	10]	d_loss: 1.3085	g_loss: 1.5307
Epoch [6/	10]	d_loss: 0.9213	g_loss: 3.0533
Epoch [6/	10]	d_loss: 0.7503	g_loss: 2.2891
Epoch [6/	10]	d_loss: 0.6195	g_loss: 1.9644
Epoch [6/	10]	d_loss: 0.8431	g_loss: 1.3764
Epoch [6/	10]	d_loss: 0.8548	g_loss: 1.9316
Epoch [6/	10]	d_loss: 0.9484	g_loss: 1.2550
Epoch [6/	10]	d_loss: 0.9404	g_loss: 1.5373
Epoch [6/	10]	d_loss: 0.9881	g_loss: 2.2734
Epoch [6/	10]	d_loss: 0.8321	g_loss: 1.7759
Epoch [6/	10]	d_loss: 0.9245	g_loss: 2.0895

Epoch [6/	10]	d_loss: 0.9039	g_loss: 1.4257
Epoch [6/	10]	d_loss: 0.7177	g_loss: 2.2871
Epoch [6/	10]	d_loss: 1.3706	g_loss: 0.9638
Epoch [6/	10]	d_loss: 1.0631	g_loss: 1.5321
Epoch [6/	10]	d_loss: 0.9055	g_loss: 3.3137
Epoch [6/	10]	d_loss: 0.9235	g_loss: 1.4158
Epoch [6/	10]	d_loss: 0.8900	g_loss: 2.1455
Epoch [7/	10]	d_loss: 2.0632	g_loss: 3.5148
Epoch [7/	10]	d_loss: 0.8493	g_loss: 2.1119
Epoch [7/	10]	d_loss: 0.6636	g_loss: 2.5023
Epoch [7/	10]	d_loss: 0.8945	g_loss: 1.5638
Epoch [7/	10]	d_loss: 0.9561	g_loss: 2.6242
Epoch [7/	10]	d_loss: 0.8261	g_loss: 2.0589
Epoch [7/	10]	d_loss: 1.0457	g_loss: 1.5512
Epoch [7/	10]	d_loss: 0.8708	g_loss: 2.0567
Epoch [7/	10]	d_loss: 0.8030	g_loss: 2.1245
Epoch [7/	10]	d_loss: 0.7316	g_loss: 1.9118
Epoch [7/	10]	d_loss: 1.1987	g_loss: 1.6804
Epoch [7/	10]	d_loss: 0.6057	g_loss: 2.2234
Epoch [7/	10]	d_loss: 0.7653	g_loss: 1.9206
Epoch [7/	10]	d_loss: 1.0960	g_loss: 2.7046
Epoch [7/	10]	d_loss: 1.0461	g_loss: 1.5116
Epoch [7/	10]	d_loss: 0.6943	g_loss: 1.9851
Epoch [7/	10]	d_loss: 1.1565	g_loss: 1.7910
Epoch [7/	10]	d_loss: 1.0646	g_loss: 1.6220
Epoch [7/	10]	d_loss: 0.7599	g_loss: 2.0011
Epoch [7/	10]	d_loss: 0.7493	g_loss: 1.6163
Epoch [7/	10]	d_loss: 0.7088	g_loss: 2.6692
Epoch [7/	10]	d_loss: 1.2151	g_loss: 1.6608
Epoch [7/	10]	d_loss: 0.7897	g_loss: 1.8464
Epoch [7/	10]	d_loss: 1.1815	g_loss: 2.1122
Epoch [7/	10]	d_loss: 1.3171	g_loss: 0.9147
Epoch [7/	10]	d_loss: 0.9301	g_loss: 2.2052
Epoch [7/	10]	d_loss: 0.9169	g_loss: 2.3204
Epoch [7/	10]	d_loss: 0.9070	g_loss: 1.2297
Epoch [7/	10]	d_loss: 0.8106	g_loss: 2.1023
Epoch [8/	10]	d_loss: 1.1497	g_loss: 3.0088
Epoch [8/	10]	d_loss: 0.6961	g_loss: 1.3878
Epoch [8/	10]	d_loss: 0.6584	g_loss: 2.5478
Epoch [8/	10]	d_loss: 0.6032	g_loss: 2.5313
Epoch [8/	10]	d_loss: 1.0421	g_loss: 1.9737
Epoch [8/	10]	d_loss: 1.3684	g_loss: 0.6272
Epoch [8/	10]	d_loss: 0.8917	g_loss: 1.4571
Epoch [8/	10]	d_loss: 0.9590	g_loss: 2.7211
Epoch [8/	10]	d_loss: 0.7889	g_loss: 2.1633
Epoch [8/	10]	d_loss: 1.3874	g_loss: 3.1819
Epoch [8/	10]	d_loss: 0.6736	g_loss: 1.8468
Epoch [8/	10]	d_loss: 0.6358	g_loss: 1.5106

Epoch [8/	10]	d_loss: 0.7493	g_loss: 1.5951
Epoch [8/	10]	d_loss: 1.1040	g_loss: 2.9449
Epoch [8/	10]	d_loss: 1.3012	g_loss: 2.0043
Epoch [8/	10]	d_loss: 0.7147	g_loss: 2.1059
Epoch [8/	10]	d_loss: 0.7713	g_loss: 2.9140
Epoch [8/	10]	d_loss: 0.5911	g_loss: 1.7512
Epoch [8/	10]	d_loss: 1.0599	g_loss: 2.7885
Epoch [8/	10]	d_loss: 0.9383	g_loss: 2.0400
Epoch [8/	10]	d_loss: 0.7437	g_loss: 1.6490
Epoch [8/	10]	d_loss: 0.9631	g_loss: 1.1954
Epoch [8/	10]	d_loss: 0.7239	g_loss: 2.1953
Epoch [8/	10]	d_loss: 0.9479	g_loss: 2.3019
Epoch [8/	10]	d_loss: 0.6340	g_loss: 2.5092
Epoch [8/	10]	d_loss: 0.9008	g_loss: 1.9137
Epoch [8/	10]	d_loss: 0.7829	g_loss: 1.8101
Epoch [8/	10]	d_loss: 0.6325	g_loss: 2.2810
Epoch [8/	10]	d_loss: 0.7352	g_loss: 2.2368
Epoch [9/	10]	d_loss: 1.1527	g_loss: 2.9975
Epoch [9/	10]	d_loss: 0.5474	g_loss: 2.5892
Epoch [9/	10]	d_loss: 0.8343	g_loss: 1.5584
Epoch [9/	10]	d_loss: 0.7694	g_loss: 2.0280
Epoch [9/	10]	d_loss: 0.8157	g_loss: 1.8087
Epoch [9/	10]	d_loss: 0.6010	g_loss: 1.6175
Epoch [9/	10]	d_loss: 0.8623	g_loss: 2.0904
Epoch [9/	10]	d_loss: 0.7609	g_loss: 1.5546
Epoch [9/	10]	d_loss: 0.7814	g_loss: 1.1856
Epoch [9/	10]	d_loss: 0.7260	g_loss: 1.8886
Epoch [9/	10]	d_loss: 0.7736	g_loss: 2.2396
Epoch [9/	10]	d_loss: 0.7328	g_loss: 2.2022
Epoch [9/	10]	d_loss: 1.0156	g_loss: 3.0174
Epoch [9/	10]	d_loss: 0.6935	g_loss: 1.7108
Epoch [9/	10]	d_loss: 0.6277	g_loss: 2.9179
Epoch [9/	10]	d_loss: 0.7642	g_loss: 2.0053
Epoch [9/	10]	d_loss: 0.7394	g_loss: 1.3220
Epoch [9/	10]	d_loss: 0.7783	g_loss: 1.9641
Epoch [9/	10]	d_loss: 0.6111	g_loss: 2.3977
Epoch [9/	10]	d_loss: 0.8223	g_loss: 1.1348
Epoch [9/	10]	d_loss: 0.8687	g_loss: 1.0894
Epoch [9/	10]	d_loss: 0.7389	g_loss: 2.8881
Epoch [9/	10]	d_loss: 0.6631	g_loss: 2.7575
Epoch [9/	10]	d_loss: 0.6162	g_loss: 1.6832
Epoch [9/	10]	d_loss: 0.5822	g_loss: 2.8105
Epoch [9/	10]	d_loss: 0.6862	g_loss: 1.9625
Epoch [9/	10]	d_loss: 0.7792	g_loss: 2.3020
Epoch [9/	10]	d_loss: 0.4846	g_loss: 3.9792
Epoch [9/	10]	d_loss: 0.8232	g_loss: 2.1081
Epoch [10/	10]	d_loss: 0.6750	g_loss: 2.3786
Epoch [10/	10]	d_loss: 0.7438	g_loss: 2.4901

```

Epoch [ 10/ 10] | d_loss: 0.7902 | g_loss: 3.5731
Epoch [ 10/ 10] | d_loss: 0.7647 | g_loss: 2.0438
Epoch [ 10/ 10] | d_loss: 0.5964 | g_loss: 1.8262
Epoch [ 10/ 10] | d_loss: 0.7856 | g_loss: 4.7687
Epoch [ 10/ 10] | d_loss: 0.6697 | g_loss: 2.3303
Epoch [ 10/ 10] | d_loss: 0.7038 | g_loss: 2.9517
Epoch [ 10/ 10] | d_loss: 0.6938 | g_loss: 3.3610
Epoch [ 10/ 10] | d_loss: 0.7254 | g_loss: 2.6644
Epoch [ 10/ 10] | d_loss: 0.8120 | g_loss: 2.9938
Epoch [ 10/ 10] | d_loss: 1.3546 | g_loss: 1.3083
Epoch [ 10/ 10] | d_loss: 0.6857 | g_loss: 1.9283
Epoch [ 10/ 10] | d_loss: 0.8688 | g_loss: 3.2819
Epoch [ 10/ 10] | d_loss: 0.8584 | g_loss: 2.2949
Epoch [ 10/ 10] | d_loss: 0.6232 | g_loss: 3.8898
Epoch [ 10/ 10] | d_loss: 0.8562 | g_loss: 1.6127
Epoch [ 10/ 10] | d_loss: 0.6300 | g_loss: 2.5275
Epoch [ 10/ 10] | d_loss: 1.6660 | g_loss: 0.8437
Epoch [ 10/ 10] | d_loss: 0.6552 | g_loss: 2.5744
Epoch [ 10/ 10] | d_loss: 0.8259 | g_loss: 1.8736
Epoch [ 10/ 10] | d_loss: 0.6817 | g_loss: 2.7600
Epoch [ 10/ 10] | d_loss: 0.7079 | g_loss: 1.6239
Epoch [ 10/ 10] | d_loss: 0.9297 | g_loss: 0.9884
Epoch [ 10/ 10] | d_loss: 0.8174 | g_loss: 2.2865
Epoch [ 10/ 10] | d_loss: 0.8270 | g_loss: 2.3938
Epoch [ 10/ 10] | d_loss: 0.8703 | g_loss: 3.0770
Epoch [ 10/ 10] | d_loss: 0.8793 | g_loss: 1.6143
Epoch [ 10/ 10] | d_loss: 0.6951 | g_loss: 2.2596

```

2.8 Training loss

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

```

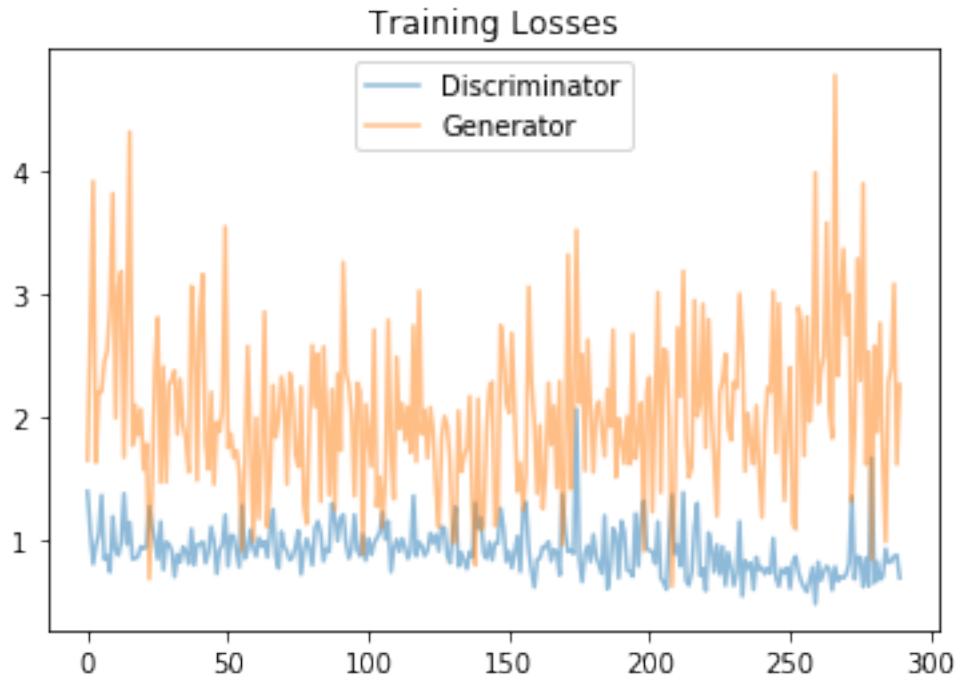
In [19]: fig, ax = plt.subplots()
         losses = np.array(losses)
         plt.plot(losses.T[0], label='Discriminator', alpha=0.5)
         plt.plot(losses.T[1], label='Generator', alpha=0.5)
         plt.title("Training Losses")
         plt.legend()

```

```

Out[19]: <matplotlib.legend.Legend at 0x7f8df004cbe0>

```



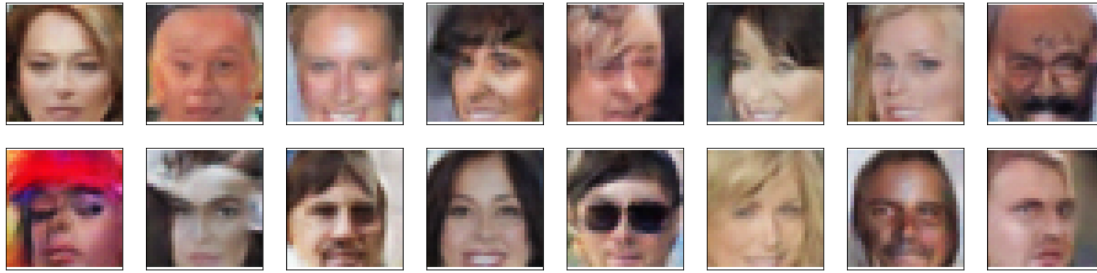
2.9 Generator samples from training

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

```
In [20]: # 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 [21]: # Load samples from generator, taken while training
with open('train_samples.pkl', 'rb') as f:
    samples = pkl.load(f)

In [22]: _ = 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. 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. 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. After taking inputs from my reviewer, I changed my batch_size from 128 to 64 which was resonable. And then I also need to reduce my learning rate accordingly. The DCGAN with this architectural structure remains stable with learning rate between 0.0001 and 0.0008. So, I changed the learning rate to 0.0004. Also, 0.5 seems to be the best choice here for beta1 as explained in this paper: <https://arxiv.org/pdf/1511.06434.pdf> and I kept the beta2 as the default value which is 0.999.

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 [25]: import os
import tarfile

def recursive_files(dir_name='.', ignore=None):
    for dir_name,subdirs,files in os.walk(dir_name):
        if ignore and os.path.basename(dir_name) in ignore:
            continue
```



```

        for file_name in files:
            if ignore and file_name in ignore:
                continue

            yield os.path.join(dir_name, file_name)

def make_tar_file(dir_name='.', target_file_name='workspace_archive.tar', ignore=None):
    tar = tarfile.open(target_file_name, 'w')

    for file_name in recursive_files(dir_name, ignore):
        tar.add(file_name)

    tar.close()

dir_name = '.'
target_file_name = 'workspace_archive.tar'
# List of files/directories to ignore
ignore = {'.ipynb_checkpoints', '__pycache__', 'processed_celeba_small.zip', 'processed

make_tar_file(dir_name, target_file_name, ignore)

```

In []: