```
# Mount Drive (optional but recommended)
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

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

```
# ================= USER CONFIG =================
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:]
```

```
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(256, input_dim=noise_dim),
        layers.LeakyReLU(0.2),

        layers.Dense(512),
        layers.LeakyReLU(0.2),

        layers.Dense(1024),
        layers.LeakyReLU(0.2),

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

```
def build_discriminator():
    model = tf.keras.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
```

```
generator = build_generator()
discriminator = build_discriminator()

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

# Combined GAN
discriminator.trainable = False
noise = layers.Input(shape=(noise_dim,))
fake_img = generator(noise)
validity = discriminator(fake_img)

gan = tf.keras.Model(noise, validity)
gan.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate),
    loss='binary_crossentropy'
)
```

```
/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)
/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)
```

```
def save_images(epoch):
    os.makedirs("generated_samples", exist_ok=True)

    noise = np.random.normal(0, 1, (25, noise_dim))
    gen_imgs = generator.predict(noise)
    gen_imgs = 0.5 * gen_imgs + 0.5

    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_imgs[cnt,:,:,0], cmap='gray')
            axs[i,j].axis('off')
            cnt += 1
    plt.savefig(f"generated_samples/epoch_{epoch:02d}.png")
    plt.close()
```

```
real = np.ones((batch_size, 1))
fake = np.zeros((batch_size, 1))

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

    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))
    gen_imgs = generator.predict(noise)

    d_loss_real = discriminator.train_on_batch(real_imgs, real)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    noise = np.random.normal(0, 1, (batch_size, noise_dim))
    g_loss = gan.train_on_batch(noise, real)

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

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

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step
/usr/local/lib/python3.12/dist-packages/keras/src/backend/tensorflow/trainer.py:83: UserWarning: The model does not have any t
  warnings.warn("The model does not have any trainable weights.")
Epoch 1/50 | D_loss: 0.98 | D_acc: 26.17% | G_loss: 0.81
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

```
Epoch 2/50 | D_loss: 0.90 | D_acc: 30.96% | G_loss: 0.70
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 3/50 | D_loss: 0.95 | D_acc: 19.77% | G_loss: 0.61
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 4/50 | D_loss: 1.00 | D_acc: 14.86% | G_loss: 0.53
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 5/50 | D_loss: 1.06 | D_acc: 11.79% | G_loss: 0.47
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 300ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 6/50 | D_loss: 1.13 | D_acc: 9.80% | G_loss: 0.42
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 7/50 | D_loss: 1.22 | D_acc: 8.52% | G_loss: 0.37
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 8/50 | D_loss: 1.30 | D_acc: 7.57% | G_loss: 0.33
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 9/50 | D_loss: 1.39 | D_acc: 6.75% | G_loss: 0.30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 10/50 | D_loss: 1.49 | D_acc: 6.29% | G_loss: 0.27
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
Epoch 11/50 | D_loss: 1.60 | D_acc: 5.78% | G_loss: 0.25
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 12/50 | D_loss: 1.70 | D_acc: 5.42% | G_loss: 0.23
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 13/50 | D_loss: 1.81 | D_acc: 5.18% | G_loss: 0.21
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 14/50 | D_loss: 1.91 | D_acc: 5.03% | G_loss: 0.20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 15/50 | D_loss: 2.02 | D_acc: 4.77% | G_loss: 0.18
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 16/50 | D_loss: 2.12 | D_acc: 4.54% | G_loss: 0.17
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 17/50 | D_loss: 2.22 | D_acc: 4.32% | G_loss: 0.16
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 18/50 | D_loss: 2.32 | D_acc: 4.14% | G_loss: 0.15
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 19/50 | D_loss: 2.42 | D_acc: 4.00% | G_loss: 0.14
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 20/50 | D_loss: 2.51 | D_acc: 4.02% | G_loss: 0.14
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 21/50 | D_loss: 2.60 | D_acc: 3.86% | G_loss: 0.13
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 22/50 | D_loss: 2.69 | D_acc: 3.74% | G_loss: 0.12
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 23/50 | D_loss: 2.77 | D_acc: 3.61% | G_loss: 0.12
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 24/50 | D_loss: 2.85 | D_acc: 3.47% | G_loss: 0.11
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 25/50 | D_loss: 2.93 | D_acc: 3.46% | G_loss: 0.11
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
```

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

noise = np.random.normal(0, 1, (100, noise_dim))
gen_imgs = generator.predict(noise)
gen_imgs = 0.5 * gen_imgs + 0.5

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

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 162ms/step
```

```python
classifier = tf.keras.Sequential([
    layers.Conv2D(32, 3, activation='relu', input_shape=img_shape),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

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

labels = np.random.randint(0, 10, x_train.shape[0])  # Dummy labels for training
classifier.fit(x_train, labels, epochs=3, batch_size=128)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shap
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.1003 - loss: 2.3051
Epoch 2/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.0992 - loss: 2.3026
Epoch 3/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1007 - loss: 2.3026
<keras.src.callbacks.history.History at 0x7f32e02b0cb0>
```

```
preds = classifier.predict(gen_imgs)
predicted_labels = np.argmax(preds, axis=1)

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

print("Label Distribution of Generated Images:")
print(label_distribution)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
Label Distribution of Generated Images:
{np.int64(0): np.int64(100)}
```

```
import os

base_path = "/content/drive/MyDrive/GEN AI/LAB 2"

os.makedirs(base_path + "/models", exist_ok=True)
os.makedirs(base_path + "/generated_samples", exist_ok=True)
os.makedirs(base_path + "/final_generated_images", exist_ok=True)

print("Folder structure created successfully!")
```

```
Folder structure created successfully!
```

```
generator.save(base_path + "/models/generator_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
```

```
discriminator.save(base_path + "/models/discriminator_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
```

```
gan.save(base_path + "/models/gan_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
```

```
def save_images(epoch):
    noise = np.random.normal(0, 1, (25, noise_dim))
    gen_imgs = generator.predict(noise)
    gen_imgs = 0.5 * gen_imgs + 0.5

    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_imgs[cnt,:,:,0], cmap='gray')
            axs[i,j].axis('off')
            cnt += 1
    plt.savefig(f"{base_path}/generated_samples/epoch_{epoch:02d}.png")
    plt.close()
```

```
noise = np.random.normal(0, 1, (100, noise_dim))
gen_imgs = generator.predict(noise)
gen_imgs = 0.5 * gen_imgs + 0.5

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

print("100 final images saved to Google Drive!")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
100 final images saved to Google Drive!
```

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