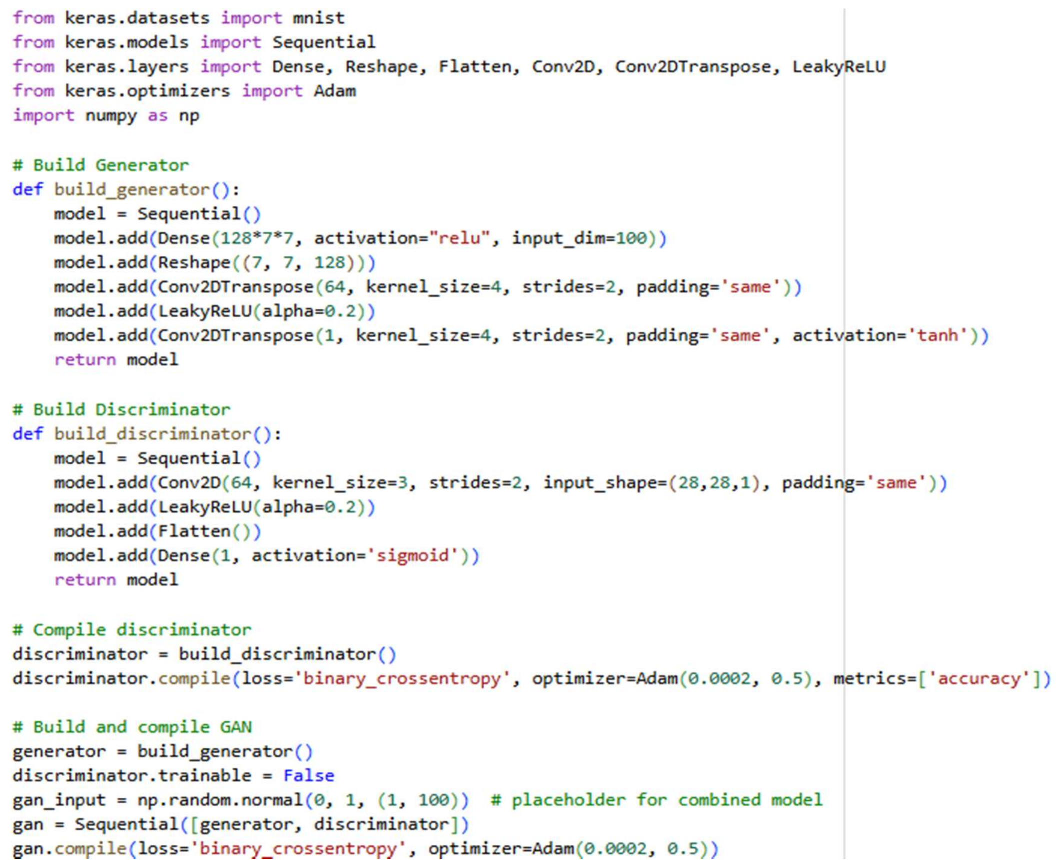
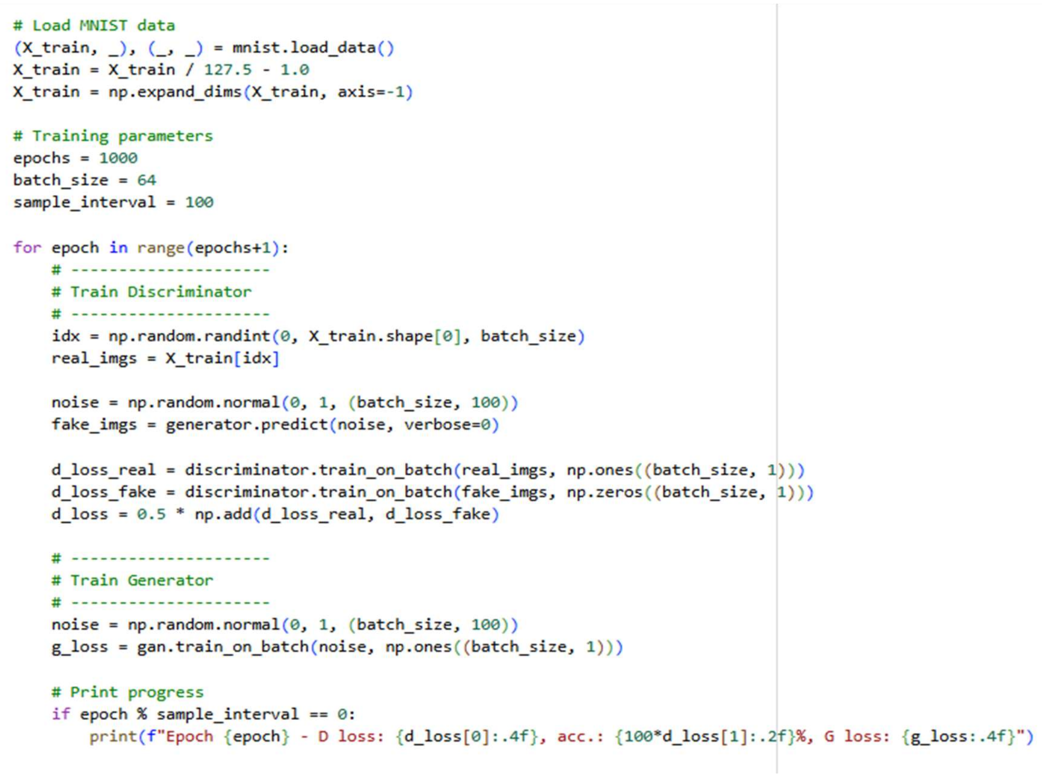
EXP NO: 8

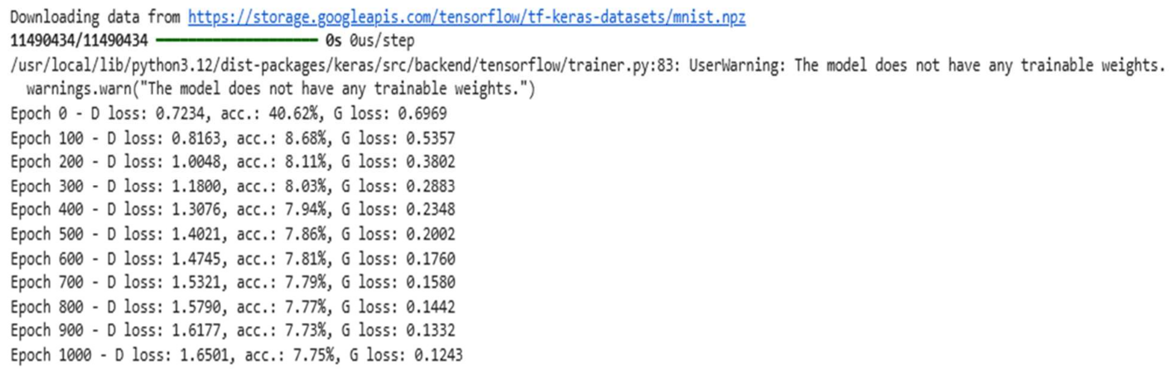


```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Reshape, Flatten, Conv2D, Conv2DTranspose, LeakyReLU
from keras.optimizers import Adam
import numpy as np

# Build Generator
def build_generator():
    model = Sequential()
    model.add(Dense(128*7*7, activation="relu", input_dim=100))
    model.add(Reshape((7, 7, 128)))
    model.add(Conv2DTranspose(64, kernel_size=4, strides=2, padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Conv2DTranspose(1, kernel_size=4, strides=2, padding='same', activation='tanh'))
    return model

# Build Discriminator
def build_discriminator():
    model = Sequential()
    model.add(Conv2D(64, kernel_size=3, strides=2, input_shape=(28,28,1), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model

# Compile discriminator
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5), metrics=['accuracy'])

# Build and compile GAN
generator = build_generator()
discriminator.trainable = False
gan_input = np.random.normal(0, 1, (1, 100))  # placeholder for combined model
gan = Sequential([generator, discriminator])
gan.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
```



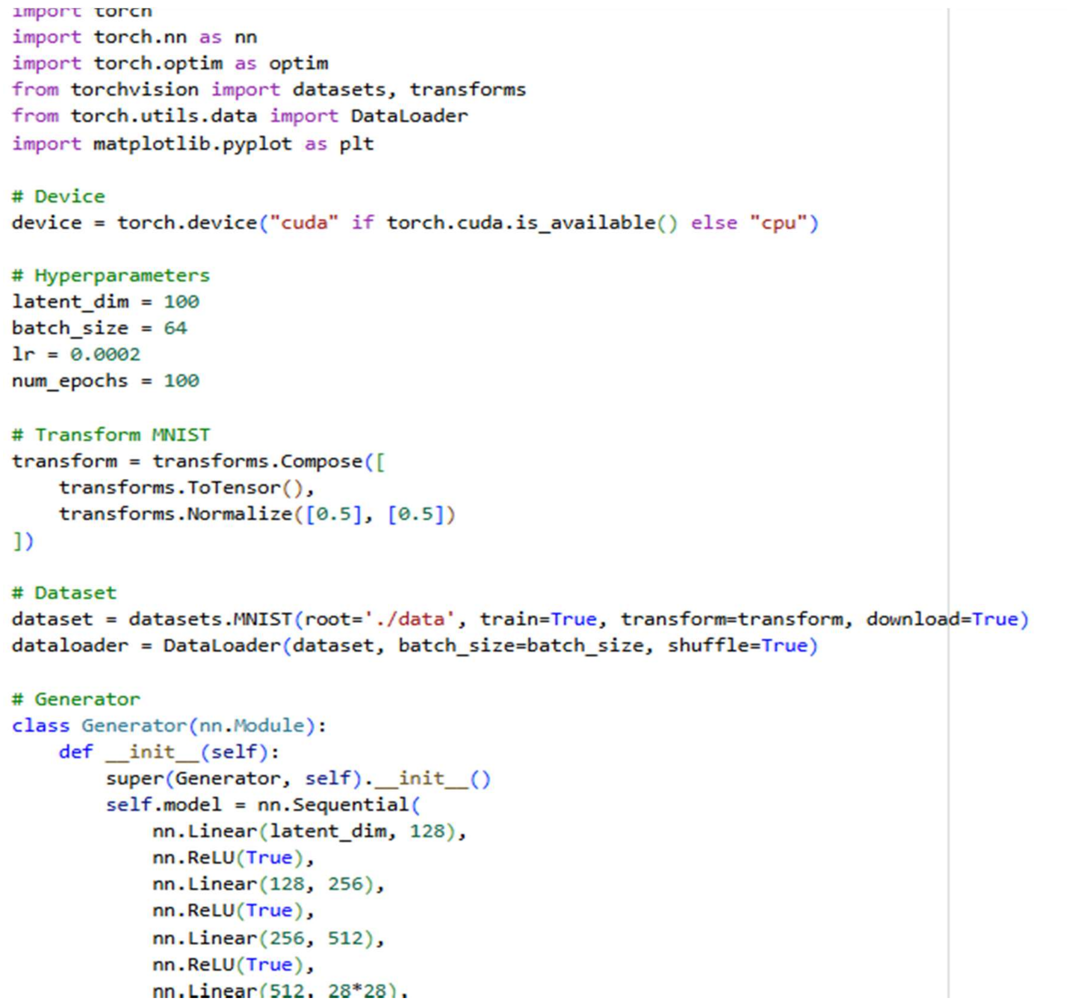
```
# Load MNIST data
(X_train, _), (_, _) = mnist.load_data()
X_train = X_train / 127.5 - 1.0
X_train = np.expand_dims(X_train, axis=-1)

# Training parameters
epochs = 1000
batch_size = 64
sample_interval = 100

for epoch in range(epochs+1):
    # ---------------------
    # 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, 100))
    fake_imgs = generator.predict(noise, verbose=0)

    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)

    # ---------------------
    # Train Generator
    # ---------------------
    noise = np.random.normal(0, 1, (batch_size, 100))
    g_loss = gan.train_on_batch(noise, np.ones((batch_size, 1)))

    # Print progress
    if epoch % sample_interval == 0:
        print(f"Epoch {epoch} - D loss: {d_loss[0]:.4f}, acc.: {100*d_loss[1]:.2f}%, G loss: {g_loss:.4f}")
```

OUTPUT:

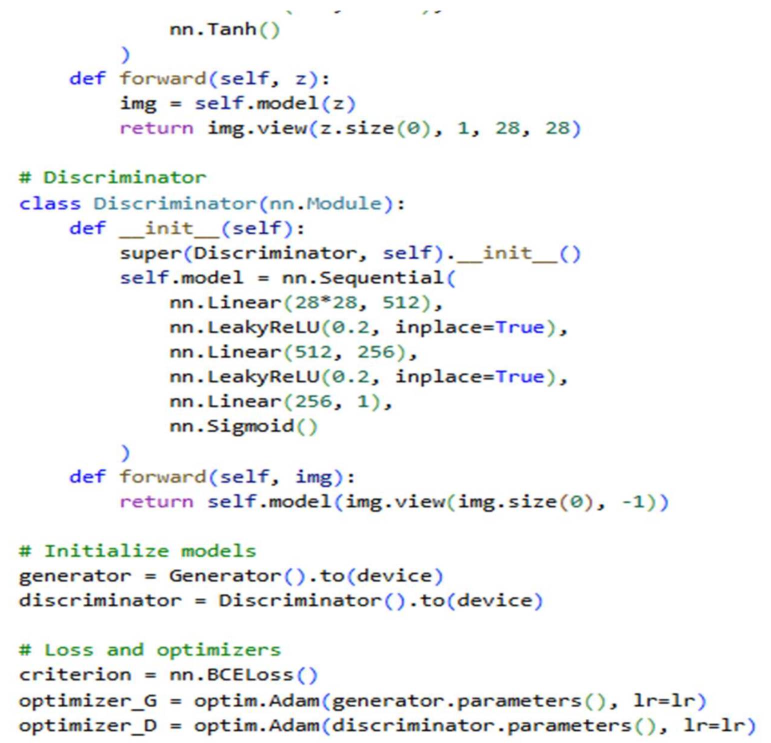


```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.12/dist-packages/keras/src/backend/tensorflow/trainer.py:83: UserWarning: The model does not have any trainable weights.
  warnings.warn("The model does not have any trainable weights.")
Epoch 0 - D loss: 0.7234, acc.: 40.62%, G loss: 0.6969
Epoch 100 - D loss: 0.8163, acc.: 8.68%, G loss: 0.5357
Epoch 200 - D loss: 1.0048, acc.: 8.11%, G loss: 0.3802
Epoch 300 - D loss: 1.1800, acc.: 8.03%, G loss: 0.2883
Epoch 400 - D loss: 1.3076, acc.: 7.94%, G loss: 0.2348
Epoch 500 - D loss: 1.4021, acc.: 7.86%, G loss: 0.2002
Epoch 600 - D loss: 1.4745, acc.: 7.81%, G loss: 0.1760
Epoch 700 - D loss: 1.5321, acc.: 7.79%, G loss: 0.1580
Epoch 800 - D loss: 1.5790, acc.: 7.77%, G loss: 0.1442
Epoch 900 - D loss: 1.6177, acc.: 7.73%, G loss: 0.1332
Epoch 1000 - D loss: 1.6501, acc.: 7.75%, G loss: 0.1243
```

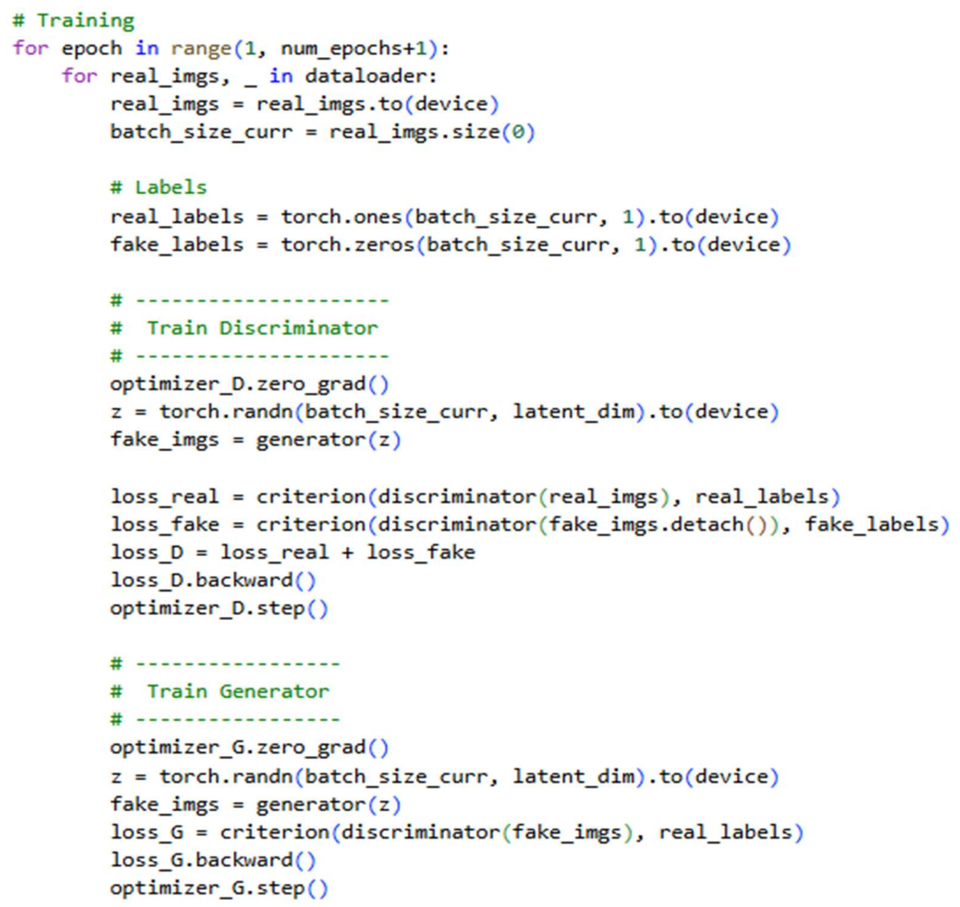
TEST CASES:



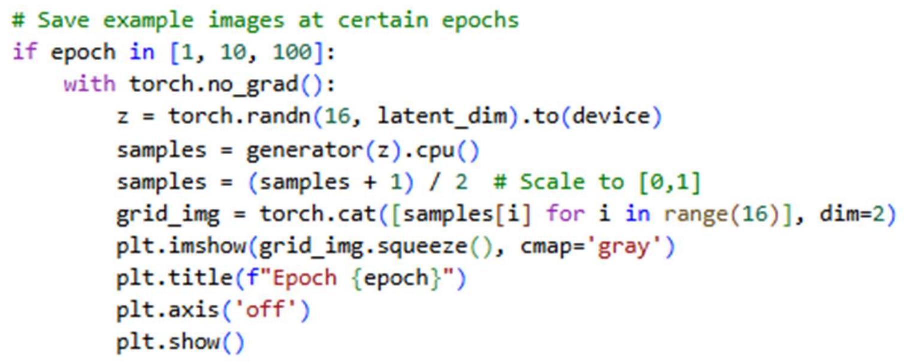
```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Hyperparameters
latent_dim = 100
batch_size = 64
lr = 0.0002
num_epochs = 100

# Transform MNIST
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])

# Dataset
dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

# Generator
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(latent_dim, 128),
            nn.ReLU(True),
            nn.Linear(128, 256),
            nn.ReLU(True),
            nn.Linear(256, 512),
            nn.ReLU(True),
            nn.Linear(512, 28*28),
```



```
            nn.Tanh()
        )
    def forward(self, z):
        img = self.model(z)
        return img.view(z.size(0), 1, 28, 28)

# Discriminator
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )
    def forward(self, img):
        return self.model(img.view(img.size(0), -1))

# Initialize models
generator = Generator().to(device)
discriminator = Discriminator().to(device)

# Loss and optimizers
criterion = nn.BCELoss()
optimizer_G = optim.Adam(generator.parameters(), lr=lr)
optimizer_D = optim.Adam(discriminator.parameters(), lr=lr)
```

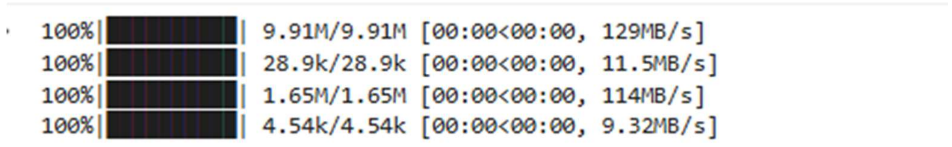


```
# Training
for epoch in range(1, num_epochs+1):
    for real_imgs, _ in dataloader:
        real_imgs = real_imgs.to(device)
        batch_size_curr = real_imgs.size(0)

        # Labels
        real_labels = torch.ones(batch_size_curr, 1).to(device)
        fake_labels = torch.zeros(batch_size_curr, 1).to(device)

        # ---------------------
        #  Train Discriminator
        # ---------------------
        optimizer_D.zero_grad()
        z = torch.randn(batch_size_curr, latent_dim).to(device)
        fake_imgs = generator(z)

        loss_real = criterion(discriminator(real_imgs), real_labels)
        loss_fake = criterion(discriminator(fake_imgs.detach()), fake_labels)
        loss_D = loss_real + loss_fake
        loss_D.backward()
        optimizer_D.step()

        # -----------------
        #  Train Generator
        # -----------------
        optimizer_G.zero_grad()
        z = torch.randn(batch_size_curr, latent_dim).to(device)
        fake_imgs = generator(z)
        loss_G = criterion(discriminator(fake_imgs), real_labels)
        loss_G.backward()
        optimizer_G.step()
```



```python
# Save example images at certain epochs
if epoch in [1, 10, 100]:
    with torch.no_grad():
        z = torch.randn(16, latent_dim).to(device)
        samples = generator(z).cpu()
        samples = (samples + 1) / 2  # Scale to [0,1]
        grid_img = torch.cat([samples[i] for i in range(16)], dim=2)
        plt.imshow(grid_img.squeeze(), cmap='gray')
        plt.title(f"Epoch {epoch}")
        plt.axis('off')
        plt.show()
```

OUTPUT:



```
100%|██████████| 9.91M/9.91M [00:00<00:00, 129MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 11.5MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 114MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 9.32MB/s]
```

Epoch 1