```
import tensorflow as tf
```

```
tf.__version__
```

```
'2.19.0'
```

```
# To generate GIFs
!pip install imageio
!pip install git+https://github.com/tensorflow/docs
```

```
Requirement already satisfied: imageio in /usr/local/lib/python3.12/dist-packages (2.37.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from imageio) (2.0.2)
Requirement already satisfied: pillow>=8.3.2 in /usr/local/lib/python3.12/dist-packages (from imageio) (11.3.0)
Collecting git+https://github.com/tensorflow/docs
  Cloning https://github.com/tensorflow/docs to /tmp/pip-req-build-d8bkt0l2
  Running command git clone --filter=blob:none --quiet https://github.com/tensorflow/docs /tmp/pip-req-build-d8bkt0l2
  Resolved https://github.com/tensorflow/docs to commit e21d085d5ed82504ffcec11aa82ebc78f1f2302e
  Preparing metadata (setup.py) ... done
Collecting astor (from tensorflow-docs==2025.3.6.10029)
  Downloading astor-0.8.1-py2.py3-none-any.whl.metadata (4.2 kB)
Requirement already satisfied: absl-py in /usr/local/lib/python3.12/dist-packages (from tensorflow-docs==2025.3.6.10029) (1.4.0
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from tensorflow-docs==2025.3.6.10029) (3.1.6)
Requirement already satisfied: nbformat in /usr/local/lib/python3.12/dist-packages (from tensorflow-docs==2025.3.6.10029) (5.10
Requirement already satisfied: protobuf>=3.12 in /usr/local/lib/python3.12/dist-packages (from tensorflow-docs==2025.3.6.10029)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dist-packages (from tensorflow-docs==2025.3.6.10029) (6.0.2)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->tensorflow-docs==2025.3
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.12/dist-packages (from nbformat->tensorflow-docs=
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.12/dist-packages (from nbformat->tensorflow-docs==2025
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in /usr/local/lib/python3.12/dist-packages (from nbformat->tensorflow
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.12/dist-packages (from nbformat->tensorflow-docs==2025
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.12/dist-packages (from jsonschema>=2.6->nbformat->tensor
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.12/dist-packages (from jsonschema
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.12/dist-packages (from jsonschema>=2.6->nbformat->
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.12/dist-packages (from jsonschema>=2.6->nbformat->tensc
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.12/dist-packages (from jupyter-core!=5.0.*,>=4.12->r
Requirement already satisfied: typing-extensions>=4.4.0 in /usr/local/lib/python3.12/dist-packages (from referencing>=0.28.4->j
Downloading astor-0.8.1-py2.py3-none-any.whl (27 kB)
Building wheels for collected packages: tensorflow-docs
  Building wheel for tensorflow-docs (setup.py) ... done
  Created wheel for tensorflow-docs: filename=tensorflow_docs-2025.3.6.10029-py3-none-any.whl size=186351 sha256=0c134631e09423
  Stored in directory: /tmp/pip-ephem-wheel-cache-v4jr9_7x/wheels/3e/88/34/48d2789bc9d37b33ddce06bccc454fae0285e5396d0a5be9d9
Successfully built tensorflow-docs
Installing collected packages: astor, tensorflow-docs
Successfully installed astor-0.8.1 tensorflow-docs-2025.3.6.10029
```

```
import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from tensorflow.keras import layers
import time

from IPython import display
```

```
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
```

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

```
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
train_images = (train_images - 127.5) / 127.5  # Normalize the images to [-1, 1]
```

```
BUFFER_SIZE = 60000
BATCH_SIZE = 256
```

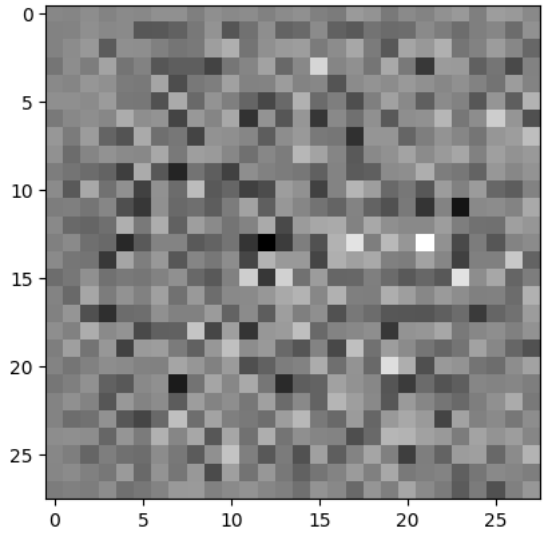
```
# Batch and shuffle the data
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)  # Note: None is the batch size
```

```
    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

```
generator = make_generator_model()

noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
<matplotlib.image.AxesImage at 0x792237953920>
```




```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                            input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

```
discriminator = make_discriminator_model()
decision = discriminator(generated_image)
print (decision)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_sha
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
tf.Tensor([[0.00086778]], shape=(1, 1), dtype=float32)
```

```
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

```
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
```

```
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

```
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)
```

```
EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16

# You will reuse this seed overtime (so it's easier)
# to visualize progress in the animated GIF)
seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

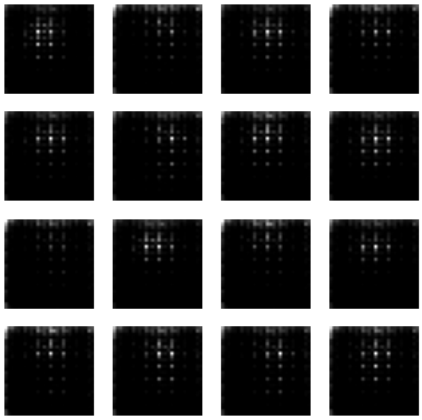
```
# Notice the use of `tf.function`
# This annotation causes the function to be "compiled".
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
```

```
def train(dataset, epochs):
  for epoch in range(epochs):
    start = time.time()

    for image_batch in dataset:
      train_step(image_batch)

    # Produce images for the GIF as you go
    # display.clear_output(wait=True)
    generate_and_save_images(generator,
                             epoch + 1,
                             seed)

    # Save the model every 15 epochs
    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-start))

  # Generate after the final epoch
  # display.clear_output(wait=True)
  generate_and_save_images(generator,
                           epochs,
                           seed)
```

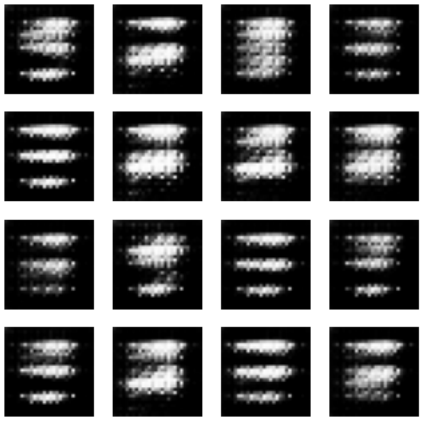
```
def generate_and_save_images(model, epoch, test_input):
  # Notice `training` is set to False.
  # This is so all layers run in inference mode (batchnorm).
  predictions = model(test_input, training=False)

  fig = plt.figure(figsize=(4, 4))
```

```
  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
```

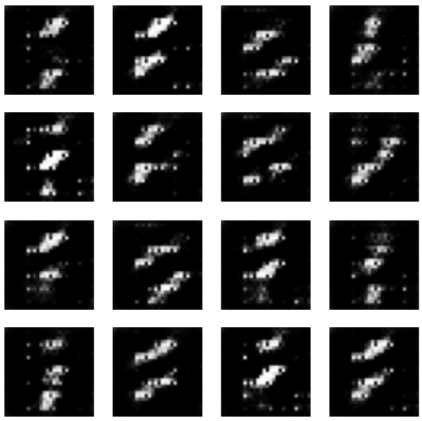
```
train(train_dataset, EPOCHS)
```



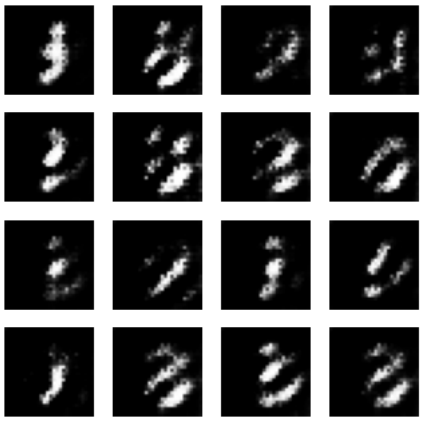
```
Time for epoch 1 is 18.12399411201477 sec
```



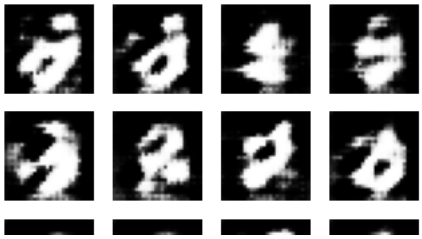
```
Time for epoch 2 is 11.612900733947754 sec
```



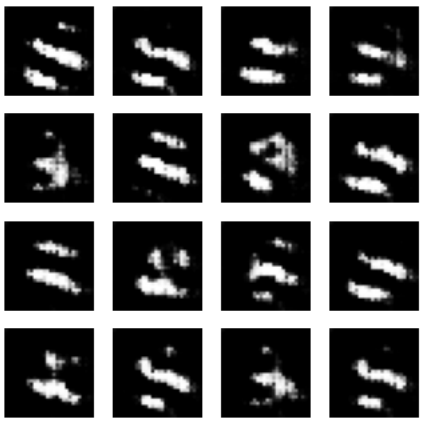
```
Time for epoch 3 is 11.84207010269165 sec
```



```
Time for epoch 4 is 11.963088750839233 sec
```



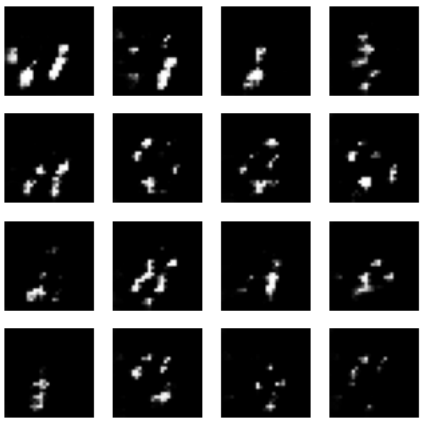
Time for epoch 5 is 12.105511188507080 sec



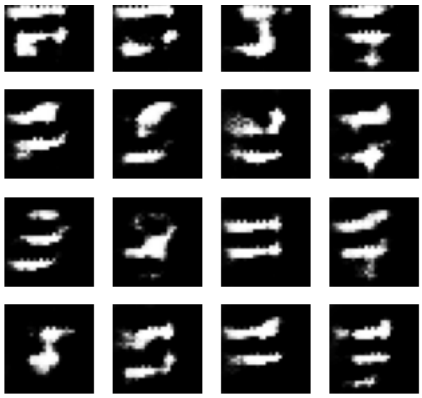
Time for epoch 6 is 11.729608058929443 sec

Time for epoch 7 is 11.864196538925171 sec

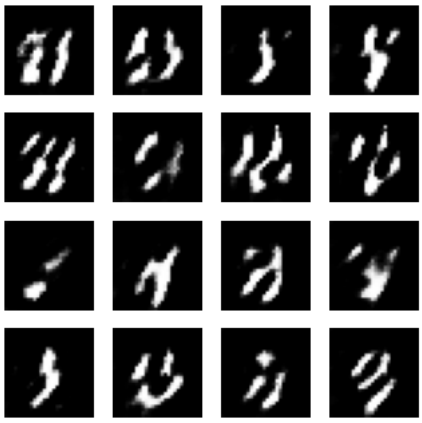
Time for epoch 8 is 11.715337991714478 sec



Time for epoch 9 is 11.662762880325317 sec

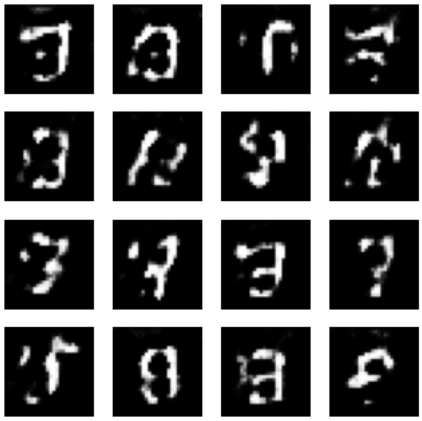


Time for epoch 10 is 11.727294206619263 sec



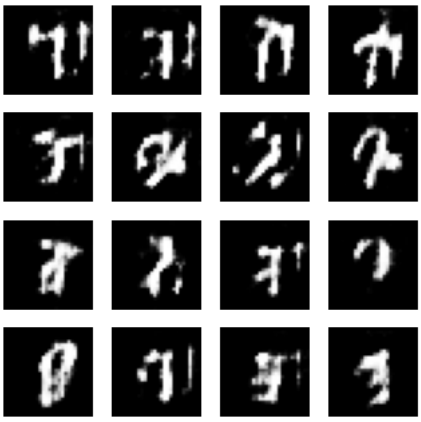
Time for epoch 11 is 12.050544738769531 sec

Time for epoch 12 is 12.069900035858154 sec

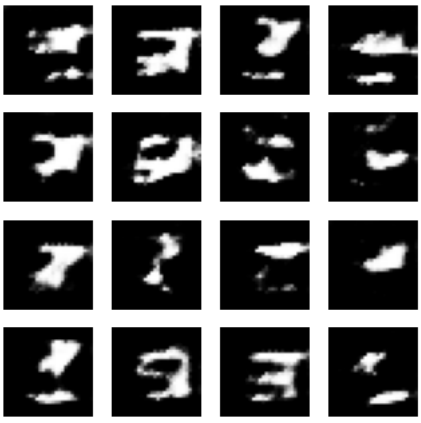


Time for epoch 13 is 11.782123804092407 sec

Time for epoch 14 is 11.735219001770020 sec



Time for epoch 15 is 11.888055086135864 sec



Time for epoch 16 is 11.74549651145935 sec

Time for epoch 17 is 12.06383490562439 sec

Time for epoch 18 is 11.739831924438477 sec

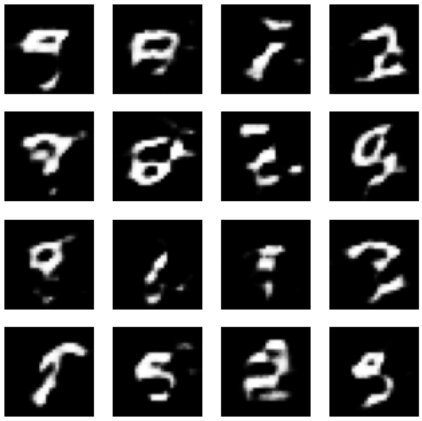
Time for epoch 19 is 11.73371934890747 sec

Time for epoch 20 is 11.757981061935425 sec

Time for epoch 21 is 11.77294373512268 sec

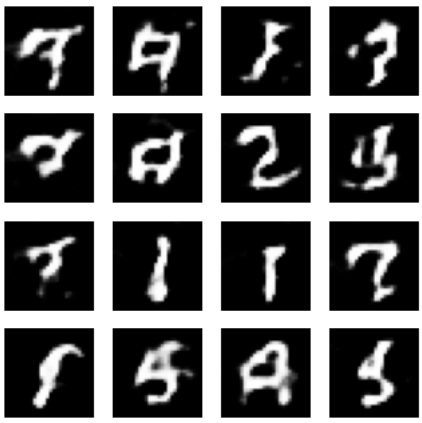
Time for epoch 22 is 11.752598285675049 sec

Time for epoch 23 is 11.782346248626709 sec

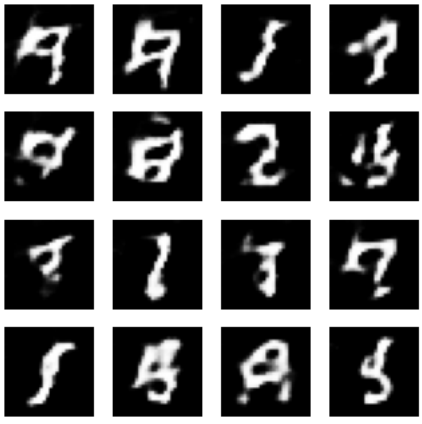


Time for epoch 24 is 12.05023980140686 sec

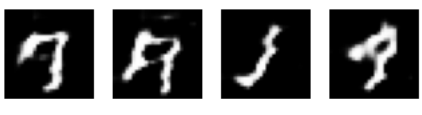
Time for epoch 25 is 11.754236459732056 sec



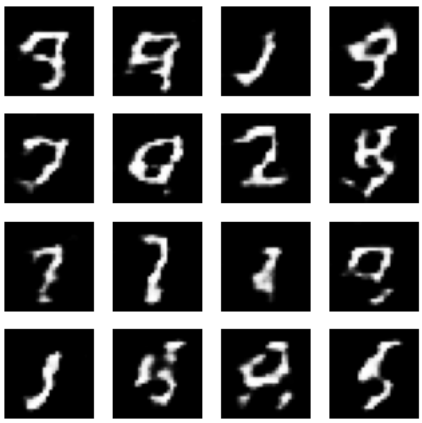
Time for epoch 26 is 11.753066301345825 sec



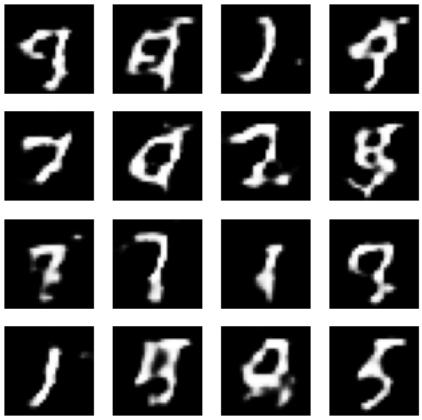
Time for epoch 27 is 11.769832611083984 sec



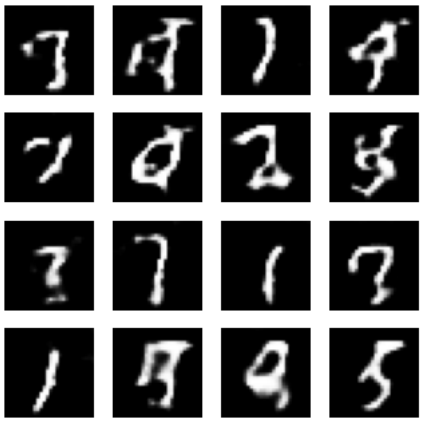
Time for epoch 28 is 11.754838228225708 sec



Time for epoch 29 is 11.743526220321655 sec



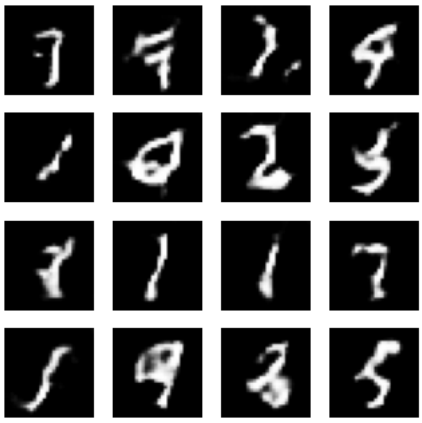
Time for epoch 30 is 12.458091735839844 sec



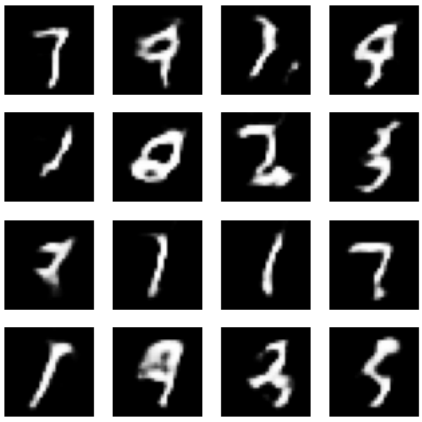
Time for epoch 31 is 11.768989086151123 sec

Time for epoch 32 is 11.754417419433594 sec

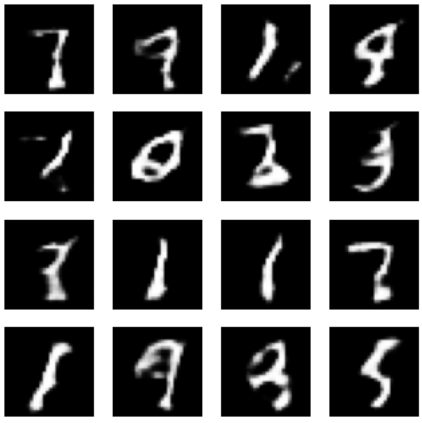
Time for epoch 33 is 11.755656957626343 sec



Time for epoch 34 is 11.742496490478516 sec

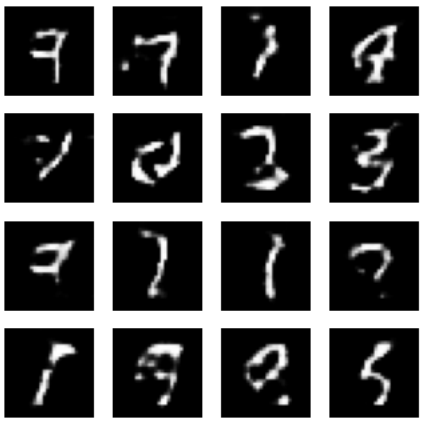


Time for epoch 35 is 11.768450736999512 sec



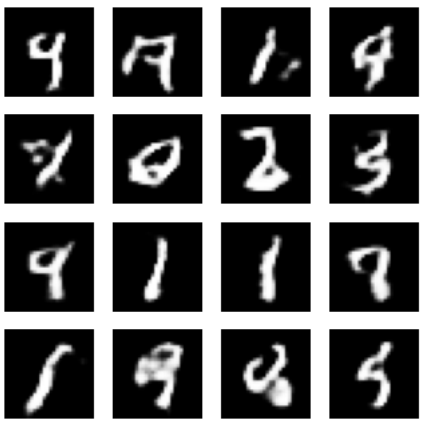
Time for epoch 36 is 12.042580604553223 sec

Time for epoch 37 is 11.760908365249634 sec

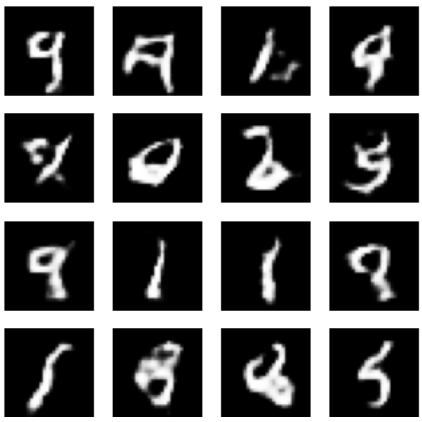


Time for epoch 38 is 11.76224660873413 sec

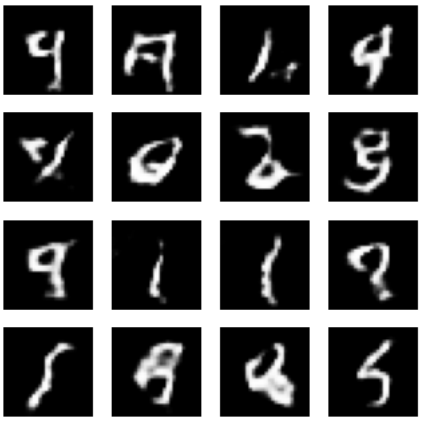
Time for epoch 39 is 11.759610176086426 sec



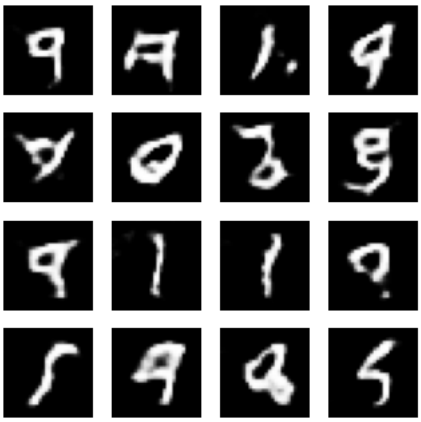
Time for epoch 40 is 11.773921966552734 sec



Time for epoch 41 is 11.756036281585693 sec



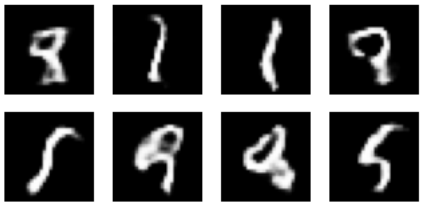
Time for epoch 42 is 12.074751377105713 sec



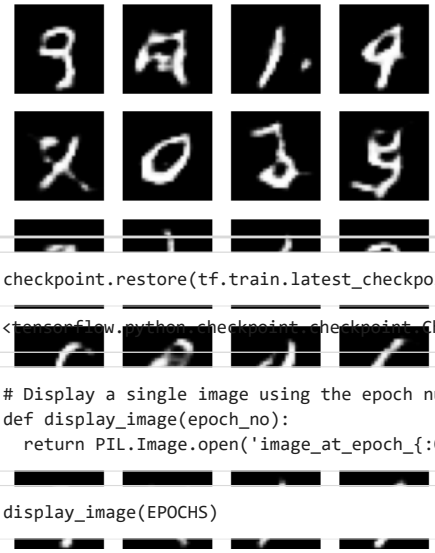
Time for epoch 43 is 11.755741596221924 sec

Time for epoch 44 is 11.788115501403809 sec

Time for epoch 45 is 11.895251035690308 sec



```
Time for epoch 46 is 11.727190017700195 sec
```



```
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7921f36fec00>
```

```
# Display a single image using the epoch number
def display_image(epoch_no):
  return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))
```

```
display_image(EPOCHS)
```