```
import tensorflow as tf
```

```
tf.__version__
```

```
'2.7.0'
```

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

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

```
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]
```

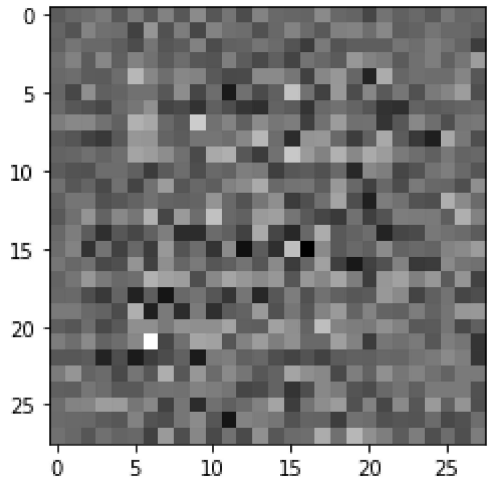
```
BUFFER_SIZE = 60000
BATCH_SIZE = 256
```

```
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(B
```

```
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=Fa
    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=Fal
    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=Fals
    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')
```

```
<matplotlib.image.AxesImage at 0x7ff8f7f02410>
```




```
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)
```

```
tf.Tensor([[0.00013076]], shape=(1, 1), dtype=float32)
```

```
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 = 21
noise_dim = 100
num_examples_to_generate = 16

# Here we are going to reuse this seed overtime (so it's easier) to visualize progress in the
seed = tf.random.normal([num_examples_to_generate, noise_dim])


# Again we are reusing this seed overtime (so it's easier) to visualize progress in the anima
seed = tf.random.normal([num_examples_to_generate, noise_dim])


# we are going to use `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_variab

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_varia
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.tra

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

    # Saving 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 and this is so all layers run in inference mode (batchn
  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)
```

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

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7ff8f84d7d10>
```

```
# Displaying 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)
```

```
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)

import tensorflow_docs.vis.embed as embed
embed.embed_file(anim_file)
```

```
pip install git+https://github.com/tensorflow/docs
```

```
Collecting git+https://github.com/tensorflow/docs
  Cloning https://github.com/tensorflow/docs to /tmp/pip-req-build-utrx5dwu
  Running command git clone -q https://github.com/tensorflow/docs /tmp/pip-req-build-utr
Requirement already satisfied: astor in /usr/local/lib/python3.7/dist-packages (from ter
Requirement already satisfied: absl-py in /usr/local/lib/python3.7/dist-packages (from t
Requirement already satisfied: protobuf>=3.14 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (from te
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.7/dist-packages (from
Building wheels for collected packages: tensorflow-docs
  Building wheel for tensorflow-docs (setup.py) ... done
  Created wheel for tensorflow-docs: filename=tensorflow_docs-0.0.0.dev0-py3-none-any.wh
  Stored in directory: /tmp/pip-ephem-wheel-cache-eencv5ol/wheels/cc/c4/d8/5341e93b6376c
Successfully built tensorflow-docs
Installing collected packages: tensorflow-docs
Successfully installed tensorflow-docs-0.0.0.dev0
```

✓ 0s completed at 11:29 AM