```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
import matplotlib.pyplot as plt

print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.19.0
```

```
dataset_choice = "mnist"       # "mnist" or "fashion"
epochs = 50
batch_size = 128
noise_dim = 100
learning_rate = 0.0002
save_interval = 5
```

```
dataset_choice = "fashion"
epochs = 50
batch_size = 128
noise_dim = 100
learning_rate = 0.0002
save_interval = 5
```

```
if dataset_choice == "mnist":
    (x_train, y_train), _ = tf.keras.datasets.mnist.load_data()
elif dataset_choice == "fashion":
    (x_train, y_train), _ = tf.keras.datasets.fashion_mnist.load_data()

# Normalize to [-1,1]
x_train = (x_train.astype("float32") - 127.5) / 127.5
x_train = np.expand_dims(x_train, axis=-1)

BUFFER_SIZE = x_train.shape[0]
train_dataset = tf.data.Dataset.from_tensor_slices(x_train)\
    .shuffle(BUFFER_SIZE).batch(batch_size)

img_shape = (28, 28, 1)
```

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

```
def build_generator():
    model = tf.keras.Sequential([
        layers.Dense(7 * 7 * 256, use_bias=False, input_shape=(noise_dim,)),
        layers.BatchNormalization(),
        layers.LeakyReLU(),

        layers.Reshape((7, 7, 256)),

        layers.Conv2DTranspose(128, 5, strides=1, padding="same", use_bias=False),
        layers.BatchNormalization(),
        layers.LeakyReLU(),

        layers.Conv2DTranspose(64, 5, strides=2, padding="same", use_bias=False),
        layers.BatchNormalization(),
        layers.LeakyReLU(),

        layers.Conv2DTranspose(1, 5, strides=2, padding="same",
                               activation="tanh", use_bias=False)
    ])
    return model

generator = build_generator()
generator.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                   │ (None, 12544)          │     1,254,400 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization             │ (None, 12544)          │        50,176 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ leaky_re_lu (LeakyReLU)         │ (None, 12544)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ reshape (Reshape)               │ (None, 7, 7, 256)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_transpose                │ (None, 7, 7, 128)      │       819,200 │
│ (Conv2DTranspose)               │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_1           │ (None, 7, 7, 128)      │           512 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ leaky_re_lu_1 (LeakyReLU)       │ (None, 7, 7, 128)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_transpose_1              │ (None, 14, 14, 64)     │       204,800 │
│ (Conv2DTranspose)               │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_2           │ (None, 14, 14, 64)     │           256 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ leaky_re_lu_2 (LeakyReLU)       │ (None, 14, 14, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_transpose_2              │ (None, 28, 28, 1)      │         1,600 │
│ (Conv2DTranspose)               │                        │               │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 2,330,944 (8.89 MB)
 Trainable params: 2,305,472 (8.79 MB)
```

```
def build_discriminator():
    model = tf.keras.Sequential([
        layers.Conv2D(64, 5, strides=2, padding="same", input_shape=img_shape),
        layers.LeakyReLU(),
        layers.Dropout(0.3),

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

        layers.Flatten(),
        layers.Dense(1, activation="sigmoid")
    ])
    return model

discriminator = build_discriminator()
discriminator.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                 │ (None, 14, 14, 64)     │         1,664 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ leaky_re_lu_3 (LeakyReLU)       │ (None, 14, 14, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 14, 14, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)               │ (None, 7, 7, 128)      │       204,928 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ leaky_re_lu_4 (LeakyReLU)       │ (None, 7, 7, 128)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_1 (Dropout)             │ (None, 7, 7, 128)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 6272)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 1)              │         6,273 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 212,865 (831.50 KB)
 Trainable params: 212,865 (831.50 KB)
 Non-trainable params: 0 (0.00 B)
```

```
loss_fn = tf.keras.losses.BinaryCrossentropy()

gen_optimizer = tf.keras.optimizers.Adam(learning_rate, beta_1=0.5)
disc_optimizer = tf.keras.optimizers.Adam(learning_rate, beta_1=0.5)
```

```
@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:
        fake_images = generator(noise, training=True)

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

        gen_loss = loss_fn(tf.ones_like(fake_output), fake_output)
        disc_loss = (
            loss_fn(tf.ones_like(real_output), real_output) +
            loss_fn(tf.zeros_like(fake_output), fake_output)
        )

    gen_gradients = gen_tape.gradient(gen_loss, generator.trainable_variables)
    disc_gradients = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    gen_optimizer.apply_gradients(zip(gen_gradients, generator.trainable_variables))
    disc_optimizer.apply_gradients(zip(disc_gradients, discriminator.trainable_variables))

    disc_acc = tf.reduce_mean(tf.cast(real_output > 0.5, tf.float32))
    return gen_loss, disc_loss, disc_acc
```

```
def save_images(epoch):
    os.makedirs("generated_samples", exist_ok=True)
    noise = tf.random.normal([25, noise_dim])
    images = generator(noise, training=False)
    images = (images + 1) / 2

    fig, axs = plt.subplots(5, 5, figsize=(5,5))
    idx = 0
    for i in range(5):
        for j in range(5):
            axs[i,j].imshow(images[idx,:,:,0], cmap="gray")
            axs[i,j].axis("off")
            idx += 1

    plt.savefig(f"generated_samples/epoch_{epoch:02d}.png")
    plt.close()
```

```
for epoch in range(1, epochs + 1):
    for batch in train_dataset:
        g_loss, d_loss, d_acc = train_step(batch)

    print(f"Epoch {epoch}/{epochs} | "
          f"D_loss: {d_loss:.2f} | "
          f"D_acc: {d_acc*100:.2f}% | "
          f"G_loss: {g_loss:.2f}")

    if epoch % save_interval == 0:
        save_images(epoch)
```

```
Epoch 1/50 | D_loss: 1.20 | D_acc: 81.25% | G_loss: 0.76
Epoch 2/50 | D_loss: 1.35 | D_acc: 14.58% | G_loss: 1.00
Epoch 3/50 | D_loss: 1.17 | D_acc: 56.25% | G_loss: 1.01
Epoch 4/50 | D_loss: 1.24 | D_acc: 29.17% | G_loss: 1.24
Epoch 5/50 | D_loss: 1.26 | D_acc: 52.08% | G_loss: 0.91
Epoch 6/50 | D_loss: 1.38 | D_acc: 69.79% | G_loss: 0.68
Epoch 7/50 | D_loss: 1.37 | D_acc: 65.62% | G_loss: 0.71
Epoch 8/50 | D_loss: 1.31 | D_acc: 51.04% | G_loss: 0.86
Epoch 9/50 | D_loss: 1.48 | D_acc: 42.71% | G_loss: 0.79
Epoch 10/50 | D_loss: 1.34 | D_acc: 60.42% | G_loss: 0.76
Epoch 11/50 | D_loss: 1.32 | D_acc: 21.88% | G_loss: 1.12
Epoch 12/50 | D_loss: 1.23 | D_acc: 72.92% | G_loss: 0.81
Epoch 13/50 | D_loss: 1.32 | D_acc: 62.50% | G_loss: 0.79
Epoch 14/50 | D_loss: 1.24 | D_acc: 71.88% | G_loss: 0.80
Epoch 15/50 | D_loss: 1.26 | D_acc: 68.75% | G_loss: 0.79
Epoch 16/50 | D_loss: 1.26 | D_acc: 65.62% | G_loss: 0.85
Epoch 17/50 | D_loss: 1.34 | D_acc: 37.50% | G_loss: 0.92
Epoch 18/50 | D_loss: 1.33 | D_acc: 76.04% | G_loss: 0.68
Epoch 19/50 | D_loss: 1.36 | D_acc: 54.17% | G_loss: 0.77
Epoch 20/50 | D_loss: 1.31 | D_acc: 76.04% | G_loss: 0.71
Epoch 21/50 | D_loss: 1.20 | D_acc: 68.75% | G_loss: 0.86
Epoch 22/50 | D_loss: 1.23 | D_acc: 67.71% | G_loss: 0.86
Epoch 23/50 | D_loss: 1.19 | D_acc: 67.71% | G_loss: 0.98
Epoch 24/50 | D_loss: 1.29 | D_acc: 72.92% | G_loss: 0.74
Epoch 25/50 | D_loss: 1.25 | D_acc: 62.50% | G_loss: 0.87
Epoch 26/50 | D_loss: 1.23 | D_acc: 62.50% | G_loss: 0.84
Epoch 27/50 | D_loss: 1.25 | D_acc: 73.96% | G_loss: 0.76
Epoch 28/50 | D_loss: 1.41 | D_acc: 37.50% | G_loss: 0.99
Epoch 29/50 | D_loss: 1.32 | D_acc: 65.62% | G_loss: 0.78
Epoch 30/50 | D_loss: 1.30 | D_acc: 71.88% | G_loss: 0.74
Epoch 31/50 | D_loss: 1.30 | D_acc: 59.38% | G_loss: 0.80
Epoch 32/50 | D_loss: 1.31 | D_acc: 76.04% | G_loss: 0.68
Epoch 33/50 | D_loss: 1.30 | D_acc: 56.25% | G_loss: 0.83
Epoch 34/50 | D_loss: 1.51 | D_acc: 33.33% | G_loss: 0.90
Epoch 35/50 | D_loss: 1.25 | D_acc: 56.25% | G_loss: 0.96
Epoch 36/50 | D_loss: 1.27 | D_acc: 61.46% | G_loss: 0.97
Epoch 37/50 | D_loss: 1.34 | D_acc: 46.88% | G_loss: 0.87
Epoch 38/50 | D_loss: 1.34 | D_acc: 61.46% | G_loss: 0.84
Epoch 39/50 | D_loss: 1.32 | D_acc: 63.54% | G_loss: 0.75
Epoch 40/50 | D_loss: 1.26 | D_acc: 45.83% | G_loss: 0.97
Epoch 41/50 | D_loss: 1.30 | D_acc: 61.46% | G_loss: 0.80
Epoch 42/50 | D_loss: 1.29 | D_acc: 63.54% | G_loss: 0.86
Epoch 43/50 | D_loss: 1.32 | D_acc: 43.75% | G_loss: 0.93
Epoch 44/50 | D_loss: 1.27 | D_acc: 54.17% | G_loss: 0.91
Epoch 45/50 | D_loss: 1.34 | D_acc: 62.50% | G_loss: 0.79
Epoch 46/50 | D_loss: 1.35 | D_acc: 54.17% | G_loss: 0.85
Epoch 47/50 | D_loss: 1.32 | D_acc: 48.96% | G_loss: 0.86
Epoch 48/50 | D_loss: 1.33 | D_acc: 57.29% | G_loss: 0.79
Epoch 49/50 | D_loss: 1.30 | D_acc: 54.17% | G_loss: 0.85
Epoch 50/50 | D_loss: 1.31 | D_acc: 79.17% | G_loss: 0.67
```

```python
os.makedirs("final_generated_images", exist_ok=True)

noise = tf.random.normal([100, noise_dim])
final_images = generator(noise, training=False)
final_images = (final_images + 1) / 2

for i in range(100):
    plt.imsave(f"final_generated_images/img_{i+1}.png",
               final_images[i,:,:,0], cmap="gray")
```

```python
classifier = tf.keras.Sequential([
    layers.Conv2D(32, 3, activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(10, activation='softmax')
])

classifier.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

classifier.fit(x_train, y_train, epochs=5, verbose=0)

preds = classifier.predict(final_images)
labels = np.argmax(preds, axis=1)

print("\nLabel Distribution of Generated Images:")
unique, counts = np.unique(labels, return_counts=True)
for u, c in zip(unique, counts):
    print(f"Label {u}: {c}")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 120ms/step

Label Distribution of Generated Images:
Label 0: 5
Label 1: 1
Label 2: 5
Label 3: 7
Label 4: 2
Label 5: 1
Label 6: 9
Label 7: 8
Label 8: 46
Label 9: 16
```