

dlnd_face_generation

February 23, 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 [ ]: # 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]

```

inflating: processed_celeba_small/celeba/New Folder With Items/051733.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051734.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051735.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051736.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051737.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051738.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051739.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051740.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051741.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051742.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051743.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051744.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051745.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051746.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051747.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051748.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051749.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051750.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051751.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051752.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051753.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051754.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051755.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051756.jpg
inflating: processed_celeba_small/celeba/New Folder With Items/051757.jpg
inflating: processed_celeba_small/

```

```
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

    # resize and normalize the images
    my_transformation = transforms.Compose([
        transforms.Resize(image_size),
        transforms.ToTensor()
    ])

    # define dataset using ImageFolder
    my_dataset = datasets.ImageFolder(data_dir,
                                     transform = my_transformation)

    # create and return DataLoader
    data_loader = torch.utils.data.DataLoader(dataset = my_dataset,
                                              batch_size = batch_size,
                                              shuffle = True)
```



```

return data_loader
#return None

```

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 [30]: # 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.

```

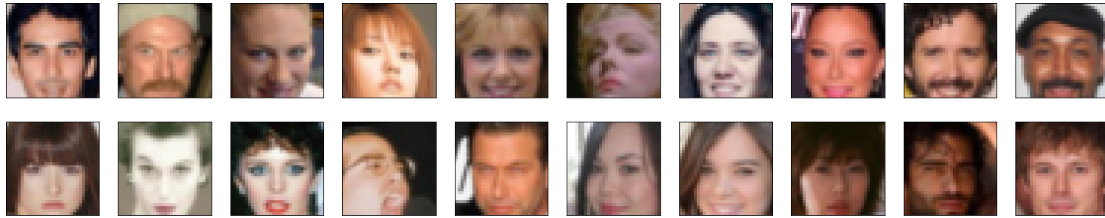
In [5]: # helper display function
def imshow(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))

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

```
In [9]: """
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""

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

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

```
Min: tensor(-1.)
Max: tensor(0.9137)
```

2 Define the Model

A GAN is comprised of two adversarial networks, a discriminator and a generator.

2.1 Discriminator

Your first task will be to define the discriminator. This is a convolutional classifier like you've built before, only without any maxpooling layers. To deal with this complex data, it's suggested you use a deep network with **normalization**. You are also allowed to create any helper functions that may be useful.

Exercise: Complete the Discriminator class

- The inputs to the discriminator are 32x32x3 tensor images
- The output should be a single value that will indicate whether a given image is real or fake

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

# Based on https://github.com/udacity/deep-learning-v2-pytorch/tree/master/cycle-gan
# helper conv function
def conv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True):
    """Creates a convolutional layer, with optional batch normalization.
    """
    layers = []
    conv_layer = nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                           kernel_size=kernel_size, stride=stride, padding=padding, bias=True)

    layers.append(conv_layer)

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

In [11]: 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
        # 32x32 input
        self.conv1 = conv(3, conv_dim, 4, batch_norm=False) # first layer, no batch_norm
        # 16x16 out
        self.conv2 = conv(conv_dim, conv_dim*2, 4)
        # 8x8 out
        self.conv3 = conv(conv_dim*2, conv_dim*4, 4)
        # 4x4 out
```

```

# final, fully-connected layer
self.fc = nn.Linear(conv_dim*4*4*4, 1) # depth=conv_dim*4, image=4*4

# dropout layer
self.dropout = nn.Dropout(0.3)

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

    # all hidden layers + leaky relu activation
    # applying a dropout layer in-between each of linear layers to ensure
    # that the network is likely to train each node evenly
    x = F.leaky_relu(self.conv1(x), 0.2)
    x = self.dropout(x)
    x = F.leaky_relu(self.conv2(x), 0.2)
    x = self.dropout(x)
    x = F.leaky_relu(self.conv3(x), 0.2)
    x = self.dropout(x)

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

    # final output layer
    x = self.fc(x)
    return x

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

tests.test_discriminator(Discriminator)

```

Tests Passed

2.2 Generator

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

Exercise: Complete the Generator class

- The inputs to the generator are vectors of some length `z_size`

- The output should be a image of shape 32x32x3

```
In [12]: # helper deconv function
def deconv(in_channels, out_channels, kernel_size, stride=2, padding=1, batch_norm=True):
    """Creates a transposed-convolutional layer, with optional batch normalization.
    """
    # create a sequence of transpose + optional batch norm layers
    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)

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

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

        # transpose conv layers
        self.t_conv1 = deconv(conv_dim*4, conv_dim*2, 4)
        self.t_conv2 = deconv(conv_dim*2, conv_dim, 4)
        self.t_conv3 = 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
        x = self.fc(x) # fc(x)--> x here is from a forward function. It's a z-vector
```

```

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

# hidden transpose conv layers + relu
x = F.relu(self.t_conv1(x))
x = F.relu(self.t_conv2(x))

# last layer + tanh activation
x = self.t_conv3(x)
x = F.tanh(x)

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]: from torch.nn import init
def weights_init_normal(m):
    """
    Applies initial weights to certain layers in a model .
    The weights are taken from a normal distribution
    with mean = 0, std dev = 0.02.
    :param m: A module or layer in a network
    """
    # classname will be something like:
    # `Conv`, `BatchNorm2d`, `Linear`, etc.
    classname = m.__class__.__name__

```

```

# TODO: Apply initial weights to convolutional and linear layers
init_gain=0.02
if hasattr(m, 'weight') and (classname.find('Conv') != -1 or classname.find('Linear') != -1):
    init.normal_(m.weight.data, 0.0, init_gain)
    if hasattr(m, 'bias') and m.bias is not None:
        init.constant_(m.bias.data, 0.0)
elif classname.find('BatchNorm2d') != -1: # BatchNorm Layer's weight is not a matrix
    init.normal_(m.weight.data, 1.0, init_gain)
    init.constant_(m.bias.data, 0.0)

```

2.4 Build complete network

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

```

In [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 = 32
g_conv_dim = 32
z_size = 100

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
D, G = build_network(d_conv_dim, g_conv_dim, z_size)

```

```

Discriminator(
  (conv1): Sequential(

```

```

        (0): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    )
    (conv2): Sequential(
      (0): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv3): Sequential(
      (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (fc): Linear(in_features=2048, out_features=1, bias=True)
    (dropout): Dropout(p=0.3)
  )

Generator(
  (fc): Linear(in_features=100, out_features=2048, bias=True)
  (t_conv1): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (t_conv2): Sequential(
    (0): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (t_conv3): Sequential(
    (0): ConvTranspose2d(32, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
)

```

2.4.1 Training on GPU

Check if you can train on GPU. Here, we'll set this as a boolean variable `train_on_gpu`. Later, you'll be responsible for making sure that >* Models, * Model inputs, and * Loss function arguments

Are moved to GPU, where appropriate.

```

In [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 [22]: 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)
    # label smoothing
    if smooth:
        # smooth, real labels = 0.9
        labels = torch.ones(batch_size)*0.9
    else:
        labels = torch.ones(batch_size) # real labels = 1
    # move labels to GPU if available
    if train_on_gpu:
        labels = labels.cuda()
    # binary cross entropy with logits loss
    criterion = nn.BCEWithLogitsLoss()
    # calculate loss
    loss = criterion(D_out.squeeze(), labels)
    return loss

def fake_loss(D_out):
    '''Calculates how close discriminator outputs are to being fake.
    param, D_out: discriminator logits
    return: fake loss'''
```

```

batch_size = D_out.size(0)
labels = torch.zeros(batch_size) # fake labels = 0
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 [23]: `import torch.optim as optim`

```

# params
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 [24]: 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)
        # no smoothing as it was found to work a little better without smoothing

        # 2. Train the generator with an adversarial loss

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

```

```

z = torch.from_numpy(z).float()
# move x to GPU, if available
if train_on_gpu:
    z = z.cuda()
fake_images = G(z)

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

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

# =====
#                      TRAIN THE GENERATOR
# =====
g_optimizer.zero_grad()

# 1. Train with fake images and flipped labels

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

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

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

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

# Print some loss stats
if batch_i % print_every == 0:
    # append discriminator loss and generator loss
    losses.append((d_loss.item(), g_loss.item()))
    # print discriminator and generator loss

```

```

        print('Epoch [{:5d}/{:5d}] | d_loss: {:.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 [25]: # 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: 1.4700 | g_loss: 0.8880
Epoch [ 1/ 10] | d_loss: 0.4251 | g_loss: 2.4885
Epoch [ 1/ 10] | d_loss: 0.1570 | g_loss: 4.3911
Epoch [ 1/ 10] | d_loss: 0.2400 | g_loss: 4.7571
Epoch [ 1/ 10] | d_loss: 0.7249 | g_loss: 6.4529
Epoch [ 1/ 10] | d_loss: 0.5930 | g_loss: 4.7274
Epoch [ 1/ 10] | d_loss: 0.3949 | g_loss: 3.0627
Epoch [ 1/ 10] | d_loss: 0.7211 | g_loss: 2.1344
Epoch [ 1/ 10] | d_loss: 0.6956 | g_loss: 2.9010
Epoch [ 1/ 10] | d_loss: 0.6042 | g_loss: 2.7611
Epoch [ 1/ 10] | d_loss: 0.6038 | g_loss: 2.3712
Epoch [ 1/ 10] | d_loss: 0.7102 | g_loss: 2.0412
Epoch [ 1/ 10] | d_loss: 0.8699 | g_loss: 2.5723
Epoch [ 1/ 10] | d_loss: 1.3123 | g_loss: 1.0970
Epoch [ 1/ 10] | d_loss: 0.6211 | g_loss: 1.8869
Epoch [ 1/ 10] | d_loss: 0.9560 | g_loss: 1.3247
Epoch [ 1/ 10] | d_loss: 0.8523 | g_loss: 1.6930

```

Epoch [1/	10]	d_loss: 0.9443	g_loss: 1.3997
Epoch [1/	10]	d_loss: 0.8191	g_loss: 1.4714
Epoch [1/	10]	d_loss: 1.0086	g_loss: 1.8679
Epoch [1/	10]	d_loss: 1.1239	g_loss: 1.4826
Epoch [1/	10]	d_loss: 1.0754	g_loss: 1.1064
Epoch [1/	10]	d_loss: 1.1550	g_loss: 1.5362
Epoch [1/	10]	d_loss: 0.8660	g_loss: 1.2419
Epoch [1/	10]	d_loss: 1.1333	g_loss: 1.8620
Epoch [1/	10]	d_loss: 0.9600	g_loss: 1.4308
Epoch [1/	10]	d_loss: 1.1397	g_loss: 1.3881
Epoch [1/	10]	d_loss: 1.2638	g_loss: 1.3908
Epoch [1/	10]	d_loss: 1.0854	g_loss: 1.0093
Epoch [2/	10]	d_loss: 1.3366	g_loss: 0.8266
Epoch [2/	10]	d_loss: 0.9802	g_loss: 0.8689
Epoch [2/	10]	d_loss: 1.0359	g_loss: 1.4364
Epoch [2/	10]	d_loss: 1.2820	g_loss: 1.0301
Epoch [2/	10]	d_loss: 1.0509	g_loss: 1.5146
Epoch [2/	10]	d_loss: 1.3702	g_loss: 1.4672
Epoch [2/	10]	d_loss: 0.9660	g_loss: 1.1944
Epoch [2/	10]	d_loss: 1.1003	g_loss: 1.3726
Epoch [2/	10]	d_loss: 1.2946	g_loss: 1.2498
Epoch [2/	10]	d_loss: 1.2333	g_loss: 1.2649
Epoch [2/	10]	d_loss: 0.9968	g_loss: 1.0619
Epoch [2/	10]	d_loss: 1.2614	g_loss: 1.4217
Epoch [2/	10]	d_loss: 1.3446	g_loss: 0.9185
Epoch [2/	10]	d_loss: 1.2176	g_loss: 1.1696
Epoch [2/	10]	d_loss: 1.2434	g_loss: 1.2470
Epoch [2/	10]	d_loss: 1.1834	g_loss: 0.8979
Epoch [2/	10]	d_loss: 1.2100	g_loss: 1.3319
Epoch [2/	10]	d_loss: 1.1607	g_loss: 0.9028
Epoch [2/	10]	d_loss: 1.3429	g_loss: 0.7372
Epoch [2/	10]	d_loss: 1.0932	g_loss: 1.1218
Epoch [2/	10]	d_loss: 1.0377	g_loss: 1.5378
Epoch [2/	10]	d_loss: 1.0113	g_loss: 1.4318
Epoch [2/	10]	d_loss: 1.3242	g_loss: 1.0005
Epoch [2/	10]	d_loss: 1.2045	g_loss: 0.9995
Epoch [2/	10]	d_loss: 1.2905	g_loss: 1.2346
Epoch [2/	10]	d_loss: 1.3069	g_loss: 1.2506
Epoch [2/	10]	d_loss: 1.4221	g_loss: 1.0381
Epoch [2/	10]	d_loss: 1.4721	g_loss: 0.9996
Epoch [2/	10]	d_loss: 1.2811	g_loss: 1.2823
Epoch [3/	10]	d_loss: 1.1461	g_loss: 0.7340
Epoch [3/	10]	d_loss: 1.1866	g_loss: 1.0748
Epoch [3/	10]	d_loss: 1.2323	g_loss: 0.9138
Epoch [3/	10]	d_loss: 1.3460	g_loss: 1.0880
Epoch [3/	10]	d_loss: 1.0476	g_loss: 1.3735
Epoch [3/	10]	d_loss: 1.0570	g_loss: 1.3292
Epoch [3/	10]	d_loss: 1.3005	g_loss: 0.9895

Epoch [3/	10]		d_loss: 1.2052		g_loss: 0.8066
Epoch [3/	10]		d_loss: 1.3367		g_loss: 1.7338
Epoch [3/	10]		d_loss: 1.1379		g_loss: 0.9770
Epoch [3/	10]		d_loss: 1.0162		g_loss: 1.4191
Epoch [3/	10]		d_loss: 1.2198		g_loss: 0.6791
Epoch [3/	10]		d_loss: 1.1604		g_loss: 0.9042
Epoch [3/	10]		d_loss: 1.3923		g_loss: 1.1504
Epoch [3/	10]		d_loss: 1.1654		g_loss: 0.9348
Epoch [3/	10]		d_loss: 1.1159		g_loss: 1.1775
Epoch [3/	10]		d_loss: 1.1391		g_loss: 1.2324
Epoch [3/	10]		d_loss: 1.2586		g_loss: 1.0634
Epoch [3/	10]		d_loss: 1.2252		g_loss: 1.1625
Epoch [3/	10]		d_loss: 1.2474		g_loss: 0.9151
Epoch [3/	10]		d_loss: 1.0826		g_loss: 1.3529
Epoch [3/	10]		d_loss: 1.3378		g_loss: 0.9827
Epoch [3/	10]		d_loss: 1.1801		g_loss: 0.7039
Epoch [3/	10]		d_loss: 1.0839		g_loss: 1.2777
Epoch [3/	10]		d_loss: 1.3095		g_loss: 1.0614
Epoch [3/	10]		d_loss: 1.4357		g_loss: 0.8241
Epoch [3/	10]		d_loss: 1.1819		g_loss: 0.8880
Epoch [3/	10]		d_loss: 1.1471		g_loss: 0.9062
Epoch [3/	10]		d_loss: 1.1793		g_loss: 0.6497
Epoch [4/	10]		d_loss: 1.2522		g_loss: 1.1134
Epoch [4/	10]		d_loss: 1.1459		g_loss: 1.1406
Epoch [4/	10]		d_loss: 1.3981		g_loss: 1.0779
Epoch [4/	10]		d_loss: 1.1669		g_loss: 0.7122
Epoch [4/	10]		d_loss: 1.3491		g_loss: 0.8014
Epoch [4/	10]		d_loss: 1.3364		g_loss: 0.9730
Epoch [4/	10]		d_loss: 1.1546		g_loss: 1.1656
Epoch [4/	10]		d_loss: 1.5158		g_loss: 0.8450
Epoch [4/	10]		d_loss: 1.3611		g_loss: 1.0797
Epoch [4/	10]		d_loss: 1.3973		g_loss: 0.8687
Epoch [4/	10]		d_loss: 0.9968		g_loss: 1.3245
Epoch [4/	10]		d_loss: 1.3149		g_loss: 1.4523
Epoch [4/	10]		d_loss: 1.2302		g_loss: 0.8995
Epoch [4/	10]		d_loss: 1.3023		g_loss: 0.9873
Epoch [4/	10]		d_loss: 1.1481		g_loss: 1.3128
Epoch [4/	10]		d_loss: 1.0703		g_loss: 0.9446
Epoch [4/	10]		d_loss: 1.3870		g_loss: 0.8813
Epoch [4/	10]		d_loss: 1.4876		g_loss: 0.9097
Epoch [4/	10]		d_loss: 1.3338		g_loss: 0.7912
Epoch [4/	10]		d_loss: 0.9730		g_loss: 1.0509
Epoch [4/	10]		d_loss: 1.2065		g_loss: 0.9431
Epoch [4/	10]		d_loss: 1.1987		g_loss: 0.9731
Epoch [4/	10]		d_loss: 1.1137		g_loss: 0.9643
Epoch [4/	10]		d_loss: 1.0831		g_loss: 1.0460
Epoch [4/	10]		d_loss: 1.3130		g_loss: 1.2180
Epoch [4/	10]		d_loss: 0.9991		g_loss: 1.0101

Epoch [4/	10]	d_loss: 1.2448	g_loss: 1.1507
Epoch [4/	10]	d_loss: 1.1648	g_loss: 1.1985
Epoch [4/	10]	d_loss: 1.3436	g_loss: 0.9330
Epoch [5/	10]	d_loss: 1.1200	g_loss: 0.9460
Epoch [5/	10]	d_loss: 1.3623	g_loss: 0.9731
Epoch [5/	10]	d_loss: 1.0899	g_loss: 1.0052
Epoch [5/	10]	d_loss: 1.1526	g_loss: 1.1232
Epoch [5/	10]	d_loss: 1.1792	g_loss: 0.9877
Epoch [5/	10]	d_loss: 1.1478	g_loss: 0.6008
Epoch [5/	10]	d_loss: 1.2038	g_loss: 1.0866
Epoch [5/	10]	d_loss: 1.0010	g_loss: 1.1985
Epoch [5/	10]	d_loss: 1.2427	g_loss: 0.9178
Epoch [5/	10]	d_loss: 1.0630	g_loss: 0.9205
Epoch [5/	10]	d_loss: 1.2579	g_loss: 1.3579
Epoch [5/	10]	d_loss: 1.2985	g_loss: 1.1281
Epoch [5/	10]	d_loss: 1.2670	g_loss: 0.8779
Epoch [5/	10]	d_loss: 1.1239	g_loss: 1.0754
Epoch [5/	10]	d_loss: 1.2922	g_loss: 0.8742
Epoch [5/	10]	d_loss: 1.2634	g_loss: 0.9878
Epoch [5/	10]	d_loss: 1.2984	g_loss: 0.9878
Epoch [5/	10]	d_loss: 1.1937	g_loss: 0.8460
Epoch [5/	10]	d_loss: 1.1990	g_loss: 0.9645
Epoch [5/	10]	d_loss: 1.4178	g_loss: 0.8617
Epoch [5/	10]	d_loss: 1.1960	g_loss: 1.1743
Epoch [5/	10]	d_loss: 1.1500	g_loss: 1.0782
Epoch [5/	10]	d_loss: 1.2434	g_loss: 1.0145
Epoch [5/	10]	d_loss: 1.1373	g_loss: 1.0369
Epoch [5/	10]	d_loss: 1.3405	g_loss: 0.9337
Epoch [5/	10]	d_loss: 1.3684	g_loss: 1.0543
Epoch [5/	10]	d_loss: 1.3464	g_loss: 0.8790
Epoch [5/	10]	d_loss: 1.1784	g_loss: 0.9263
Epoch [5/	10]	d_loss: 1.3026	g_loss: 1.3046
Epoch [6/	10]	d_loss: 1.3113	g_loss: 0.8179
Epoch [6/	10]	d_loss: 1.1855	g_loss: 0.9334
Epoch [6/	10]	d_loss: 1.1229	g_loss: 0.9656
Epoch [6/	10]	d_loss: 1.2729	g_loss: 1.1526
Epoch [6/	10]	d_loss: 1.2367	g_loss: 0.8374
Epoch [6/	10]	d_loss: 1.2956	g_loss: 1.1615
Epoch [6/	10]	d_loss: 1.1718	g_loss: 0.8399
Epoch [6/	10]	d_loss: 1.1839	g_loss: 0.9898
Epoch [6/	10]	d_loss: 1.1582	g_loss: 1.2594
Epoch [6/	10]	d_loss: 1.2434	g_loss: 1.0625
Epoch [6/	10]	d_loss: 1.3493	g_loss: 1.1043
Epoch [6/	10]	d_loss: 1.2628	g_loss: 0.9166
Epoch [6/	10]	d_loss: 1.1387	g_loss: 1.1104
Epoch [6/	10]	d_loss: 1.3852	g_loss: 0.9558
Epoch [6/	10]	d_loss: 1.4288	g_loss: 0.9298
Epoch [6/	10]	d_loss: 1.2232	g_loss: 1.0014

Epoch [6/	10]	d_loss: 1.4434	g_loss: 1.0584
Epoch [6/	10]	d_loss: 1.1488	g_loss: 0.9304
Epoch [6/	10]	d_loss: 1.3384	g_loss: 0.8666
Epoch [6/	10]	d_loss: 0.9801	g_loss: 1.2457
Epoch [6/	10]	d_loss: 1.2241	g_loss: 1.1045
Epoch [6/	10]	d_loss: 1.2128	g_loss: 0.7986
Epoch [6/	10]	d_loss: 1.1309	g_loss: 0.9417
Epoch [6/	10]	d_loss: 1.0522	g_loss: 1.1458
Epoch [6/	10]	d_loss: 1.2330	g_loss: 0.9512
Epoch [6/	10]	d_loss: 1.0651	g_loss: 1.2413
Epoch [6/	10]	d_loss: 1.2327	g_loss: 0.9925
Epoch [6/	10]	d_loss: 1.1042	g_loss: 1.0987
Epoch [6/	10]	d_loss: 1.4625	g_loss: 0.8554
Epoch [7/	10]	d_loss: 1.1553	g_loss: 1.0541
Epoch [7/	10]	d_loss: 1.2941	g_loss: 1.1630
Epoch [7/	10]	d_loss: 1.1930	g_loss: 1.0722
Epoch [7/	10]	d_loss: 1.3547	g_loss: 1.3097
Epoch [7/	10]	d_loss: 1.4457	g_loss: 1.0158
Epoch [7/	10]	d_loss: 1.4251	g_loss: 1.0158
Epoch [7/	10]	d_loss: 1.1293	g_loss: 1.2996
Epoch [7/	10]	d_loss: 1.2557	g_loss: 1.0512
Epoch [7/	10]	d_loss: 1.3712	g_loss: 0.9624
Epoch [7/	10]	d_loss: 1.1583	g_loss: 0.7633
Epoch [7/	10]	d_loss: 1.1256	g_loss: 0.7830
Epoch [7/	10]	d_loss: 1.1962	g_loss: 0.9581
Epoch [7/	10]	d_loss: 1.2114	g_loss: 0.9903
Epoch [7/	10]	d_loss: 1.1911	g_loss: 1.0938
Epoch [7/	10]	d_loss: 1.3808	g_loss: 0.8572
Epoch [7/	10]	d_loss: 1.1458	g_loss: 1.1487
Epoch [7/	10]	d_loss: 1.2880	g_loss: 0.9261
Epoch [7/	10]	d_loss: 1.3385	g_loss: 0.9878
Epoch [7/	10]	d_loss: 1.4816	g_loss: 0.9535
Epoch [7/	10]	d_loss: 1.2055	g_loss: 1.1806
Epoch [7/	10]	d_loss: 1.3698	g_loss: 0.9677
Epoch [7/	10]	d_loss: 1.0888	g_loss: 1.1377
Epoch [7/	10]	d_loss: 1.3311	g_loss: 0.7654
Epoch [7/	10]	d_loss: 1.2916	g_loss: 1.0941
Epoch [7/	10]	d_loss: 1.0593	g_loss: 0.9455
Epoch [7/	10]	d_loss: 1.3259	g_loss: 1.0420
Epoch [7/	10]	d_loss: 1.3447	g_loss: 0.8980
Epoch [7/	10]	d_loss: 1.0623	g_loss: 0.8661
Epoch [7/	10]	d_loss: 1.3554	g_loss: 0.7139
Epoch [8/	10]	d_loss: 1.3771	g_loss: 0.9959
Epoch [8/	10]	d_loss: 1.4271	g_loss: 0.6083
Epoch [8/	10]	d_loss: 1.2317	g_loss: 1.1492
Epoch [8/	10]	d_loss: 1.4580	g_loss: 0.8989
Epoch [8/	10]	d_loss: 1.3450	g_loss: 0.9139
Epoch [8/	10]	d_loss: 1.2489	g_loss: 1.0065

Epoch [8/	10]	d_loss: 1.2666	g_loss: 0.8847
Epoch [8/	10]	d_loss: 1.3528	g_loss: 0.9676
Epoch [8/	10]	d_loss: 1.1229	g_loss: 1.1373
Epoch [8/	10]	d_loss: 1.3126	g_loss: 0.8018
Epoch [8/	10]	d_loss: 1.3030	g_loss: 1.0511
Epoch [8/	10]	d_loss: 1.3059	g_loss: 0.8024
Epoch [8/	10]	d_loss: 1.2333	g_loss: 0.9223
Epoch [8/	10]	d_loss: 1.1612	g_loss: 1.2163
Epoch [8/	10]	d_loss: 1.3063	g_loss: 0.8269
Epoch [8/	10]	d_loss: 1.1201	g_loss: 1.0244
Epoch [8/	10]	d_loss: 1.4444	g_loss: 0.8099
Epoch [8/	10]	d_loss: 1.2185	g_loss: 0.9337
Epoch [8/	10]	d_loss: 1.1699	g_loss: 0.9643
Epoch [8/	10]	d_loss: 1.1815	g_loss: 1.0284
Epoch [8/	10]	d_loss: 1.3872	g_loss: 0.8185
Epoch [8/	10]	d_loss: 1.1327	g_loss: 0.9891
Epoch [8/	10]	d_loss: 1.3784	g_loss: 0.8816
Epoch [8/	10]	d_loss: 1.3321	g_loss: 0.9770
Epoch [8/	10]	d_loss: 1.2738	g_loss: 0.9370
Epoch [8/	10]	d_loss: 1.0663	g_loss: 1.1150
Epoch [8/	10]	d_loss: 1.2732	g_loss: 0.9180
Epoch [8/	10]	d_loss: 1.3261	g_loss: 1.0478
Epoch [8/	10]	d_loss: 1.1241	g_loss: 0.8770
Epoch [9/	10]	d_loss: 1.2698	g_loss: 0.9613
Epoch [9/	10]	d_loss: 1.2891	g_loss: 1.1542
Epoch [9/	10]	d_loss: 1.2510	g_loss: 1.1476
Epoch [9/	10]	d_loss: 1.1773	g_loss: 0.8670
Epoch [9/	10]	d_loss: 1.2472	g_loss: 0.9761
Epoch [9/	10]	d_loss: 1.1537	g_loss: 0.8364
Epoch [9/	10]	d_loss: 1.1720	g_loss: 0.9785
Epoch [9/	10]	d_loss: 1.2464	g_loss: 1.0173
Epoch [9/	10]	d_loss: 1.3715	g_loss: 1.1436
Epoch [9/	10]	d_loss: 1.3024	g_loss: 0.8958
Epoch [9/	10]	d_loss: 1.3261	g_loss: 0.8753
Epoch [9/	10]	d_loss: 1.2103	g_loss: 0.9016
Epoch [9/	10]	d_loss: 1.3909	g_loss: 0.5502
Epoch [9/	10]	d_loss: 1.3035	g_loss: 0.9937
Epoch [9/	10]	d_loss: 1.4184	g_loss: 0.9198
Epoch [9/	10]	d_loss: 1.4132	g_loss: 1.0217
Epoch [9/	10]	d_loss: 1.2735	g_loss: 0.8842
Epoch [9/	10]	d_loss: 1.5238	g_loss: 0.8907
Epoch [9/	10]	d_loss: 1.2574	g_loss: 1.1718
Epoch [9/	10]	d_loss: 1.7234	g_loss: 0.9651
Epoch [9/	10]	d_loss: 1.3466	g_loss: 0.9404
Epoch [9/	10]	d_loss: 1.3295	g_loss: 0.9198
Epoch [9/	10]	d_loss: 1.3145	g_loss: 0.8945
Epoch [9/	10]	d_loss: 1.3815	g_loss: 0.9309
Epoch [9/	10]	d_loss: 1.2922	g_loss: 0.7477

```

Epoch [ 9/ 10] | d_loss: 1.2568 | g_loss: 0.6240
Epoch [ 9/ 10] | d_loss: 1.1769 | g_loss: 0.8698
Epoch [ 9/ 10] | d_loss: 1.4335 | g_loss: 0.6805
Epoch [ 9/ 10] | d_loss: 1.3504 | g_loss: 0.8911
Epoch [ 10/ 10] | d_loss: 1.1212 | g_loss: 0.9933
Epoch [ 10/ 10] | d_loss: 1.3929 | g_loss: 0.8499
Epoch [ 10/ 10] | d_loss: 1.2407 | g_loss: 0.9947
Epoch [ 10/ 10] | d_loss: 1.1119 | g_loss: 0.9207
Epoch [ 10/ 10] | d_loss: 1.3652 | g_loss: 0.9551
Epoch [ 10/ 10] | d_loss: 1.3789 | g_loss: 0.9373
Epoch [ 10/ 10] | d_loss: 1.2374 | g_loss: 0.9114
Epoch [ 10/ 10] | d_loss: 1.2864 | g_loss: 1.0930
Epoch [ 10/ 10] | d_loss: 1.6527 | g_loss: 0.9479
Epoch [ 10/ 10] | d_loss: 1.3960 | g_loss: 0.9701
Epoch [ 10/ 10] | d_loss: 1.3105 | g_loss: 0.9012
Epoch [ 10/ 10] | d_loss: 1.4606 | g_loss: 1.0054
Epoch [ 10/ 10] | d_loss: 1.4507 | g_loss: 0.8719
Epoch [ 10/ 10] | d_loss: 1.4056 | g_loss: 0.8449
Epoch [ 10/ 10] | d_loss: 1.3565 | g_loss: 0.9061
Epoch [ 10/ 10] | d_loss: 1.0628 | g_loss: 0.9575
Epoch [ 10/ 10] | d_loss: 1.4035 | g_loss: 1.0265
Epoch [ 10/ 10] | d_loss: 1.2747 | g_loss: 1.0855
Epoch [ 10/ 10] | d_loss: 1.0833 | g_loss: 0.9281
Epoch [ 10/ 10] | d_loss: 0.9872 | g_loss: 0.8838
Epoch [ 10/ 10] | d_loss: 0.9663 | g_loss: 1.2958
Epoch [ 10/ 10] | d_loss: 1.3694 | g_loss: 0.7770
Epoch [ 10/ 10] | d_loss: 1.2609 | g_loss: 0.8387
Epoch [ 10/ 10] | d_loss: 1.4186 | g_loss: 0.8994
Epoch [ 10/ 10] | d_loss: 1.2103 | g_loss: 1.2235
Epoch [ 10/ 10] | d_loss: 1.2129 | g_loss: 0.7963
Epoch [ 10/ 10] | d_loss: 1.4001 | g_loss: 0.9450
Epoch [ 10/ 10] | d_loss: 1.2546 | g_loss: 1.0183
Epoch [ 10/ 10] | d_loss: 1.5298 | g_loss: 0.6739

```

2.8 Training loss

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

```

In [26]: 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[26]: <matplotlib.legend.Legend at 0x7fa9cc181198>

```



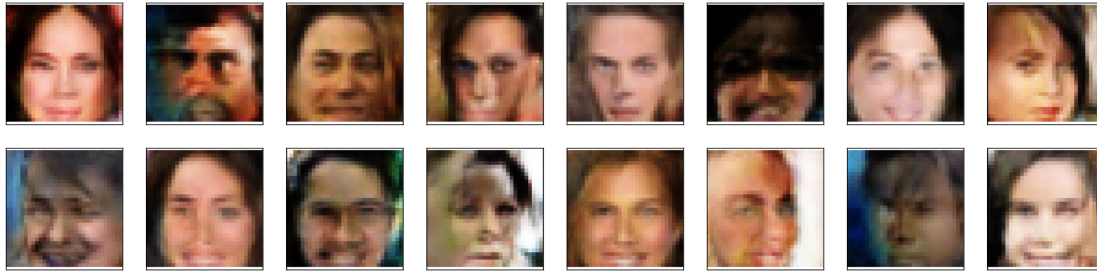
2.9 Generator samples from training

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

```
In [27]: # 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 [28]: # Load samples from generator, taken while training
with open('train_samples.pkl', 'rb') as f:
    samples = pkl.load(f)

In [29]: _ = 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)

- Discriminator is very steady over time. We can observe squiggling up and down behavior as it's typical for adversarial network.
- Generator steadily decreases over time. For the most parts, generator loss value is slightly smaller than discriminator loss. Generator loss decreases a lot at the start of training (we can observe a huge spike). It is likely because it starts off producing really bad fakes. Then it made a big improvement, and after that it refined over time.
- Dropout layer in-between each of linear layers has been applied to ensure the network is likely to train each node evenly. - Having label smoothing will smooth our ground truth labels. It will make discriminator numerically stable and it will help discriminator to generalize better.
- Model produces low-resolution images.
- If weights are initialized in the network, it might help the model to converge faster.
- Higher number of epochs in most cases will produce better results at the price of training taking much longer time.
- Model can be improved even more by better tuning and optimization any of hyperparameters (like learning rate, batch size, etc.), as well as increasing depth by additional hidden layers.

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.