```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, LeakyReLU, BatchNormalization, Reshape, Flatten, Input
from tensorflow.keras.optimizers import Adam



(X_train, _), (_, _) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
X_train = X_train / 127.5 - 1.0
X_train = np.expand_dims(X_train, axis=3)


def build_generator(latent_dim):
    model = Sequential()
    model.add(Dense(128 * 7 * 7, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((7, 7, 128)))
    model.add(BatchNormalization())
    model.add(Flatten())
    model.add(Dense(28 * 28, activation='tanh'))
    model.add(Reshape((28, 28, 1)))
    noise = Input(shape=(latent_dim,))
    img = model(noise)
    return Model(noise, img)


def build_discriminator(img_shape):
    model = Sequential()
    model.add(Flatten(input_shape=img_shape))
    model.add(Dense(128))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(1, activation='sigmoid'))
    img = Input(shape=img_shape)
    validity = model(img)
    return Model(img, validity)


img_shape = X_train[0].shape
discriminator = build_discriminator(img_shape)
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002, beta_1=0.5), metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.leg
```

```
latent_dim = 100
generator = build_generator(latent_dim)



z = Input(shape=(latent_dim,))
img = generator(z)
discriminator.trainable = False
validity = discriminator(img)
combined = Model(z, validity)
combined.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002, beta_1=0.5))
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.leg
```

```
epochs = 10000
batch_size = 128
sample_interval = 1000




for epoch in range(epochs):
    # Train discriminator
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    real_imgs = X_train[idx]
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
```

```
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    fake_imgs = generator.predict(noise)
    d_loss_real = discriminator.train_on_batch(real_imgs, np.ones((batch_size, 1)))
    d_loss_fake = discriminator.train_on_batch(fake_imgs, np.zeros((batch_size, 1)))
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
```

```
Streaming output truncated to the last 5000 lines.
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 24ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 26ms/step
4/4 [==============================] - 0s 27ms/step
4/4 [==============================] - 0s 33ms/step
4/4 [==============================] - 0s 22ms/step
4/4 [==============================] - 0s 22ms/step
4/4 [==============================] - 0s 30ms/step
4/4 [==============================] - 0s 33ms/step
4/4 [==============================] - 0s 29ms/step
4/4 [==============================] - 0s 22ms/step
4/4 [==============================] - 0s 30ms/step
4/4 [==============================] - 0s 23ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 22ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 21ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 22ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 21ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 22ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 21ms/step
4/4 [==============================] - 0s 18ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 21ms/step
4/4 [==============================] - 0s 21ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 21ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 26ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 19ms/step
```

```
noise = np.random.normal(0, 1, (batch_size, latent_dim))
g_loss = combined.train_on_batch(noise, np.ones((batch_size, 1)))


 if epoch % sample_interval == 0:
        print(f"{epoch} [D loss: {d_loss[0]}, acc.: {100 * d_loss[1]}] [G loss: {g_loss}]")
```