4. BUILD AND TRAIN A GAN FOR GENERATING HAND-WRITTEN DIGITS

EX.N0 : 10	**BUILD AND TRAIN A GAN FOR GENERATING HAND-WRITTEN DIGITS**
DATE : 08/04/2025	

AIM:

To build and train a Generative Adversarial Network (GAN) for generating hand-written digits using the MNIST dataset.

ALGORITHM:

Step 1: Import required libraries (TensorFlow, Keras, NumPy, Matplotlib).

Step 2: Load and preprocess the MNIST dataset for training.

Step 3: Build the Generator and Discriminator models using Keras.

Step 4: Define the loss functions and optimizers for both models.

Step 5: Train the GAN by alternately training the discriminator and generator.

Step 6: Generate and visualize synthetic digit images.

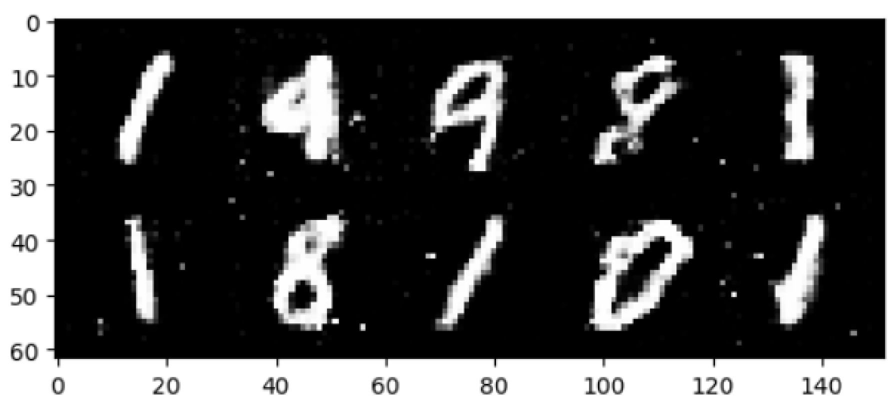
PROGRAM:

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize([0.5], [0.5])])
train_loader = torch.utils.data.DataLoader( torchvision.datasets.MNIST('.', train=True,
download=True, transform=transform), batch_size=128, shuffle=True )
```

```
class Generator(nn.Module):
def __init__(self):
super().__init__()
self.model = nn.Sequential(
nn.Linear(100, 256), nn.ReLU(True),
nn.Linear(256, 512), nn.ReLU(True),
nn.Linear(512, 784), nn.Tanh() )
def forward(self, x): return self.model(x).view(-1, 1, 28, 28)
class Discriminator(nn.Module):
def __init__(self):
super().__init__()
self.model = nn.Sequential(
nn.Flatten(),
nn.Linear(784, 512), nn.LeakyReLU(0.2),
nn.Linear(512, 1), nn.Sigmoid() )
def forward(self, x): return self.model(x)
generator = Generator().to(device)
discriminator = Discriminator().to(device)
criterion = nn.BCELoss()
optimizer_G = torch.optim.Adam(generator.parameters(), lr=0.0002)
optimizer_D = torch.optim.Adam(discriminator.parameters(), lr=0.0002)
epochs = 30
fixed_noise = torch.randn(64, 100, device=device)
for epoch in range(epochs):
for imgs, _ in train_loader:
imgs = imgs.to(device)
batch_size = imgs.size(0)
real_labels = torch.ones(batch_size, 1, device=device)
fake_labels = torch.zeros(batch_size, 1, device=device)
z = torch.randn(batch_size, 100, device=device)
fake_imgs = generator(z)
```

```
d_loss = criterion(discriminator(imgs), real_labels) + criterion(discriminator(fake_imgs.detach()), fake_labels)
optimizer_D.zero_grad()
d_loss.backward()
optimizer_D.step()
g_loss = criterion(discriminator(fake_imgs), real_labels)
optimizer_G.zero_grad()
g_loss.backward()
optimizer_G.step()
print(f"Epoch [{epoch+1}/{epochs}], D Loss: {d_loss.item():.4f}, G Loss: {g_loss.item():.4f}")
if (epoch + 1) % 5 == 0:
with torch.no_grad():
generated = generator(fixed_noise).cpu()
grid = torchvision.utils.make_grid(generated, nrow=8, normalize=True)
plt.imshow(np.transpose(grid, (1, 2, 0)))
plt.title(f"Epoch {epoch+1}")
plt.axis("off")
plt.show()
```

OUTPUT:



RESULT:

Thus the Program has been executed successfully and verified.