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

```
BUFFER_SIZE = 60000
BATCH_SIZE = 256
EPOCHS = 100
NOISE_DIM = 100
NUM_EXAMPLES_TO_GENERATE = 16
```

```
BUFFER_SIZE = 60000
BATCH_SIZE = 256

(train_images, _), (_, _) = tf.keras.datasets.mnist.load_data()

train_images = train_images.reshape(-1, 28, 28, 1)
train_images = train_images.astype("float32")
train_images = (train_images - 127.5) / 127.5

train_dataset = tf.data.Dataset.from_tensor_slices(train_images) \
    .shuffle(BUFFER_SIZE) \
    .batch(BATCH_SIZE)
```

```
def make_generator_model():
    model = tf.keras.Sequential([
        layers.Dense(7 * 7 * 256, use_bias=False, input_shape=(NOISE_
        layers.BatchNormalization(),
        layers.LeakyReLU(),

        layers.Reshape((7, 7, 256)),

        layers.Conv2DTranspose(128, (5, 5), strides=(1, 1),
                               padding='same', use_bias=False),
        layers.BatchNormalization(),
        layers.LeakyReLU(),

        layers.Conv2DTranspose(64, (5, 5), strides=(2, 2),
                               padding='same', use_bias=False),
        layers.BatchNormalization(),
        layers.LeakyReLU(),

        layers.Conv2DTranspose(1, (5, 5), strides=(2, 2),
                               padding='same', use_bias=False, activa
    ])
```

```
    return model
```

```
generator = make_generator_model()
```

```
def make_discriminator_model():
    model = tf.keras.Sequential([
        layers.Conv2D(64, (5, 5), strides=(2, 2),
                      padding='same', input_shape=[28, 28, 1]),
        layers.LeakyReLU(),
        layers.Dropout(0.3),

        layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'),
        layers.LeakyReLU(),
        layers.Dropout(0.3),

        layers.Flatten(),
        layers.Dense(1)
    ])
    return model
```

```
discriminator = make_discriminator_model()
```

```
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output
    return real_loss + fake_loss
```

```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

```
def discriminator_accuracy(real_output, fake_output):
    real_pred = tf.cast(real_output > 0, tf.float32)
    fake_pred = tf.cast(fake_output < 0, tf.float32)

    real_acc = tf.reduce_mean(real_pred)
    fake_acc = tf.reduce_mean(fake_pred)

    return (real_acc + fake_acc) / 2.0
```

```
def generator_accuracy(fake_output):
    fake_pred = tf.cast(fake_output > 0, tf.float32)
    return tf.reduce_mean(fake_pred)
```

```
@tf.function
def train_step(images):
    noise = tf.random.normal([tf.shape(images)[0], 100])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tap
        generated_images = generator(noise, training=True)

        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

    generator_optimizer.apply_gradients(
        zip(gen_tape.gradient(gen_loss, generator.trainable_variables
            generator.trainable_variables))

    discriminator_optimizer.apply_gradients(
        zip(disc_tape.gradient(disc_loss, discriminator.trainable_var
            discriminator.trainable_variables))

    d_acc = discriminator_accuracy(real_output, fake_output)
    g_acc = generator_accuracy(fake_output)

    return d_acc, g_acc
```

```
seed = tf.random.normal([NUM_EXAMPLES_TO_GENERATE, NOISE_DIM])

def save_images(epoch):
    predictions = generator(seed, training=False)

    fig = plt.figure(figsize=(4, 4))
    for i in range(predictions.shape[0]):
        plt.subplot(4, 4, i + 1)
        plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gra
        plt.axis('off')

    os.makedirs("gan_images", exist_ok=True)
    plt.savefig(f"gan_images/image_epoch_{epoch:03d}.png")
    plt.close()
```

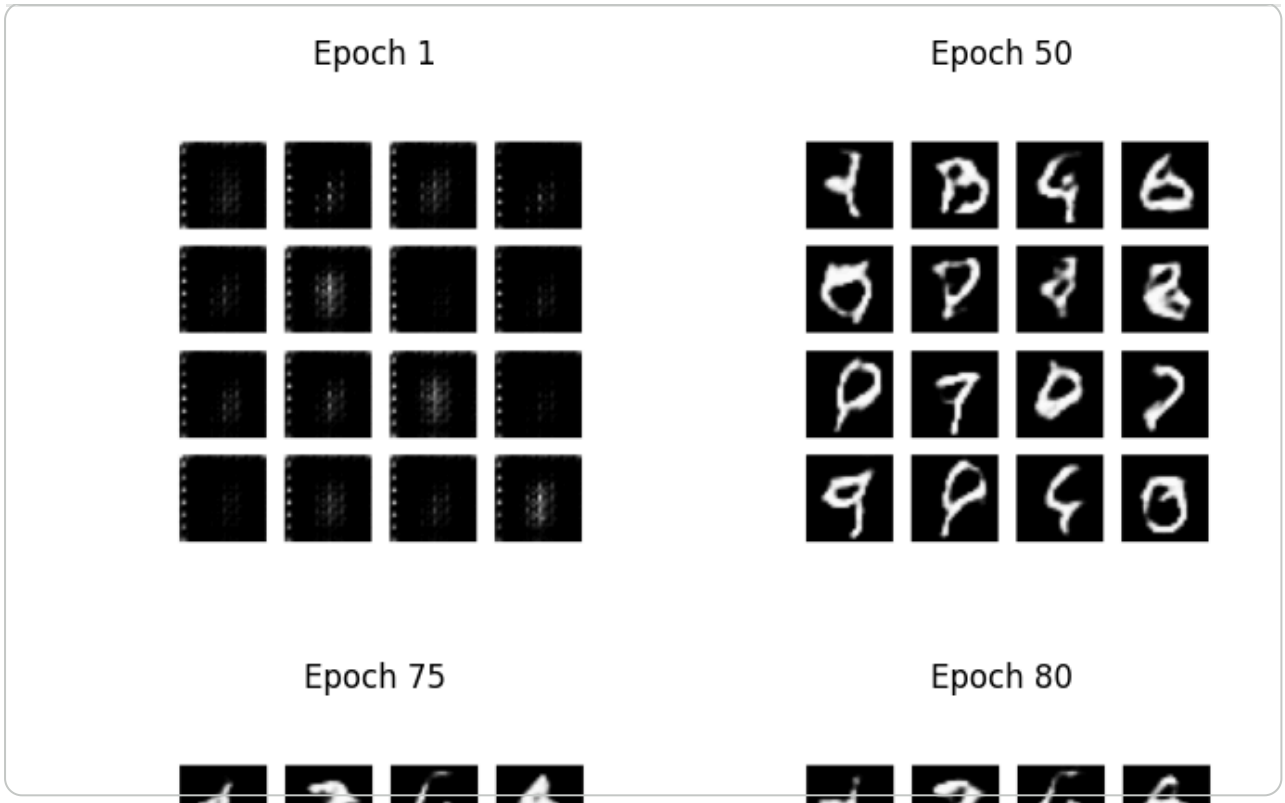
```
for epoch in range(1, EPOCHS + 1):

    d_acc_list = []
    g_acc_list = []
```

```
    for image_batch in train_dataset:
        d_acc, g_acc = train_step(image_batch)
        d_acc_list.append(d_acc)
        g_acc_list.append(g_acc)

    avg_d_acc = tf.reduce_mean(d_acc_list)
    avg_g_acc = tf.reduce_mean(g_acc_list)

    save_images(epoch)

    print(
        f"Epoch {epoch}/{EPOCHS} | "
        f"D Accuracy: {avg_d_acc:.4f} | "
        f"G Accuracy: {avg_g_acc:.4f}"
    )
```

```
Epoch 1/100 | D Accuracy: 0.7456 | G Accuracy: 0.4568
Epoch 2/100 | D Accuracy: 0.7006 | G Accuracy: 0.2937
Epoch 3/100 | D Accuracy: 0.7976 | G Accuracy: 0.1868
Epoch 4/100 | D Accuracy: 0.6845 | G Accuracy: 0.2930
Epoch 5/100 | D Accuracy: 0.6937 | G Accuracy: 0.2877
Epoch 6/100 | D Accuracy: 0.6469 | G Accuracy: 0.3273
Epoch 7/100 | D Accuracy: 0.6624 | G Accuracy: 0.3234
Epoch 8/100 | D Accuracy: 0.6384 | G Accuracy: 0.3488
Epoch 9/100 | D Accuracy: 0.6878 | G Accuracy: 0.2829
Epoch 10/100 | D Accuracy: 0.6773 | G Accuracy: 0.3125
Epoch 11/100 | D Accuracy: 0.6795 | G Accuracy: 0.2979
Epoch 12/100 | D Accuracy: 0.7236 | G Accuracy: 0.2530
Epoch 13/100 | D Accuracy: 0.6902 | G Accuracy: 0.2929
Epoch 14/100 | D Accuracy: 0.7120 | G Accuracy: 0.2734
Epoch 15/100 | D Accuracy: 0.7337 | G Accuracy: 0.2471
Epoch 16/100 | D Accuracy: 0.7069 | G Accuracy: 0.2802
Epoch 17/100 | D Accuracy: 0.7219 | G Accuracy: 0.2622
Epoch 18/100 | D Accuracy: 0.7346 | G Accuracy: 0.2456
Epoch 19/100 | D Accuracy: 0.7278 | G Accuracy: 0.2508
Epoch 20/100 | D Accuracy: 0.7487 | G Accuracy: 0.2301
Epoch 21/100 | D Accuracy: 0.7550 | G Accuracy: 0.2265
Epoch 22/100 | D Accuracy: 0.7809 | G Accuracy: 0.1999
Epoch 23/100 | D Accuracy: 0.7603 | G Accuracy: 0.2180
Epoch 24/100 | D Accuracy: 0.7420 | G Accuracy: 0.2408
Epoch 25/100 | D Accuracy: 0.7539 | G Accuracy: 0.2240
Epoch 26/100 | D Accuracy: 0.7647 | G Accuracy: 0.2133
Epoch 27/100 | D Accuracy: 0.7653 | G Accuracy: 0.2146
Epoch 28/100 | D Accuracy: 0.7322 | G Accuracy: 0.2502
Epoch 29/100 | D Accuracy: 0.7191 | G Accuracy: 0.2646
Epoch 30/100 | D Accuracy: 0.7283 | G Accuracy: 0.2558
Epoch 31/100 | D Accuracy: 0.7149 | G Accuracy: 0.2690
Epoch 32/100 | D Accuracy: 0.6985 | G Accuracy: 0.2839
Epoch 33/100 | D Accuracy: 0.7099 | G Accuracy: 0.2770
Epoch 34/100 | D Accuracy: 0.6947 | G Accuracy: 0.2917
Epoch 35/100 | D Accuracy: 0.6792 | G Accuracy: 0.3086
Epoch 36/100 | D Accuracy: 0.6805 | G Accuracy: 0.3073
Epoch 37/100 | D Accuracy: 0.6855 | G Accuracy: 0.2995
Epoch 38/100 | D Accuracy: 0.6767 | G Accuracy: 0.3129
Epoch 39/100 | D Accuracy: 0.6743 | G Accuracy: 0.3140
Epoch 40/100 | D Accuracy: 0.6673 | G Accuracy: 0.3213
Epoch 41/100 | D Accuracy: 0.6725 | G Accuracy: 0.3178
```

```
Epoch 42/100 | D Accuracy: 0.7000 | G Accuracy: 0.2778
Epoch 43/100 | D Accuracy: 0.6682 | G Accuracy: 0.3263
Epoch 44/100 | D Accuracy: 0.6616 | G Accuracy: 0.3270
Epoch 45/100 | D Accuracy: 0.6522 | G Accuracy: 0.3392
Epoch 46/100 | D Accuracy: 0.6663 | G Accuracy: 0.3196
Epoch 47/100 | D Accuracy: 0.6763 | G Accuracy: 0.3147
Epoch 48/100 | D Accuracy: 0.7013 | G Accuracy: 0.2789
Epoch 49/100 | D Accuracy: 0.6735 | G Accuracy: 0.3177
Epoch 50/100 | D Accuracy: 0.6587 | G Accuracy: 0.3316
Epoch 51/100 | D Accuracy: 0.6517 | G Accuracy: 0.3404
Epoch 52/100 | D Accuracy: 0.6602 | G Accuracy: 0.3307
Epoch 53/100 | D Accuracy: 0.6582 | G Accuracy: 0.3354
Epoch 54/100 | D Accuracy: 0.6692 | G Accuracy: 0.3208
Epoch 55/100 | D Accuracy: 0.6918 | G Accuracy: 0.2880
Epoch 56/100 | D Accuracy: 0.6828 | G Accuracy: 0.2984
Epoch 57/100 | D Accuracy: 0.6608 | G Accuracy: 0.3312
```

```
import matplotlib.pyplot as plt
import os

epochs_to_show = [1, 50, 75, 80]
image_folder = "gan_images"

plt.figure(figsize=(8, 8))

for i, epoch in enumerate(epochs_to_show):
    img_path = os.path.join(
        image_folder, f"image_epoch_{epoch:03d}.png"
    )

    if os.path.exists(img_path):
        img = plt.imread(img_path)
        plt.subplot(2, 2, i + 1)
        plt.imshow(img)
        plt.title(f"Epoch {epoch}")
        plt.axis("off")
    else:
        print(f"Image not found for Epoch {epoch} -> {img_path}")

plt.show()
```



Start coding or generate with AI.