Lab 4 & 5

Daniel Mehta

Part 1a: Follow the tutorial code given in the lecture (https://www.tensorflow.org/tutorials/genera

In [5]:
```
import tensorflow as tf
tf.__version__
```

Out[5]: '2.19.0'

In [6]:
```
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 [7]:
```
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mn
ist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

In [8]:
```
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float
train_images = (train_images - 127.5) / 127.5  # Normalize the images to [-1, 1]
```

In [9]:
```
BUFFER_SIZE = 60000
BATCH_SIZE = 256
```

In [10]:
```
# Batch and shuffle the data
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZ
```

In [11]:
```
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
```

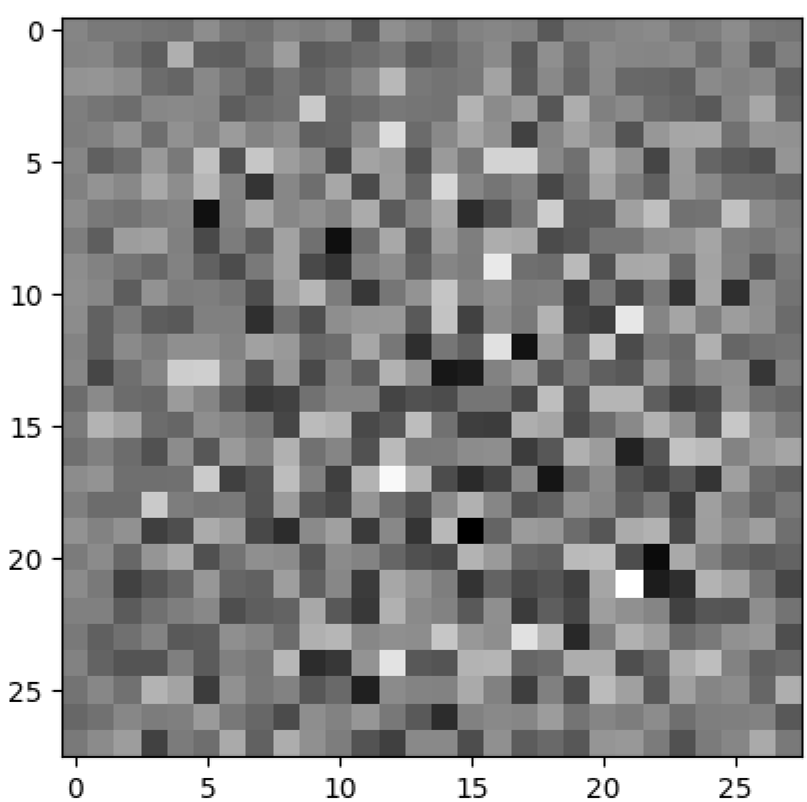
```
    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', u
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', us
    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
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

In [12]:

```
generator = make_generator_model()

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

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

```
C:\Users\danie\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\l
ayers\core\dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argume
nt to a layer. When using Sequential models, prefer using an `Input(shape)` object a
s the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Out[12]: <matplotlib.image.AxesImage at 0x16d5f1bbcd0>



In [13]:
```
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 [14]:
```
discriminator = make_discriminator_model()
decision = discriminator(generated_image)
print (decision)
```

```
tf.Tensor([[-0.00176129]], shape=(1, 1), dtype=float32)
```

```
C:\Users\danie\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\l
ayers\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`inp
ut_dim` argument to a layer. When using Sequential models, prefer using an `Input(sh
ape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [15]:
```
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

In [16]:
```
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 [17]:
```
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

In [18]:
```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

In [19]:
```
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 [26]:
```
#EPOCHS = 50
EPOCHS = 10 # lowered to ten for time
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])
```

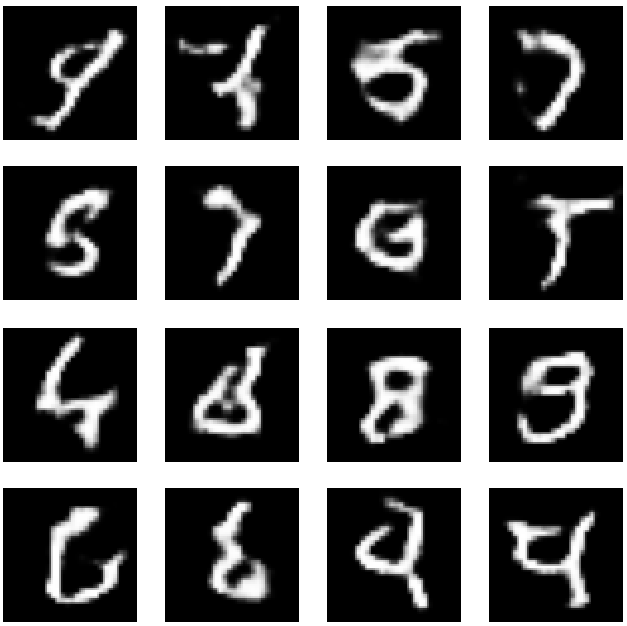
In [21]:
```
# Notice the use of `tf.function`
# This annotation causes the function to be "compiled".
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    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_variab
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.traina

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.train
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discrim
```

In [22]:
```
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 [23]:
```
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 [24]:
```
train(train_dataset, EPOCHS)
```



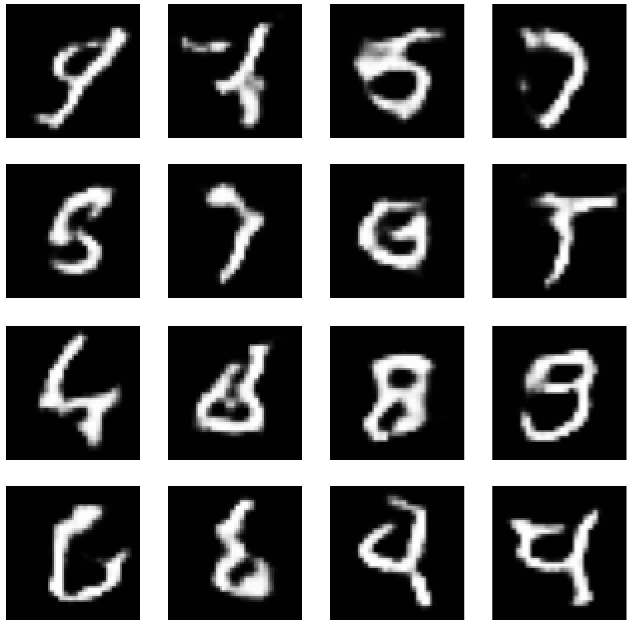
In [25]:
```
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

Out[25]:
```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x16d605772e0>
```

In [27]:
```
# 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 [29]:
```
display_image(50)
```

Out[29]:

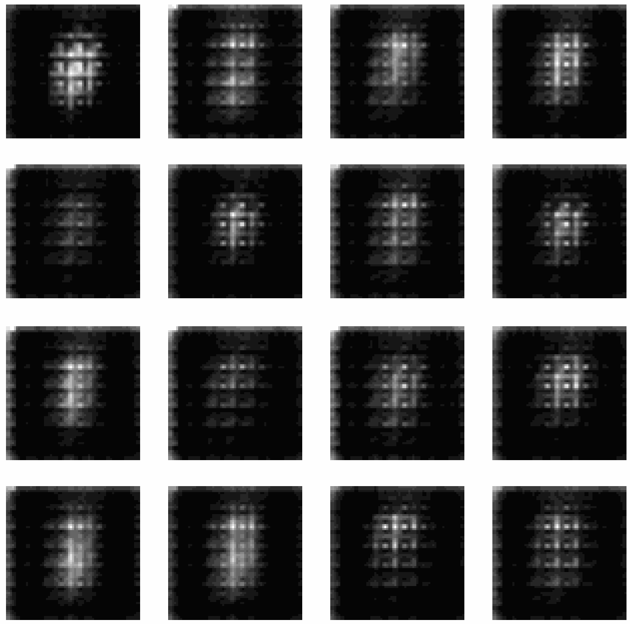


In [30]:
```
anim_file = 'dcgan.gif'

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)
```

```
C:\Users\danie\AppData\Local\Temp\ipykernel_18276\1982054950.py:7: DeprecationWarnin
g: Starting with ImageIO v3 the behavior of this function will switch to that of ii
o.v3.imread. To keep the current behavior (and make this warning disappear) use `imp
ort imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread(filename)
C:\Users\danie\AppData\Local\Temp\ipykernel_18276\1982054950.py:9: DeprecationWarnin
g: Starting with ImageIO v3 the behavior of this function will switch to that of ii
o.v3.imread. To keep the current behavior (and make this warning disappear) use `imp
ort imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread(filename)
```

In [31]:
```
import tensorflow_docs.vis.embed as embed
embed.embed_file(anim_file)
```

Out[31]:



Part 1b: choose areal-life problem where GANs can be used as a full or part of the solution

Dataset: Fashion MNIST

Retailers often need to generate synthetic fashion items for prototyping or virtual fitting rooms

In [34]:
```
(train_images, train_labels), (_, _) = tf.keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

In [35]:
```
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float
train_images = (train_images-127.5)/127.5
```

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

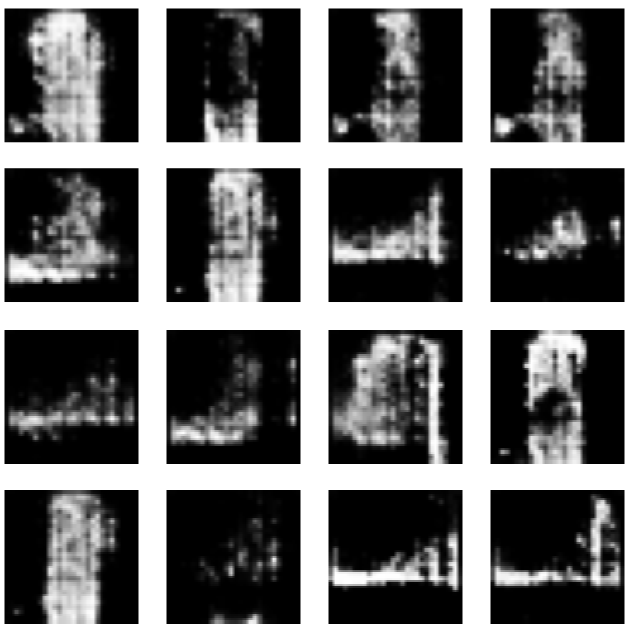
In [36]:
```
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZ
```

In [37]:
```
make_generator_model()
make_discriminator_model()
```

Out[37]: <Sequential name=sequential_3, built=True>

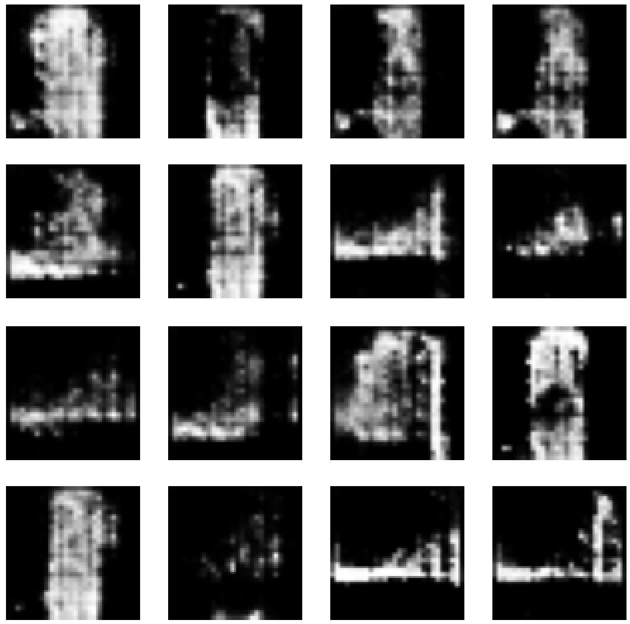
In [39]:
```
EPOCHS = 10
noise_dim = 100
num_examples_to_generate = 16
seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

In [40]:
```
train(train_dataset, EPOCHS)
```



In [41]:
```
display_image(EPOCHS)
```

Out[41]:



Part 2: Evaluating Generator Performance

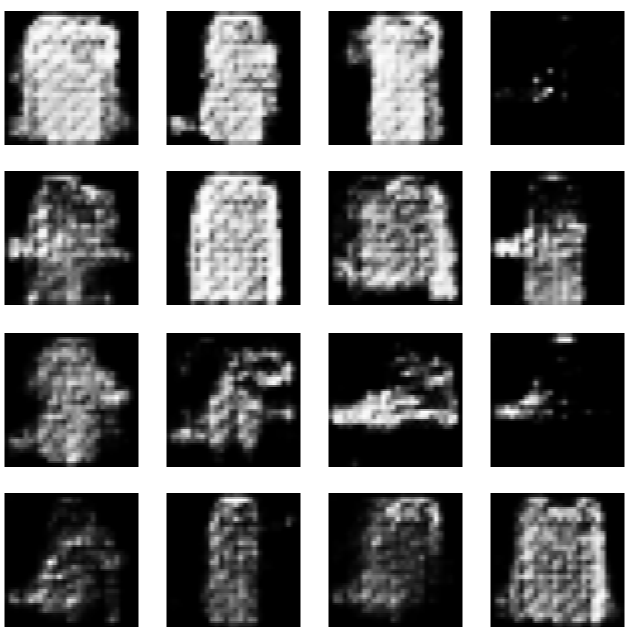
In [42]:
```
gen_losses = []
disc_losses = []
```

In [43]:
```
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE,100])

    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_variab
    gradients_of_discriminator = disc_tape.gradient(disc_loss,discriminator.trainab

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.train
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,discrimi

    return gen_loss, disc_loss
```

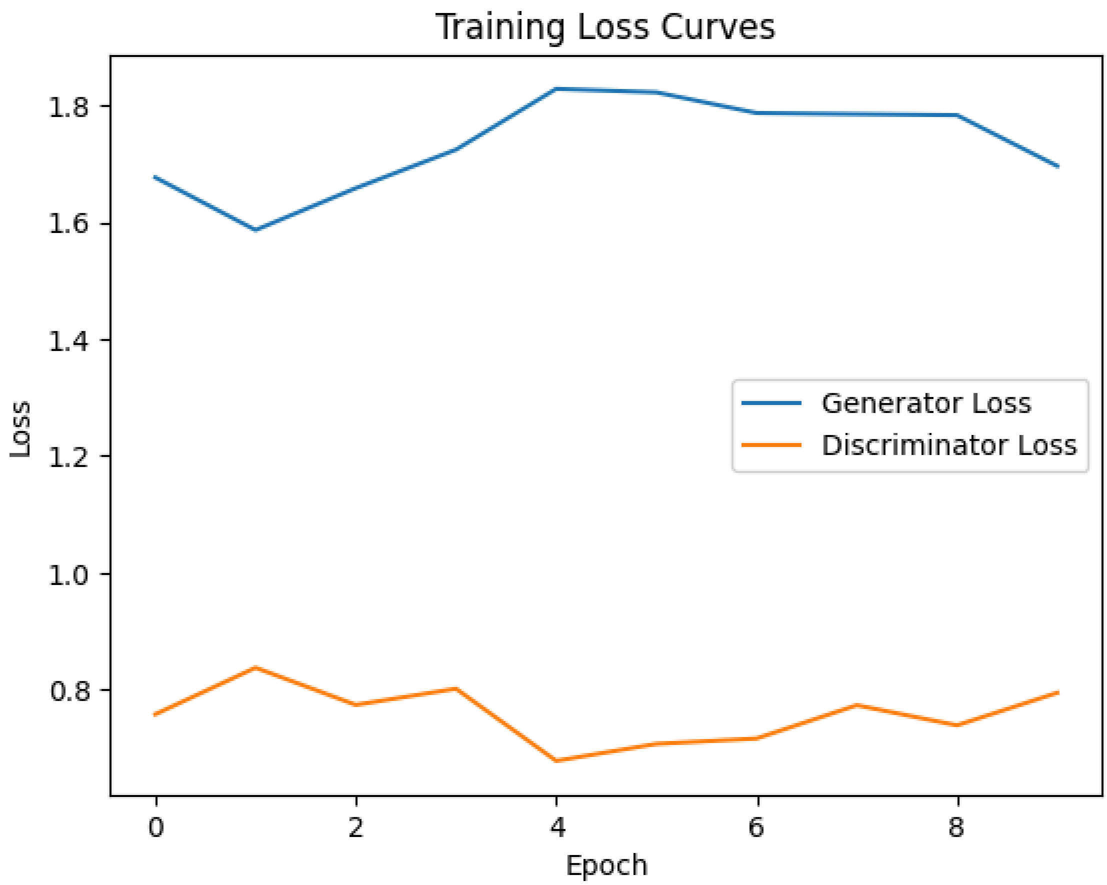
In [48]:
```
def train(dataset, epochs):
    for epoch in range(epochs):
        start =time.time()
        gen_total = 0
```

```
        disc_total =0
        batches = 0

        for image_batch in dataset:
            gen_loss, disc_loss = train_step(image_batch)
            gen_total += gen_loss
            disc_total += disc_loss
            batches+=1

        gen_losses.append(gen_total / batches)
        disc_losses.append(disc_total / batches)

        display.clear_output(wait=True)
        generate_and_save_images(generator, epoch +1,seed)

        print(f'Epoch {epoch +1}, Gen Loss: {gen_losses[-1]:.4f}, Disc Loss: {disc_
        print(f'Time for epoch {epoch +1} is {time.time() -start:.2f} sec')

    display.clear_output(wait=True)
    generate_and_save_images(generator, epochs, seed)
```

In [49]:
```
EPOCHS = 10
seed = tf.random.normal([16, 100])

train(train_dataset, EPOCHS)
```

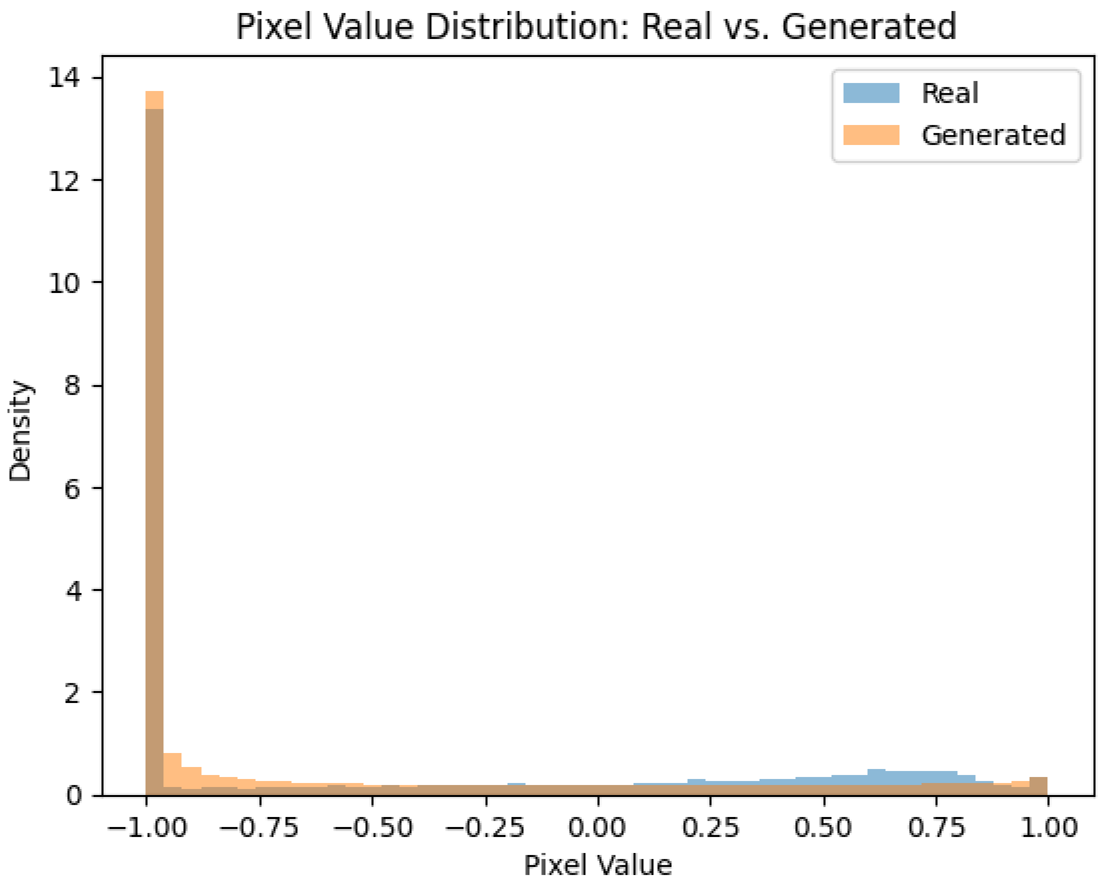


In [50]:
```
plt.plot(gen_losses, label='Generator Loss')
plt.plot(disc_losses, label='Discriminator Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Curves')
plt.legend()
plt.show()
```



In [51]:

```
#sample 1000 generated images
generated =generator(tf.random.normal([1000, 100]), training=False).numpy()
generated = generated.reshape(-1)

#Sample 1000 real images
real = train_images[:1000].reshape(-1)

#histogram
plt.hist(real, bins=50, alpha=0.5,label='Real', density=True)
plt.hist(generated, bins=50, alpha=0.5,label='Generated', density=True)
plt.title('Pixel Value Distribution: Real vs. Generated')
plt.xlabel('Pixel Value')
plt.ylabel('Density')
plt.legend()
plt.show()
```



Pixel Value Distribution: Real vs. Generated
Real
Generated
Density
Pixel Value
0
2
4
6
8
10
12
14
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

In [ ]: