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 (https://www.floydhub.com/), set `data_dir` to "/input" and use the FloydHub data ID (http://docs.floydhub.com/home/using_datasets/) "R5KrjnANiKVhLWAkpXhNBe".

In [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)
```

```
Found mnist Data
Found celeba Data
```

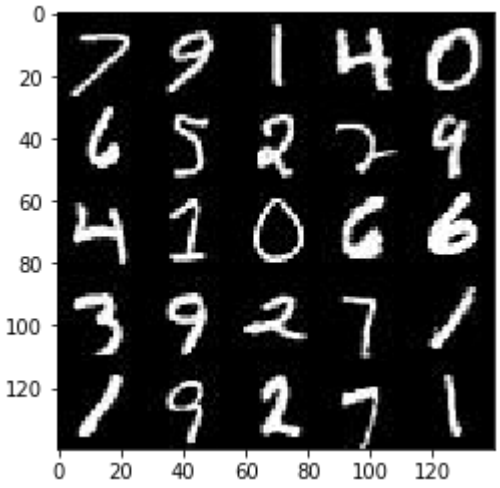
Explore the Data

MNIST

As you're aware, the MNIST (http://yann.lecun.com/exdb/mnist/) dataset contains images of handwritten digits. You can view the first number of examples by changing `show_n_images`.

In [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/*.jpg'))[:s
how_n_images], 28, 28, 'L')
pyplot.imshow(helper.images_square_grid(mnist_images, 'L'), cmap='gray')
```

Out[2]: <matplotlib.image.AxesImage at 0x7fac1baa8128>

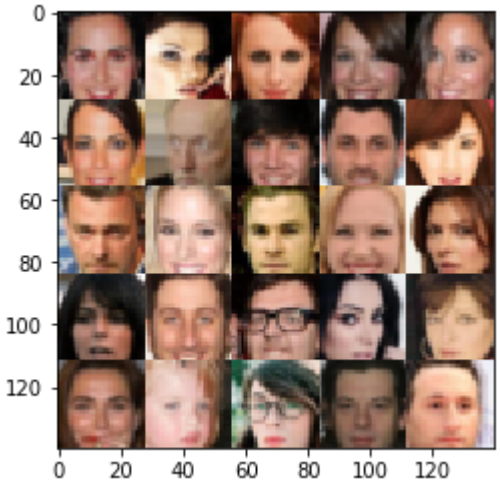


CelebA

The CelebFaces Attributes Dataset (CelebA) (http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html) 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`.

In [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_celeba/*.jpg'))[:show_n_images], 28, 28, 'RGB')
pyplot.imshow(helper.images_square_grid(mnist_images, 'RGB'))
```

Out[3]: <matplotlib.image.AxesImage at 0x7fac1b9dc160>



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

In [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 Tensor
Flow version 1.0 or newer.  You are using {}'.format(tf.__version__)
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 networ
k.')
else:
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
```

```
TensorFlow Version: 1.0.0
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)

In [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, learning
 rate)
    """
    # TODO: Implement Functionmodel_inputs
    inputs_real = tf.placeholder(tf.float32, (None, image_width, image_height,
 image_channels), name = 'input_real')
    inputs_z = tf.placeholder(tf.float32, (None, z_dim), name = 'input_z')
    learning_rate = tf.placeholder(tf.float32, (None))
    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)
```

```
Tests Passed
```

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` (https://www.tensorflow.org/api_docs/python/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).

In [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 logits of th
e discriminator)
    """
    # TODO: Implement Function

    alpha = 0.2
    with tf.variable_scope('discriminator', reuse = reuse):
        x1 = tf.layers.conv2d(images, 64, 4, strides = 2, padding = "same")
        bn1 = tf.layers.batch_normalization(x1, training = True)
        relu1 = tf.maximum(alpha * bn1, bn1)
        d1 = relu1
        #d1 = tf.layers.dropout(relu1, 0.5)

        x2 = tf.layers.conv2d(d1, 128, 4, strides = 2, padding = "same")
        bn2 = tf.layers.batch_normalization(x2, training = True)
        relu2 = tf.maximum(alpha * bn2, bn2)
        d2 = relu2
        #d2 = tf.layers.dropout(relu2, 0.5)

        x3 = tf.layers.conv2d(d2, 256, 4, strides = 2, padding = "same")
        bn3 = tf.layers.batch_normalization(x3, training = True)
        relu3 = tf.maximum(alpha * bn3, bn3)
        d3 = relu3
        #d3 = tf.layers.dropout(relu3, 0.5)

        flat = tf.reshape(d3, (-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` (https://www.tensorflow.org/api_docs/python/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.

In [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
    """
    # TODO: Implement Function

    reuse = not is_train
    alpha = 0.2
    with tf.variable_scope('generator', reuse = reuse):
        x1 = tf.layers.dense(z, 4 * 4 * 512)
        x1 = tf.reshape(x1, (-1, 4, 4, 512))
        bn1 = tf.layers.batch_normalization(x1, training = is_train)
        relu1 = tf.maximum(alpha * bn1, bn1)
        #d1 = relu1
        d1 = tf.layers.dropout(relu1, 0.5)

        x2 = tf.layers.conv2d_transpose(d1, 256, 4, strides = 1, padding = "va
lid")
        bn2 = tf.layers.batch_normalization(x2, training = is_train)
        relu2 = tf.maximum(alpha * bn2, bn2)
        #d2 = relu2
        d2 = tf.layers.dropout(relu2, 0.5)

        x3 = tf.layers.conv2d_transpose(d2, 128, 4, strides = 2, padding = "sa
me")
        bn3 = tf.layers.batch_normalization(x3, training = is_train)
        relu3 = tf.maximum(alpha * bn3, bn3)
        #d3 = relu3
        d3 = tf.layers.dropout(relu3, 0.5)

        logits = tf.layers.conv2d_transpose(d3, out_channel_dim, 4, strides =
2, padding = "same")
        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)`

In [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)
    """
    # TODO: Implement Function

    smooth = 0.1
    g_model = generator(input_z, out_channel_dim, is_train = True)
    d_model_real, d_logits_real = discriminator(input_real, reuse = False)
    d_model_fake, d_logits_fake = discriminator(g_model, reuse = True)

    # Calculate losses
    d_loss_real = tf.reduce_mean(
                        tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_
real,
                                                                labels=tf.ones_l
ike(d_model_real) * (1 - smooth)))
    d_loss_fake = tf.reduce_mean(
                        tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_
fake,
                                                                labels=tf.zeros_
like(d_logits_fake)))
    g_loss = tf.reduce_mean(
                 tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_fake,
                                                         labels=tf.ones_like(d
_model_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`](https://www.tensorflow.org/api_docs/python/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).

In [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 optimiz
er
    :return: A tuple of (discriminator training operation, generator training
 operation)
    """
    # TODO: Implement Function

    t_vars = tf.trainable_variables()
    g_vars = [var for var in t_vars if var.name.startswith('generator')]
    d_vars = [var for var in t_vars if var.name.startswith('discriminator')]
    all_update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)

    g_update_ops = [var for var in all_update_ops if var.name.startswith('gene
rator')]
    d_update_ops = [var for var in all_update_ops if var.name.startswith('disc
riminator')]

    with tf.control_dependencies(d_update_ops):
        d_train_opt = tf.train.AdamOptimizer(learning_rate,beta1=beta1).minimi
ze(d_loss, var_list=d_vars)
    with tf.control_dependencies(g_update_ops):
        g_train_opt = tf.train.AdamOptimizer(learning_rate,beta1=beta1).minimi
ze(g_loss, var_list=g_vars)

    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.

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

def show_generator_output(sess, n_images, input_z, out_channel_dim, image_mode
):
    """
    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.

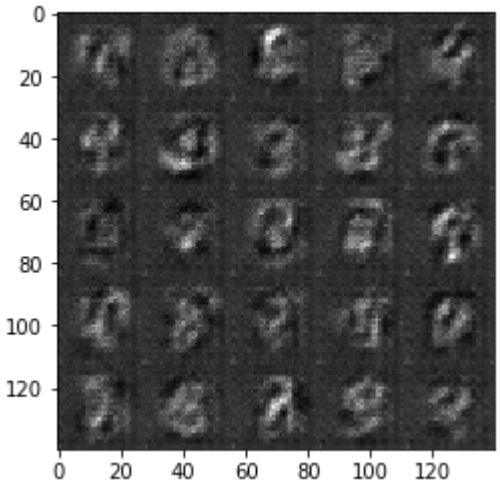
In [11]:
```
def train(epoch_count, batch_size, z_dim, learning_rate, beta1, get_batches
, data_shape, data_image_mode):
    """
    Train the GAN
    :param epoch_count: Number of epochs
    :param batch_size: Batch Size
    :param z_dim: Z dimension
    :param learning_rate: Learning Rate
    :param beta1: The exponential decay rate for the 1st moment in the opti
mizer
    :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 "L")
    """
    # TODO: Build Model
    inputs_real, inputs_z, lr = model_inputs(data_shape[1], data_shape[2],
data_shape[3], z_dim)
    d_loss, g_loss = model_loss(inputs_real, inputs_z, data_shape[-1])
    d_train_opt, g_train_opt = model_opt(d_loss, g_loss, learning_rate, bet
a1)
    step = 0

    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):
                # TODO: Train Model
                step = step + 1
                batch_images = batch_images * 2
                # Sample random noise for G
                batch_z = np.random.uniform(-1, 1, size = (batch_size, z_di
m))
                #print('batch_z shape=',batch_z.shape)
                # Run optimizers
                _ = sess.run(d_train_opt, feed_dict = {inputs_real: batch_i
mages, inputs_z: batch_z, lr:learning_rate})
                _ = sess.run(g_train_opt, feed_dict = {inputs_z: batch_z, l
r:learning_rate})

                if step % 100 == 0:
                    train_loss_d = d_loss.eval({inputs_z:batch_z, inputs_re
al: batch_images})
                    train_loss_g = g_loss.eval({inputs_z:batch_z})
                    print("Epoch {}/{} Step {}...".format(epoch_i+1, epoch_
count, step),
                      "Discriminator Loss: {:.4f}...".format(train_loss_d),
                      "Generator Loss: {:.4f}".format(train_loss_g))

                if step % 200 == 0:
                    show_generator_output(sess, 25, inputs_z, data_shape[3
], data_image_mode)
```

MNIST

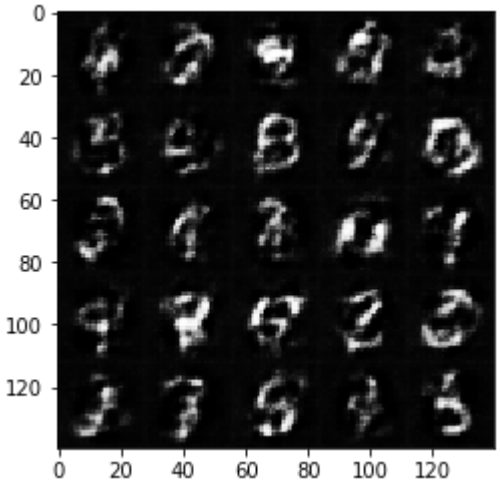
Test your GANs architecture on MNIST. After 2 epochs, the GANs should be able to generate images that look like handwritten digits. Make sure the loss of the generator is lower than the loss of the discriminator or close to 0.

In [12]:
```
batch_size = 100
z_dim = 100
learning_rate = 0.0001
beta1 = 0.2


"""
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, 'mnist/*.j
pg')))
with tf.Graph().as_default():
    train(epochs, batch_size, z_dim, learning_rate, beta1, mnist_dataset.get_b
atches,
          mnist_dataset.shape, mnist_dataset.image_mode)
```

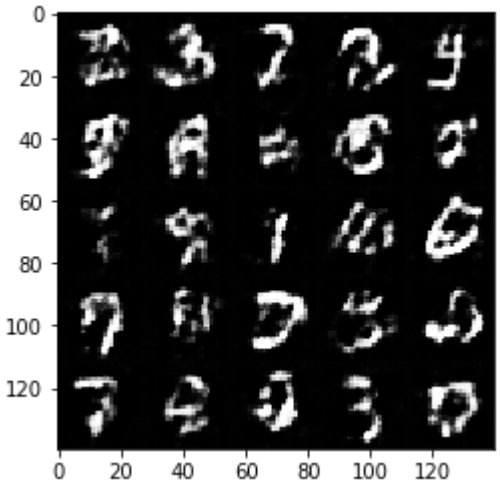
```
Epoch 1/2 Step 100... Discriminator Loss: 1.1081... Generator Loss: 0.7781
Epoch 1/2 Step 200... Discriminator Loss: 1.1718... Generator Loss: 0.6604
```



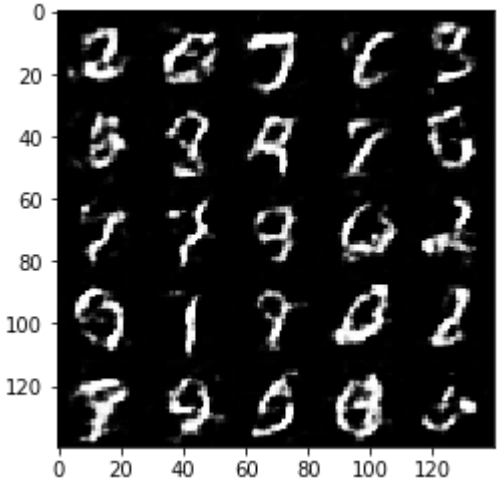

```
Epoch 1/2 Step 300... Discriminator Loss: 0.8743... Generator Loss: 1.2917
Epoch 1/2 Step 400... Discriminator Loss: 1.0337... Generator Loss: 0.7762
```



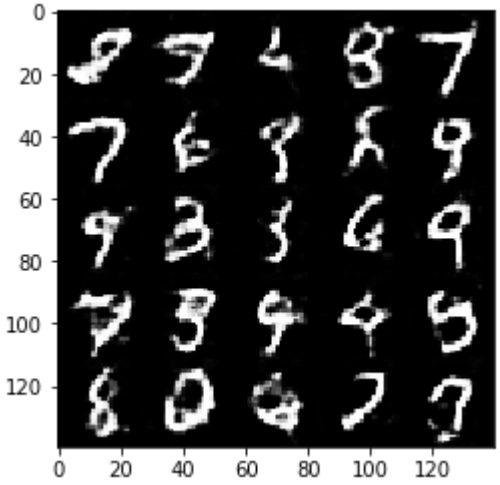

```
Epoch 1/2 Step 500... Discriminator Loss: 0.9871... Generator Loss: 0.8365
Epoch 1/2 Step 600... Discriminator Loss: 1.0110... Generator Loss: 2.0424
```



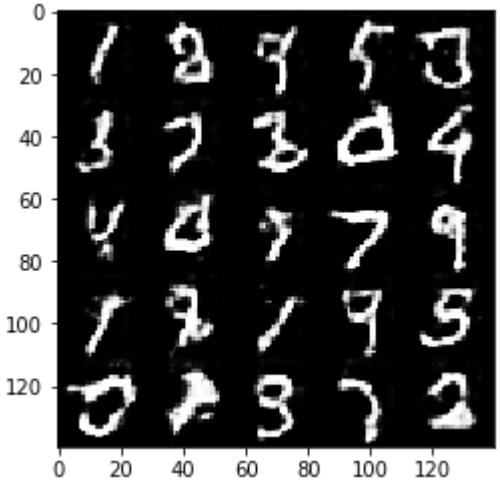

```
Epoch 2/2 Step 700... Discriminator Loss: 0.9823... Generator Loss: 1.6035
Epoch 2/2 Step 800... Discriminator Loss: 0.9671... Generator Loss: 1.5338
```



```
Epoch 2/2 Step 900... Discriminator Loss: 1.0328... Generator Loss: 0.8993
Epoch 2/2 Step 1000... Discriminator Loss: 1.1566... Generator Loss: 0.6283
```



```
Epoch 2/2 Step 1100... Discriminator Loss: 1.1967... Generator Loss: 1.5912
Epoch 2/2 Step 1200... Discriminator Loss: 0.9186... Generator Loss: 1.4104
```

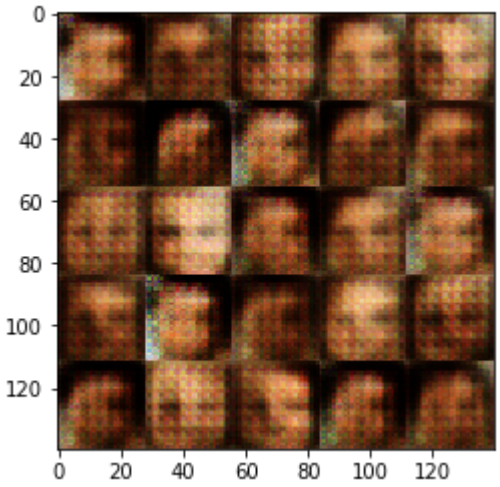


CelebA

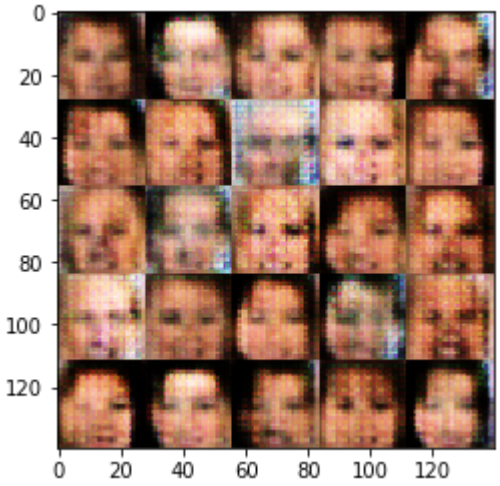
Run your GANs on CelebA. It will take around 20 minutes on the average GPU to run one epoch. You can run the whole epoch or stop when it starts to generate realistic faces.

In [13]:
```
batch_size = 100
z_dim = 100
learning_rate = 0.001
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, 'img_ali
gn_celeba/*.jpg')))
with tf.Graph().as_default():
    train(epochs, batch_size, z_dim, learning_rate, beta1, celeba_dataset.get_
batches,
          celeba_dataset.shape, celeba_dataset.image_mode)
```

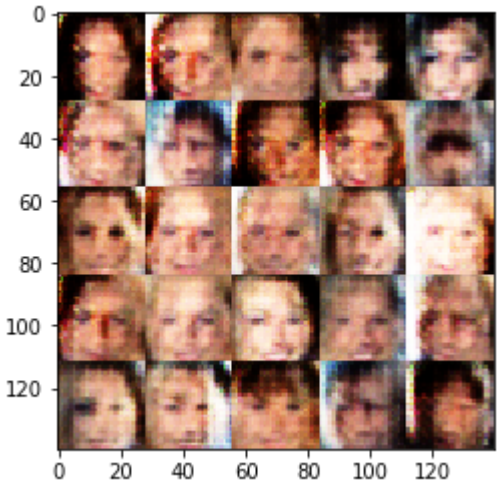
```
Epoch 1/10 Step 100... Discriminator Loss: 0.8909... Generator Loss: 7.2165
Epoch 1/10 Step 200... Discriminator Loss: 1.3784... Generator Loss: 0.9670
```



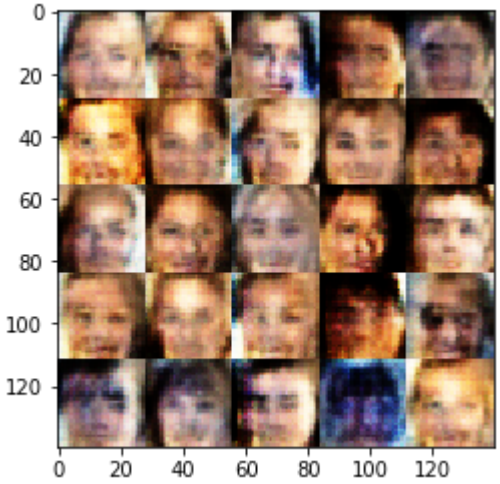

```
Epoch 1/10 Step 300... Discriminator Loss: 1.8202... Generator Loss: 0.3601
Epoch 1/10 Step 400... Discriminator Loss: 1.5800... Generator Loss: 1.5312
```



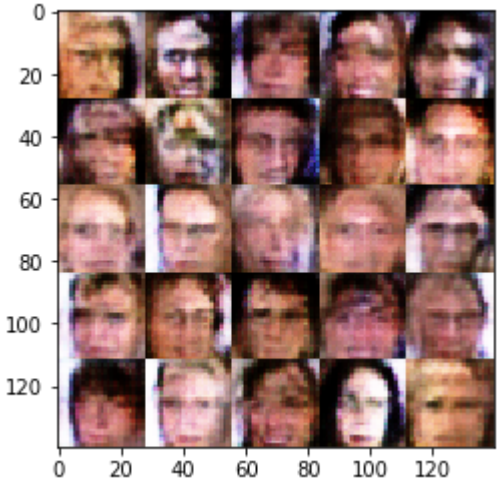

```
Epoch 1/10 Step 500... Discriminator Loss: 1.0135... Generator Loss: 1.2996
Epoch 1/10 Step 600... Discriminator Loss: 1.4273... Generator Loss: 0.6740
```



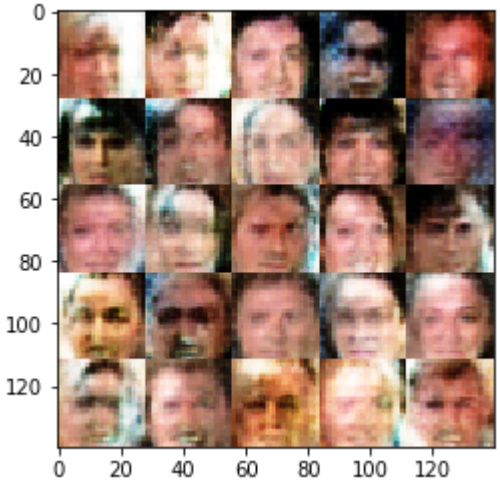

```
Epoch 1/10 Step 700... Discriminator Loss: 1.1787... Generator Loss: 1.1168
Epoch 1/10 Step 800... Discriminator Loss: 1.1926... Generator Loss: 1.0748
```



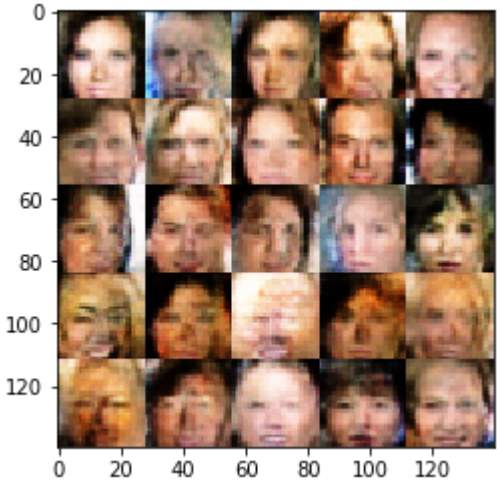
```
Epoch 1/10 Step 900... Discriminator Loss: 1.0586... Generator Loss: 1.4138
Epoch 1/10 Step 1000... Discriminator Loss: 1.0887... Generator Loss: 1.0351
```



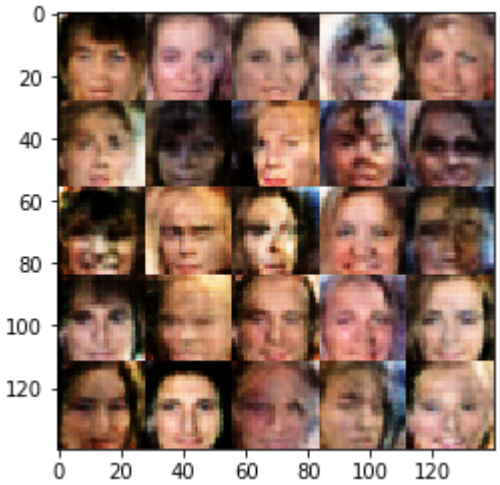
```
Epoch 1/10 Step 1100... Discriminator Loss: 1.2731... Generator Loss: 0.5898
Epoch 1/10 Step 1200... Discriminator Loss: 1.2558... Generator Loss: 0.8986
```



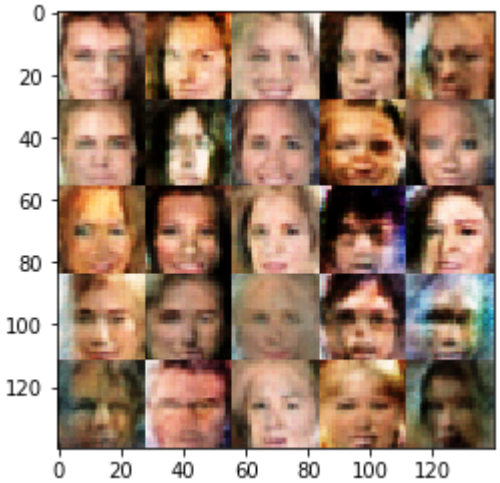
```
Epoch 1/10 Step 1300... Discriminator Loss: 1.0778... Generator Loss: 0.9165
Epoch 1/10 Step 1400... Discriminator Loss: 1.2917... Generator Loss: 1.5605
```



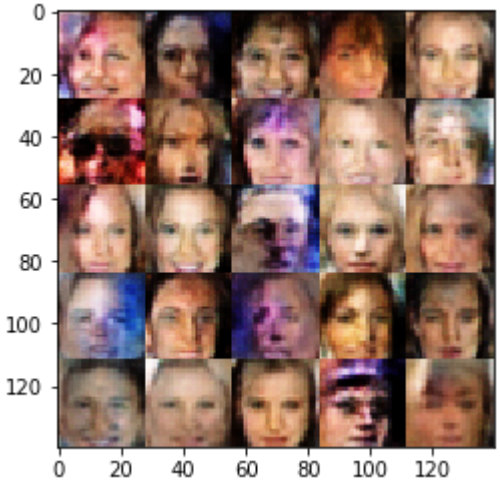

```
Epoch 1/10 Step 1500... Discriminator Loss: 1.1434... Generator Loss: 1.0066
Epoch 1/10 Step 1600... Discriminator Loss: 1.4270... Generator Loss: 0.5738
```



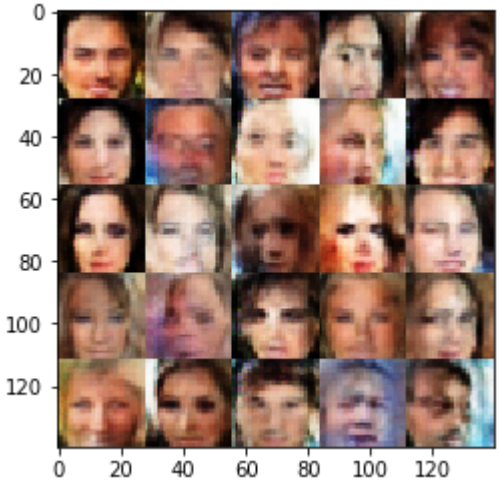

```
Epoch 1/10 Step 1700... Discriminator Loss: 1.5123... Generator Loss: 0.4768
Epoch 1/10 Step 1800... Discriminator Loss: 1.4688... Generator Loss: 0.4953
```



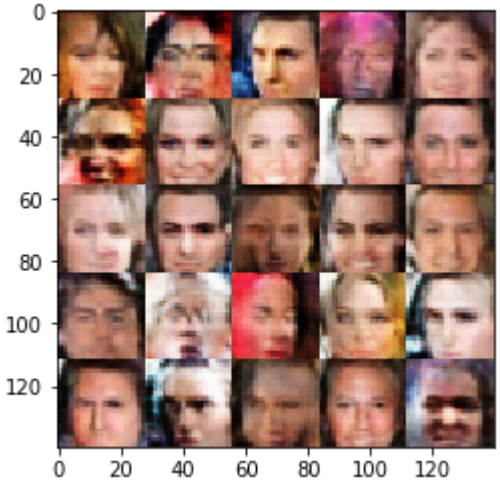

```
Epoch 1/10 Step 1900... Discriminator Loss: 1.3410... Generator Loss: 0.7351
Epoch 1/10 Step 2000... Discriminator Loss: 1.1605... Generator Loss: 1.0251
```



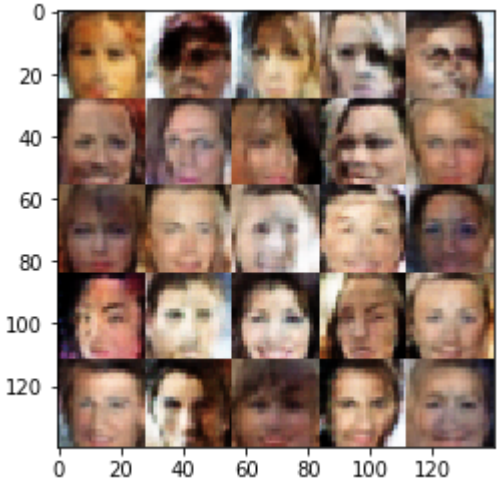

```
Epoch 2/10 Step 2100... Discriminator Loss: 1.1804... Generator Loss: 1.3038
Epoch 2/10 Step 2200... Discriminator Loss: 1.2133... Generator Loss: 0.8016
```



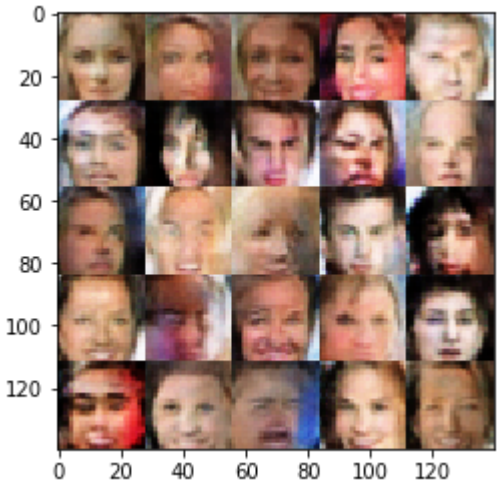

```
Epoch 2/10 Step 2300... Discriminator Loss: 1.2107... Generator Loss: 0.7572
Epoch 2/10 Step 2400... Discriminator Loss: 1.1345... Generator Loss: 0.8287
```



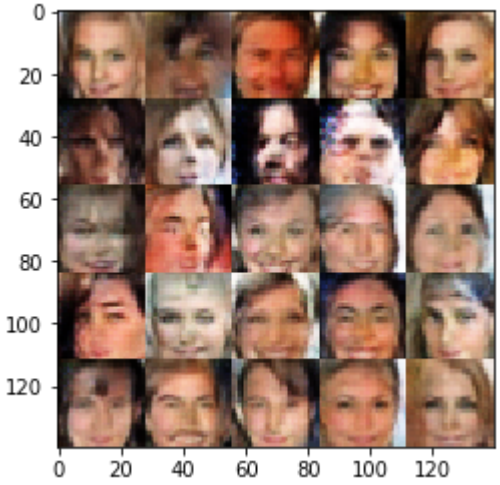

```
Epoch 2/10 Step 2500... Discriminator Loss: 1.1583... Generator Loss: 0.8788
Epoch 2/10 Step 2600... Discriminator Loss: 1.2045... Generator Loss: 0.8100
```



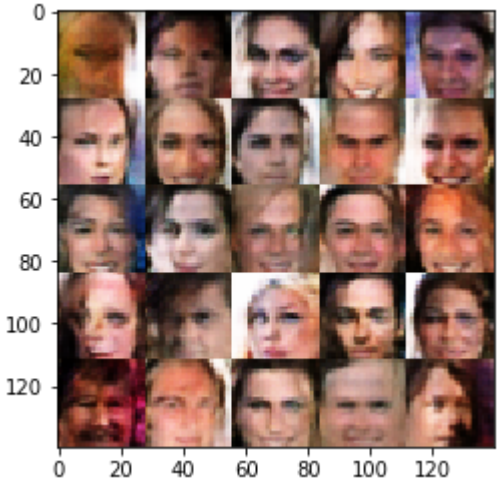

```
Epoch 2/10 Step 2700... Discriminator Loss: 1.2114... Generator Loss: 0.8019
Epoch 2/10 Step 2800... Discriminator Loss: 1.4035... Generator Loss: 1.0449
```



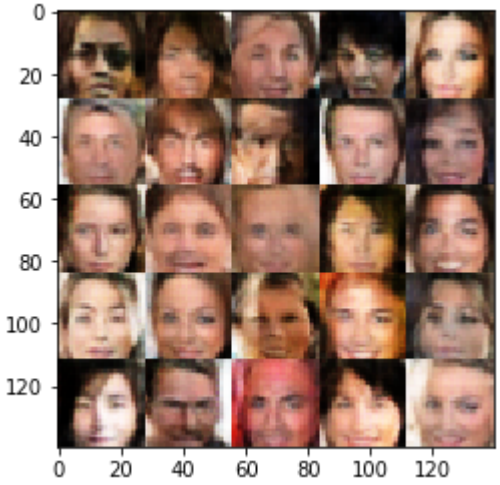

```
Epoch 2/10 Step 2900... Discriminator Loss: 1.2717... Generator Loss: 1.1017
Epoch 2/10 Step 3000... Discriminator Loss: 1.3051... Generator Loss: 0.9425
```



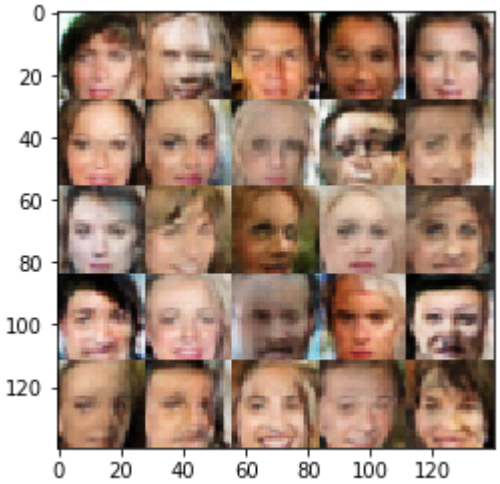

```
Epoch 2/10 Step 3100... Discriminator Loss: 1.4092... Generator Loss: 0.5075
Epoch 2/10 Step 3200... Discriminator Loss: 1.2189... Generator Loss: 0.9134
```



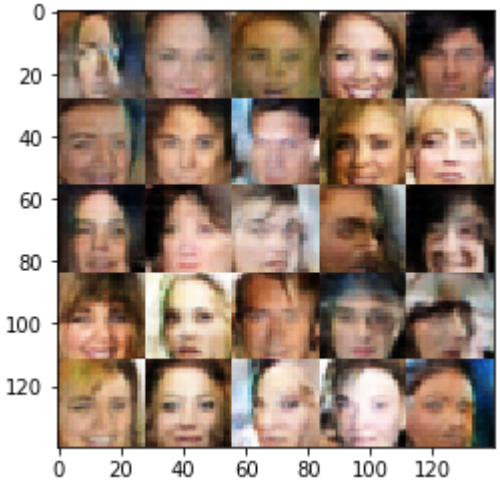

```
Epoch 2/10 Step 3300... Discriminator Loss: 1.3483... Generator Loss: 0.6103
Epoch 2/10 Step 3400... Discriminator Loss: 1.2157... Generator Loss: 0.8205
```



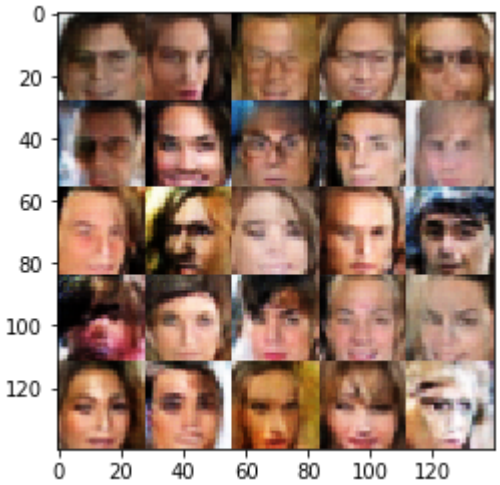

```
Epoch 2/10 Step 3500... Discriminator Loss: 1.1891... Generator Loss: 0.8521
Epoch 2/10 Step 3600... Discriminator Loss: 1.3039... Generator Loss: 0.6588
```



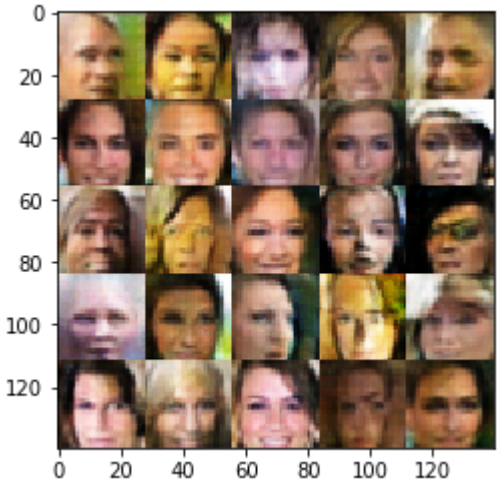

```
Epoch 2/10 Step 3700... Discriminator Loss: 1.3965... Generator Loss: 0.9548
Epoch 2/10 Step 3800... Discriminator Loss: 1.2117... Generator Loss: 0.7812
```



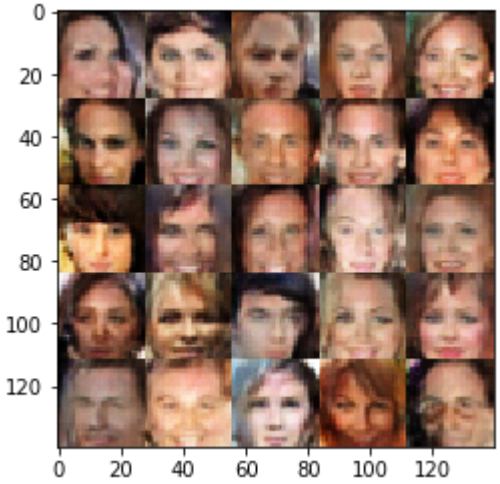

```
Epoch 2/10 Step 3900... Discriminator Loss: 1.1089... Generator Loss: 1.0154
Epoch 2/10 Step 4000... Discriminator Loss: 1.1935... Generator Loss: 0.7056
```



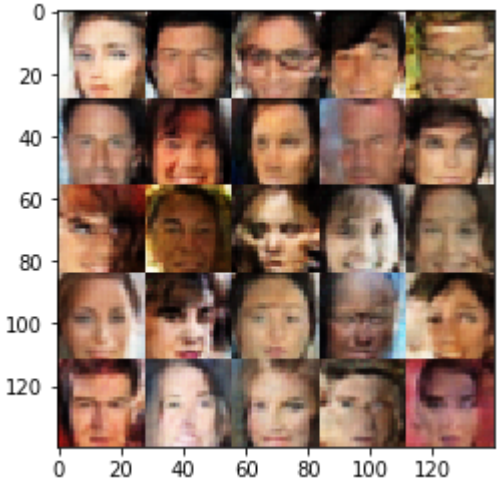

```
Epoch 3/10 Step 4100... Discriminator Loss: 1.2100... Generator Loss: 0.9396
Epoch 3/10 Step 4200... Discriminator Loss: 1.2863... Generator Loss: 0.6047
```



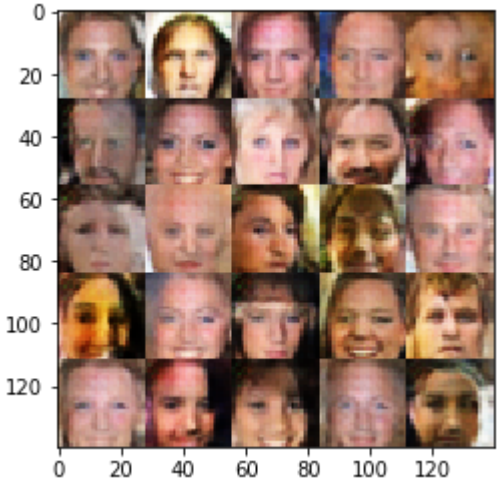

```
Epoch 3/10 Step 4300... Discriminator Loss: 1.2858... Generator Loss: 0.7618
Epoch 3/10 Step 4400... Discriminator Loss: 1.4174... Generator Loss: 1.1773
```



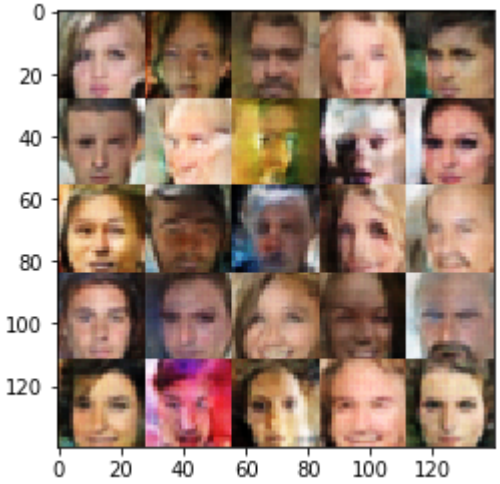

```
Epoch 3/10 Step 4500... Discriminator Loss: 1.2477... Generator Loss: 1.2386
Epoch 3/10 Step 4600... Discriminator Loss: 1.3692... Generator Loss: 1.2904
```



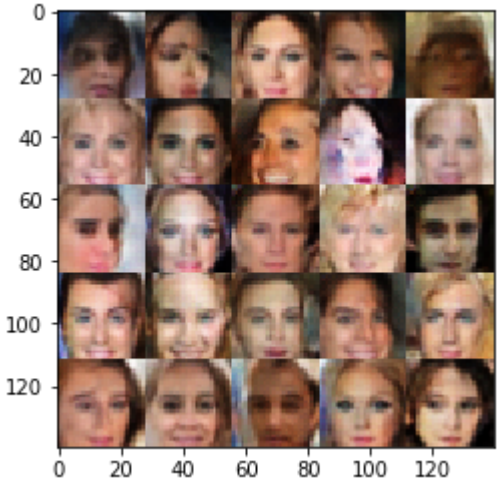

```
Epoch 3/10 Step 4700... Discriminator Loss: 1.2473... Generator Loss: 0.6488
Epoch 3/10 Step 4800... Discriminator Loss: 1.1414... Generator Loss: 0.8399
```



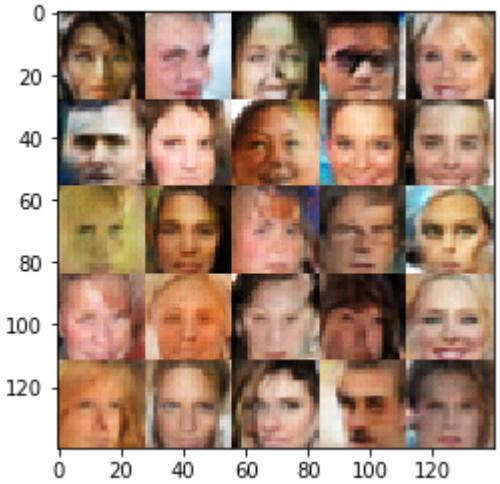

```
Epoch 3/10 Step 4900... Discriminator Loss: 1.2873... Generator Loss: 0.7177
Epoch 3/10 Step 5000... Discriminator Loss: 1.1839... Generator Loss: 1.0469
```



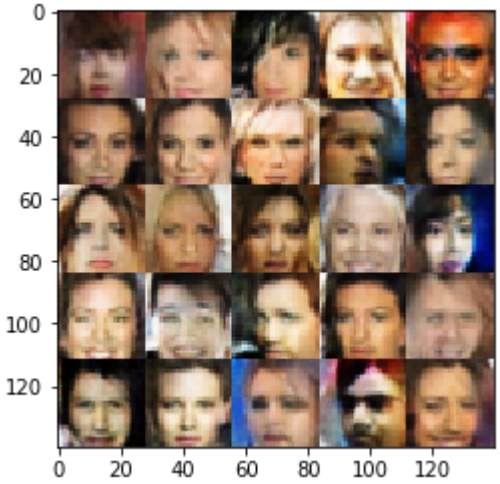

```
Epoch 3/10 Step 5100... Discriminator Loss: 1.3760... Generator Loss: 0.5581
Epoch 3/10 Step 5200... Discriminator Loss: 1.1877... Generator Loss: 0.9223
```



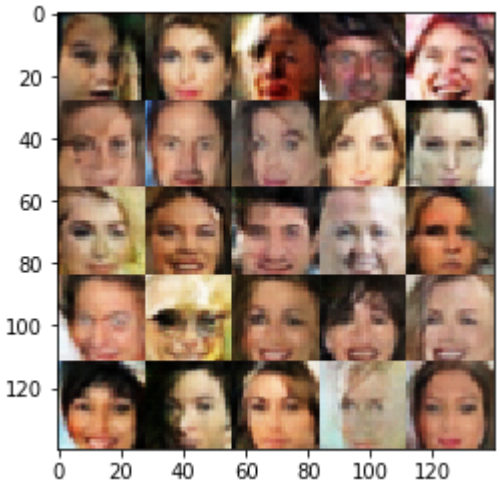

```
Epoch 3/10 Step 5300... Discriminator Loss: 1.3307... Generator Loss: 0.6413
Epoch 3/10 Step 5400... Discriminator Loss: 1.0907... Generator Loss: 0.9363
```



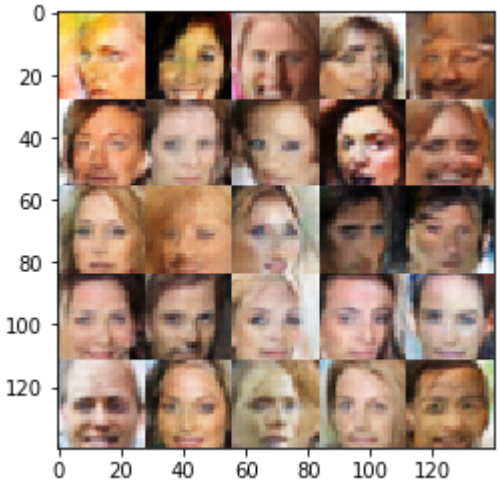

```
Epoch 3/10 Step 5500... Discriminator Loss: 1.1473... Generator Loss: 0.8261
Epoch 3/10 Step 5600... Discriminator Loss: 1.2798... Generator Loss: 0.9936
```



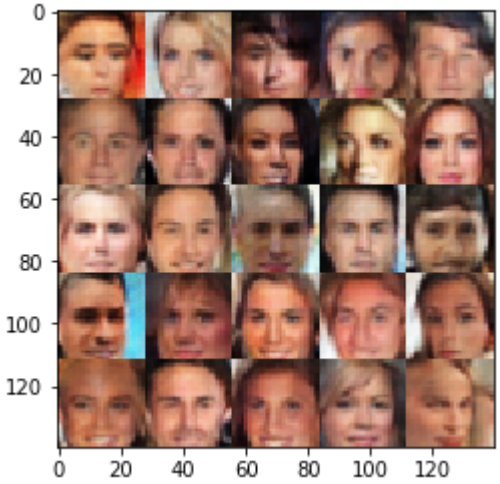

```
Epoch 3/10 Step 5700... Discriminator Loss: 1.2338... Generator Loss: 0.7025
Epoch 3/10 Step 5800... Discriminator Loss: 1.3385... Generator Loss: 0.8517
```



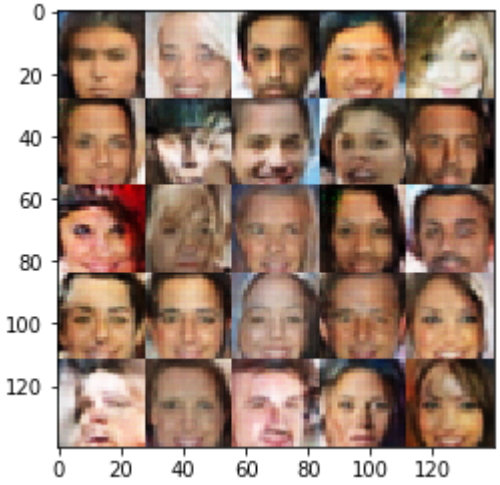

```
Epoch 3/10 Step 5900... Discriminator Loss: 1.1964... Generator Loss: 0.7898
Epoch 3/10 Step 6000... Discriminator Loss: 1.2315... Generator Loss: 0.9980
```



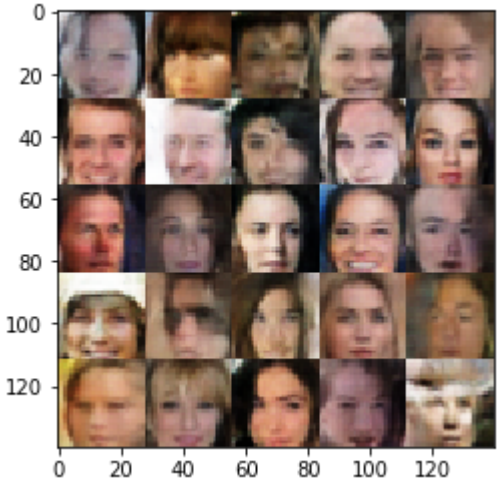

```
Epoch 4/10 Step 6100... Discriminator Loss: 1.3021... Generator Loss: 0.6611
Epoch 4/10 Step 6200... Discriminator Loss: 1.1418... Generator Loss: 0.8423
```



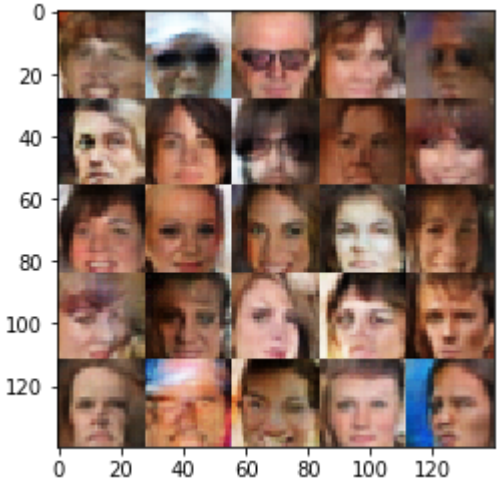

```
Epoch 4/10 Step 6300... Discriminator Loss: 0.9679... Generator Loss: 1.0665
Epoch 4/10 Step 6400... Discriminator Loss: 1.2072... Generator Loss: 1.1677
```



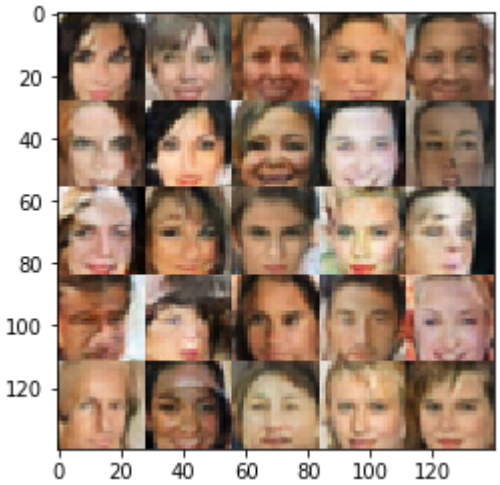

```
Epoch 4/10 Step 6500... Discriminator Loss: 1.2659... Generator Loss: 0.9358
Epoch 4/10 Step 6600... Discriminator Loss: 1.5664... Generator Loss: 0.4079
```



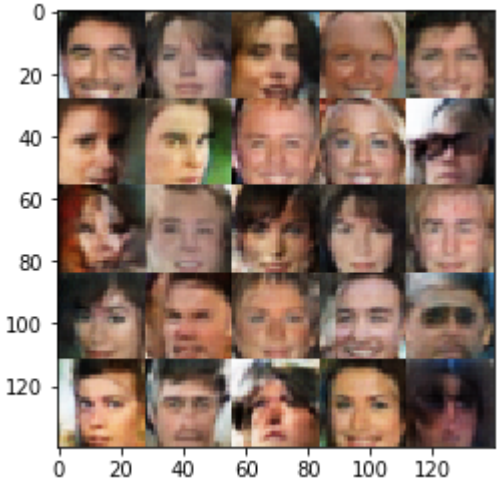

```
Epoch 4/10 Step 6700... Discriminator Loss: 1.3034... Generator Loss: 0.9231
Epoch 4/10 Step 6800... Discriminator Loss: 1.4979... Generator Loss: 1.5128
```



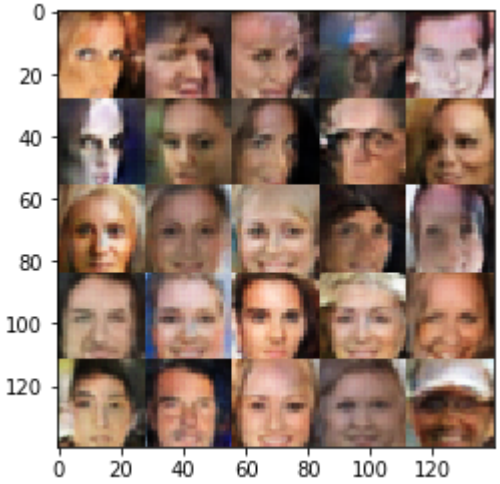

```
Epoch 4/10 Step 6900... Discriminator Loss: 1.2241... Generator Loss: 1.1783
Epoch 4/10 Step 7000... Discriminator Loss: 1.1465... Generator Loss: 1.1680
```



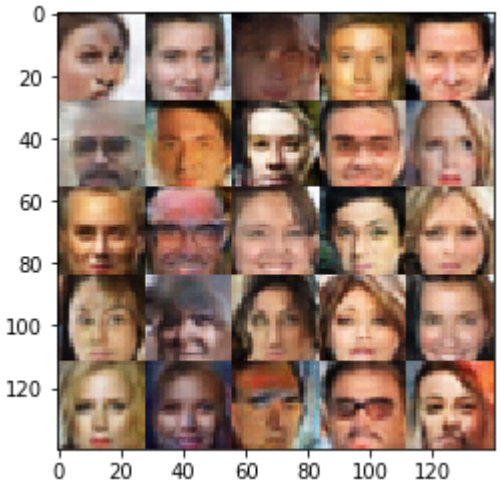

```
Epoch 4/10 Step 7100... Discriminator Loss: 1.4609... Generator Loss: 0.5132
Epoch 4/10 Step 7200... Discriminator Loss: 0.8539... Generator Loss: 1.7703
```



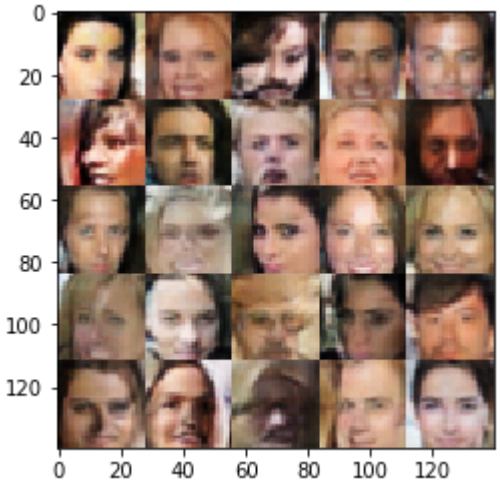

```
Epoch 4/10 Step 7300... Discriminator Loss: 1.3062... Generator Loss: 0.5626
Epoch 4/10 Step 7400... Discriminator Loss: 1.0064... Generator Loss: 1.0605
```



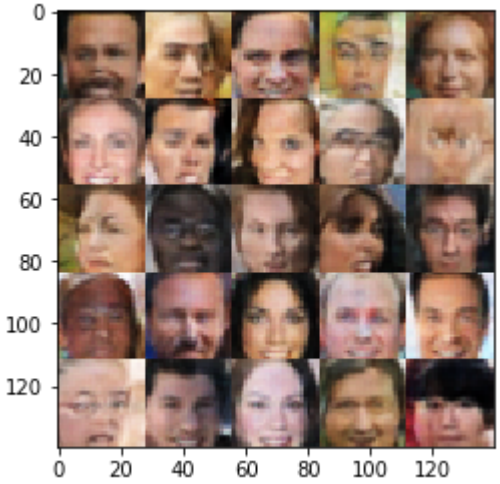

```
Epoch 4/10 Step 7500... Discriminator Loss: 1.2085... Generator Loss: 0.6595
Epoch 4/10 Step 7600... Discriminator Loss: 1.1357... Generator Loss: 0.7785
```



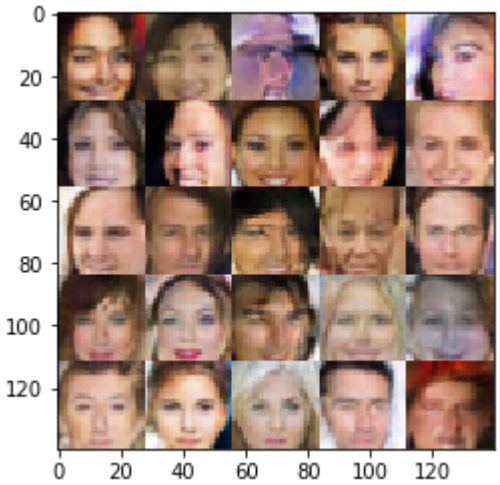

```
Epoch 4/10 Step 7700... Discriminator Loss: 0.9850... Generator Loss: 0.9850
Epoch 4/10 Step 7800... Discriminator Loss: 1.1037... Generator Loss: 1.5030
```



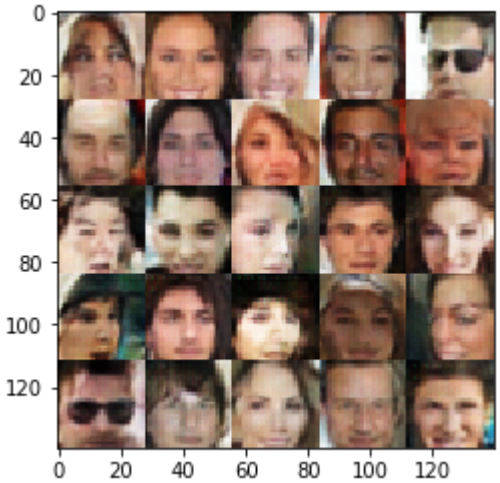

```
Epoch 4/10 Step 7900... Discriminator Loss: 1.0389... Generator Loss: 1.0344
Epoch 4/10 Step 8000... Discriminator Loss: 0.9718... Generator Loss: 1.0571
```



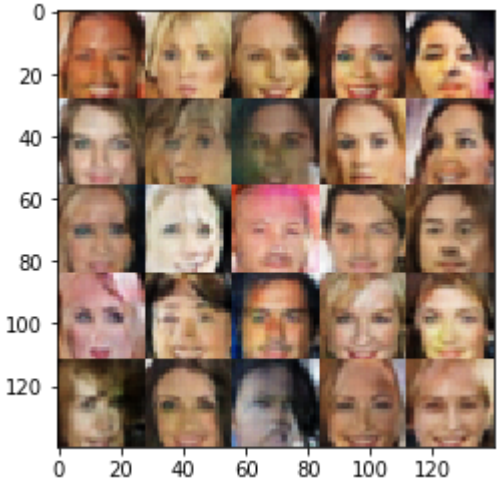

```
Epoch 4/10 Step 8100... Discriminator Loss: 1.6344... Generator Loss: 0.3823
Epoch 5/10 Step 8200... Discriminator Loss: 0.8499... Generator Loss: 1.3415
```



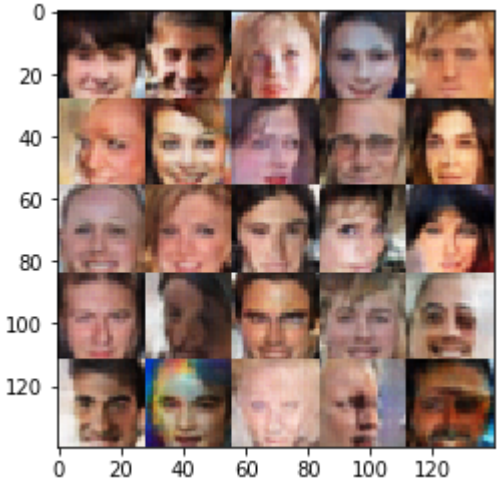

```
Epoch 5/10 Step 8300... Discriminator Loss: 0.9825... Generator Loss: 0.9410
Epoch 5/10 Step 8400... Discriminator Loss: 0.9521... Generator Loss: 2.3838
```



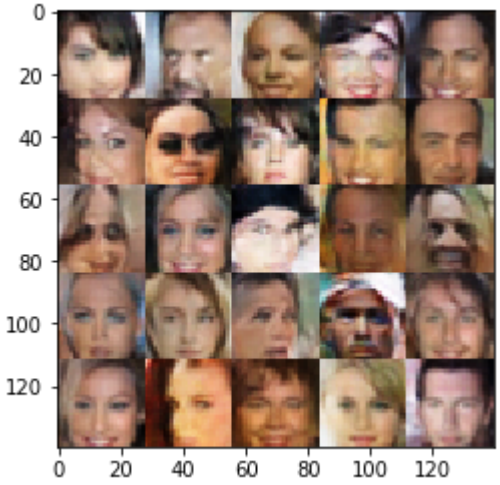

```
Epoch 5/10 Step 8500... Discriminator Loss: 1.4490... Generator Loss: 0.4666
Epoch 5/10 Step 8600... Discriminator Loss: 0.9270... Generator Loss: 1.0435
```



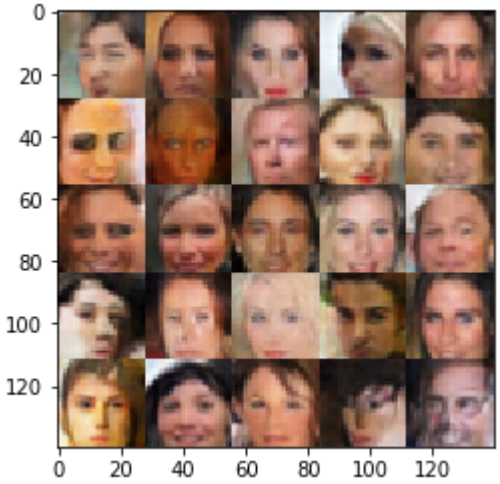

```
Epoch 5/10 Step 8700... Discriminator Loss: 1.3896... Generator Loss: 0.5044
Epoch 5/10 Step 8800... Discriminator Loss: 1.5966... Generator Loss: 0.4126
```



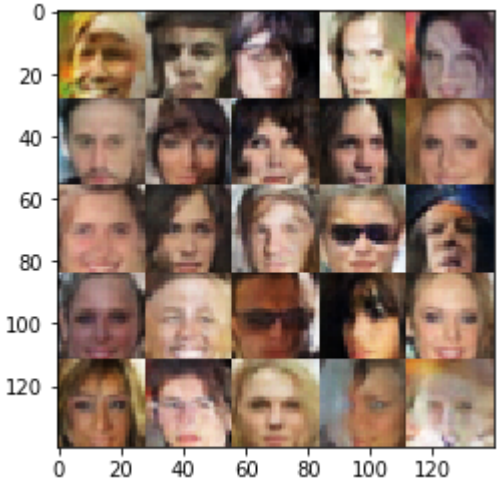

```
Epoch 5/10 Step 8900... Discriminator Loss: 1.0084... Generator Loss: 0.8425
Epoch 5/10 Step 9000... Discriminator Loss: 1.6626... Generator Loss: 0.4017
```



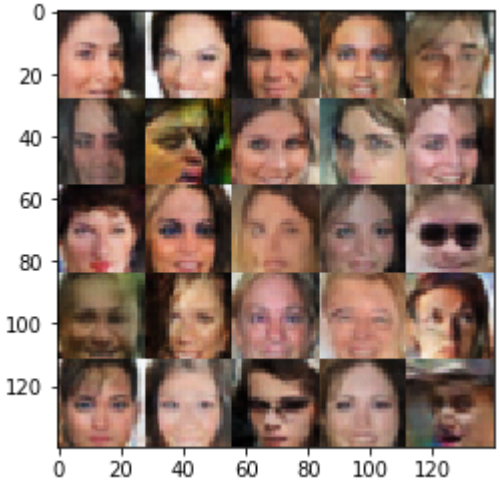

```
Epoch 5/10 Step 9100... Discriminator Loss: 0.7331... Generator Loss: 1.6519
Epoch 5/10 Step 9200... Discriminator Loss: 1.2794... Generator Loss: 0.6354
```



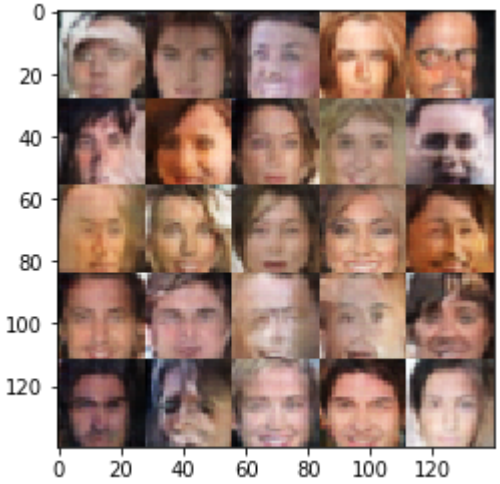

```
Epoch 5/10 Step 9300... Discriminator Loss: 1.1200... Generator Loss: 2.5173
Epoch 5/10 Step 9400... Discriminator Loss: 0.8361... Generator Loss: 1.2461
```



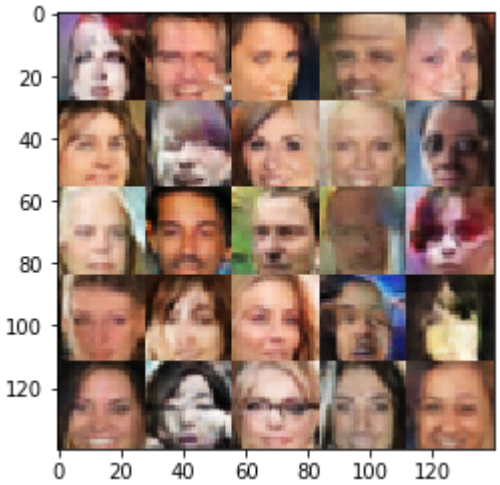

```
Epoch 5/10 Step 9500... Discriminator Loss: 1.0366... Generator Loss: 1.3121
Epoch 5/10 Step 9600... Discriminator Loss: 1.0346... Generator Loss: 0.8390
```



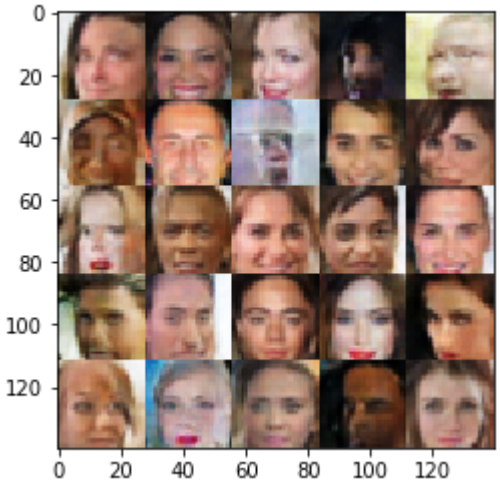

```
Epoch 5/10 Step 9700... Discriminator Loss: 1.0464... Generator Loss: 0.8066
Epoch 5/10 Step 9800... Discriminator Loss: 1.0441... Generator Loss: 2.6045
```



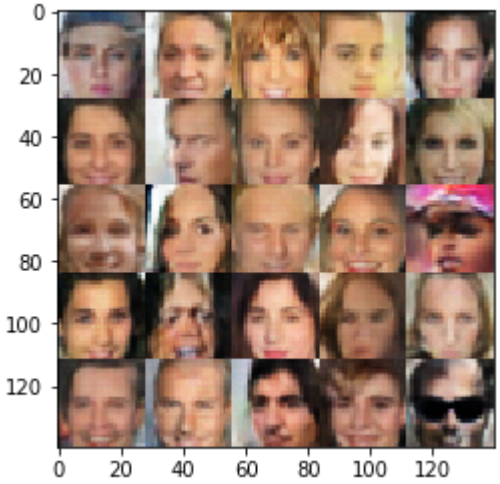

```
Epoch 5/10 Step 9900... Discriminator Loss: 1.1521... Generator Loss: 0.7071
Epoch 5/10 Step 10000... Discriminator Loss: 1.0889... Generator Loss: 0.7792
```



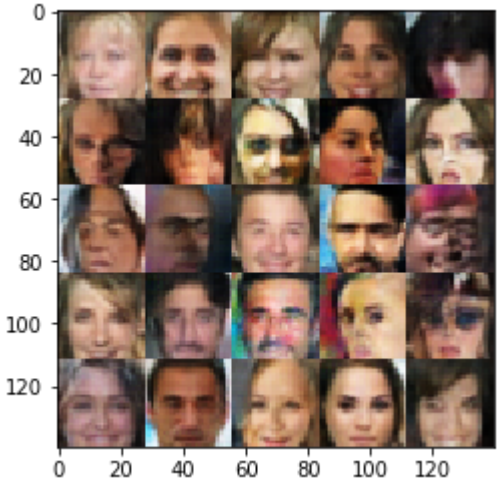

```
Epoch 5/10 Step 10100... Discriminator Loss: 1.1419... Generator Loss: 0.7081
Epoch 6/10 Step 10200... Discriminator Loss: 0.8788... Generator Loss: 1.0564
```



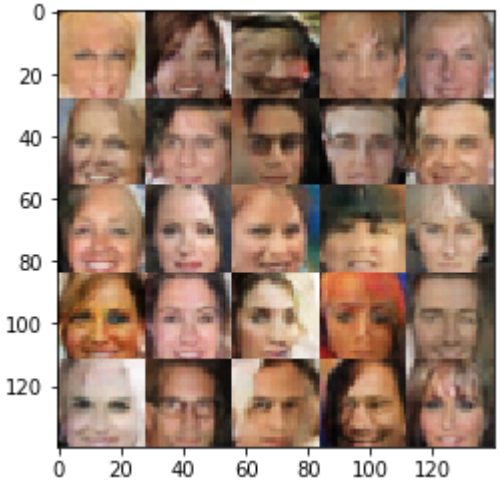

```
Epoch 6/10 Step 10300... Discriminator Loss: 0.6937... Generator Loss: 2.0354
Epoch 6/10 Step 10400... Discriminator Loss: 1.1404... Generator Loss: 0.7320
```



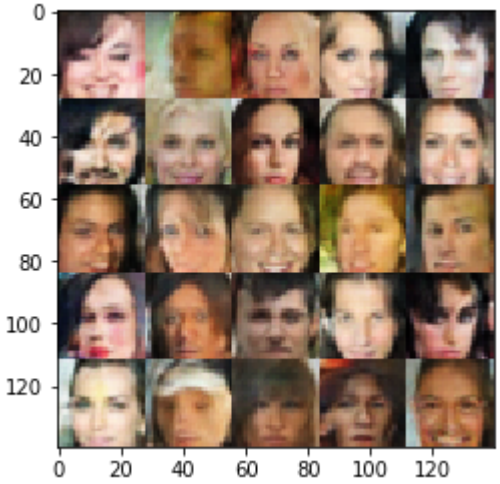

```
Epoch 6/10 Step 10500... Discriminator Loss: 1.2913... Generator Loss: 0.6209
Epoch 6/10 Step 10600... Discriminator Loss: 1.1040... Generator Loss: 1.8562
```



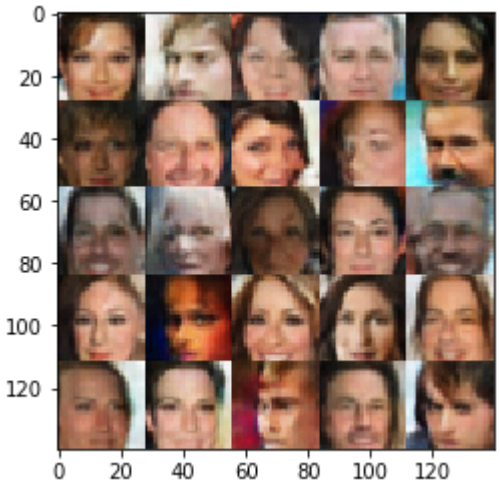

```
Epoch 6/10 Step 10700... Discriminator Loss: 1.0881... Generator Loss: 0.7519
Epoch 6/10 Step 10800... Discriminator Loss: 1.1079... Generator Loss: 0.7465
```



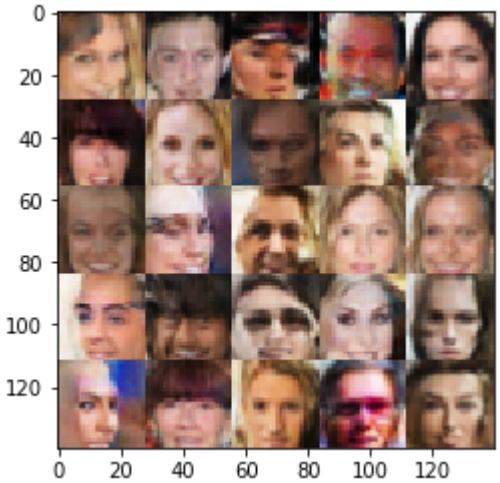

```
Epoch 6/10 Step 10900... Discriminator Loss: 1.3706... Generator Loss: 0.6503
Epoch 6/10 Step 11000... Discriminator Loss: 0.6757... Generator Loss: 1.5784
```



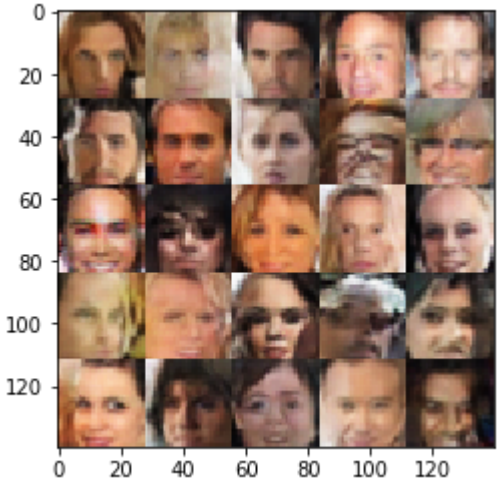

```
Epoch 6/10 Step 11100... Discriminator Loss: 0.5609... Generator Loss: 1.9248
Epoch 6/10 Step 11200... Discriminator Loss: 0.6826... Generator Loss: 1.5734
```



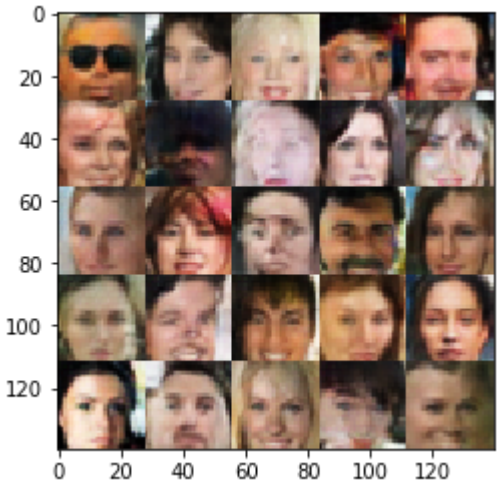

```
Epoch 6/10 Step 11300... Discriminator Loss: 0.8602... Generator Loss: 1.1141
Epoch 6/10 Step 11400... Discriminator Loss: 0.6684... Generator Loss: 1.7148
```



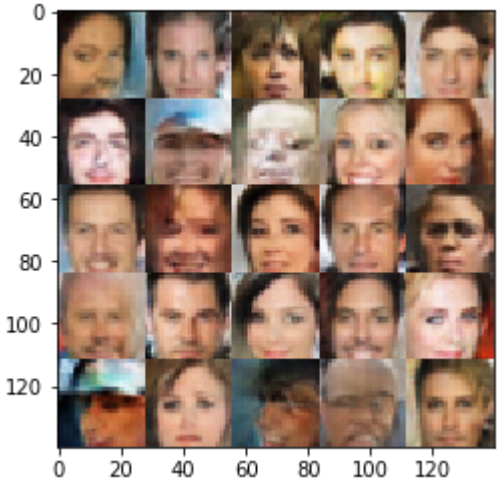

```
Epoch 6/10 Step 11500... Discriminator Loss: 0.7721... Generator Loss: 1.4165
Epoch 6/10 Step 11600... Discriminator Loss: 0.7372... Generator Loss: 1.3463
```



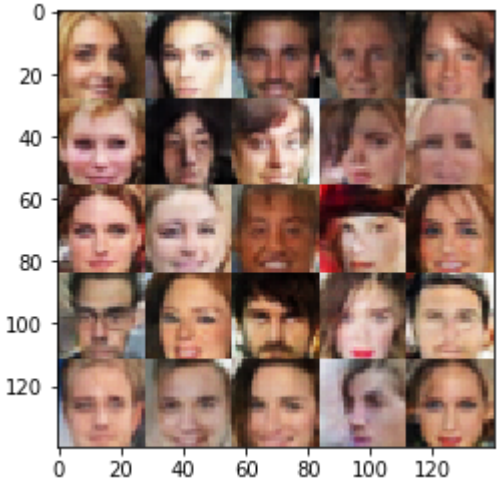

```
Epoch 6/10 Step 11700... Discriminator Loss: 1.1804... Generator Loss: 1.9755
Epoch 6/10 Step 11800... Discriminator Loss: 1.0765... Generator Loss: 0.8019
```



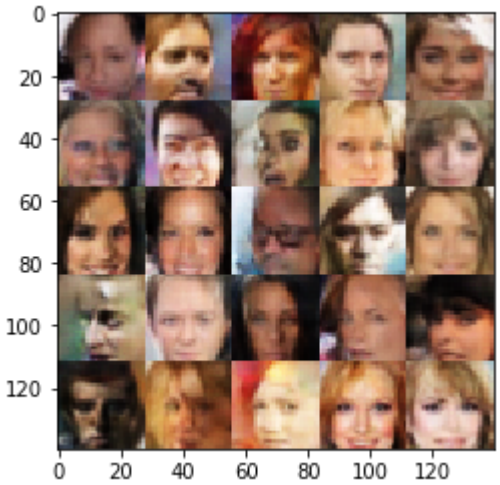

```
Epoch 6/10 Step 11900... Discriminator Loss: 0.8584... Generator Loss: 1.1284
Epoch 6/10 Step 12000... Discriminator Loss: 0.7901... Generator Loss: 1.5687
```



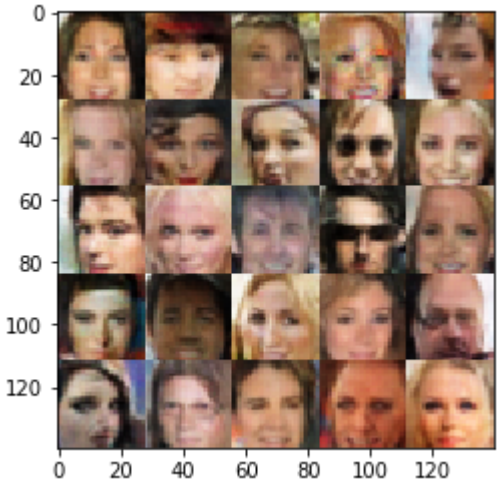

```
Epoch 6/10 Step 12100... Discriminator Loss: 0.6936... Generator Loss: 1.9752
Epoch 7/10 Step 12200... Discriminator Loss: 1.1518... Generator Loss: 0.7564
```



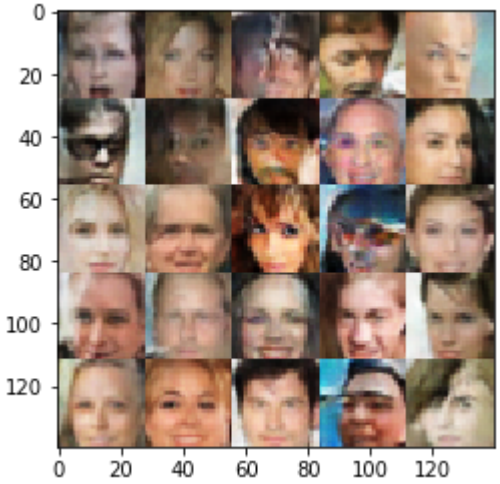
```
Epoch 7/10 Step 12300... Discriminator Loss: 0.6208... Generator Loss: 1.6586
Epoch 7/10 Step 12400... Discriminator Loss: 1.9795... Generator Loss: 0.3403
```



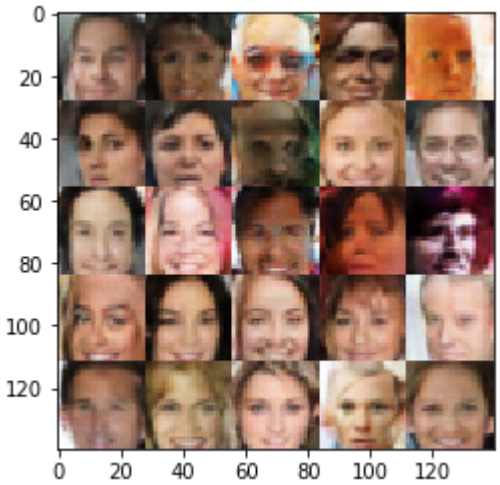
```
Epoch 7/10 Step 12500... Discriminator Loss: 0.8976... Generator Loss: 1.1428
Epoch 7/10 Step 12600... Discriminator Loss: 1.2575... Generator Loss: 0.7040
```



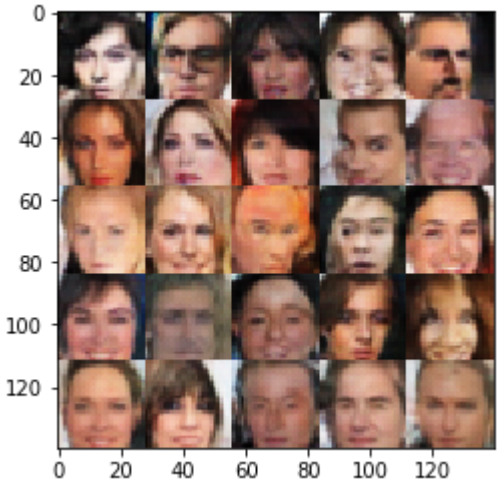
```
Epoch 7/10 Step 12700... Discriminator Loss: 0.9803... Generator Loss: 1.6467
Epoch 7/10 Step 12800... Discriminator Loss: 1.6282... Generator Loss: 0.4597
```



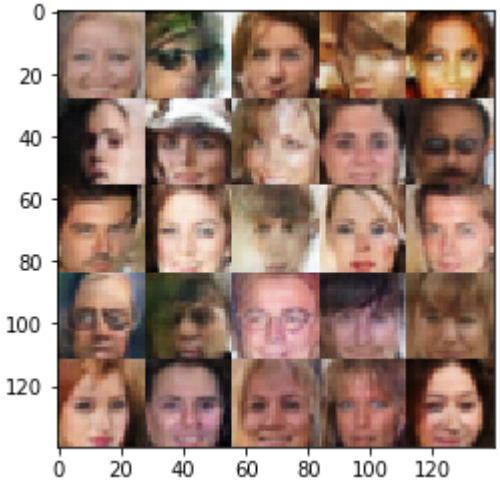

```
Epoch 7/10 Step 12900... Discriminator Loss: 0.6216... Generator Loss: 1.9313
Epoch 7/10 Step 13000... Discriminator Loss: 0.6875... Generator Loss: 1.5094
```



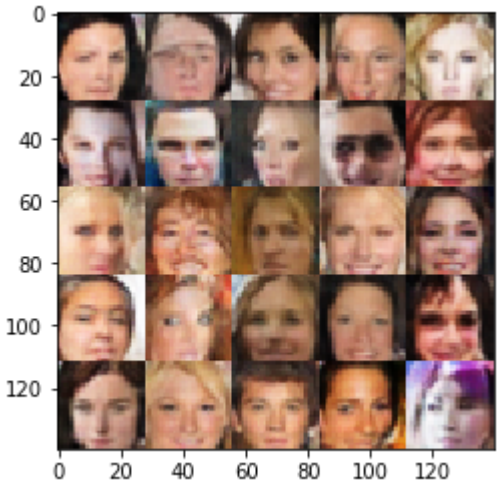

```
Epoch 7/10 Step 13100... Discriminator Loss: 0.5660... Generator Loss: 2.2910
Epoch 7/10 Step 13200... Discriminator Loss: 0.6470... Generator Loss: 1.6638
```



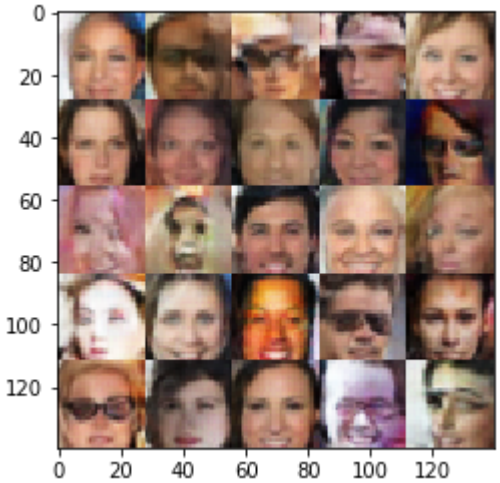

```
Epoch 7/10 Step 13300... Discriminator Loss: 0.7188... Generator Loss: 1.7542
Epoch 7/10 Step 13400... Discriminator Loss: 0.8508... Generator Loss: 1.2997
```



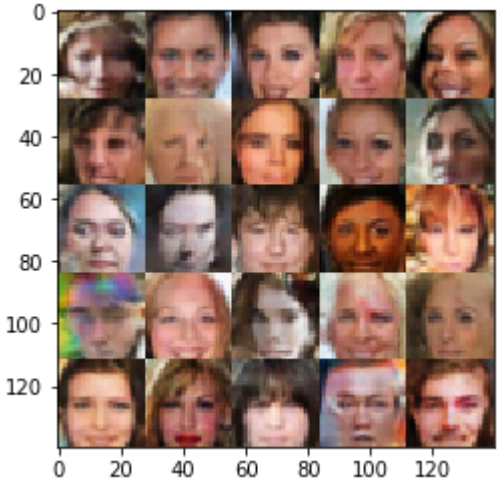

```
Epoch 7/10 Step 13500... Discriminator Loss: 0.5814... Generator Loss: 1.9570
Epoch 7/10 Step 13600... Discriminator Loss: 0.4921... Generator Loss: 2.7285
```



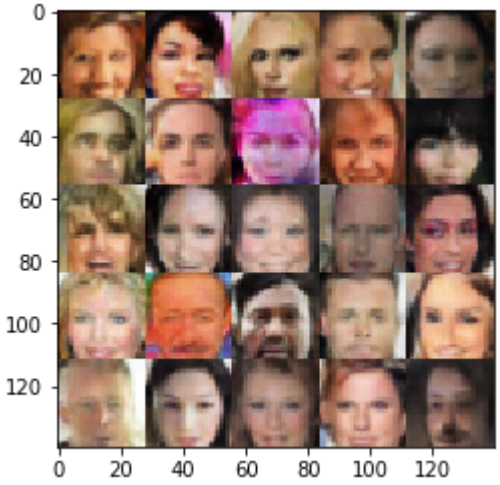

```
Epoch 7/10 Step 13700... Discriminator Loss: 0.8080... Generator Loss: 1.9230
Epoch 7/10 Step 13800... Discriminator Loss: 0.7941... Generator Loss: 1.3360
```



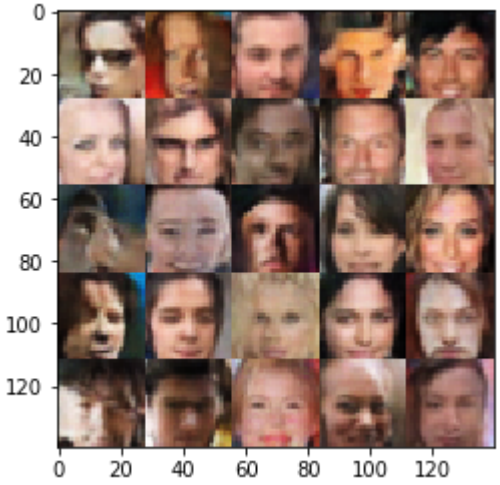

```
Epoch 7/10 Step 13900... Discriminator Loss: 1.3405... Generator Loss: 2.5133
Epoch 7/10 Step 14000... Discriminator Loss: 0.5740... Generator Loss: 3.2807
```



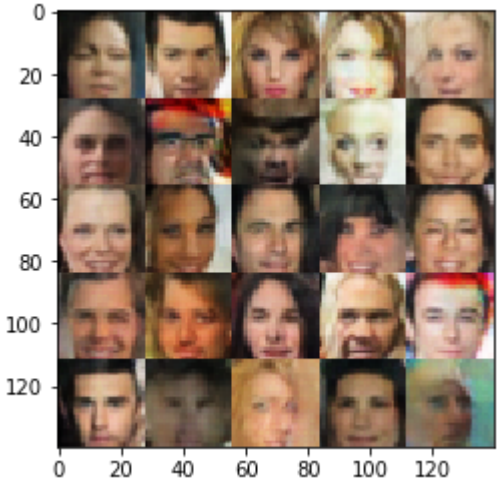

```
Epoch 7/10 Step 14100... Discriminator Loss: 0.9635... Generator Loss: 0.9789
Epoch 8/10 Step 14200... Discriminator Loss: 0.6221... Generator Loss: 2.0480
```



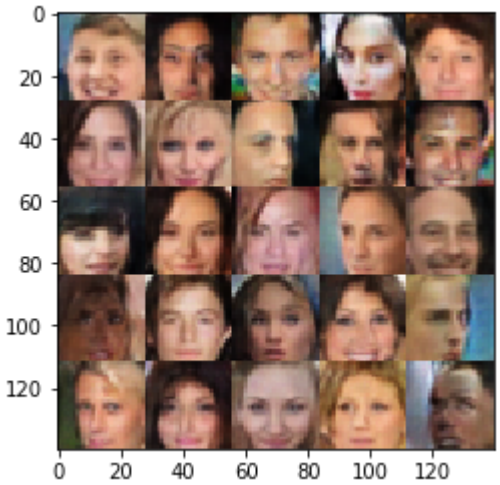

```
Epoch 8/10 Step 14300... Discriminator Loss: 0.5757... Generator Loss: 2.4019
Epoch 8/10 Step 14400... Discriminator Loss: 1.2835... Generator Loss: 0.6926
```



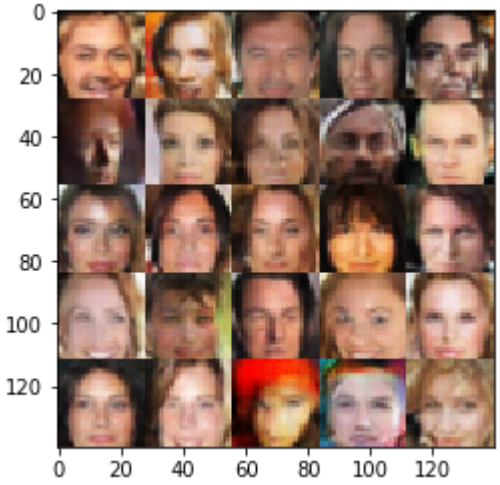

```
Epoch 8/10 Step 14500... Discriminator Loss: 1.1732... Generator Loss: 2.7163
Epoch 8/10 Step 14600... Discriminator Loss: 0.6589... Generator Loss: 1.9700
```



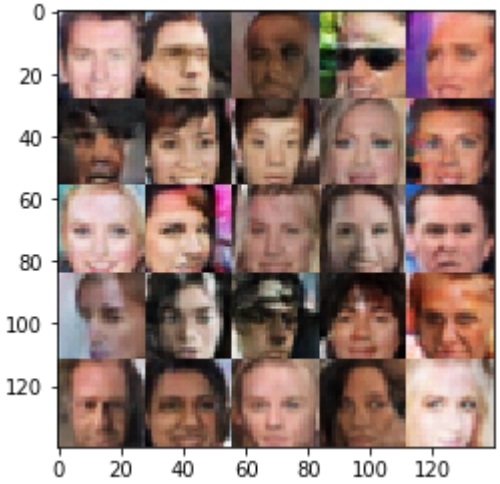

```
Epoch 8/10 Step 14700... Discriminator Loss: 0.7288... Generator Loss: 1.4022
Epoch 8/10 Step 14800... Discriminator Loss: 1.0543... Generator Loss: 3.3078
```



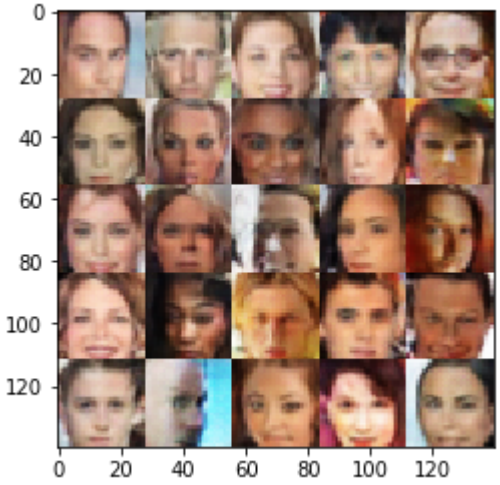

```
Epoch 8/10 Step 14900... Discriminator Loss: 0.9807... Generator Loss: 0.9442
Epoch 8/10 Step 15000... Discriminator Loss: 1.2173... Generator Loss: 0.7143
```



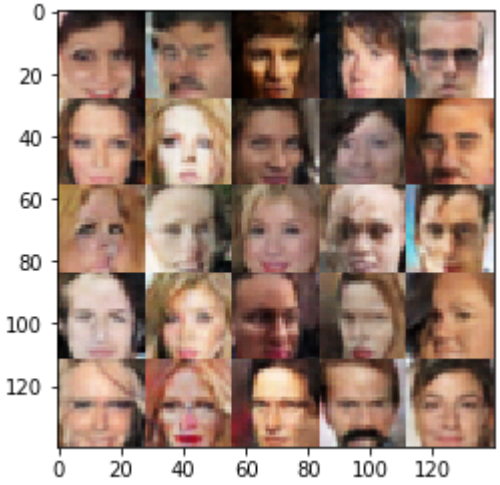

```
Epoch 8/10 Step 15100... Discriminator Loss: 0.5644... Generator Loss: 2.6923
Epoch 8/10 Step 15200... Discriminator Loss: 0.6143... Generator Loss: 1.7630
```



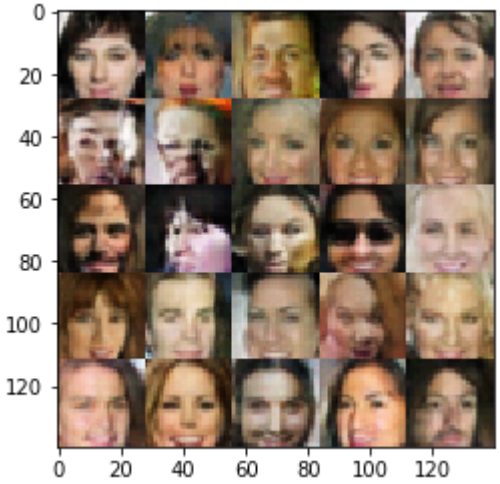

```
Epoch 8/10 Step 15300... Discriminator Loss: 1.0501... Generator Loss: 0.9630
Epoch 8/10 Step 15400... Discriminator Loss: 0.8715... Generator Loss: 1.1768
```



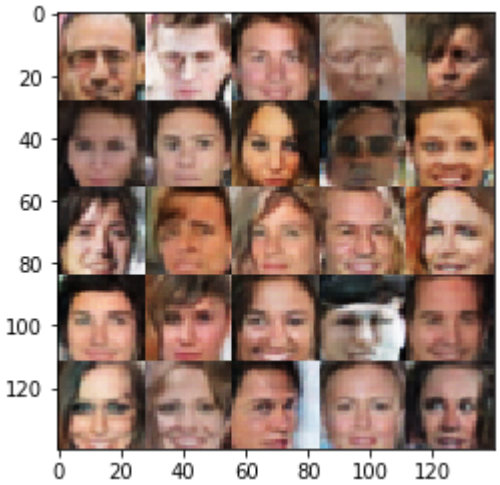

```
Epoch 8/10 Step 15500... Discriminator Loss: 1.4208... Generator Loss: 3.3123
Epoch 8/10 Step 15600... Discriminator Loss: 0.7820... Generator Loss: 1.2411
```



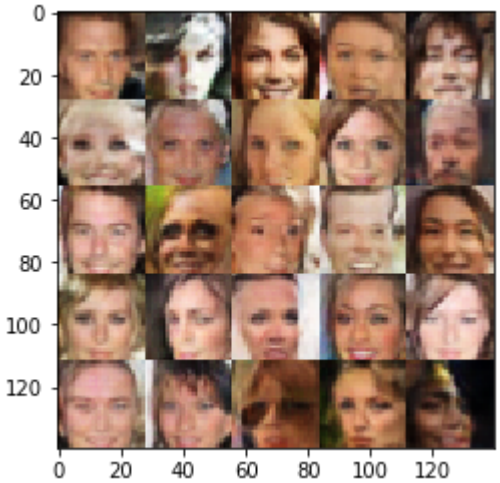

```
Epoch 8/10 Step 15700... Discriminator Loss: 1.0957... Generator Loss: 0.8920
Epoch 8/10 Step 15800... Discriminator Loss: 0.5388... Generator Loss: 2.1672
```



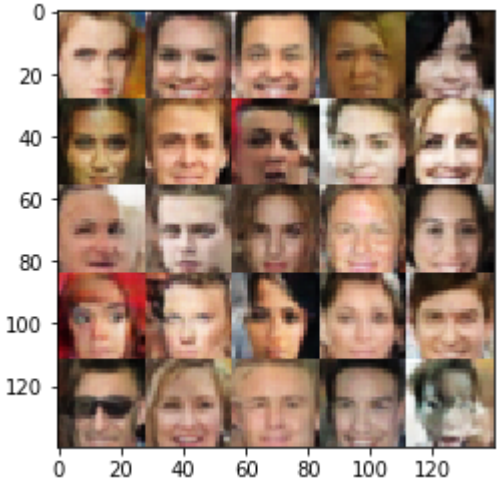

```
Epoch 8/10 Step 15900... Discriminator Loss: 0.6758... Generator Loss: 2.6309
Epoch 8/10 Step 16000... Discriminator Loss: 0.8373... Generator Loss: 1.3934
```



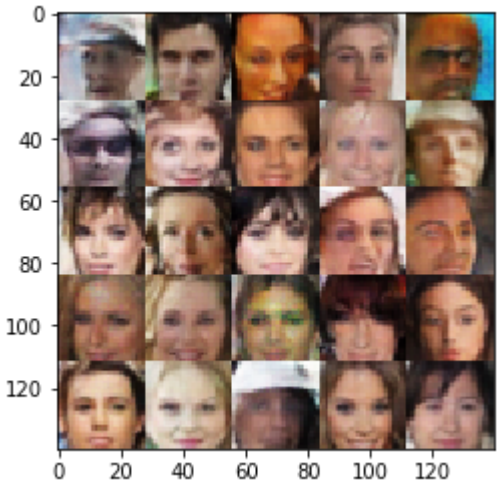

```
Epoch 8/10 Step 16100... Discriminator Loss: 1.0906... Generator Loss: 0.9577
Epoch 8/10 Step 16200... Discriminator Loss: 0.4826... Generator Loss: 3.0848
```



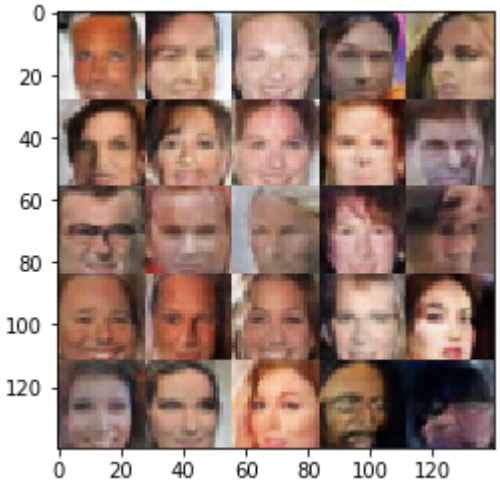

```
Epoch 9/10 Step 16300... Discriminator Loss: 0.5476... Generator Loss: 3.5772
Epoch 9/10 Step 16400... Discriminator Loss: 0.5251... Generator Loss: 2.9838
```



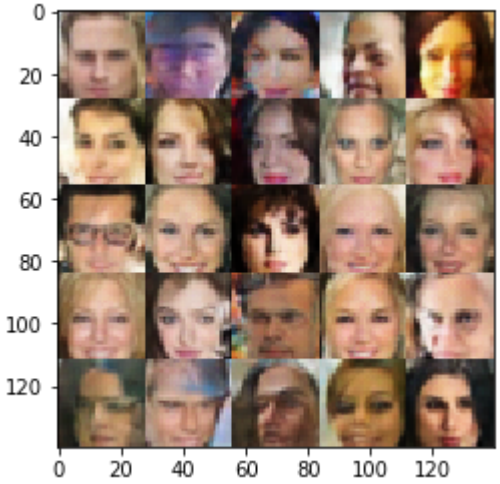

```
Epoch 9/10 Step 16500... Discriminator Loss: 0.6470... Generator Loss: 2.1271
Epoch 9/10 Step 16600... Discriminator Loss: 1.6525... Generator Loss: 3.7084
```



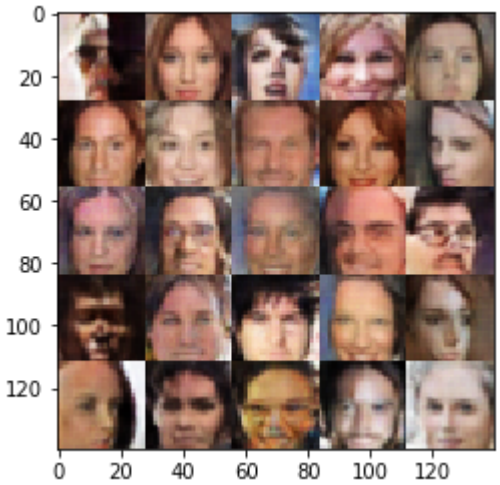

```
Epoch 9/10 Step 16700... Discriminator Loss: 0.6384... Generator Loss: 1.7020
Epoch 9/10 Step 16800... Discriminator Loss: 1.0606... Generator Loss: 1.4058
```



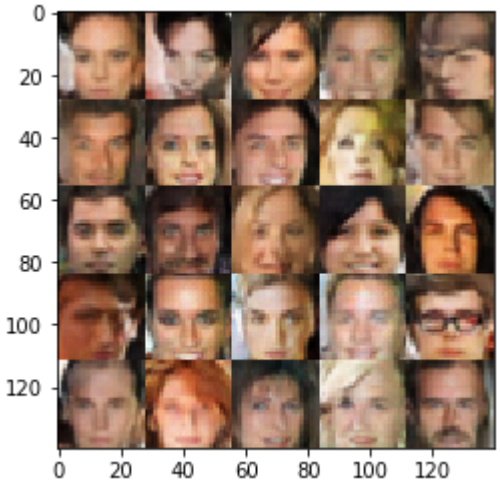

```
Epoch 9/10 Step 16900... Discriminator Loss: 0.5297... Generator Loss: 2.2234
Epoch 9/10 Step 17000... Discriminator Loss: 0.7934... Generator Loss: 1.2441
```



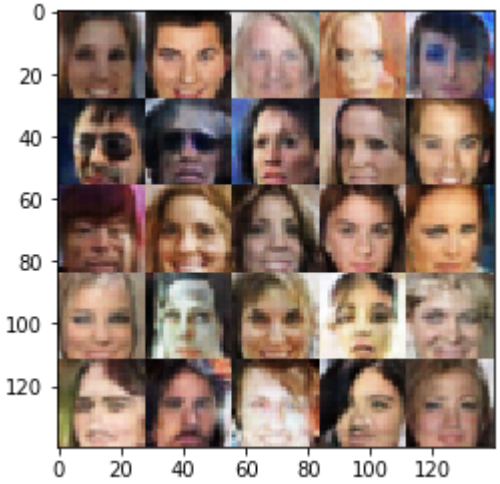

```
Epoch 9/10 Step 17100... Discriminator Loss: 1.5492... Generator Loss: 0.5308
Epoch 9/10 Step 17200... Discriminator Loss: 0.5186... Generator Loss: 2.9459
```



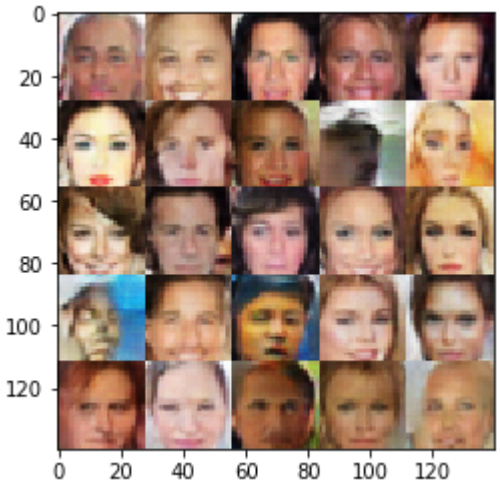

```
Epoch 9/10 Step 17300... Discriminator Loss: 1.0485... Generator Loss: 1.9147
Epoch 9/10 Step 17400... Discriminator Loss: 1.0553... Generator Loss: 0.8876
```



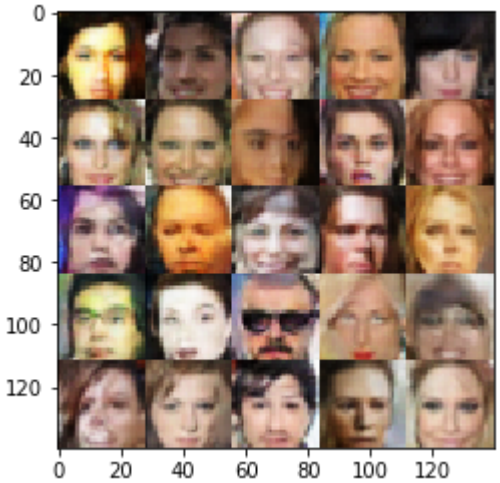

```
Epoch 9/10 Step 17500... Discriminator Loss: 0.6622... Generator Loss: 1.7790
Epoch 9/10 Step 17600... Discriminator Loss: 0.9243... Generator Loss: 2.3801
```



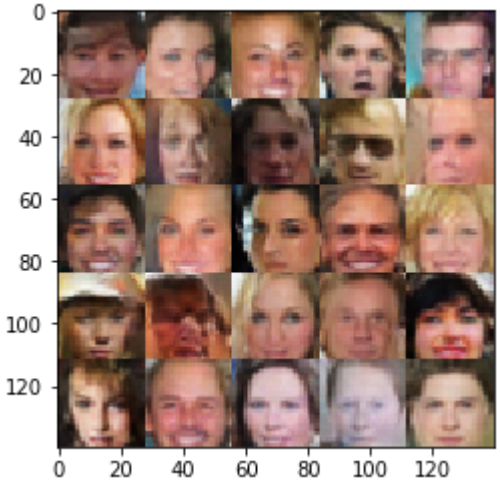

```
Epoch 9/10 Step 17700... Discriminator Loss: 0.4541... Generator Loss: 3.0104
Epoch 9/10 Step 17800... Discriminator Loss: 0.7639... Generator Loss: 1.3596
```



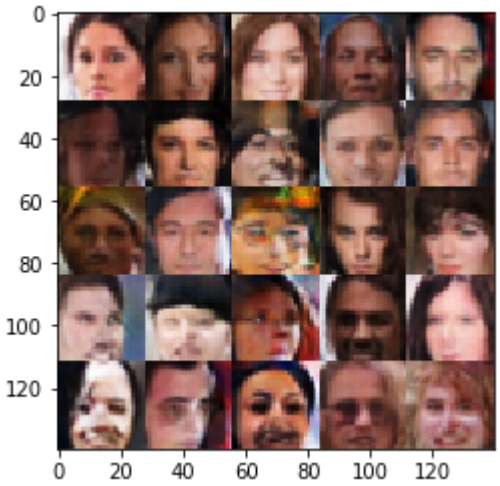

```
Epoch 9/10 Step 17900... Discriminator Loss: 0.5773... Generator Loss: 1.8643
Epoch 9/10 Step 18000... Discriminator Loss: 0.7622... Generator Loss: 1.4037
```



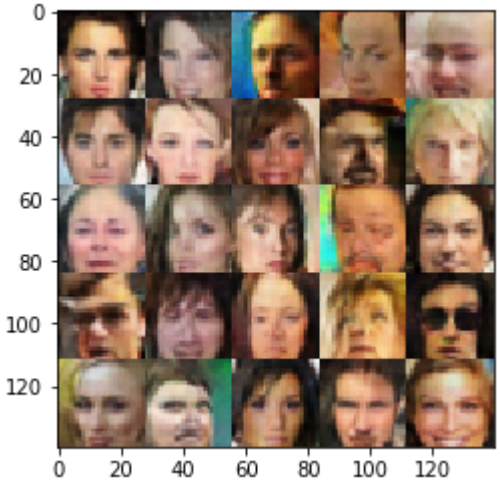

```
Epoch 9/10 Step 18100... Discriminator Loss: 0.5029... Generator Loss: 2.8727
Epoch 9/10 Step 18200... Discriminator Loss: 2.6602... Generator Loss: 0.1742
```



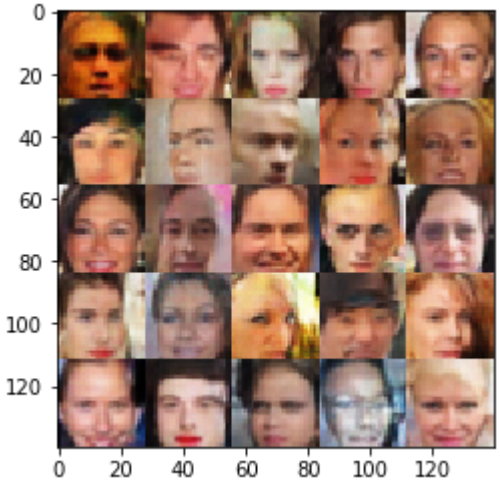
```
Epoch 10/10 Step 18300... Discriminator Loss: 0.5795... Generator Loss: 1.971
6
Epoch 10/10 Step 18400... Discriminator Loss: 0.5329... Generator Loss: 2.097
0
```



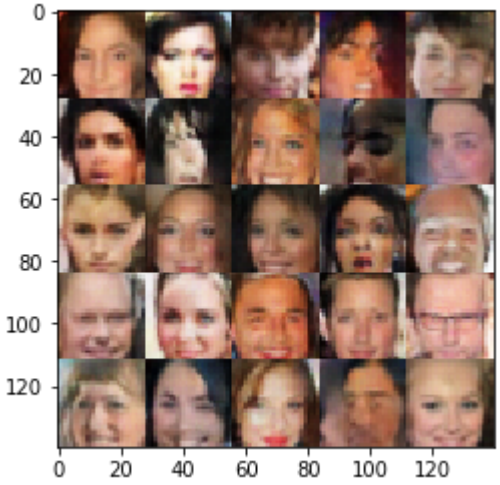
```
Epoch 10/10 Step 18500... Discriminator Loss: 2.0892... Generator Loss: 0.363
4
Epoch 10/10 Step 18600... Discriminator Loss: 0.7803... Generator Loss: 1.367
8
```



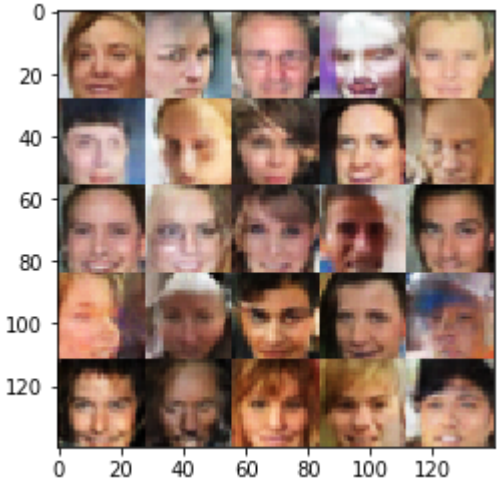
```
Epoch 10/10 Step 18700... Discriminator Loss: 0.5692... Generator Loss: 2.441
0
Epoch 10/10 Step 18800... Discriminator Loss: 0.9679... Generator Loss: 1.039
1
```



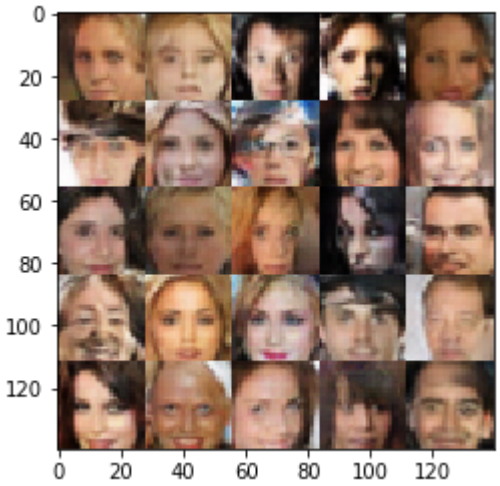

```
Epoch 10/10 Step 18900... Discriminator Loss: 0.8768... Generator Loss: 1.173
7
Epoch 10/10 Step 19000... Discriminator Loss: 0.8692... Generator Loss: 1.294
5
```



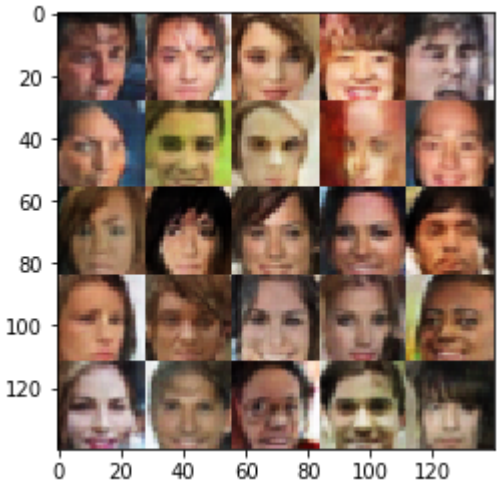

```
Epoch 10/10 Step 19100... Discriminator Loss: 0.7383... Generator Loss: 1.379
3
Epoch 10/10 Step 19200... Discriminator Loss: 0.8215... Generator Loss: 1.279
3
```



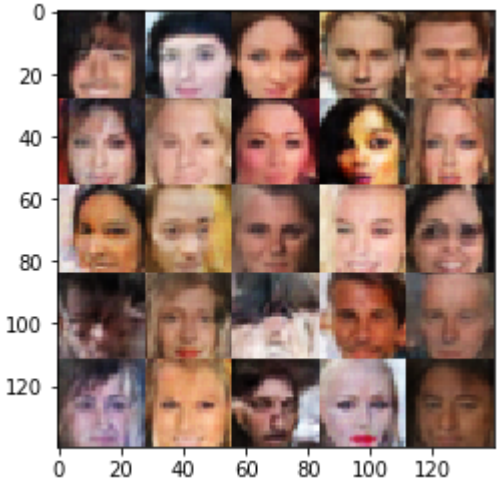

```
Epoch 10/10 Step 19300... Discriminator Loss: 0.4919... Generator Loss: 2.895
9
Epoch 10/10 Step 19400... Discriminator Loss: 0.5650... Generator Loss: 1.937
1
```



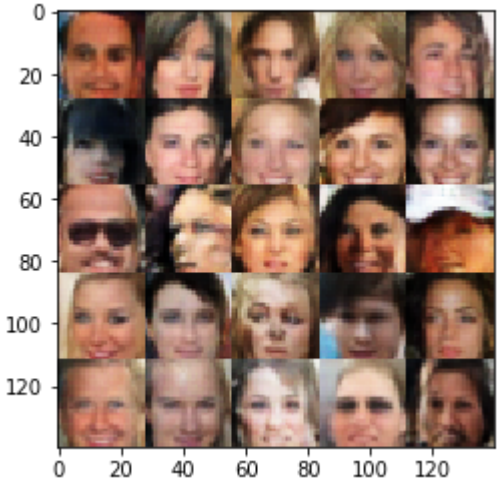

```
Epoch 10/10 Step 19500... Discriminator Loss: 1.5104... Generator Loss: 0.605
8
Epoch 10/10 Step 19600... Discriminator Loss: 0.6400... Generator Loss: 3.641
5
```



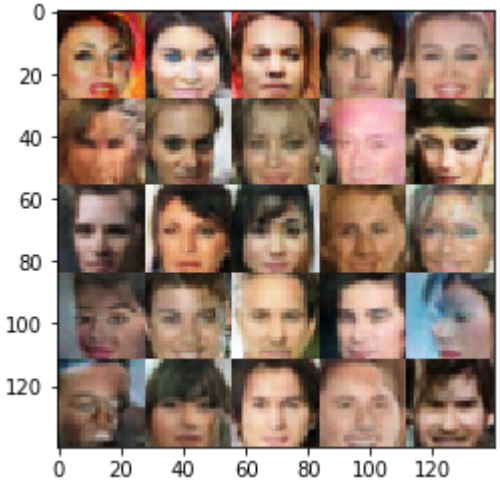
```
Epoch 10/10 Step 19700... Discriminator Loss: 0.6411... Generator Loss: 1.643
2
Epoch 10/10 Step 19800... Discriminator Loss: 0.5168... Generator Loss: 2.379
8
```




```
Epoch 10/10 Step 19900... Discriminator Loss: 0.9472... Generator Loss: 1.000
1
Epoch 10/10 Step 20000... Discriminator Loss: 1.8459... Generator Loss: 0.460
3
```




```
Epoch 10/10 Step 20100... Discriminator Loss: 0.4933... Generator Loss: 2.597
4
Epoch 10/10 Step 20200... Discriminator Loss: 0.5595... Generator Loss: 2.417
0
```



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.