In [1]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import os
from tensorflow.keras import layers
import time
from IPython import display
import datetime
```

In [2]:
```
%load_ext tensorboard
!rm -rf ./logs/wgan_gradient_tape/
```

In [3]:
```
def load_data():
    (train_images, train_labels), (valid_images, valid_labels) = tf.keras.datasets.mnist.load_data()
  # preprocess the images
    train_images = (train_images.astype(np.float32) - 127.5)/127.5 # standardize to [-1, 1]
    train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)

    valid_images = (valid_images.astype(np.float32) - 127.5)/127.5 # standardize to [-1, 1]
    valid_images = valid_images.reshape(valid_images.shape[0], 28, 28, 1)
    return (train_images, train_labels, valid_images, valid_labels)
```

In [4]:
```
train_images, train_labels, valid_images, valid_labels = load_data()
train_images.shape, valid_images.shape
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

Out[4]:
```
((60000, 28, 28, 1), (10000, 28, 28, 1))
```

In [5]:
```
batch_size = 64
n_critic = 1
num_batches = len(train_images)//batch_size
```

Self Define Generator:

In [6]:

```
def build_generator():
  model = tf.keras.Sequential([
        layers.Dense(128, use_bias=False, input_shape=(100,)),
        layers.BatchNormalization(),
        layers.LeakyReLU(0),

        layers.Dropout(0.2),

        layers.Dense(7*7*256, use_bias=False),
        layers.BatchNormalization(),
        layers.LeakyReLU(0),
        layers.Reshape((7,7,256)),

        layers.Conv2DTranspose(128, (5,5), strides=1, padding='same', use_bias=False),
# 7,7,128
        layers.BatchNormalization(),
        layers.LeakyReLU(0),

        layers.Conv2DTranspose(64, (5,5), strides=2, padding='same', use_bias=False),
#14,14,64
        layers.BatchNormalization(),
        layers.LeakyReLU(0),

        layers.Conv2DTranspose(1, (5,5), strides=2, padding='same', use_bias=False), #
28,28,1
        layers.Activation('tanh')
  ])

  return model

generator = build_generator()
generator.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	12800
batch_normalization (BatchNo	(None, 128)	512
leaky_re_lu (LeakyReLU)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 12544)	1605632
batch_normalization_1 (Batch	(None, 12544)	50176
leaky_re_lu_1 (LeakyReLU)	(None, 12544)	0
reshape (Reshape)	(None, 7, 7, 256)	0
conv2d_transpose (Conv2DTran	(None, 7, 7, 128)	819200
batch_normalization_2 (Batch	(None, 7, 7, 128)	512
leaky_re_lu_2 (LeakyReLU)	(None, 7, 7, 128)	0
conv2d_transpose_1 (Conv2DTr	(None, 14, 14, 64)	204800
batch_normalization_3 (Batch	(None, 14, 14, 64)	256
leaky_re_lu_3 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_transpose_2 (Conv2DTr	(None, 28, 28, 1)	1600
activation (Activation)	(None, 28, 28, 1)	0

Total params: 2,695,488
Trainable params: 2,669,760
Non-trainable params: 25,728

Self Define Discriminator:

In [7]:
```
def build_discriminator():
  model = tf.keras.Sequential([
          layers.Conv2D(128, (5,5), strides=2, padding='same',
                        input_shape=(28,28,1)), #14,14,64
          layers.LeakyReLU(0.2),

          layers.Conv2D(64, (5,5), strides=2, padding='same'), #7,7,128
          layers.BatchNormalization(),
          layers.LeakyReLU(0.2),
          layers.Dropout(0.2),

          layers.Flatten(),
          layers.Dense(128, activation='relu'),
          layers.Dropout(0.2),
          layers.Dense(1)
  ])
  return model

discriminator = build_discriminator()
discriminator.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 14, 14, 128)       3328
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 64)          204864
_________________________________________________________________
batch_normalization_4 (Batch (None, 7, 7, 64)          256
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 7, 7, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               401536
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129
=================================================================
Total params: 610,113
Trainable params: 609,985
Non-trainable params: 128
_________________________________________________________________
```

In [8]:
```
generator_optimizer = tf.keras.optimizers.RMSprop(learning_rate=5e-5)
discriminator_optimizer = tf.keras.optimizers.RMSprop(learning_rate=5e-5)
```

In [9]:
```
def show_images(generated_images):
  n_images = len(generated_images)
  cols = 8
  rows = n_images//cols

  plt.figure(figsize=(8, 8))
  for i in range(n_images):
      img = generated_images[i,:,:,0]*127.5+127.5
      ax = plt.subplot(rows, cols, i+1)
      plt.imshow(img, cmap='gray')
      plt.xticks([])
      plt.yticks([])
  plt.tight_layout()
  plt.show()
```

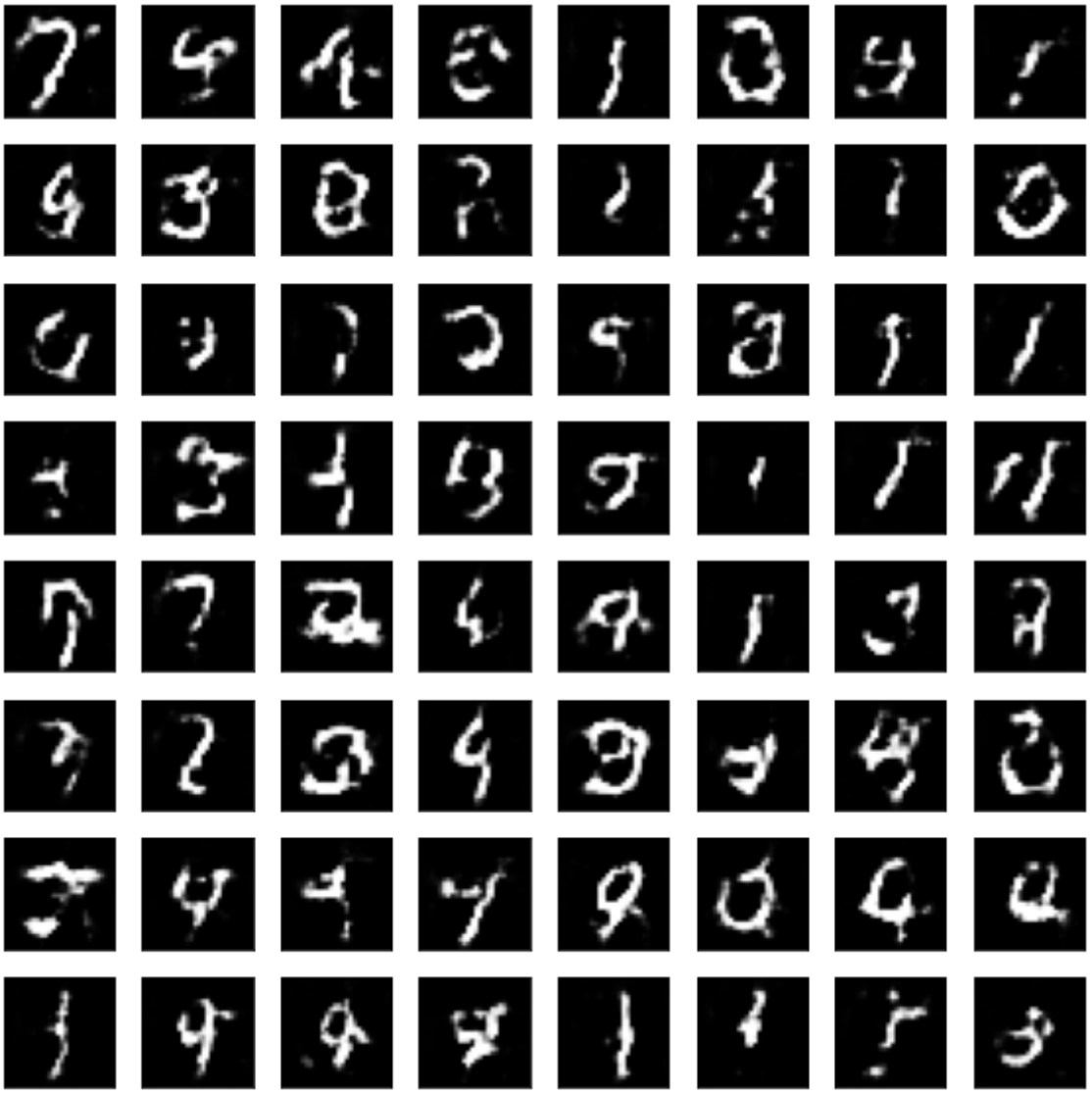
In [10]:
```
EPOCHS = 100
np.random.seed(2020)
seed = np.random.normal(loc=0, scale=1, size=(64, 100))

current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
gen_log_dir = 'logs/wgan_gradient_tape/' + current_time + '/gen'
disc_log_dir = 'logs/wgan_gradient_tape/' + current_time + '/disc'
gen_summary_writer = tf.summary.create_file_writer(gen_log_dir)
disc_summary_writer = tf.summary.create_file_writer(disc_log_dir)
```

In [11]:
```
@tf.function
def generator_gradient():
    # step 9
    noise = tf.random.normal([batch_size, 100])
    with tf.GradientTape() as gen_tape:
        # step 10
        generated_images = generator(noise, training=True)
        gen_loss_val = -tf.reduce_mean(discriminator(generated_images, training=True))
    # step 10
    gradients_of_generator = gen_tape.gradient(gen_loss_val, generator.trainable_variables)
    # step 11
    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    return gen_loss_val
```

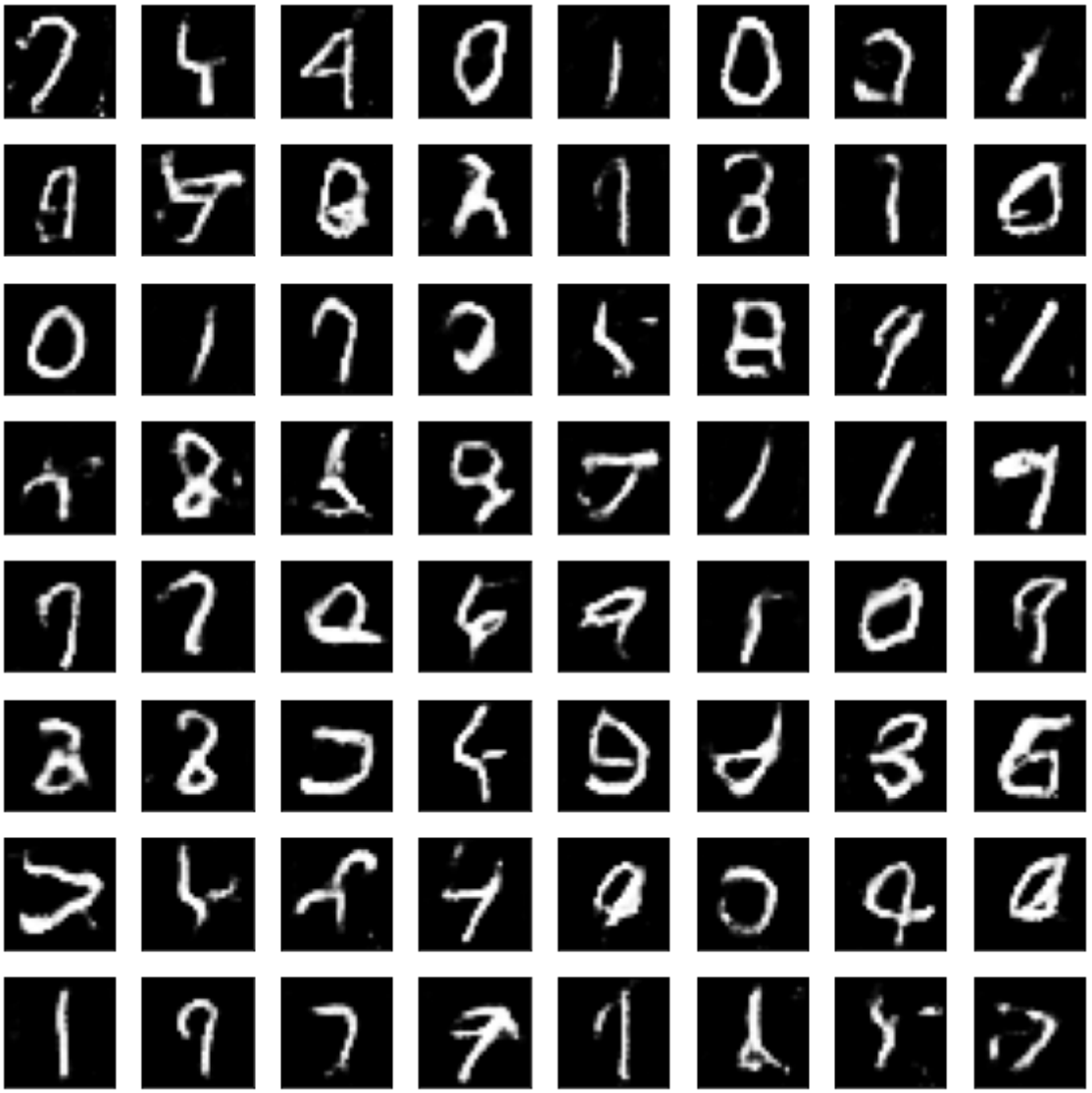
In [12]:
```
@tf.function
def discriminator_gradient():
    # step 3
    noise = tf.random.normal([batch_size, 100])
    # step 4
    images = train_images[np.random.choice(len(train_images), size=batch_size, replace
=False)]

    with tf.GradientTape() as disc_tape:
        # step 5
        generated_images = generator(noise, training=True)
        dis_loss_val = -tf.reduce_mean(discriminator(images, training=True)) + tf.redu
ce_mean(discriminator(generated_images, training=True))
    # step 5
    gradients_of_discriminator = disc_tape.gradient(dis_loss_val, discriminator.traina
ble_variables)
    # step 6
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discrimina
tor.trainable_variables))

    # step 7, clip
    for p in discriminator.trainable_variables:
      p.assign(tf.clip_by_value(p, -0.01, 0.01))

    return dis_loss_val
```

In [14]:
```
# the number of step corresponding to Algorithm 1 in "Wasserstein GAN"
def train(epochs):

  start = time.time()
  # step 1
  for e in range(epochs):
    start2 = time.time()
    for i in range(num_batches):
      # step 2
      for _ in range(n_critic):
        dis_loss_val = discriminator_gradient()
      # step 8
      gen_loss_val = generator_gradient()
      #if (i+1) % 50 == 0:
        #print('This is the {}/{} of epoch {}.'.format(i+1, num_batches, e+1))


    with gen_summary_writer.as_default():
      tf.summary.scalar('loss', gen_loss_val, step=e)
    with disc_summary_writer.as_default():
      tf.summary.scalar('loss', dis_loss_val, step=e)


    #if e==0 or (e+1) % 5 == 0:
    if e in [0,4,9,49,99,149,199]:
      imgs = generator.predict_on_batch(seed)
      show_images(imgs)
      print("Epoch: {}/{} Discriminator Loss: {:.4f}  Generator Loss: {:.4f}".format
(e+1, epochs, dis_loss_val, gen_loss_val))
      print('Time for epoch {} is {} sec'.format(e + 1, time.time()-start2))
  # step 12
  end = time.time()
  print(end-start)
```

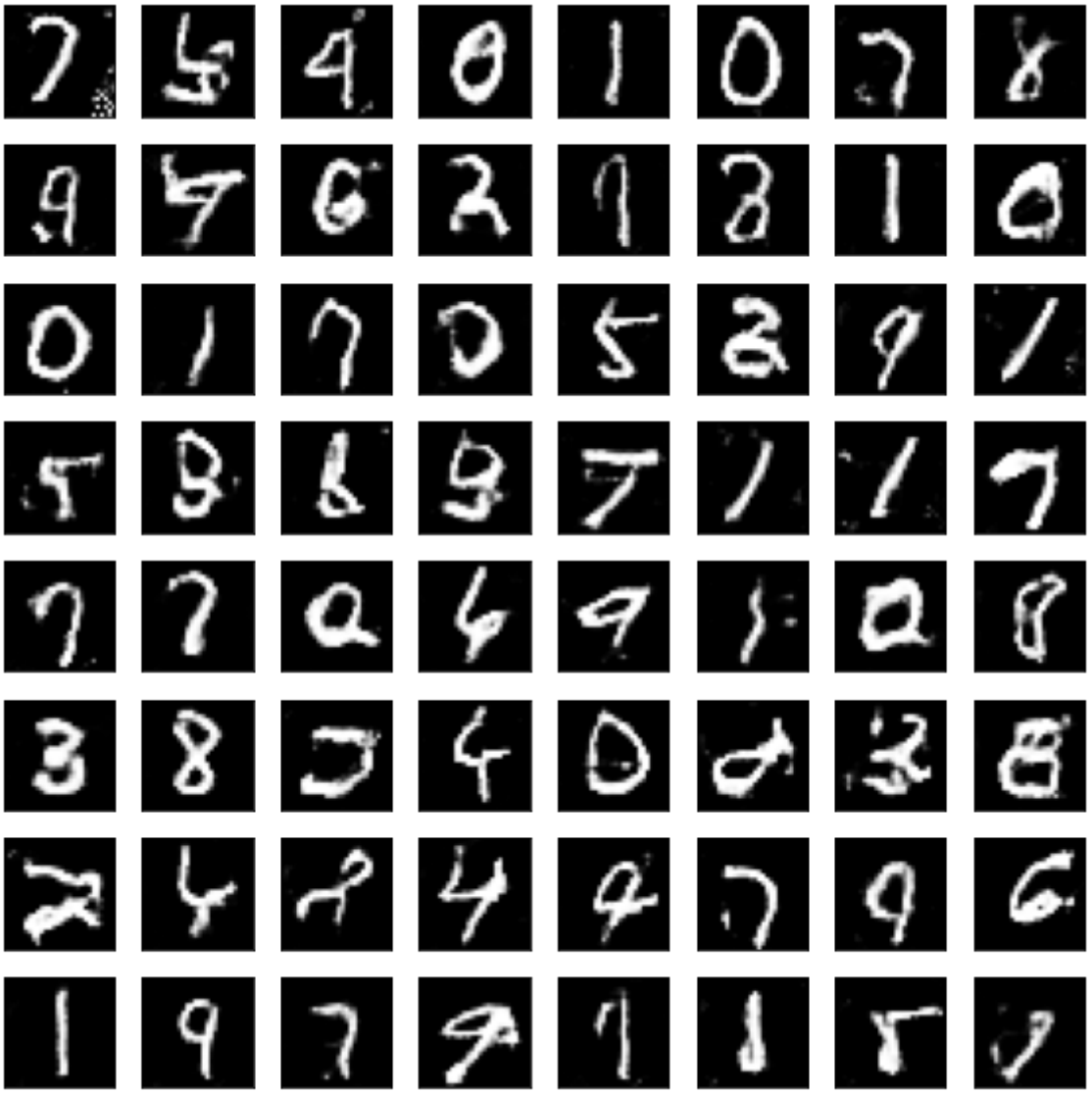
In [15]:
```
train(200)
```



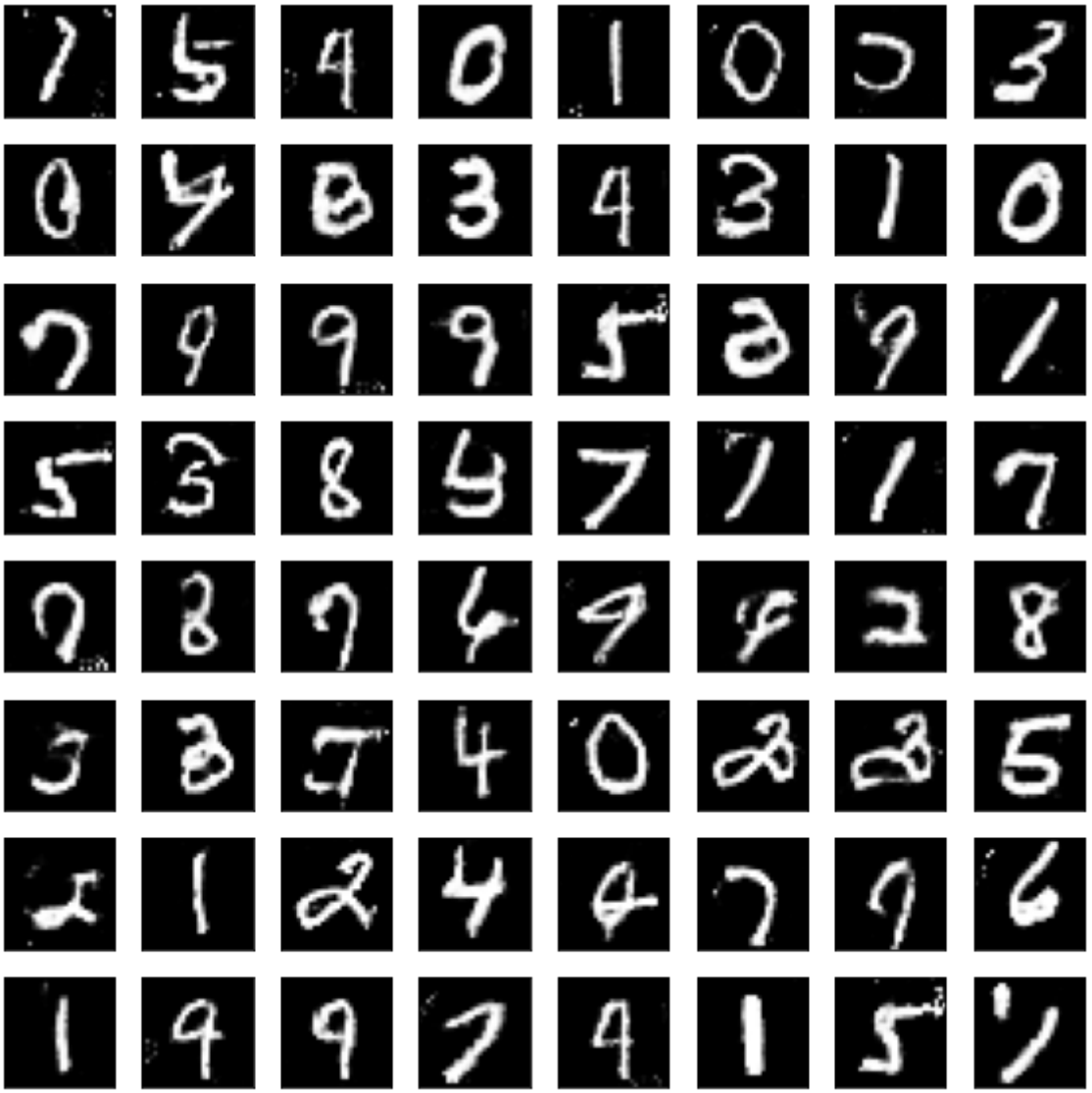
Epoch: 1/200 Discriminator Loss: -0.0071 Generator Loss: -0.0022
Time for epoch 1 is 27.301555156707764 sec



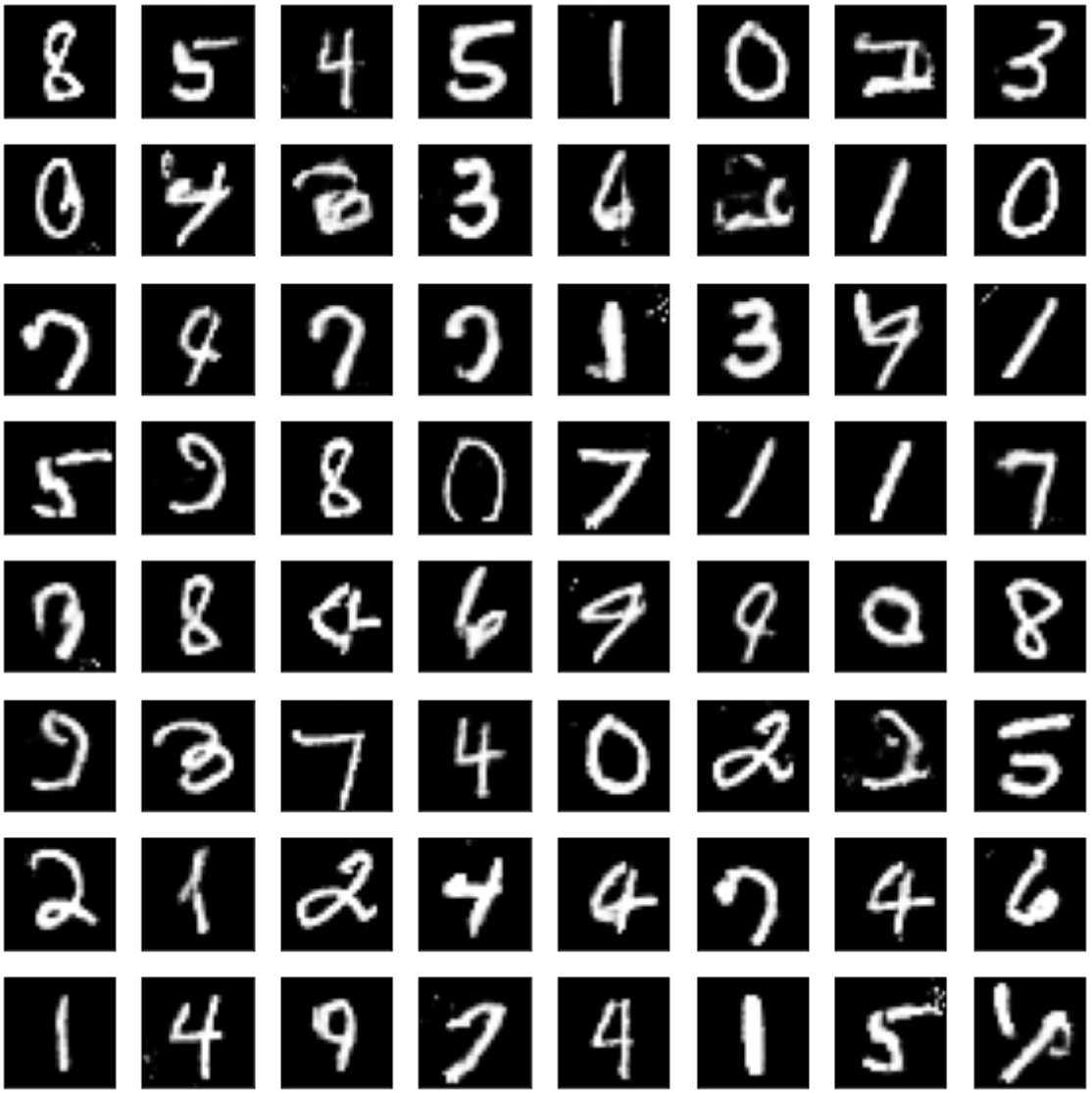
Epoch: 5/200 Discriminator Loss: -0.0105 Generator Loss: 0.0073
Time for epoch 5 is 19.333012104034424 sec



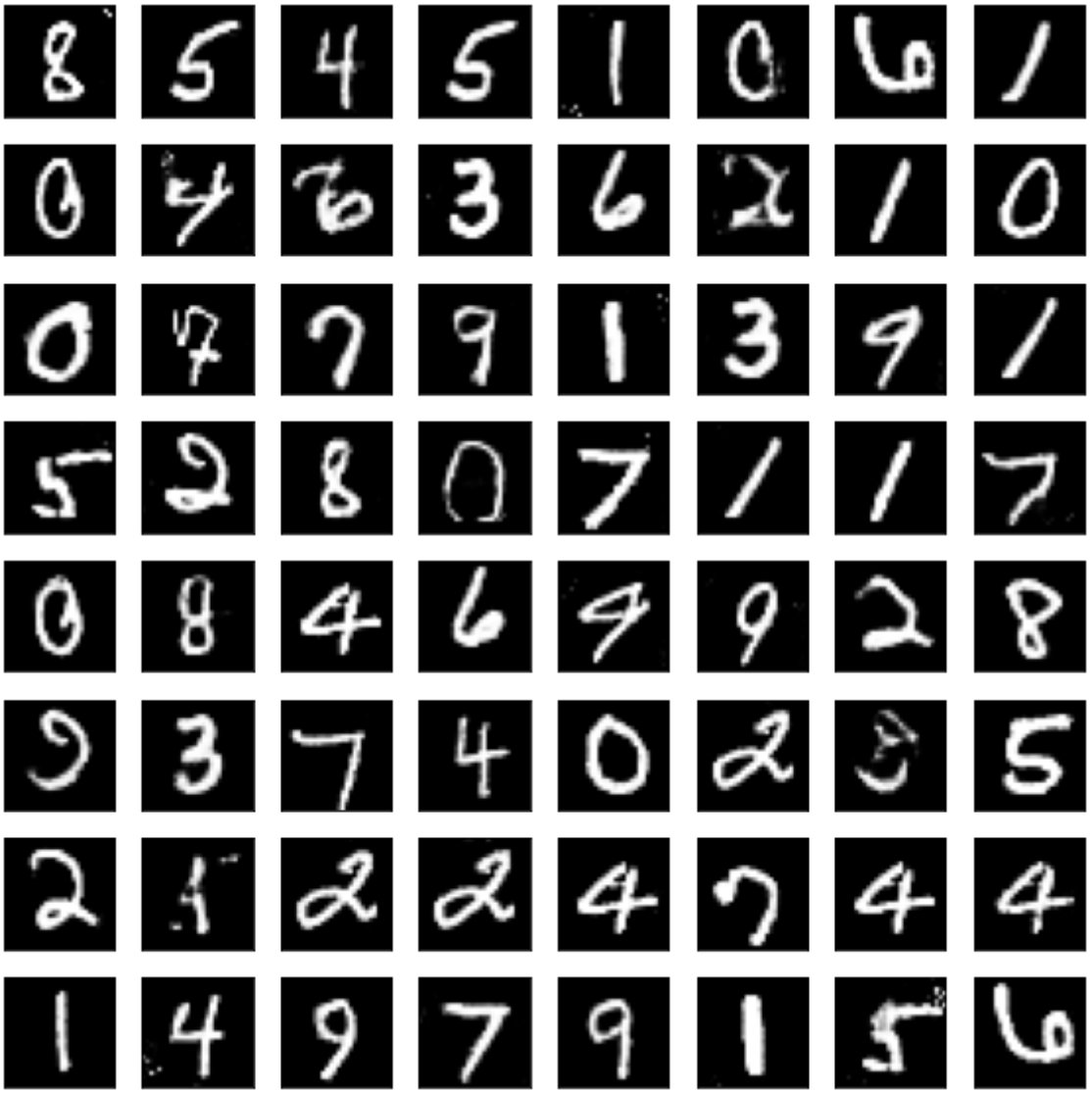
Epoch: 10/200 Discriminator Loss: -0.0091 Generator Loss: 0.0050
Time for epoch 10 is 19.399072647094727 sec



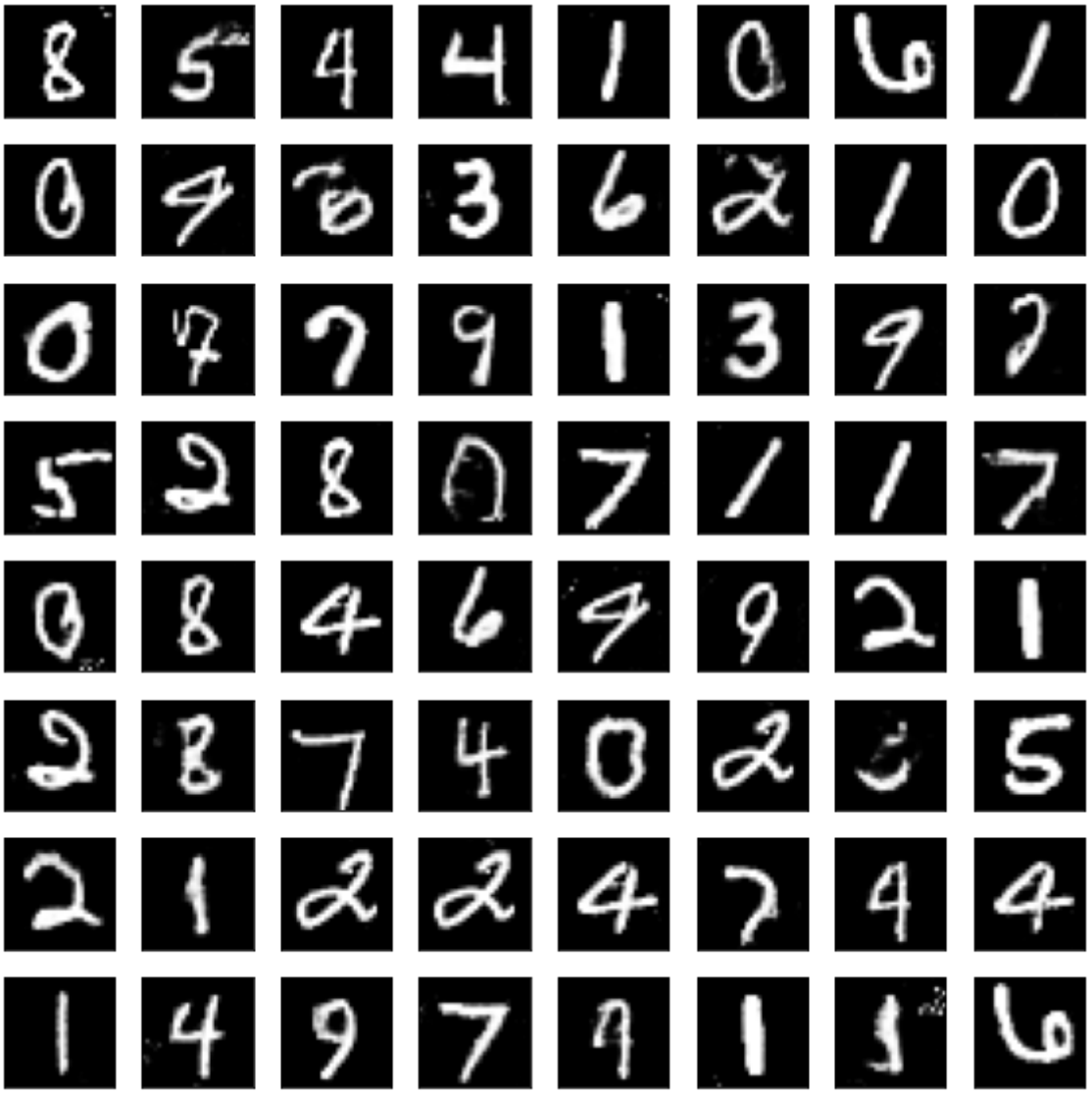
Epoch: 50/200 Discriminator Loss: -0.0069 Generator Loss: 0.0047
Time for epoch 50 is 19.267515420913696 sec



Epoch: 100/200 Discriminator Loss: -0.0049 Generator Loss: 0.0012
Time for epoch 100 is 19.16631507873535 sec



Epoch: 150/200 Discriminator Loss: -0.0036 Generator Loss: -0.0001
Time for epoch 150 is 19.192378520965576 sec



```
Epoch: 200/200 Discriminator Loss: -0.0021  Generator Loss: -0.0040
Time for epoch 200 is 19.184732675552368 sec
3471.508542060852
```

In [16]:
```
%tensorboard --logdir logs/wgan_gradient_tape
```