

dlnd_face_generation

July 8, 2020

1 Face Generation

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

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

1.0.1 Get the Data

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

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

1.0.2 Pre-processed Data

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

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

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

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

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


[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
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
```

```
    transform = transforms.Compose([transforms.Resize(image_size), transforms.ToTensor()])
    image_dataset = datasets.ImageFolder(data_dir, transform)
    data_loader = torch.utils.data.DataLoader(image_dataset, batch_size = batch_size, sh

    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 = 256
        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.

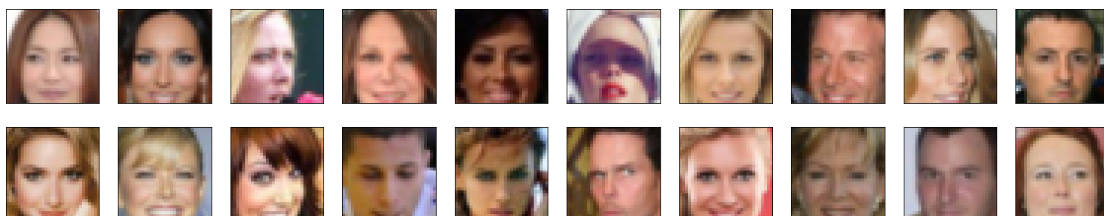
Note: You'll need to convert the Tensor images into a NumPy type and transpose the dimensions to correctly display an image, suggested `imshow` code is below, but it may not be perfect.

```
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
    x = (max-min)*x + min
    return x
```

```
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(-0.8588)

Max: tensor(1.)

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

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

            layers.append(conv_layer)

            if batch_norm:
                layers.append(nn.BatchNorm2d(out_channels))

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

In [10]: 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
            self.conv1 = conv(3, conv_dim, 4, batch_norm=False)
            self.conv2 = conv(conv_dim, conv_dim*2, 4)
            self.conv3 = conv(conv_dim*2, conv_dim*4, 4)
            self.conv4 = conv(conv_dim*4, conv_dim*8, 4)

            self.classifier = 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)

```

```

        out = out.view(-1, self.conv_dim*8*2*2)
        x = self.classifier(out)
        return x

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

```

Tests Passed

2.2 Generator

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

Exercise: Complete the Generator class

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

```

In [12]: def deconv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True):
    """Creates a transposed-convolutional layer, with optional batch normalization.
    """
    layers = []
    transpose_conv_layer = nn.ConvTranspose2d(in_channels, out_channels,
                                              kernel_size, stride, padding, bias=False)

    layers.append(transpose_conv_layer)

    if batch_norm:
        layers.append(nn.BatchNorm2d(out_channels))

    return nn.Sequential(*layers)

In [13]: 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.conv1 = deconv(conv_dim*8, conv_dim*4, 4)
self.conv2 = deconv(conv_dim*4, conv_dim*2, 4)
self.conv3 = deconv(conv_dim*2, conv_dim, 4)
self.conv4 = deconv(conv_dim, 3, 4, batch_norm=False)

def forward(self, x):
    """
    Forward propagation of the neural network
    :param x: The input to the neural network
    :return: A 32x32x3 Tensor image as output
    """
    # define feedforward behavior
    out = self.fc(x)
    out = out.view(-1, self.conv_dim*8, 2, 2)

    out = F.relu(self.conv1(out))
    out = F.relu(self.conv2(out))
    out = F.relu(self.conv3(out))

    out = self.conv4(out)
    x = F.tanh(out)

    return x

    """
    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 [14]: 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') or classname.find('Linear') != -1:
            m.weight.data.normal_(0.0, 0.02)
            if hasattr(m, 'bias') and m.bias is not None:
                m.bias.data.fill_(0)
```

2.4 Build complete network

Define your models' hyperparameters and instantiate the discriminator and generator from the classes defined above. Make sure you've passed in the correct input arguments.

```
In [15]: """
        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 [16]: # 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)
```

```
    )
```

```
    (classifier): Linear(in_features=2048, out_features=1, bias=True)
```

```
)
```

```
Generator(
```

```
    (fc): Linear(in_features=100, out_features=2048, bias=True)
```

```
    (conv1): 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)
```

```
    )
```

```
    (conv2): 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)
```

```
    )
```

```
    (conv3): 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)
```

```
    )
```

```
    (conv4): 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 [17]: """
          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 [18]: def real_loss(D_out):
          '''Calculates how close discriminator outputs are to being real.
          param, D_out: discriminator logits
```

```

        return: real loss'''

    batch_size = D_out.size(0)

    labels = torch.ones(batch_size)*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)
    if train_on_gpu:
        labels = labels.cuda()

    criterion = nn.BCEWithLogitsLoss()
    loss = criterion(D_out.squeeze(), labels)
    return loss

```

2.6 Optimizers

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

In [20]: `import torch.optim as optim`

```

lr = 0.0002
beta1=0.5
beta2=0.999

# Create optimizers for the discriminator D and generator G
d_optimizer = optim.Adam(D.parameters(), lr, [beta1, beta2])
g_optimizer = optim.Adam(G.parameters(), lr, [beta1, beta2])

```

2.7 Training

Training will involve alternating between training the discriminator and the generator. You'll use your functions `real_loss` and `fake_loss` to help you calculate the discriminator losses.

- You should train the discriminator by alternating on real and fake images
- Then the generator, which tries to trick the discriminator and should have an opposing loss function

Saving Samples You've been given some code to print out some loss statistics and save some generated "fake" samples.

Exercise: Complete the training function Keep in mind that, if you've moved your models to GPU, you'll also have to move any model inputs to GPU.

```
In [21]: def train(D, G, n_epochs, print_every=50):
    '''Trains adversarial networks for some number of epochs
    param, D: the discriminator network
    param, G: the generator network
    param, n_epochs: number of epochs to train for
    param, print_every: when to print and record the models' losses
    return: D and G losses'''

    # move models to GPU
    if train_on_gpu:
        D.cuda()
        G.cuda()

    # keep track of loss and generated, "fake" samples
    samples = []
    losses = []

    # Get some fixed data for sampling. These are images that are held
    # constant throughout training, and allow us to inspect the model's performance
    sample_size=16
    fixed_z = np.random.uniform(-1, 1, size=(sample_size, z_size))
    fixed_z = torch.from_numpy(fixed_z).float()
    # move z to GPU if available
    if train_on_gpu:
        fixed_z = fixed_z.cuda()

    # epoch training loop
    for epoch in range(n_epochs):

        # batch training loop
        for batch_i, (real_images, _) in enumerate(celeba_train_loader):

            batch_size = real_images.size(0)
            real_images = scale(real_images)

            # =====
            #          YOUR CODE HERE: TRAIN THE NETWORKS
```

```

# =====

# 1. Train the discriminator on real and fake images
d_optimizer.zero_grad()
if train_on_gpu:
    real_images = real_images.cuda()

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

z = np.random.uniform(-1, 1, size = (batch_size, z_size))
z = torch.from_numpy(z).float()
if train_on_gpu:
    z = z.cuda()

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

d_loss = d_real_loss + d_fake_loss
d_loss.backward()
d_optimizer.step()
# 2. Train the generator with an adversarial loss
g_optimizer.zero_grad()

z = np.random.uniform(-1, 1, size = (batch_size, z_size))
z = torch.from_numpy(z).float()
if train_on_gpu:
    z = z.cuda()

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

g_loss.backward()
g_optimizer.step()

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

# Print some loss stats
if batch_i % print_every == 0:
    # append discriminator loss and generator loss
    losses.append((d_loss.item(), g_loss.item()))
    # print discriminator and generator loss
    print('Epoch [{:5d}/{:5d}] | d_loss: {:.64f} | g_loss: {:.64f}'.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 [22]: # set number of epochs
         n_epochs = 20

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

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

Epoch [ 1/ 20] | d_loss: 1.3864 | g_loss: 0.7033
Epoch [ 1/ 20] | d_loss: 0.6597 | g_loss: 1.3549
Epoch [ 1/ 20] | d_loss: 0.6907 | g_loss: 1.5369
Epoch [ 1/ 20] | d_loss: 1.4717 | g_loss: 0.9373
Epoch [ 1/ 20] | d_loss: 0.8921 | g_loss: 1.3614
Epoch [ 1/ 20] | d_loss: 1.3341 | g_loss: 1.0616
Epoch [ 1/ 20] | d_loss: 1.3893 | g_loss: 1.5188
Epoch [ 1/ 20] | d_loss: 0.8384 | g_loss: 1.8915
Epoch [ 2/ 20] | d_loss: 0.8279 | g_loss: 1.7958
Epoch [ 2/ 20] | d_loss: 1.1749 | g_loss: 2.7044
Epoch [ 2/ 20] | d_loss: 0.9474 | g_loss: 2.3451
Epoch [ 2/ 20] | d_loss: 1.1527 | g_loss: 2.6467
Epoch [ 2/ 20] | d_loss: 1.0646 | g_loss: 2.0390
Epoch [ 2/ 20] | d_loss: 0.9155 | g_loss: 2.4012
Epoch [ 2/ 20] | d_loss: 0.9270 | g_loss: 2.2659
Epoch [ 2/ 20] | d_loss: 0.9299 | g_loss: 1.9765
Epoch [ 3/ 20] | d_loss: 0.9501 | g_loss: 1.6112
Epoch [ 3/ 20] | d_loss: 0.9577 | g_loss: 1.7272

```


Epoch [3/	20]	d_loss: 0.8875	g_loss: 2.4082
Epoch [3/	20]	d_loss: 0.8967	g_loss: 1.9506
Epoch [3/	20]	d_loss: 0.8767	g_loss: 2.2992
Epoch [3/	20]	d_loss: 1.3912	g_loss: 3.5839
Epoch [3/	20]	d_loss: 0.9523	g_loss: 2.0185
Epoch [3/	20]	d_loss: 0.9825	g_loss: 2.8159
Epoch [4/	20]	d_loss: 0.9431	g_loss: 2.3975
Epoch [4/	20]	d_loss: 0.9103	g_loss: 2.5782
Epoch [4/	20]	d_loss: 0.8600	g_loss: 2.1430
Epoch [4/	20]	d_loss: 1.0067	g_loss: 2.2828
Epoch [4/	20]	d_loss: 0.9660	g_loss: 2.3855
Epoch [4/	20]	d_loss: 0.9455	g_loss: 2.3270
Epoch [4/	20]	d_loss: 1.0242	g_loss: 3.8285
Epoch [4/	20]	d_loss: 0.8374	g_loss: 2.0971
Epoch [5/	20]	d_loss: 0.8933	g_loss: 2.0292
Epoch [5/	20]	d_loss: 1.0458	g_loss: 3.3985
Epoch [5/	20]	d_loss: 0.9222	g_loss: 1.8243
Epoch [5/	20]	d_loss: 0.7567	g_loss: 2.9879
Epoch [5/	20]	d_loss: 1.2006	g_loss: 3.0545
Epoch [5/	20]	d_loss: 0.8821	g_loss: 2.5465
Epoch [5/	20]	d_loss: 0.9386	g_loss: 1.5161
Epoch [5/	20]	d_loss: 0.9135	g_loss: 2.2428
Epoch [6/	20]	d_loss: 0.8680	g_loss: 2.1946
Epoch [6/	20]	d_loss: 1.0490	g_loss: 1.5994
Epoch [6/	20]	d_loss: 0.9722	g_loss: 1.8113
Epoch [6/	20]	d_loss: 0.8675	g_loss: 2.9207
Epoch [6/	20]	d_loss: 1.0543	g_loss: 1.3391
Epoch [6/	20]	d_loss: 1.0134	g_loss: 1.7103
Epoch [6/	20]	d_loss: 0.6628	g_loss: 3.0019
Epoch [6/	20]	d_loss: 0.9967	g_loss: 2.5393
Epoch [7/	20]	d_loss: 0.9507	g_loss: 1.7183
Epoch [7/	20]	d_loss: 1.0022	g_loss: 1.4839
Epoch [7/	20]	d_loss: 0.9599	g_loss: 2.7764
Epoch [7/	20]	d_loss: 0.7887	g_loss: 2.0450
Epoch [7/	20]	d_loss: 0.8877	g_loss: 1.9552
Epoch [7/	20]	d_loss: 0.7817	g_loss: 2.2201
Epoch [7/	20]	d_loss: 0.9147	g_loss: 1.6934
Epoch [7/	20]	d_loss: 1.0851	g_loss: 0.9659
Epoch [8/	20]	d_loss: 0.7836	g_loss: 2.5120
Epoch [8/	20]	d_loss: 0.8942	g_loss: 2.0857
Epoch [8/	20]	d_loss: 1.1823	g_loss: 3.8538
Epoch [8/	20]	d_loss: 0.8995	g_loss: 2.3304
Epoch [8/	20]	d_loss: 1.3191	g_loss: 1.1175
Epoch [8/	20]	d_loss: 0.9104	g_loss: 1.3774
Epoch [8/	20]	d_loss: 1.0744	g_loss: 3.1075
Epoch [8/	20]	d_loss: 0.8763	g_loss: 2.3564
Epoch [9/	20]	d_loss: 0.8837	g_loss: 2.1829
Epoch [9/	20]	d_loss: 0.8056	g_loss: 2.0879

Epoch [9/	20]	d_loss: 0.9406	g_loss: 2.1567
Epoch [9/	20]	d_loss: 0.8465	g_loss: 2.8899
Epoch [9/	20]	d_loss: 0.9440	g_loss: 2.3125
Epoch [9/	20]	d_loss: 1.0174	g_loss: 2.4646
Epoch [9/	20]	d_loss: 0.9331	g_loss: 2.5347
Epoch [9/	20]	d_loss: 0.9105	g_loss: 2.7108
Epoch [10/	20]	d_loss: 0.9829	g_loss: 3.3116
Epoch [10/	20]	d_loss: 0.7657	g_loss: 2.0601
Epoch [10/	20]	d_loss: 1.0136	g_loss: 1.9130
Epoch [10/	20]	d_loss: 0.8751	g_loss: 2.4060
Epoch [10/	20]	d_loss: 1.0189	g_loss: 1.0385
Epoch [10/	20]	d_loss: 0.8880	g_loss: 3.1827
Epoch [10/	20]	d_loss: 1.0245	g_loss: 3.8835
Epoch [10/	20]	d_loss: 1.3779	g_loss: 3.7298
Epoch [11/	20]	d_loss: 0.8869	g_loss: 2.4939
Epoch [11/	20]	d_loss: 0.8584	g_loss: 2.8206
Epoch [11/	20]	d_loss: 0.7157	g_loss: 1.9733
Epoch [11/	20]	d_loss: 0.9955	g_loss: 3.4021
Epoch [11/	20]	d_loss: 0.9927	g_loss: 1.6526
Epoch [11/	20]	d_loss: 0.8762	g_loss: 1.6408
Epoch [11/	20]	d_loss: 1.3608	g_loss: 2.6946
Epoch [11/	20]	d_loss: 0.8410	g_loss: 2.1167
Epoch [12/	20]	d_loss: 0.8861	g_loss: 1.4496
Epoch [12/	20]	d_loss: 0.9533	g_loss: 2.5393
Epoch [12/	20]	d_loss: 0.8750	g_loss: 2.3195
Epoch [12/	20]	d_loss: 0.8682	g_loss: 1.6609
Epoch [12/	20]	d_loss: 0.9162	g_loss: 2.1389
Epoch [13/	20]	d_loss: 0.8783	g_loss: 2.5385
Epoch [13/	20]	d_loss: 1.0065	g_loss: 1.3929
Epoch [13/	20]	d_loss: 0.9653	g_loss: 2.0237
Epoch [13/	20]	d_loss: 0.8915	g_loss: 1.6056
Epoch [13/	20]	d_loss: 0.8253	g_loss: 1.7009
Epoch [13/	20]	d_loss: 1.0406	g_loss: 1.5626
Epoch [13/	20]	d_loss: 0.7383	g_loss: 2.4248
Epoch [13/	20]	d_loss: 1.1656	g_loss: 4.3934
Epoch [14/	20]	d_loss: 0.9189	g_loss: 3.0985
Epoch [14/	20]	d_loss: 0.9286	g_loss: 2.8823
Epoch [14/	20]	d_loss: 0.8690	g_loss: 1.7336
Epoch [14/	20]	d_loss: 1.2637	g_loss: 3.9985
Epoch [14/	20]	d_loss: 0.7875	g_loss: 1.8116
Epoch [14/	20]	d_loss: 0.8293	g_loss: 2.0744
Epoch [14/	20]	d_loss: 0.9154	g_loss: 1.4625
Epoch [14/	20]	d_loss: 0.9810	g_loss: 2.0203
Epoch [15/	20]	d_loss: 0.8600	g_loss: 1.8336
Epoch [15/	20]	d_loss: 0.7573	g_loss: 2.1040
Epoch [15/	20]	d_loss: 0.8822	g_loss: 2.4342
Epoch [15/	20]	d_loss: 0.7796	g_loss: 2.4214
Epoch [15/	20]	d_loss: 0.9916	g_loss: 1.1974

Epoch [15/	20]		d_loss: 0.8675		g_loss: 2.0871
Epoch [15/	20]		d_loss: 0.8769		g_loss: 2.5197
Epoch [15/	20]		d_loss: 0.9246		g_loss: 2.4413
Epoch [16/	20]		d_loss: 0.7727		g_loss: 2.3268
Epoch [16/	20]		d_loss: 0.8374		g_loss: 2.4033
Epoch [16/	20]		d_loss: 0.8716		g_loss: 1.6029
Epoch [16/	20]		d_loss: 0.7712		g_loss: 1.6855
Epoch [16/	20]		d_loss: 1.5011		g_loss: 3.5838
Epoch [16/	20]		d_loss: 0.7666		g_loss: 1.8799
Epoch [16/	20]		d_loss: 1.0451		g_loss: 3.3615
Epoch [16/	20]		d_loss: 0.8351		g_loss: 1.8880
Epoch [17/	20]		d_loss: 0.8923		g_loss: 1.2893
Epoch [17/	20]		d_loss: 0.7693		g_loss: 2.5531
Epoch [17/	20]		d_loss: 0.9128		g_loss: 2.0762
Epoch [17/	20]		d_loss: 1.2333		g_loss: 3.2861
Epoch [17/	20]		d_loss: 0.9197		g_loss: 1.3869
Epoch [17/	20]		d_loss: 1.1637		g_loss: 1.2628
Epoch [17/	20]		d_loss: 0.7449		g_loss: 2.0741
Epoch [17/	20]		d_loss: 0.7893		g_loss: 2.1157
Epoch [18/	20]		d_loss: 0.9332		g_loss: 3.1243
Epoch [18/	20]		d_loss: 1.2740		g_loss: 4.1060
Epoch [18/	20]		d_loss: 0.7132		g_loss: 2.5374
Epoch [18/	20]		d_loss: 0.8921		g_loss: 1.6716
Epoch [18/	20]		d_loss: 0.8232		g_loss: 1.4164
Epoch [18/	20]		d_loss: 0.8254		g_loss: 2.5703
Epoch [18/	20]		d_loss: 1.0060		g_loss: 1.5270
Epoch [18/	20]		d_loss: 0.7394		g_loss: 2.3146
Epoch [19/	20]		d_loss: 0.8967		g_loss: 3.3604
Epoch [19/	20]		d_loss: 0.9354		g_loss: 2.6435
Epoch [19/	20]		d_loss: 0.8164		g_loss: 1.8524
Epoch [19/	20]		d_loss: 0.8925		g_loss: 1.6977
Epoch [19/	20]		d_loss: 0.9006		g_loss: 1.8906
Epoch [19/	20]		d_loss: 0.7375		g_loss: 2.5834
Epoch [19/	20]		d_loss: 0.9857		g_loss: 3.2732
Epoch [19/	20]		d_loss: 0.8860		g_loss: 2.1955
Epoch [20/	20]		d_loss: 0.7182		g_loss: 2.6756
Epoch [20/	20]		d_loss: 0.6548		g_loss: 2.8317
Epoch [20/	20]		d_loss: 0.6314		g_loss: 2.3337
Epoch [20/	20]		d_loss: 0.9339		g_loss: 1.3754
Epoch [20/	20]		d_loss: 0.9451		g_loss: 1.3578
Epoch [20/	20]		d_loss: 0.8174		g_loss: 2.4458
Epoch [20/	20]		d_loss: 0.7719		g_loss: 1.9192
Epoch [20/	20]		d_loss: 0.7564		g_loss: 1.8983

2.8 Training loss

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

```
In [23]: 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[23]: <matplotlib.legend.Legend at 0x7fdbec2d2c88>
```



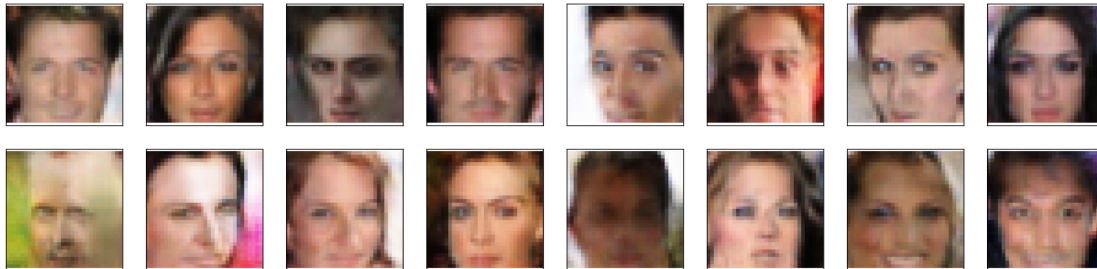
2.9 Generator samples from training

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

```
In [24]: # 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 [25]: # Load samples from generator, taken while training
         with open('train_samples.pkl', 'rb') as f:
             samples = pickle.load(f)
```

```
In [26]: _ = 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) As mentioned in above first point that dataset is biased, the generated images are not so clear. Some features are recognizable, like eyes, lips, nose, etc, but still image is not so clear. Generated faces are mostly white. The low resolution images makes harder for CNN to identify features. This can be improved by increasing network depth but it will take more training time. On emore way to improve quality to have more diverse dataset, and hyper parameter tuning. Which can be clearly seen ny changing batch_size, learning rate , z_size and conv_dim.

As currently i have used Adam optimizer, we can check with RMSprop optimizer and discriminator output with sigmoid and binary cross entropy to get more realistic images. since GAN contains two separately trained networks its convergence is hard to identify. RMSProp as an optimizer generates more realistic fake images compared to Adam for this case.

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.