```
# ================================
# CSET419 – Introduction to Generative AI
# Lab 2: Basic GAN for Image Generation
# Single-Cell Complete Implementation
# ================================

# ---------- IMPORT LIBRARIES ----------
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, load_model
from tensorflow.keras.layers import Dense, Flatten, Reshape, LeakyReLU
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy
import tensorflow as tf

# ---------- USER INPUT PARAMETERS ----------
dataset_choice = 'mnist'        # 'mnist' or 'fashion'
epochs = 30                     # recommended 30–100
batch_size = 128                # recommended 64 or 128
noise_dim = 100                 # recommended 50 or 100
learning_rate = 0.0002
save_interval = 5               # save images every k epochs

# ---------- CREATE OUTPUT DIRECTORIES ----------
os.makedirs("generated_samples", exist_ok=True)
os.makedirs("final_generated_images", exist_ok=True)

# ---------- LOAD DATASET ----------
if dataset_choice == 'mnist':
    (X_train, y_train), _ = mnist.load_data()
else:
    (X_train, y_train), _ = fashion_mnist.load_data()

# Normalize images to [0,1]
X_train = X_train.astype("float32") / 255.0
X_train = np.expand_dims(X_train, axis=-1)

img_shape = X_train.shape[1:]  # (28,28,1)

# ---------- GENERATOR MODEL ----------
generator = Sequential([
    Dense(256, input_dim=noise_dim),
    LeakyReLU(0.2),
    Dense(512),
    LeakyReLU(0.2),
    Dense(1024),
    LeakyReLU(0.2),
    Dense(np.prod(img_shape), activation='sigmoid'),
    Reshape(img_shape)
])

# ---------- DISCRIMINATOR MODEL ----------
discriminator = Sequential([
    Flatten(input_shape=img_shape),
    Dense(512),
    LeakyReLU(0.2),
    Dense(256),
    LeakyReLU(0.2),
    Dense(1, activation='sigmoid')
])

discriminator.compile(
    optimizer=Adam(learning_rate),
    loss=BinaryCrossentropy(),
    metrics=['accuracy']
)

# ---------- GAN MODEL ----------
discriminator.trainable = False

gan = Sequential([generator, discriminator])
gan.compile(
    optimizer=Adam(learning_rate),
    loss=BinaryCrossentropy()
)

# ---------- IMAGE SAVING FUNCTION ----------
def save_generated_images(epoch):
    noise = np.random.normal(0, 1, (25, noise_dim))
    generated_images = generator.predict(noise, verbose=0)

    plt.figure(figsize=(5,5))
    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.imshow(generated_images[i].reshape(28,28), cmap='gray')
        plt.axis('off')

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

# ---------- TRAINING LOOP ----------
```

```
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_images = X_train[idx]

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

    real_labels = np.ones((half_batch, 1))
    fake_labels = np.zeros((half_batch, 1))

    d_loss_real = discriminator.train_on_batch(real_images, real_labels)
    d_loss_fake = discriminator.train_on_batch(fake_images, 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))
    valid_labels = np.ones((batch_size, 1))

    g_loss = gan.train_on_batch(noise, valid_labels)

    # ----- PRINT REQUIRED LOG FORMAT -----
    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}"
    )

    # ----- SAVE IMAGES PERIODICALLY -----
    if epoch % save_interval == 0:
        save_generated_images(epoch)

# ---------- FINAL IMAGE GENERATION (100 IMAGES) ----------
noise = np.random.normal(0, 1, (100, noise_dim))
final_images = generator.predict(noise, verbose=0)

for i in range(100):
    plt.imsave(
        f"final_generated_images/img_{i+1}.png",
        final_images[i].reshape(28,28),
        cmap='gray'
    )

# ---------- PRE-TRAINED CLASSIFIER (TRANSFER LEARNING) ----------
classifier = Sequential([
    Flatten(input_shape=img_shape),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

classifier.fit(X_train, y_train, epochs=3, batch_size=256, verbose=0)

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

# ---------- LABEL DISTRIBUTION OUTPUT ----------
print("\nLabel Distribution of Generated Images:")
unique, counts = np.unique(predicted_labels, return_counts=True)
for label, count in zip(unique, counts):
    print(f"Label {label}: {count} images")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/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
  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
  super().__init__(**kwargs)
/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 1/30 | D_loss: 0.71 | D_acc: 66.80% | G_loss: 0.42
Epoch 2/30 | D_loss: 0.79 | D_acc: 52.41% | G_loss: 0.39
Epoch 3/30 | D_loss: 0.84 | D_acc: 48.41% | G_loss: 0.37
Epoch 4/30 | D_loss: 0.87 | D_acc: 47.08% | G_loss: 0.35
Epoch 5/30 | D_loss: 0.90 | D_acc: 47.17% | G_loss: 0.33
Epoch 6/30 | D_loss: 0.93 | D_acc: 47.37% | G_loss: 0.31
Epoch 7/30 | D_loss: 0.96 | D_acc: 47.40% | G_loss: 0.29
Epoch 8/30 | D_loss: 0.99 | D_acc: 47.43% | G_loss: 0.28
Epoch 9/30 | D_loss: 1.02 | D_acc: 47.27% | G_loss: 0.26
Epoch 10/30 | D_loss: 1.06 | D_acc: 47.07% | G_loss: 0.24
Epoch 11/30 | D_loss: 1.10 | D_acc: 46.90% | G_loss: 0.23
Epoch 12/30 | D_loss: 1.14 | D_acc: 46.43% | G_loss: 0.22
Epoch 13/30 | D_loss: 1.19 | D_acc: 46.03% | G_loss: 0.20
Epoch 14/30 | D_loss: 1.24 | D_acc: 46.04% | G_loss: 0.19
Epoch 15/30 | D_loss: 1.29 | D_acc: 45.88% | G_loss: 0.18
Epoch 16/30 | D_loss: 1.34 | D_acc: 45.80% | G_loss: 0.17
Epoch 17/30 | D_loss: 1.39 | D_acc: 45.63% | G_loss: 0.16
```

```
Epoch 18/30 | D_loss: 1.44 | D_acc: 45.70% | G_loss: 0.15
Epoch 19/30 | D_loss: 1.50 | D_acc: 45.63% | G_loss: 0.15
Epoch 20/30 | D_loss: 1.55 | D_acc: 45.54% | G_loss: 0.14
Epoch 21/30 | D_loss: 1.60 | D_acc: 45.56% | G_loss: 0.13
Epoch 22/30 | D_loss: 1.66 | D_acc: 45.44% | G_loss: 0.13
Epoch 23/30 | D_loss: 1.71 | D_acc: 45.40% | G_loss: 0.12
Epoch 24/30 | D_loss: 1.77 | D_acc: 45.40% | G_loss: 0.12
Epoch 25/30 | D_loss: 1.82 | D_acc: 45.21% | G_loss: 0.11
Epoch 26/30 | D_loss: 1.88 | D_acc: 45.27% | G_loss: 0.11
Epoch 27/30 | D_loss: 1.93 | D_acc: 45.30% | G_loss: 0.11
Epoch 28/30 | D_loss: 1.98 | D_acc: 45.36% | G_loss: 0.10
Epoch 29/30 | D_loss: 2.03 | D_acc: 45.25% | G_loss: 0.10
Epoch 30/30 | D_loss: 2.08 | D_acc: 45.30% | G_loss: 0.09
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step

Label Distribution of Generated Images:
Label 2: 88 images
Label 5: 12 images
```

```
import shutil

shutil.make_archive("gan_outputs", 'zip', ".")
```

```
'/content/gan_outputs.zip'
```