Face Generation

In this project, you'll use generative adversarial networks to generate new images of faces.

Get the Data

You'll be using two datasets in this project:

- MNIST
- CelebA

Since the celebA dataset is complex and you're doing GANs in a project for the first time, we want you to test your neural network on MNIST before CelebA. Running the GANs on MNIST will allow you to see how well your model trains sooner.

If you're using FloydHub, set `data_dir` to "/input" and use the FloydHub data ID "R5KrjnANiKVhLWAkpXhNBe".

[1]
```
#data_dir = './data'

# FloydHub - Use with data ID "R5KrjnANiKVhLWAkpXhNBe"
data_dir = '/input'


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

helper.download_extract('mnist', data_dir)
helper.download_extract('celeba', data_dir)
```

```
Downloading mnist: 9.92MB [00:01, 7.80MB/s]
Extracting mnist: 100%|██████████| 60.0K/60.0K [00:10<00:00,
5.48KFile/s]
Downloading celeba: 1.44GB [00:25, 56.6MB/s]
Extracting celeba...
```

Explore the Data

MNIST

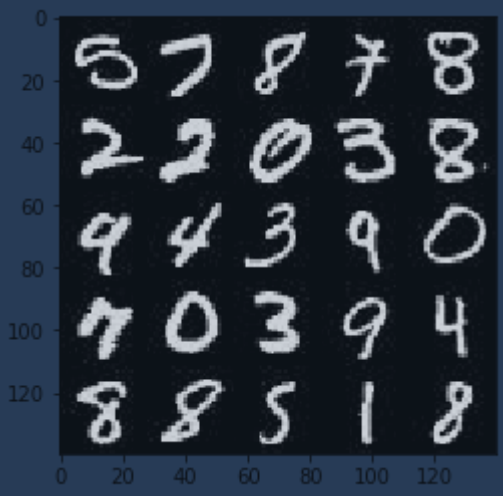
As you're aware, the MNIST dataset contains images of handwritten digits. You can view the first number of examples by changing `show_n_images`.

[2]
```
show_n_images = 25

"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
%matplotlib inline
import os
from glob import glob
from matplotlib import pyplot

mnist_images = helper.get_batch(glob(os.path.join(data_dir, 'mnist/*.jp
pyplot.imshow(helper.images_square_grid(mnist_images, 'L'), cmap='gray'
```

```
<matplotlib.image.AxesImage at 0x7efc492c4240>
```



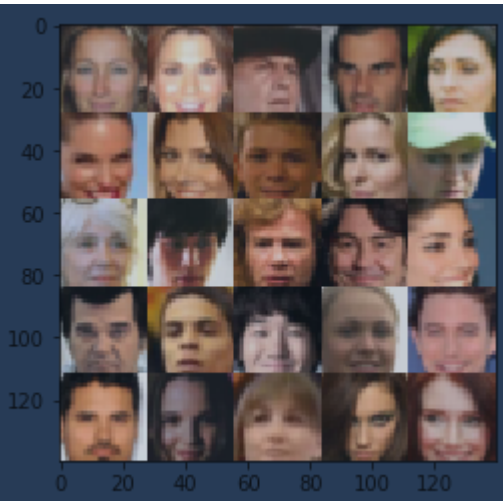
CelebA

The CelebFaces Attributes Dataset (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 can view the first number of examples by changing `show_n_images`.

[3]
```
show_n_images = 25

"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
mnist_images = helper.get_batch(glob(os.path.join(data_dir, 'img_align_
pyplot.imshow(helper.images_square_grid(mnist_images, 'RGB'))
```

```
<matplotlib.image.AxesImage at 0x7efc491f8cf8>
```



Preprocess the Data

Since the project's main focus is on building the GANs, we'll preprocess the data for you. The values of the MNIST and CelebA dataset will be in the range of -0.5 to 0.5 of 28x28 dimensional images. The CelebA images will be cropped to remove parts of the image that don't include a face, then resized down to 28x28.

The MNIST images are black and white images with a single [color channel] (https://en.wikipedia.org/wiki/Channel_(digital_image%29) while the CelebA images have [3 color channels (RGB color channel)] (https://en.wikipedia.org/wiki/Channel_(digital_image%29#RGB_Images).

Build the Neural Network

You'll build the components necessary to build a GANs by implementing the following functions below:

- `model_inputs`
- `discriminator`
- `generator`
- `model_loss`
- `model_opt`
- `train`

Check the Version of TensorFlow and Access to GPU

This will check to make sure you have the correct version of TensorFlow and access to a GPU

[4]
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
from distutils.version import LooseVersion
import warnings
import tensorflow as tf
```

```
# Check TensorFlow Version
assert LooseVersion(tf.__version__) >= LooseVersion('1.0'), 'Please use
print('TensorFlow Version: {}'.format(tf.__version__))

# Check for a GPU
if not tf.test.gpu_device_name():
    warnings.warn('No GPU found. Please use a GPU to train your neural
else:
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
```

```
TensorFlow Version: 1.2.1
Default GPU Device: /gpu:0
```

Input

Implement the `model_inputs` function to create TF Placeholders for the Neural Network. It should create the following placeholders:

- Real input images placeholder with rank 4 using `image_width`, `image_height`, and `image_channels`.
- Z input placeholder with rank 2 using `z_dim`.
- Learning rate placeholder with rank 0.

Return the placeholders in the following the tuple (tensor of real input images, tensor of z data)

[5]
```
import problem_unittests as tests

def model_inputs(image_width, image_height, image_channels, z_dim):
    """
    Create the model inputs
    :param image_width: The input image width
    :param image_height: The input image height
    :param image_channels: The number of image channels
    :param z_dim: The dimension of Z
    :return: Tuple of (tensor of real input images, tensor of z data,
    """
    inputs_real = tf.placeholder(tf.float32, (None, image_width, image
    inputs_z = tf.placeholder(tf.float32, (None, z_dim), name='input_z
    learning_rate = tf.placeholder(tf.float32, None, name='learning_ra

    return inputs_real, inputs_z, learning_rate


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_inputs(model_inputs)
```

```
ERROR:tensorflow:==================================
Object was never used (type <class
'tensorflow.python.framework.ops.Operation'>):
<tf.Operation 'assert_rank_2/Assert/Assert' type=Assert>
If you want to mark it as used call its "mark_used()" method.
It was originally created here:
['File "/usr/local/lib/python3.5/runpy.py", line 193, in
_run_module_as_main\n    "__main__", mod_spec)', 'File
"/usr/local/lib/python3.5/runpy.py", line 85, in _run_code\n
exec(code, run_globals)', 'File "/usr/local/lib/python3.5/site-
packages/ipykernel_launcher.py", line 16, in <module>\n
app.launch_new_instance()', 'File "/usr/local/lib/python3.5/site-
packages/traitlets/config/application.py", line 658, in
launch_instance\n    app.start()', 'File
"/usr/local/lib/python3.5/site-packages/ipykernel/kernelapp.py", line
477, in start\n    ioloop.IOLoop.instance().start()', 'File
"/usr/local/lib/python3.5/site-packages/zmq/eventloop/ioloop.py", line
177, in start\n    super(ZMQIOLoop, self).start()', 'File
"/usr/local/lib/python3.5/site-packages/tornado/ioloop.py", line 888,
in start\n    handler_func(fd_obj, events)', 'File
"/usr/local/lib/python3.5/site-packages/tornado/stack_context.py",
line 277, in null_wrapper\n    return fn(*args, **kwargs)', 'File
"/usr/local/lib/python3.5/site-packages/zmq/eventloop/zmqstream.py",
line 440, in _handle_events\n    self._handle_recv()', 'File
"/usr/local/lib/python3.5/site-packages/zmq/eventloop/zmqstream.py",
line 472, in _handle_recv\n    self._run_callback(callback, msg)',
'File "/usr/local/lib/python3.5/site-
packages/zmq/eventloop/zmqstream.py", line 414, in _run_callback\n
callback(*args, **kwargs)', 'File "/usr/local/lib/python3.5/site-
packages/tornado/stack_context.py", line 277, in null_wrapper\n
```

Discriminator

Implement `discriminator` to create a discriminator neural network that discriminates on `images`. This function should be able to reuse the variables in the neural network. Use tf.variable_scope with a scope name of "discriminator" to allow the variables to be reused. The function should return a tuple of (tensor output of the discriminator, tensor logits of the discriminator).

[6]

```
def discriminator(images, reuse=False):
    """
    Create the discriminator network
    :param images: Tensor of input image(s)
    :param reuse: Boolean if the weights should be reused
    :return: Tuple of (tensor output of the discriminator, tensor logi
    """
    alpha = 0.2
    with tf.variable_scope('discriminator', reuse=reuse):
```

```
        # Input layer is 28x28x3
        x1 = tf.layers.conv2d(images, 64, 5, strides=2, padding='same'
        relu1 = tf.maximum(alpha * x1, x1)
        # 14x14x64

        x2 = tf.layers.conv2d(relu1, 128, 5, strides=2, padding='same'
        bn2 = tf.layers.batch_normalization(x2, training=True)
        relu2 = tf.maximum(alpha * bn2, bn2)
        # 7x7x128

        x3 = tf.layers.conv2d(relu2, 256, 5, strides=2, padding='same'
        bn3 = tf.layers.batch_normalization(x3, training=True)
        relu3 = tf.maximum(alpha * bn3, bn3)
        # 4x4x256

        # Flatten it
        flat = tf.reshape(relu3, (-1, 4*4*256))
        logits = tf.layers.dense(flat, 1)
        out = tf.sigmoid(logits)

        return out, logits


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_discriminator(discriminator, tf)
```

```
Tests Passed
```

Generator

Implement `generator` to generate an image using `z`. This function should be able to reuse the variables in the neural network. Use [`tf.variable_scope`] with a scope name of "generator" to allow the variables to be reused. The function should return the generated 28 x 28 x `out_channel_dim` images.

[7]

```
def generator(z, out_channel_dim, is_train=True):
    """
    Create the generator network
    :param z: Input z
    :param out_channel_dim: The number of channels in the output image
    :param is_train: Boolean if generator is being used for training
    :return: The tensor output of the generator
    """
    alpha = 0.2
    with tf.variable_scope('generator', reuse=not is_train):
        # First fully connected layer
        x1 = tf.layers.dense(z, 7*7*256)
        # Reshape it to start the convolutional stack
        x1 = tf.reshape(x1, (-1, 7, 7, 256))
        x1 = tf.layers.batch_normalization(x1, training=is_train)
        x1 = tf.maximum(alpha * x1, x1)
```

```
        # 7x7x256

        x2 = tf.layers.conv2d_transpose(x1, 128, 5, strides=2, padding
        x2 = tf.layers.batch_normalization(x2, training=is_train)
        x2 = tf.maximum(alpha * x2, x2)
        # 14x14x128

        x3 = tf.layers.conv2d_transpose(x2, 64, 5, strides=2, padding=
        x3 = tf.layers.batch_normalization(x3, training=is_train)
        x3 = tf.maximum(alpha * x3, x3)
        # 28x28x64

        # Output layer
        logits = tf.layers.conv2d_transpose(x3, out_channel_dim, 5, st
        # 28x28x3

        out = tf.tanh(logits)

        return out


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_generator(generator, tf)
```

```
Tests Passed
```

Loss

Implement `model_loss` to build the GANs for training and calculate the loss. The function should return a tuple of (discriminator loss, generator loss). Use the following functions you implemented:

- `discriminator(images, reuse=False)`
- `generator(z, out_channel_dim, is_train=True)`

[8]

```
def model_loss(input_real, input_z, out_channel_dim):
    """
    Get the loss for the discriminator and generator
    :param input_real: Images from the real dataset
    :param input_z: Z input
    :param out_channel_dim: The number of channels in the output image
    :return: A tuple of (discriminator loss, generator loss)
    """
    g_model = generator(input_z, out_channel_dim)
    d_model_real, d_logits_real = discriminator(input_real)
    d_model_fake, d_logits_fake = discriminator(g_model, reuse=True)

    d_loss_real = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_real,
    d_loss_fake = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_fake,
```

```
    g_loss = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_fake,

    d_loss = d_loss_real + d_loss_fake

    return d_loss, g_loss


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_loss(model_loss)
```

```
Tests Passed
```

Optimization

Implement `model_opt` to create the optimization operations for the GANs. Use tf.trainable_variables to get all the trainable variables. Filter the variables with names that are in the discriminator and generator scope names. The function should return a tuple of (discriminator training operation, generator training operation).

[9]

```
def model_opt(d_loss, g_loss, learning_rate, beta1):
    """
    Get optimization operations
    :param d_loss: Discriminator loss Tensor
    :param g_loss: Generator loss Tensor
    :param learning_rate: Learning Rate Placeholder
    :param beta1: The exponential decay rate for the 1st moment in the
    :return: A tuple of (discriminator training operation, generator t
    """
    # Get weights and bias to update
    t_vars = tf.trainable_variables()
    d_vars = [var for var in t_vars if var.name.startswith('discrimina
    g_vars = [var for var in t_vars if var.name.startswith('generator'

    # Optimize
    with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_
        d_train_opt = tf.train.AdamOptimizer(learning_rate, beta1=beta
        g_train_opt = tf.train.AdamOptimizer(learning_rate, beta1=beta

    return d_train_opt, g_train_opt


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_opt(model_opt, tf)
```

```
Tests Passed
```

Neural Network Training

Show Output

Use this function to show the current output of the generator during training. It will help you determine how well the GANs is training.

[10]
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import numpy as np

def show_generator_output(sess, n_images, input_z, out_channel_dim, ima
    """
    Show example output for the generator
    :param sess: TensorFlow session
    :param n_images: Number of Images to display
    :param input_z: Input Z Tensor
    :param out_channel_dim: The number of channels in the output image
    :param image_mode: The mode to use for images ("RGB" or "L")
    """
    cmap = None if image_mode == 'RGB' else 'gray'
    z_dim = input_z.get_shape().as_list()[-1]
    example_z = np.random.uniform(-1, 1, size=[n_images, z_dim])

    samples = sess.run(
        generator(input_z, out_channel_dim, False),
        feed_dict={input_z: example_z})

    images_grid = helper.images_square_grid(samples, image_mode)
    pyplot.imshow(images_grid, cmap=cmap)
    pyplot.show()
```

Train

Implement `train` to build and train the GANs. Use the following functions you implemented:

- `model_inputs(image_width, image_height, image_channels, z_dim)`
- `model_loss(input_real, input_z, out_channel_dim)`
- `model_opt(d_loss, g_loss, learning_rate, beta1)`

Use the `show_generator_output` to show `generator` output while you train. Running `show_generator_output` for every batch will drastically increase training time and increase the size of the notebook. It's recommended to print the `generator` output every 100 batches.

[11]
```
def train(epoch_count, batch_size, z_dim, learning_rate, beta1, get_ba
    """
    Train the GAN
```

```
    :param epoch_count: Number of epochs
    :param batch_size: Batch Size
    :param z_dim: Z dimension
    :param learning_rate: Learning Rate
    :param beta1: The exponential decay rate for the 1st moment in the
    :param get_batches: Function to get batches
    :param data_shape: Shape of the data
    :param data_image_mode: The image mode to use for images ("RGB" or
    """
    samples, width, height, channels = data_shape

    input_real, input_z, lr = model_inputs(width, height, channels, z_
    d_loss, g_loss = model_loss(input_real, input_z, channels)
    d_opt, g_opt = model_opt(d_loss, g_loss, learning_rate, beta1)

    steps = 0

    print_every = 10
    show_every = 100

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for epoch_i in range(epoch_count):
            for batch_images in get_batches(batch_size):
                steps += 1

                # Rescale real image input to (-1, 1)
                batch_images *= 2.0

                # Sample random noise for G
                batch_z = np.random.uniform(-1, 1, size=(batch_size,

                # Run optimizers
                _ = sess.run(d_opt, feed_dict={input_real: batch_imag
                _ = sess.run(g_opt, feed_dict={input_z: batch_z, inpu

                if steps % print_every == 0:
                    # At the end of each epoch, get the losses and p
                    train_loss_d = d_loss.eval({input_z: batch_z, inp
                    train_loss_g = g_loss.eval({input_z: batch_z})

                    print("Epoch {}/{}...".format(epoch_i+1, epoch_co
                          "Discriminator Loss: {:.4f}...".format(tra
                          "Generator Loss: {:.4f}".format(train_loss

                if steps % show_every == 0:
                    show_generator_output(sess, 16, input_z, channels
```

MNIST

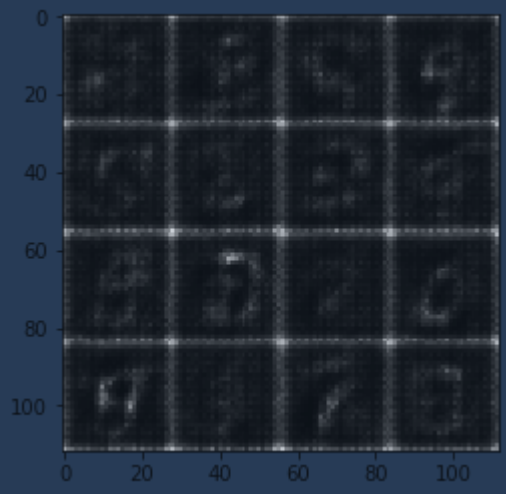
Test your GANs architecture on MNIST. After 2 epochs, the GANs should be able to generate images that look like handwritten digits. Make sure the loss of the generator is lower than the loss of the discriminator or close to 0.

[12]
```
batch_size = 64
z_dim = 100
learning_rate = 0.0002
beta1 = 0.5

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
epochs = 2

mnist_dataset = helper.Dataset('mnist', glob(os.path.join(data_dir, 'mr
with tf.Graph().as_default():
    train(epochs, batch_size, z_dim, learning_rate, beta1, mnist_datas
          mnist_dataset.shape, mnist_dataset.image_mode)
```

```
Epoch 1/2... Discriminator Loss: 1.2176... Generator Loss: 0.6714
Epoch 1/2... Discriminator Loss: 1.4056... Generator Loss: 0.6099
Epoch 1/2... Discriminator Loss: 1.1522... Generator Loss: 0.7128
Epoch 1/2... Discriminator Loss: 1.4629... Generator Loss: 0.4873
Epoch 1/2... Discriminator Loss: 1.3913... Generator Loss: 0.4417
Epoch 1/2... Discriminator Loss: 1.5108... Generator Loss: 1.2677
Epoch 1/2... Discriminator Loss: 1.3556... Generator Loss: 1.3759
Epoch 1/2... Discriminator Loss: 1.1695... Generator Loss: 0.7526
Epoch 1/2... Discriminator Loss: 1.1281... Generator Loss: 1.2136
Epoch 1/2... Discriminator Loss: 1.1452... Generator Loss: 1.0603
```




```
Epoch 1/2... Discriminator Loss: 1.3193... Generator Loss: 0.4210
Epoch 1/2... Discriminator Loss: 1.0953... Generator Loss: 1.3878
Epoch 1/2... Discriminator Loss: 1.0981... Generator Loss: 0.7554
Epoch 1/2... Discriminator Loss: 1.0849... Generator Loss: 0.6998
Epoch 1/2... Discriminator Loss: 1.0303... Generator Loss: 1.2040
Epoch 1/2... Discriminator Loss: 0.8796... Generator Loss: 0.9640
Epoch 1/2... Discriminator Loss: 1.0268... Generator Loss: 1.7176
Epoch 1/2... Discriminator Loss: 1.0429... Generator Loss: 0.7760
```

CelebA

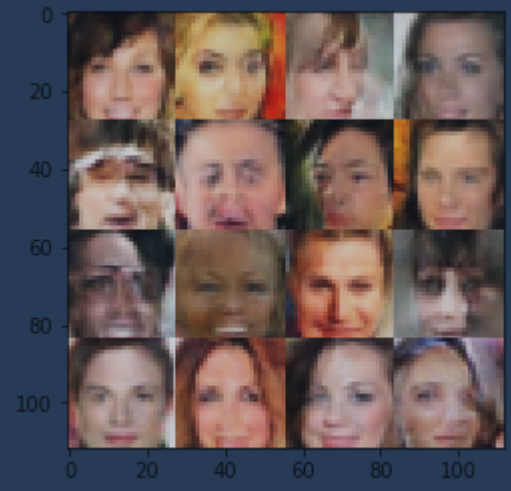
Run your GANs on CelebA. It will take around 20 minutes on the average GPU to run one epoch. You can run the whole epoch or stop when it starts to generate realistic faces.

```
[ ]
batch_size = 64
z_dim = 100
learning_rate = 0.0002
beta1 = 0.5

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
epochs = 10

celeba_dataset = helper.Dataset('celeba', glob(os.path.join(data_dir, '
with tf.Graph().as_default():
    train(epochs, batch_size, z_dim, learning_rate, beta1, celeba_data
          celeba_dataset.shape, celeba_dataset.image_mode)
```


```
Epoch 7/10... Discriminator Loss: 0.4829... Generator Loss: 2.0844
Epoch 7/10... Discriminator Loss: 1.5151... Generator Loss: 0.3434
Epoch 7/10... Discriminator Loss: 0.6994... Generator Loss: 0.9582
Epoch 7/10... Discriminator Loss: 0.6847... Generator Loss: 1.0741
Epoch 7/10... Discriminator Loss: 0.5882... Generator Loss: 1.4045
Epoch 7/10... Discriminator Loss: 0.8017... Generator Loss: 0.8570
Epoch 7/10... Discriminator Loss: 0.5083... Generator Loss: 1.6035
Epoch 7/10... Discriminator Loss: 0.4992... Generator Loss: 3.3632
Epoch 7/10... Discriminator Loss: 1.1290... Generator Loss: 0.5877
Epoch 7/10... Discriminator Loss: 0.7333... Generator Loss: 1.3097
```




```
Epoch 7/10... Discriminator Loss: 1.4246... Generator Loss: 0.4000
Epoch 7/10... Discriminator Loss: 0.3947... Generator Loss: 1.5444
Epoch 7/10... Discriminator Loss: 0.3315... Generator Loss: 1.7507
Epoch 7/10... Discriminator Loss: 1.3416... Generator Loss: 0.4524
```

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 "helper.py" and "problem_unittests.py" files in your submission.