```
import tensorflow as tf
```

```
tf.__version__
```

```
'2.15.0'
```

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

```
Requirement already satisfied: imageio in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pillow<10.1.0,>=8.3.2 in /usr/local/lib/python3
Collecting git+https://github.com/tensorflow/docs
  Cloning https://github.com/tensorflow/docs to /tmp/pip-req-build-q6ndd4q5
  Running command git clone --filter=blob:none --quiet https://github.com/tens
  Resolved https://github.com/tensorflow/docs to commit 42550ed44d5bfa9f372bc5
  Preparing metadata (setup.py) ... done
Collecting astor (from tensorflow-docs==2024.2.5.73858)
  Downloading astor-0.8.1-py2.py3-none-any.whl (27 kB)
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: protobuf>=3.12 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.10/dis
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/di
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.10/dist-
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/lc
```

```
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.1
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/
Building wheels for collected packages: tensorflow-docs
  Building wheel for tensorflow-docs (setup.py) ... done
  Created wheel for tensorflow-docs: filename=tensorflow_docs-2024.2.5.73858-p
  Stored in directory: /tmp/pip-ephem-wheel-cache-bnw8z5w1/wheels/86/0f/1e/3b6
Successfully built tensorflow-docs
Installing collected packages: astor, tensorflow-docs
Successfully installed astor-0.8.1 tensorflow-docs-2024.2.5.73858
```

```
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
```

Load and prepare the dataset

You will use the MNIST dataset to train the generator and the discriminator. The generator will generate handwritten digits resembling the MNIST data.

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
11490434/11490434 [==============================] - 2s 0us/step
```

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

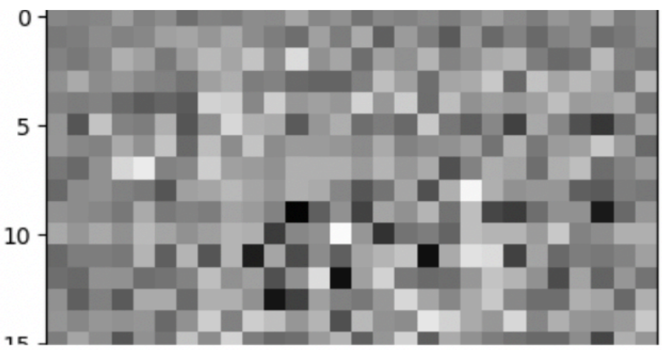
```
BUFFER_SIZE = 60000
BATCH_SIZE = 256
```

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

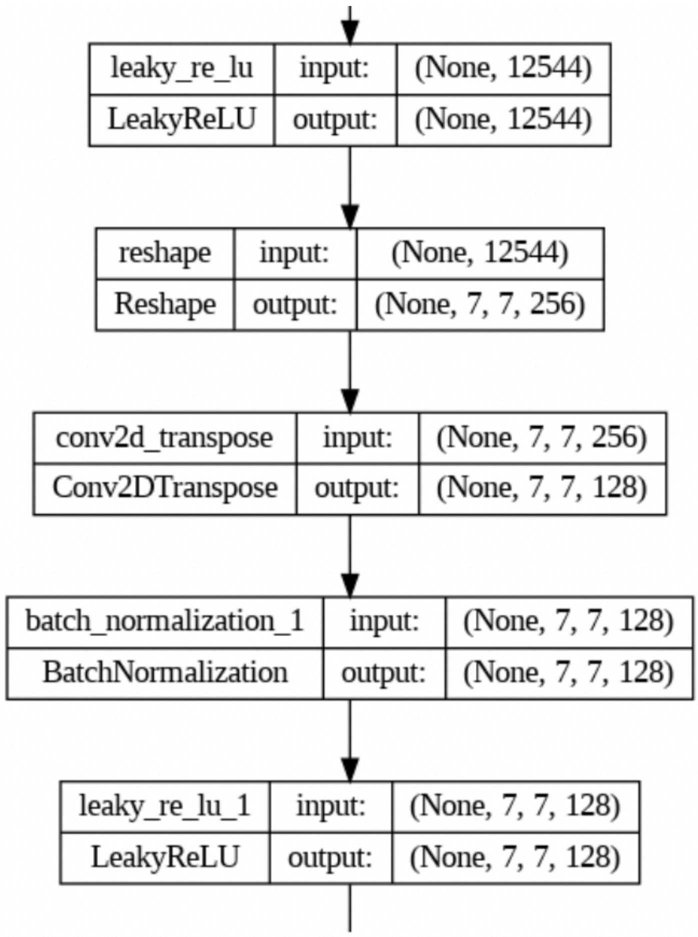
```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    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',
    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', u
    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', us
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

Use the (as yet untrained) generator to create an image.

```
generator = make_generator_model()

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

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




```
from tensorflow.keras.utils import plot_model
plot_model(generator, to_file='model_plot.png', show_shapes=True, show_layer_names
```



The Discriminator

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
```

Use the (as yet untrained) discriminator to classify the generated images as real or fake. The model will be trained to output positive values for real images, and negative values for fake

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

```
tf.Tensor([[-0.0020717]], shape=(1, 1), dtype=float32)
```

Define the loss and optimizers

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)
```

Discriminator loss

This method quantifies how well the discriminator is able to distinguish real images from fakes. It compares the discriminator's predictions on real images to an array of 1s, and the discriminator's predictions on fake (generated) images to an array of 0s.

```
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
```

Generator loss

The generator's loss quantifies how well it was able to trick the discriminator. Intuitively, if the generator is performing well, the discriminator will classify the fake images as real (or 1). Here, compare the discriminators decisions on the generated images to an array of 1s.

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

The discriminator and the generator optimizers are different since you will train two networks separately.

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

Save checkpoints

This notebook also demonstrates how to save and restore models, which can be helpful in case a long running training task is interrupted.

```
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)
```

Define the training loop

```
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])
```

The training loop begins with generator receiving a random seed as input. That seed is used to produce an image. The discriminator is then used to classify real images (drawn from the training set) and fakes images (produced by the generator). The loss is calculated for each of these models, and the gradients are used to update the generator and discriminator.

```
# 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_varia
    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
```

```
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)
```

Generate and save images

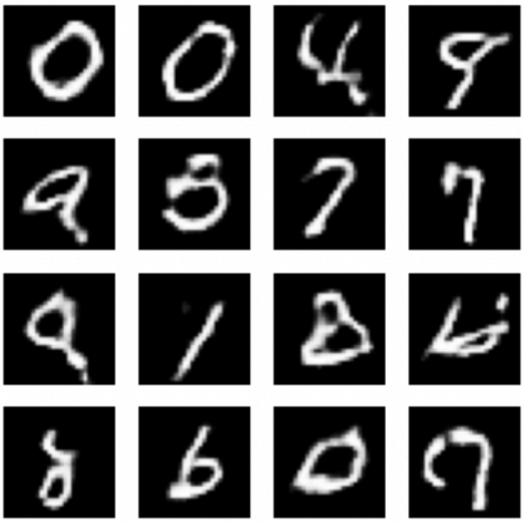
```
def generate_and_save_images(model, epoch, test_input):
  # Notice `training` is set to False.
  # This is so all layers run in inference mode (batchnorm).
  predictions = model(test_input, training=False)

  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 the model

Call the `train()` method defined above to train the generator and discriminator simultaneously. Note, training GANs can be tricky. It's important that the generator and discriminator do not overpower each other (e.g., that they train at a similar rate).

At the beginning of the training, the generated images look like random noise. As training progresses, the generated digits will look increasingly real. After about 50 epochs, they resemble MNIST digits. This may take about one minute / epoch with the default settings on Colab

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



Restore the latest checkpoint.

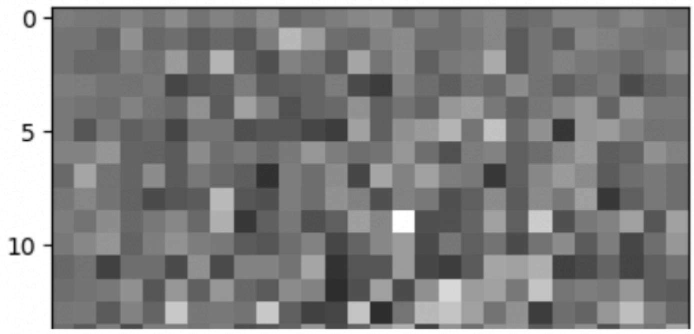
```
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at
0x7abcb4406920>
```

```
seed = tf.random.normal([1, noise_dim])
```

```
generator = make_generator_model()

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

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

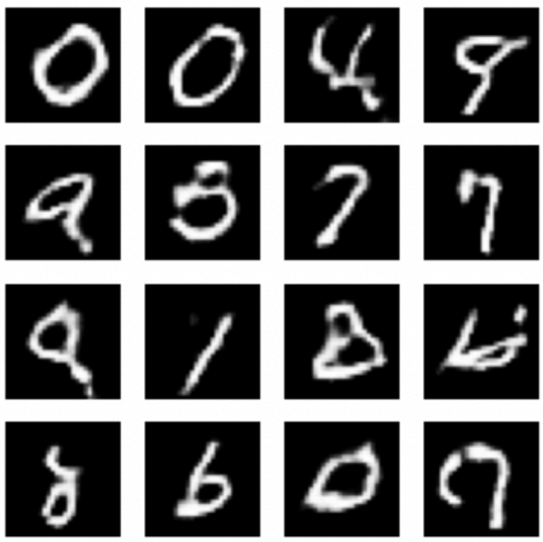
```
<matplotlib.image.AxesImage at 0x7abc002d69e0>
```



Create a GIF

```
# 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)
```



Use `imageio` to create an animated gif using the images saved during training.

```
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)

    <ipython-input-31-56bb6d34be2e>:7: DeprecationWarning: Starting with ImageIO v
      image = imageio.imread(filename)
    <ipython-input-31-56bb6d34be2e>:9: DeprecationWarning: Starting with ImageIO v
      image = imageio.imread(filename)
```

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