In [1]:
```
import tensorflow as tf
```

In [2]:
```
# To generate GIFs
```

In [3]:
```
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
```

In [4]:
```
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
```

In [5]:
```
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('flo
train_images = (train_images - 127.5) / 127.5
# Normalize the images to [-1, 1]
```

In [6]:
```
BUFFER_SIZE = 60000
BATCH_SIZE = 256
```

In [7]:
```
# Batch and shuffle the data
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_S
```

In [8]:
```
#Model creation
```

In [9]:

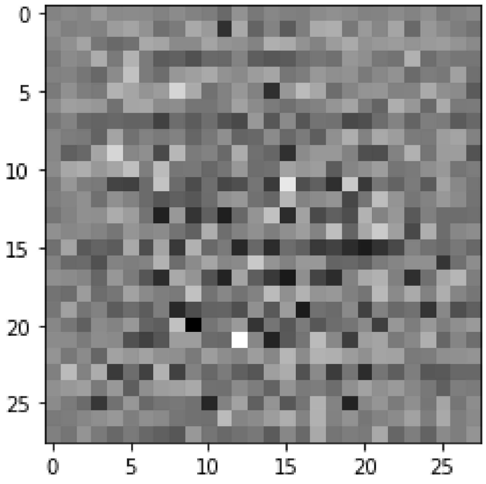
```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

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

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same',
    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', 
    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', u
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

In [10]:

```
generator = make_generator_model()

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

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

Out[10]:

```
<matplotlib.image.AxesImage at 0x24aab5cdaf0>
```



In [11]:

```
#The discriminator is a CNN-based image classifier.
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
```

In [12]:

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

```
tf.Tensor([[0.0013039]], shape=(1, 1), dtype=float32)
```

In [13]:

```
#Define loss functions and optimizers for both models.
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

In [14]:

```
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
```

In [15]:

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

In [16]:

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

In [17]:

```
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)
```

In [18]:

```
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])
```

In [19]:

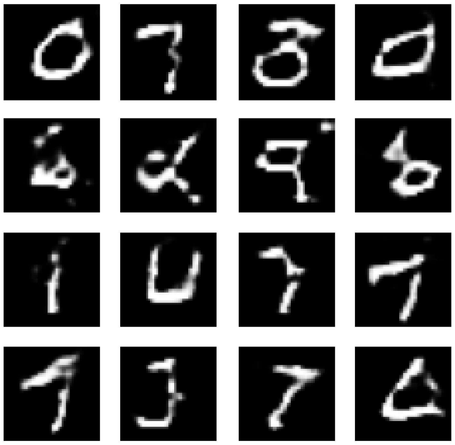
```
# Notice the use of `tf.function`
# This annotation causes the function to be "compiled".
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    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_vari
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trai

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.tra
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discr
```

In [20]:

```
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)
```

In [21]:
```
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()
```

In [22]:
```
train(train_dataset, EPOCHS)
```



In [23]:
```
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

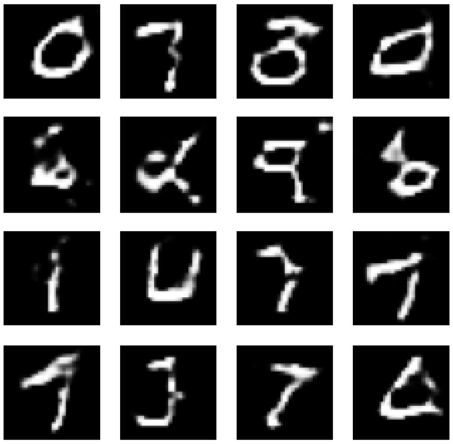
Out[23]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x24aa5444c10
>
```

In [24]:
```
# 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))
```

In [25]:

```
display_image(EPOCHS)
```

Out[25]:



In [26]:

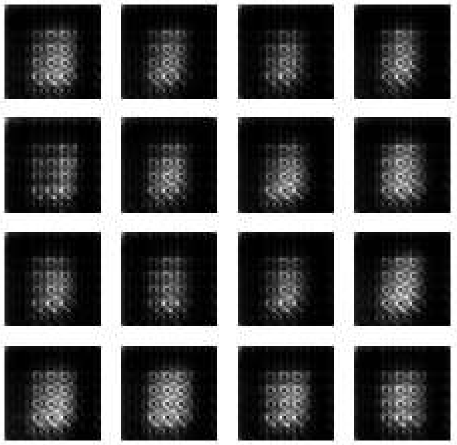
```
anim_file = 'dcgan.gif'

with imageio.get_writer(anim_file, mode='I') as writer:
  filenames = glob.glob('image*.png')
  filenames = sorted(filenames)
  for filename in filenames:
    image = imageio.imread(filename)
    writer.append_data(image)
  image = imageio.imread(filename)
  writer.append_data(image)
```

In [28]:
```
pip install git+https://github.com/tensorflow/docs
```

```
Collecting git+https://github.com/tensorflow/docs
  Cloning https://github.com/tensorflow/docs (https://github.com/tensorflow/doc
s) to c:\users\administrator\appdata\local\temp\pip-req-build-433mdsfq
Collecting astor
  Downloading astor-0.8.1-py2.py3-none-any.whl (27 kB)
Requirement already satisfied: absl-py in c:\programdata\anaconda3\lib\site-pac
kages (from tensorflow-docs===0.0.04df4b1d50e0016b80ca295e2117b92757c8040ed-)
 (0.12.0)
Requirement already satisfied: protobuf>=3.14 in c:\programdata\anaconda3\lib\s
ite-packages (from tensorflow-docs===0.0.04df4b1d50e0016b80ca295e2117b92757c804
0ed-) (3.15.7)
Requirement already satisfied: pyyaml in c:\programdata\anaconda3\lib\site-pack
ages (from tensorflow-docs===0.0.04df4b1d50e0016b80ca295e2117b92757c8040ed-)
 (5.3.1)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-package
s (from absl-py->tensorflow-docs===0.0.04df4b1d50e0016b80ca295e2117b92757c8040e
d-) (1.15.0)
Building wheels for collected packages: tensorflow-docs
  Building wheel for tensorflow-docs (setup.py): started
  Building wheel for tensorflow-docs (setup.py): finished with status 'done'
  Created wheel for tensorflow-docs: filename=tensorflow_docs-0.0.04df4b1d50e00
16b80ca295e2117b92757c8040ed_-py3-none-any.whl size=133194 sha256=2680bf90749dd
a506cf45e8d45aaf6d9d63698efd8a0b5c932b096a86a0e5e1f
  Stored in directory: C:\Users\Administrator\AppData\Local\Temp\pip-ephem-whee
l-cache-h2g5pmum\wheels\3b\ee\a2\ab4d36a9a4af495bcb936f3e849d4b497b65fa40548a68
d6c3
Successfully built tensorflow-docs
Installing collected packages: astor, tensorflow-docs
Successfully installed astor-0.8.1 tensorflow-docs-0.0.04df4b1d50e0016b80ca295e
2117b92757c8040ed-
Note: you may need to restart the kernel to use updated packages.
```

In [29]:
```
import tensorflow_docs.vis.embed as embed
embed.embed_file(anim_file)
```

Out[29]:



In [ ]: