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

def load_and_preprocess_mnist(batch_size=128):
    (x_train, _), (_, _) = keras.datasets.mnist.load_data()
    x_train = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32')
    x_train = (x_train - 127.5) / 127.5  # Normalize the images to [-1, 1]

    # Use tf.data for efficient data loading and batching
    dataset = tf.data.Dataset.from_tensor_slices(x_train)
    dataset = dataset.shuffle(buffer_size=1024).batch(batch_size, drop_remainder=True)  # drop_remainder to ensure all batches are the same s
    return dataset

def build_generator():
    model = keras.Sequential([
        layers.Dense(256, input_dim=100, activation='relu'),
        layers.BatchNormalization(),
        layers.Dense(512, activation='relu'),
        layers.BatchNormalization(),
        layers.Dense(1024, activation='relu'),
        layers.BatchNormalization(),
        layers.Dense(784, activation='tanh'),
        layers.Reshape((28, 28, 1))
    ])
    return model

def build_discriminator():
    model = keras.Sequential([
        layers.Flatten(input_shape=(28, 28, 1)),
        layers.Dense(512, activation='relu'),
        layers.Dense(256, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
    return model

def build_gan(generator, discriminator):
    discriminator.trainable = False
    model = keras.Sequential([generator, discriminator])
    return model

def train_gan(epochs=10000, batch_size=128, sample_interval=1000):
    dataset = load_and_preprocess_mnist(batch_size)  # Use tf.data for batching and memory efficiency

    generator = build_generator()
    discriminator = build_discriminator()

    discriminator.compile(optimizer='adam', loss='binary_crossentropy')

    gan = build_gan(generator, discriminator)
    gan.compile(optimizer='adam', loss='binary_crossentropy')

    for epoch in range(epochs):
        for real_imgs in dataset:
            current_batch_size = real_imgs.shape[0]  # get the current batch size
            noise = np.random.normal(0, 1, (current_batch_size, 100))  # adjust noise to current batch size
            generated_imgs = generator.predict(noise)

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

            d_loss_real = discriminator.train_on_batch(real_imgs, real_labels)
            d_loss_fake = discriminator.train_on_batch(generated_imgs, fake_labels)

            noise = np.random.normal(0, 1, (current_batch_size, 100))  # adjust noise to current batch size
            g_loss = gan.train_on_batch(noise, real_labels)

            # Print the loss for every sample interval
            if epoch % sample_interval == 0:
                print(f"Epoch {epoch}, D Loss: {0.5*(d_loss_real + d_loss_fake)}, G Loss: {g_loss}")

                # Save or display a limited number of images
                if epoch % sample_interval == 0:
                    noise = np.random.normal(0, 1, (25, 100))
```

```
                    generated_imgs = generator.predict(noise)

                    plt.figure(figsize=(5, 5))
                    for i in range(5):  # Reduced to only 5 images for lower memory usage
                        plt.subplot(1, 5, i+1)
                        plt.imshow(generated_imgs[i].reshape(28, 28), cmap='gray')
                        plt.axis('off')
                    plt.tight_layout()
                    plt.show()

    return generator

# Start training with reduced memory usage
generator = train_gan()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
Epoch 0, D Loss: 0.6962174773216248, G Loss: [array(0.68648314, dtype=float32), array(0.68648314, dtype=float32)]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 299ms/step
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Epoch 0, D Loss: 0.6769695281982422, G Loss: [array(0.6809544, dtype=float32), array(0.6809544, dtype=float32)]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
Epoch 0, D Loss: 0.6784374117851257, G Loss: [array(0.6843593, dtype=float32), array(0.6843593, dtype=float32)]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
Epoch 0, D Loss: 0.6881504654884338, G Loss: [array(0.6945397, dtype=float32), array(0.6945397, dtype=float32)]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
```

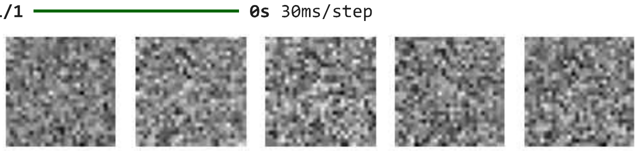
```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 0, D Loss: 0.6980311870574951, G Loss: [array(0.70502317, dtype=float32), array(0.70502317, dtype=float32)]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
Epoch 0, D Loss: 0.7083938121795654, G Loss: [array(0.71576977, dtype=float32), array(0.71576977, dtype=float32)]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
Epoch 0, D Loss: 0.7175439596176147, G Loss: [array(0.7251874, dtype=float32), array(0.7251874, dtype=float32)]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
Epoch 0, D Loss: 0.7280389070510864, G Loss: [array(0.7362927, dtype=float32), array(0.7362927, dtype=float32)]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Epoch 0, D Loss: 0.7394824028015137, G Loss: [array(0.7479398, dtype=float32), array(0.7479398, dtype=float32)]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```



```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
Epoch 0, D Loss: 0.7513222694396973, G Loss: [array(0.759879, dtype=float32), array(0.759879, dtype=float32)]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
```