```
import os
import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist, fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Reshape, Flatten, LeakyReLU
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy
```

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

```
# -------- Load Dataset --------
if dataset_choice == "mnist":
    (X_train, _), (_, _) = mnist.load_data()
else:
    (X_train, _), (_, _) = fashion_mnist.load_data()

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

print("Dataset Shape:", X_train.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Dataset Shape: (60000, 28, 28, 1)
```

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

```
def build_generator():
    model = Sequential()
    model.add(Dense(256, input_dim=noise_dim))
    model.add(LeakyReLU(0.2))

    model.add(Dense(512))
    model.add(LeakyReLU(0.2))

    model.add(Dense(28 * 28, activation="tanh"))
    model.add(Reshape((28, 28, 1)))

    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_sha
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	25,856
leaky_re_lu (LeakyReLU)	(None, 256)	0
dense_1 (Dense)	(None, 512)	131,584
leaky_re_lu_1 (LeakyReLU)	(None, 512)	0
dense_2 (Dense)	(None, 784)	402,192
reshape (Reshape)	(None, 28, 28, 1)	0

```
Total params: 559,632 (2.13 MB)
Trainable params: 559,632 (2.13 MB)
Non-trainable params: 0 (0.00 B)
```

```
def build_discriminator():
    model = Sequential()
    model.add(Flatten(input_shape=(28, 28, 1)))

    model.add(Dense(512))
    model.add(LeakyReLU(0.2))

    model.add(Dense(256))
    model.add(LeakyReLU(0.2))

    model.add(Dense(1, activation="sigmoid"))
    return model

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

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `in
  super().__init__(**kwargs)
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense_3 (Dense)	(None, 512)	401,920
leaky_re_lu_2 (LeakyReLU)	(None, 512)	0
dense_4 (Dense)	(None, 256)	131,328
leaky_re_lu_3 (LeakyReLU)	(None, 256)	0
dense_5 (Dense)	(None, 1)	257

```
Total params: 533,505 (2.04 MB)
Trainable params: 533,505 (2.04 MB)
Non-trainable params: 0 (0.00 B)
```

```
optimizer = Adam(learning_rate)
loss_fn = BinaryCrossentropy()

# Compile Discriminator
discriminator.compile(
    loss=loss_fn,
    optimizer=optimizer,
    metrics=["accuracy"]
)

# Combined GAN
discriminator.trainable = False
gan = Sequential([generator, discriminator])
gan.compile(loss=loss_fn, optimizer=optimizer)
```

```
def save_generated_images(epoch):
    noise = np.random.normal(0, 1, (25, noise_dim))
    gen_images = generator.predict(noise)
    gen_images = (gen_images + 1) / 2  # Rescale to [0,1]

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

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

```
for epoch in range(1, epochs + 1):

    # ----- Train Discriminator -----
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    real_imgs = X_train[idx]

    noise = np.random.normal(0, 1, (batch_size, noise_dim))
    fake_imgs = generator.predict(noise)

    real_labels = np.ones((batch_size, 1))
```

```
    fake_labels = np.zeros((batch_size, 1))

    d_loss_real = discriminator.train_on_batch(real_imgs, real_labels)
    d_loss_fake = discriminator.train_on_batch(fake_imgs, fake_labels)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    # ----- Train Generator -----
    noise = np.random.normal(0, 1, (batch_size, noise_dim))
    g_loss = gan.train_on_batch(noise, real_labels)

    # ----- Print Logs -----
    print(f"Epoch {epoch}/{epochs} | "
          f"D_loss: {d_loss[0]:.2f} | "
          f"D_acc: {d_loss[1]*100:.2f}% | "
          f"G_loss: {g_loss:.2f}")

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

```
Epoch 25/50 | D_loss: 1.96 | D_acc: 50.60% | G_loss: 0.10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
Epoch 26/50 | D_loss: 2.02 | D_acc: 50.58% | G_loss: 0.09
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
Epoch 27/50 | D_loss: 2.07 | D_acc: 50.56% | G_loss: 0.09
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Epoch 28/50 | D_loss: 2.12 | D_acc: 50.54% | G_loss: 0.09
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
Epoch 29/50 | D_loss: 2.17 | D_acc: 50.52% | G_loss: 0.09
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Epoch 30/50 | D_loss: 2.22 | D_acc: 50.50% | G_loss: 0.08
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 31/50 | D_loss: 2.27 | D_acc: 50.49% | G_loss: 0.08
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 32/50 | D_loss: 2.31 | D_acc: 50.47% | G_loss: 0.08
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 33/50 | D_loss: 2.36 | D_acc: 50.46% | G_loss: 0.07
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
Epoch 34/50 | D_loss: 2.40 | D_acc: 50.44% | G_loss: 0.07
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
Epoch 35/50 | D_loss: 2.44 | D_acc: 50.43% | G_loss: 0.07
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 36/50 | D_loss: 2.48 | D_acc: 50.42% | G_loss: 0.07
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 37/50 | D_loss: 2.52 | D_acc: 50.41% | G_loss: 0.07
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 38/50 | D_loss: 2.55 | D_acc: 50.40% | G_loss: 0.07
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 39/50 | D_loss: 2.59 | D_acc: 50.39% | G_loss: 0.06
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 40/50 | D_loss: 2.62 | D_acc: 50.38% | G_loss: 0.06
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 41/50 | D_loss: 2.66 | D_acc: 50.37% | G_loss: 0.06
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 42/50 | D_loss: 2.69 | D_acc: 50.36% | G_loss: 0.06
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
Epoch 43/50 | D_loss: 2.72 | D_acc: 50.35% | G_loss: 0.06
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 44/50 | D_loss: 2.75 | D_acc: 50.34% | G_loss: 0.06
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 45/50 | D_loss: 2.78 | D_acc: 50.33% | G_loss: 0.06
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 46/50 | D_loss: 2.81 | D_acc: 50.33% | G_loss: 0.05
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
Epoch 47/50 | D_loss: 2.84 | D_acc: 50.32% | G_loss: 0.05
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 48/50 | D_loss: 2.86 | D_acc: 50.31% | G_loss: 0.05
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 49/50 | D_loss: 2.89 | D_acc: 50.31% | G_loss: 0.05
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 50/50 | D_loss: 2.92 | D_acc: 50.30% | G_loss: 0.05
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
```

```
noise = np.random.normal(0, 1, (100, noise_dim))
final_images = generator.predict(noise)
final_images = (final_images + 1) / 2

for i in range(100):
    plt.imshow(final_images[i, :, :, 0], cmap="gray")
    plt.axis("off")
    plt.savefig(f"final_generated_images/img_{i}.png")
```

```
    plt.close()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
```

```
from tensorflow.keras.models import load_model

# Load a pretrained MNIST classifier (assumed)
# classifier = load_model("mnist_classifier.h5")

# Example placeholder label distribution
labels = np.random.randint(0, 10, 100)

unique, counts = np.unique(labels, return_counts=True)
label_distribution = dict(zip(unique, counts))

print("Predicted Label Distribution:")
print(label_distribution)
```

```
Predicted Label Distribution:
{np.int64(0): np.int64(13), np.int64(1): np.int64(9), np.int64(2): np.int64(6), np.int64(3): np.int64(9), np.int6
```

```
Start coding or generate with AI.
```