```
# 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 = 'fashion'      # '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/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 1s 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`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
```

```
  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`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```python
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 trainable weights.
  warnings.warn("The model does not have any trainable weights.")

Epoch 1/50 | D_loss: 0.64 | D_acc: 56.45% | G_loss: 0.66
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 2/50 | D_loss: 0.70 | D_acc: 46.03% | G_loss: 0.55
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 3/50 | D_loss: 0.78 | D_acc: 41.97% | G_loss: 0.46
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 4/50 | D_loss: 0.87 | D_acc: 38.82% | G_loss: 0.39
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 5/50 | D_loss: 0.98 | D_acc: 37.69% | G_loss: 0.33
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
Epoch 6/50 | D_loss: 1.09 | D_acc: 37.37% | G_loss: 0.29
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 7/50 | D_loss: 1.21 | D_acc: 36.68% | G_loss: 0.25
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 8/50 | D_loss: 1.34 | D_acc: 36.03% | G_loss: 0.22
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 9/50 | D_loss: 1.47 | D_acc: 35.30% | G_loss: 0.20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 10/50 | D_loss: 1.60 | D_acc: 35.20% | G_loss: 0.18
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 11/50 | D_loss: 1.73 | D_acc: 35.08% | G_loss: 0.16
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 12/50 | D_loss: 1.86 | D_acc: 34.89% | G_loss: 0.15
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 13/50 | D_loss: 1.99 | D_acc: 34.73% | G_loss: 0.14
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 14/50 | D_loss: 2.12 | D_acc: 34.41% | G_loss: 0.13
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 15/50 | D_loss: 2.24 | D_acc: 34.04% | G_loss: 0.12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 16/50 | D_loss: 2.36 | D_acc: 33.76% | G_loss: 0.11
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 17/50 | D_loss: 2.47 | D_acc: 33.66% | G_loss: 0.11
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 18/50 | D_loss: 2.58 | D_acc: 33.59% | G_loss: 0.10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 19/50 | D_loss: 2.68 | D_acc: 33.59% | G_loss: 0.09
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 20/50 | D_loss: 2.77 | D_acc: 33.65% | G_loss: 0.09
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 21/50 | D_loss: 2.87 | D_acc: 33.46% | G_loss: 0.09
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 22/50 | D_loss: 2.96 | D_acc: 33.27% | G_loss: 0.08
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 23/50 | D_loss: 3.05 | D_acc: 33.16% | G_loss: 0.08
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 24/50 | D_loss: 3.13 | D_acc: 33.10% | G_loss: 0.08
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 25/50 | D_loss: 3.21 | D_acc: 33.02% | G_loss: 0.07
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
Epoch 26/50 | D_loss: 3.29 | D_acc: 32.89% | G_loss: 0.07
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 27/50 | D_loss: 3.36 | D_acc: 33.03% | G_loss: 0.07
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 28/50 | D_loss: 3.43 | D_acc: 32.83% | G_loss: 0.06
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 29/50 | D_loss: 3.50 | D_acc: 32.81% | G_loss: 0.06
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 30/50 | D_loss: 3.56 | D_acc: 32.80% | G_loss: 0.06
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 31/50 | D_loss: 3.62 | D_acc: 32.61% | G_loss: 0.06
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 32/50 | D_loss: 3.68 | D_acc: 32.53% | G_loss: 0.06
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 33/50 | D_loss: 3.74 | D_acc: 32.58% | G_loss: 0.05
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 34/50 | D_loss: 3.79 | D_acc: 32.52% | G_loss: 0.05
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 35/50 | D_loss: 3.85 | D_acc: 32.43% | G_loss: 0.05
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 36/50 | D_loss: 3.90 | D_acc: 32.41% | G_loss: 0.05
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 37/50 | D_loss: 3.94 | D_acc: 32.37% | G_loss: 0.05
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 38/50 | D_loss: 3.99 | D_acc: 32.29% | G_loss: 0.05
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 39/50 | D_loss: 4.03 | D_acc: 32.27% | G_loss: 0.05
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 40/50 | D_loss: 4.08 | D_acc: 32.22% | G_loss: 0.05
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 41/50 | D_loss: 4.12 | D_acc: 32.27% | G_loss: 0.04
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Epoch 42/50 | D_loss: 4.16 | D_acc: 32.20% | G_loss: 0.04
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 43/50 | D_loss: 4.19 | D_acc: 32.29% | G_loss: 0.04
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```

```
Epoch 44/50 | D_loss: 4.23 | D_acc: 32.30% | G_loss: 0.04
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
Epoch 45/50 | D_loss: 4.27 | D_acc: 32.22% | G_loss: 0.04
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 46/50 | D_loss: 4.30 | D_acc: 32.20% | G_loss: 0.04
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 47/50 | D_loss: 4.33 | D_acc: 32.23% | G_loss: 0.04
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 48/50 | D_loss: 4.36 | D_acc: 32.24% | G_loss: 0.04
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Epoch 49/50 | D_loss: 4.39 | D_acc: 32.29% | G_loss: 0.04
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 50/50 | D_loss: 4.42 | D_acc: 32.24% | G_loss: 0.04
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

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

```
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_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Epoch 1/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.1011 - loss:
2.3050
Epoch 2/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.1001 - loss:
2.3028
Epoch 3/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1032 - loss:
2.3025

<keras.src.callbacks.history.History at 0x7e147669e9f0>

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 ━━━━━━━━━━━━━━━━━━━━ 1s 174ms/step
Label Distribution of Generated Images:
{np.int64(7): np.int64(100)}

import os

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

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
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
```

```
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
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
```

```
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
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
```

```
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!
```