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

import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
```

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

```
if dataset_choice == 'mnist':
    (x_train, _), (_, _) = tf.keras.datasets.mnist.load_data()
elif dataset_choice == 'fashion':
    (x_train, _), (_, _) = tf.keras.datasets.fashion_mnist.load_data()
else:
    raise ValueError("Invalid dataset choice")

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

img_shape = x_train.shape[1:]
```

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

```
def build_generator():
    model = tf.keras.Sequential([
        layers.Dense(256, input_dim=noise_dim),
        layers.LeakyReLU(0.2),
        layers.BatchNormalization(momentum=0.8),

        layers.Dense(512),
        layers.LeakyReLU(0.2),
        layers.BatchNormalization(momentum=0.8),

        layers.Dense(1024),
        layers.LeakyReLU(0.2),
        layers.BatchNormalization(momentum=0.8),

        layers.Dense(np.prod(img_shape), activation='tanh'),
        layers.Reshape(img_shape)
    ])
    return model
```

```
def build_discriminator():
    model = models.Sequential([
        layers.Flatten(input_shape=img_shape),
        layers.Dense(512),
        layers.LeakyReLU(0.2),

        layers.Dense(256),
        layers.LeakyReLU(0.2),

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

discriminator = build_discriminator()
```

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

```
discriminator.compile(
    loss='binary_crossentropy',
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001, beta_1=0.5),
    metrics=['accuracy']
)
```

Gemini

```
discriminator.trainable = False

generator = build_generator()
gan_input = layers.Input(shape=(noise_dim,))
fake_img = generator(gan_input)
gan_output = discriminator(fake_img)

gan = models.Model(gan_input, gan_output)

gan.compile(
    loss='binary_crossentropy',
    optimizer=optimizers.Adam(learning_rate, 0.5)
)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
def save_images(epoch):
    noise = np.random.normal(0, 1, (25, noise_dim))
    gen_imgs = generator.predict(noise)

    gen_imgs = (gen_imgs + 1) / 2.0

    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(gen_imgs[idx, :, :, 0], cmap='gray')
            axs[i, j].axis('off')
            idx += 1

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

```
half_batch = batch_size // 2

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

    # Train Discriminator
    idx = np.random.randint(0, x_train.shape[0], half_batch)
    real_imgs = x_train[idx]

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

    d_loss_real = discriminator.train_on_batch(real_imgs, np.ones((half_batch, 1)))
    d_loss_fake = discriminator.train_on_batch(fake_imgs, np.zeros((half_batch, 1)))

    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, np.ones((batch_size, 1)))

    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_images(epoch)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Epoch 25/50 | D_loss: 1.33 | D_acc: 5.90% | G_loss: 0.43
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Epoch 26/50 | D_loss: 1.34 | D_acc: 5.67% | G_loss: 0.42
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Epoch 27/50 | D_loss: 1.36 | D_acc: 5.46% | G_loss: 0.41
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Epoch 28/50 | D_loss: 1.37 | D_acc: 5.27% | G_loss: 0.40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Epoch 29/50 | D_loss: 1.38 | D_acc: 5.08% | G_loss: 0.39
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
Epoch 30/50 | D_loss: 1.39 | D_acc: 4.94% | G_loss: 0.39
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Epoch 31/50 | D_loss: 1.41 | D_acc: 4.83% | G_loss: 0.38
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Epoch 32/50 | D_loss: 1.42 | D_acc: 4.70% | G_loss: 0.37
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Epoch 33/50 | D_loss: 1.43 | D_acc: 4.56% | G_loss: 0.36
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Epoch 34/50 | D_loss: 1.44 | D_acc: 4.42% | G_loss: 0.36
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
Epoch 35/50 | D_loss: 1.45 | D_acc: 4.32% | G_loss: 0.35
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Epoch 36/50 | D_loss: 1.47 | D_acc: 4.22% | G_loss: 0.34
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Epoch 37/50 | D_loss: 1.48 | D_acc: 4.10% | G_loss: 0.34
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Epoch 38/50 | D_loss: 1.49 | D_acc: 4.02% | G_loss: 0.33
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Epoch 39/50 | D_loss: 1.50 | D_acc: 3.91% | G_loss: 0.33
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
Epoch 40/50 | D_loss: 1.51 | D_acc: 3.83% | G_loss: 0.32
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Epoch 41/50 | D_loss: 1.52 | D_acc: 3.74% | G_loss: 0.31
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Epoch 42/50 | D_loss: 1.53 | D_acc: 3.65% | G_loss: 0.31
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Epoch 43/50 | D_loss: 1.54 | D_acc: 3.56% | G_loss: 0.30
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
Epoch 44/50 | D_loss: 1.55 | D_acc: 3.48% | G_loss: 0.30
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Epoch 45/50 | D_loss: 1.56 | D_acc: 3.40% | G_loss: 0.29
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
Epoch 46/50 | D_loss: 1.57 | D_acc: 3.33% | G_loss: 0.29
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Epoch 47/50 | D_loss: 1.59 | D_acc: 3.26% | G_loss: 0.29
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Epoch 48/50 | D_loss: 1.60 | D_acc: 3.21% | G_loss: 0.28
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Epoch 49/50 | D_loss: 1.61 | D_acc: 3.14% | G_loss: 0.28
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
Epoch 50/50 | D_loss: 1.62 | D_acc: 3.08% | G_loss: 0.27
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
```

```
noise = np.random.normal(0, 1, (100, noise_dim))
final_images = generator.predict(noise)
final_images = (final_images + 1) / 2.0

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

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 124ms/step
```

```
classifier = tf.keras.Sequential([
    layers.Input(shape=(28, 28, 1)),

    layers.Conv2D(32, (3,3)),
```

```
    layers.LeakyReLU(0.2),
    layers.MaxPooling2D(),

    layers.Conv2D(64, (3,3)),
    layers.LeakyReLU(0.2),
    layers.MaxPooling2D(),

    layers.Flatten(),

    layers.Dense(128),
    layers.LeakyReLU(0.2),

    layers.Dense(10, activation='softmax')
])

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

```
# Reload labels properly
if dataset_choice == 'mnist':
    (_, y_train), (_, _) = tf.keras.datasets.mnist.load_data()
else:
    (_, y_train), (_, _) = tf.keras.datasets.fashion_mnist.load_data()

classifier.fit(
    x_train, y_train,
    epochs=5,
    batch_size=128,
    verbose=1
)
```

```
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.8799 - loss: 0.3965
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9852 - loss: 0.0469
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9897 - loss: 0.0345
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9930 - loss: 0.0221
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9948 - loss: 0.0170
<keras.src.callbacks.history.History at 0x7d88d2d07650>
```

```
generated_imgs = final_images.reshape(-1, 28, 28, 1)

predictions = classifier.predict(generated_imgs)
predicted_labels = np.argmax(predictions, axis=1)

unique, counts = np.unique(predicted_labels, return_counts=True)

print("Label Distribution of Generated Images:")
for u, c in zip(unique, counts):
    print(f"Label {u}: {c} images")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 139ms/step
Label Distribution of Generated Images:
Label 8: 100 images
```