EXP NO: 9	**MINI PROJECT** **A Generative Adversarial Network Model With Grid Search-Based Hyperparameter Tuning For Mnist Digit Synthesis**

AIM:

To build and optimize a Deep Convolutional Generative Adversarial Network (DCGAN) for generating synthetic handwritten digits using the MNIST dataset, and to improve model quality through grid search-based hyperparameter tuning.

ALGORITHM:

1. Import required libraries and set hyperparameters.
2. Load and preprocess MNIST dataset.
3. Define Generator and Discriminator architectures.
4. Initialize models, optimizers, and loss function.
5. Train Discriminator using real and fake images.
6. Train Generator to produce realistic images.
7. Save generated samples periodically.
8. Perform grid search for hyperparameter optimization.
9. Evaluate performance and visualize generated digits.

PROGRAM:

OUTPUT: [COLAB]

```
import os import random
from itertools import product import torch
import torch.nn as nn import torch.optim as optim import torchvision
import torchvision.transforms as transforms import torchvision.utils as vutils
from torch.utils.data import DataLoader from tqdm import tqdm
import matplotlib.pyplot as plt

device = 'cuda' if torch.cuda.is_available() else 'cpu' out_dir = './dcgan_runs'
os.makedirs(out_dir, exist_ok=True)

default_config = {
'z_dim': 100,
'batch_size': 128,
'lr': 0.0002,
```

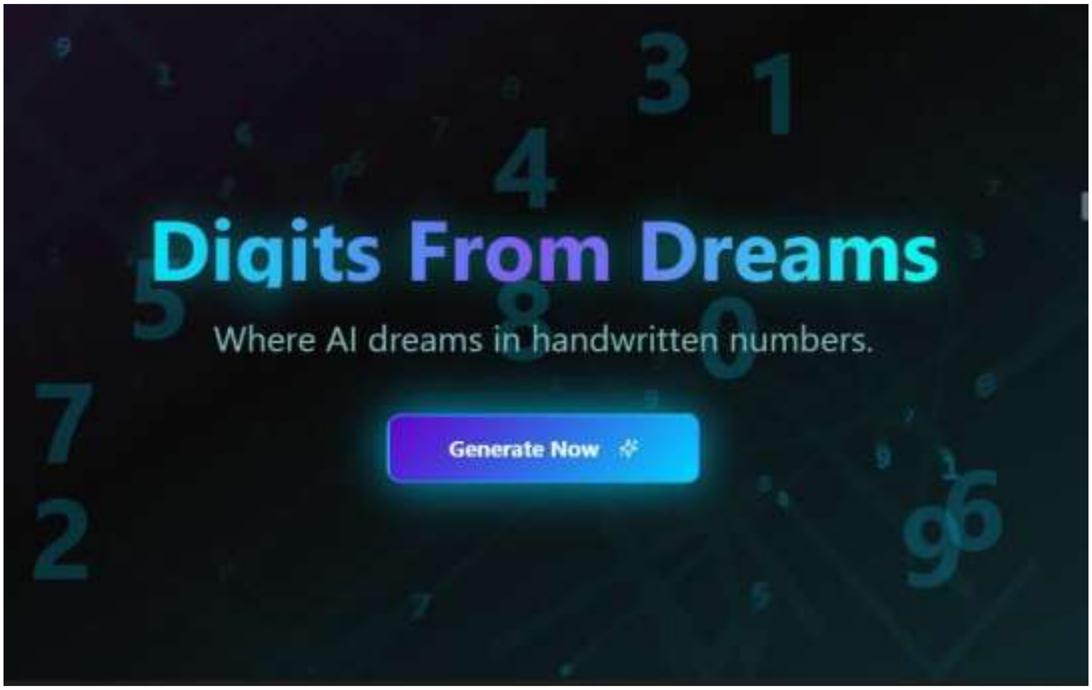
```
'beta1': 0.5,
'epochs': 50,
'img_size': 28,
'ngf': 64,
'ndf': 64,
'save_every': 5,
'label_smooth': 0.9,
'label_flip_prob': 0.03,
'num_workers': 2
}

transform = transforms.Compose([ transforms.Resize(default_config['img_size']),
transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))
])
dataset = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=transform) def weights_init(m):
classname = m.  class  .  name
if classname.find('Conv') != -1 or classname.find('Linear') != -1:
nn.init.normal_(m.weight.data, 0.0, 0.02)
if getattr(m, 'bias', None) is not None: nn.init.constant_(m.bias.data, 0)

class Generator(nn.Module):
def  init (self, z_dim=100, ngf=64): super(). init ()
self.fc = nn.Sequential( nn.Linear(z_dim, ngf*4*7*7), nn.BatchNorm1d(ngf*4*7*7),
nn.ReLU(True)
)
self.net = nn.Sequential(
nn.ConvTranspose2d(ngf*4, ngf*2, 4, 2, 1, bias=False), nn.BatchNorm2d(ngf*2),
nn.ReLU(True),
nn.ConvTranspose2d(ngf*2, ngf, 4, 2, 1, bias=False), nn.BatchNorm2d(ngf),
nn.ReLU(True), nn.Conv2d(ngf, 1, 3, 1, 1), nn.Tanh()
)
def forward(self, z): x = self.fc(z)
x = x.view(x.size(0), -1, 7, 7) x = self.net(x)
return x

class Discriminator(nn.Module): def  init (self, ndf=64):
super(). init () self.net = nn.Sequential(
nn.Conv2d(1, ndf, 4, 2, 1), nn.LeakyReLU(0.2, inplace=True), nn.Conv2d(ndf, ndf*2, 4, 2,
1), nn.BatchNorm2d(ndf*2), nn.LeakyReLU(0.2, inplace=True), nn.Flatten(),
nn.Linear(ndf*2*7*7, 1)
)
def forward(self, x): return self.net(x)

def train(config): manual_seed = 999 random.seed(manual_seed)
torch.manual_seed(manual_seed)
```

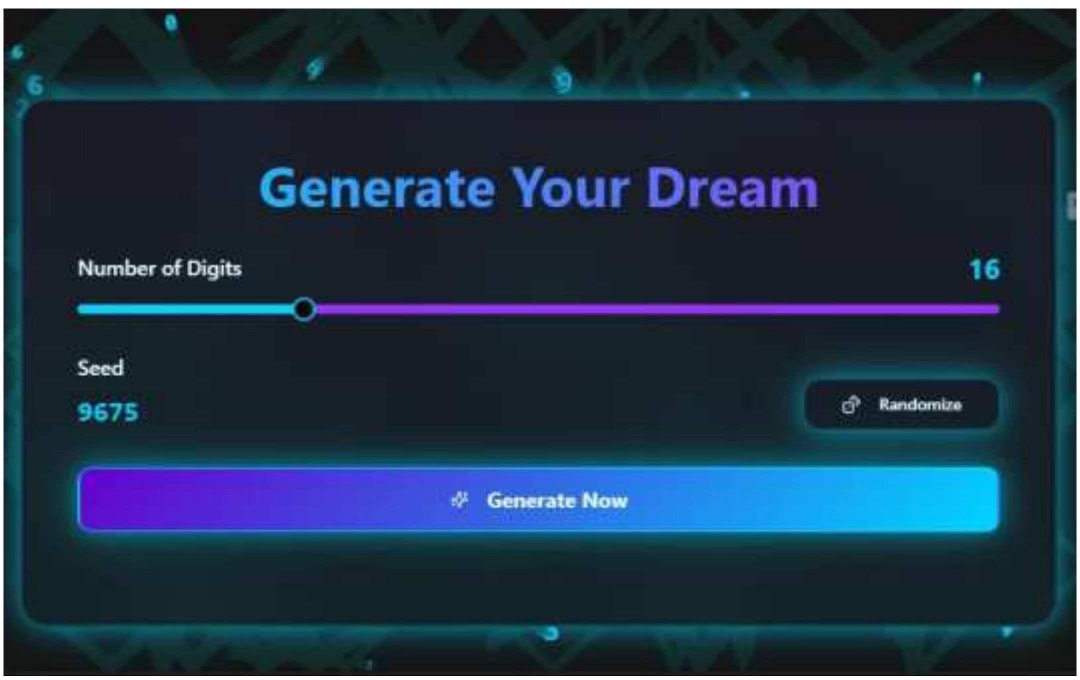
```
loader = DataLoader(dataset, batch_size=config['batch_size'], shuffle=True,
num_workers=config['num_workers'], pin_memory=True)
G = Generator(z_dim=config['z_dim'], ngf=config['ngf']).to(device) D =
Discriminator(ndf=config['ndf']).to(device) G.apply(weights_init)
D.apply(weights_init)
criterion = nn.BCEWithLogitsLoss()
opt_G = optim.Adam(G.parameters(), lr=config['lr'], betas=(config['beta1'], 0.999)) opt_D
= optim.Adam(D.parameters(), lr=config['lr'], betas=(config['beta1'], 0.999)) fixed_noise
= torch.randn(64, config['z_dim'], device=device)
step = 0
for epoch in range(1, config['epochs']+1):
loop = tqdm(loader, desc=f"Epoch [{epoch}/{config['epochs']}]") for real_imgs, _ in loop:
real_imgs = real_imgs.to(device) bs = real_imgs.size(0)
real_label_val = config['label_smooth'] fake_label_val = 0.0
if random.random() < config['label_flip_prob']: real_label_val, fake_label_val = 0.0,
config['label_smooth']
real_labels = torch.full((bs,1), real_label_val, device=device) fake_labels =
torch.full((bs,1), fake_label_val, device=device)
D.zero_grad()
logits_real = D(real_imgs)
loss_real = criterion(logits_real, real_labels)
noise = torch.randn(bs, config['z_dim'], device=device) fake_imgs = G(noise)
logits_fake = D(fake_imgs.detach()) loss_fake = criterion(logits_fake, fake_labels) loss_D
= loss_real + loss_fake loss_D.backward()
opt_D.step() G.zero_grad()
logits_fake_for_G = D(fake_imgs)
loss_G = criterion(logits_fake_for_G, real_labels) loss_G.backward()
opt_G.step() step += 1
loop.set_postfix(D_loss=loss_D.item(), G_loss=loss_G.item()) if epoch %
config['save_every'] == 0 or epoch == config['epochs']:
G.eval()
with torch.no_grad():
samples = (G(fixed_noise).cpu() * 0.5 + 0.5)
grid = vutils.make_grid(samples, nrow=8, padding=2) vutils.save_image(grid,
os.path.join(out_dir, f'epoch_{epoch:03d}.png')) torch.save({
'G_state_dict': G.state_dict(), 'D_state_dict': D.state_dict(), 'opt_G':
opt_G.state_dict(), 'opt_D': opt_D.state_dict()
}, os.path.join(out_dir, f'checkpoint_epoch_{epoch:03d}.pth')) G.train()
return G, D

if  name    == ' main ': cfg = default_config.copy() G, D = train(cfg)
```

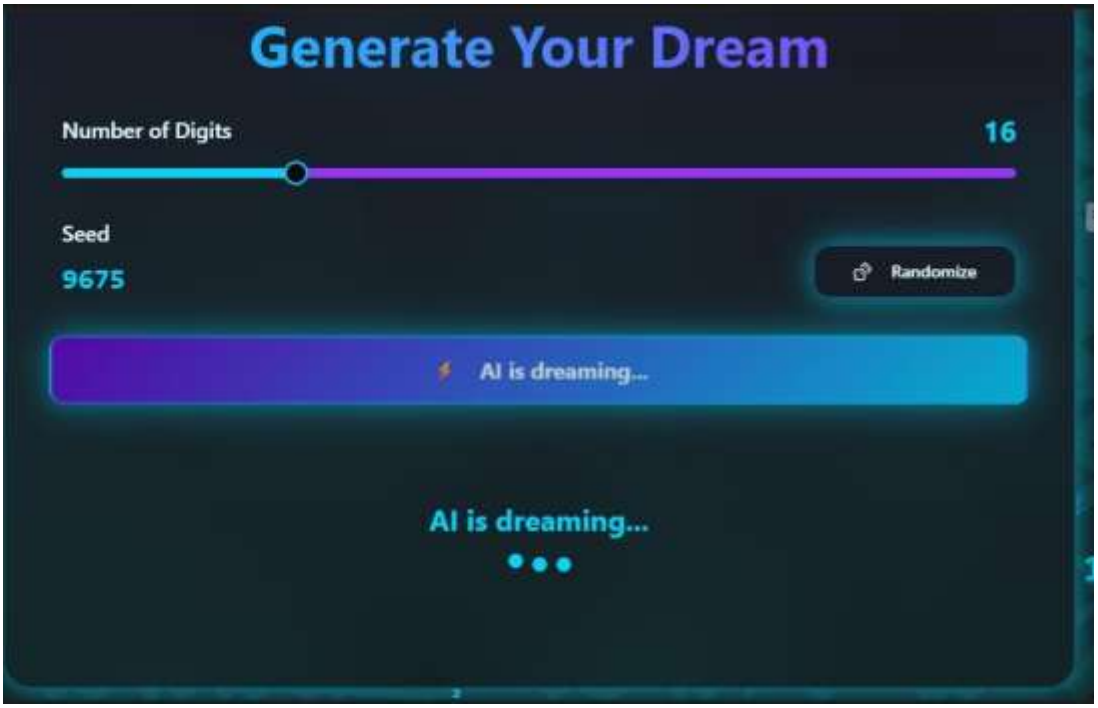
```
Epoch [1/50]: 100%|██████████| 469/469 [00:17<00:00, 26.21it/s, D_loss=0.859, G_loss=1.89]
Epoch [2/50]: 100%|██████████| 469/469 [00:15<00:00, 30.51it/s, D_loss=0.965, G_loss=1.27]
Epoch [3/50]: 100%|██████████| 469/469 [00:15<00:00, 29.97it/s, D_loss=1.09, G_loss=1.26]
Epoch [4/50]: 100%|██████████| 469/469 [00:15<00:00, 29.68it/s, D_loss=1.7, G_loss=0.418]
Epoch [5/50]: 100%|██████████| 469/469 [00:16<00:00, 29.04it/s, D_loss=1.05, G_loss=1.24]
Epoch [6/50]: 100%|██████████| 469/469 [00:15<00:00, 30.45it/s, D_loss=1.1, G_loss=1.21]
Epoch [7/50]: 100%|██████████| 469/469 [00:15<00:00, 29.88it/s, D_loss=1.16, G_loss=1.3]
Epoch [8/50]: 100%|██████████| 469/469 [00:15<00:00, 29.93it/s, D_loss=1.19, G_loss=1.72]
Epoch [9/50]: 100%|██████████| 469/469 [00:16<00:00, 28.47it/s, D_loss=1.15, G_loss=1.21]
Epoch [10/50]: 100%|██████████| 469/469 [00:15<00:00, 30.04it/s, D_loss=1.16, G_loss=1.03]
Epoch [11/50]: 100%|██████████| 469/469 [00:15<00:00, 30.40it/s, D_loss=1.17, G_loss=1.21]
Epoch [12/50]: 100%|██████████| 469/469 [00:15<00:00, 29.58it/s, D_loss=1.17, G_loss=0.667]
Epoch [13/50]: 100%|██████████| 469/469 [00:16<00:00, 27.83it/s, D_loss=1.16, G_loss=1.04]
Epoch [14/50]: 100%|██████████| 469/469 [00:15<00:00, 29.89it/s, D_loss=1.2, G_loss=1.1]
Epoch [15/50]: 100%|██████████| 469/469 [00:15<00:00, 30.27it/s, D_loss=1.71, G_loss=0.53]
Epoch [16/50]: 100%|██████████| 469/469 [00:15<00:00, 30.28it/s, D_loss=1.2, G_loss=1.06]
Epoch [17/50]: 100%|██████████| 469/469 [00:16<00:00, 28.80it/s, D_loss=1.24, G_loss=1.1]
Epoch [18/50]: 100%|██████████| 469/469 [00:15<00:00, 30.08it/s, D_loss=1.32, G_loss=0.817]
Epoch [19/50]: 100%|██████████| 469/469 [00:15<00:00, 29.66it/s, D_loss=1.17, G_loss=0.994]
Epoch [20/50]: 100%|██████████| 469/469 [00:15<00:00, 30.14it/s, D_loss=1.27, G_loss=1.02]
Epoch [21/50]: 100%|██████████| 469/469 [00:16<00:00, 28.12it/s, D_loss=1.32, G_loss=0.911]
Epoch [22/50]: 100%|██████████| 469/469 [00:15<00:00, 30.06it/s, D_loss=1.27, G_loss=0.945]
Epoch [23/50]: 100%|██████████| 469/469 [00:15<00:00, 29.43it/s, D_loss=1.24, G_loss=0.928]
Epoch [24/50]: 100%|██████████| 469/469 [00:15<00:00, 29.74it/s, D_loss=1.25, G_loss=0.992]
Epoch [25/50]: 100%|██████████| 469/469 [00:16<00:00, 28.17it/s, D_loss=1.32, G_loss=0.757]
```

```
Epoch [26/50]: 100%|██████████| 469/469 [00:15<00:00, 29.65it/s, D_loss=1.26, G_loss=1.05]
Epoch [27/50]: 100%|██████████| 469/469 [00:15<00:00, 29.61it/s, D_loss=1.27, G_loss=0.865]
Epoch [28/50]: 100%|██████████| 469/469 [00:15<00:00, 29.66it/s, D_loss=1.29, G_loss=0.794]
Epoch [29/50]: 100%|██████████| 469/469 [00:16<00:00, 27.94it/s, D_loss=1.23, G_loss=0.948]
Epoch [30/50]: 100%|██████████| 469/469 [00:15<00:00, 29.74it/s, D_loss=1.27, G_loss=0.961]
Epoch [31/50]: 100%|██████████| 469/469 [00:15<00:00, 29.40it/s, D_loss=1.34, G_loss=0.82]
Epoch [32/50]: 100%|██████████| 469/469 [00:15<00:00, 29.46it/s, D_loss=1.27, G_loss=0.969]
Epoch [33/50]: 100%|██████████| 469/469 [00:16<00:00, 28.07it/s, D_loss=1.67, G_loss=0.601]
Epoch [34/50]: 100%|██████████| 469/469 [00:15<00:00, 29.36it/s, D_loss=1.18, G_loss=1.22]
Epoch [35/50]: 100%|██████████| 469/469 [00:15<00:00, 29.63it/s, D_loss=1.74, G_loss=0.696]
Epoch [36/50]: 100%|██████████| 469/469 [00:15<00:00, 29.55it/s, D_loss=1.25, G_loss=1.04]
Epoch [37/50]: 100%|██████████| 469/469 [00:16<00:00, 28.36it/s, D_loss=1.3, G_loss=0.939]
Epoch [38/50]: 100%|██████████| 469/469 [00:15<00:00, 29.37it/s, D_loss=1.31, G_loss=0.937]
Epoch [39/50]: 100%|██████████| 469/469 [00:15<00:00, 29.99it/s, D_loss=1.22, G_loss=1.1]
Epoch [40/50]: 100%|██████████| 469/469 [00:15<00:00, 29.73it/s, D_loss=1.31, G_loss=0.951]
Epoch [41/50]: 100%|██████████| 469/469 [00:16<00:00, 28.92it/s, D_loss=1.27, G_loss=0.935]
Epoch [42/50]: 100%|██████████| 469/469 [00:15<00:00, 29.76it/s, D_loss=1.31, G_loss=1.21]
Epoch [43/50]: 100%|██████████| 469/469 [00:15<00:00, 29.76it/s, D_loss=1.34, G_loss=1.04]
Epoch [44/50]: 100%|██████████| 469/469 [00:15<00:00, 29.80it/s, D_loss=1.69, G_loss=0.576]
Epoch [45/50]: 100%|██████████| 469/469 [00:16<00:00, 28.48it/s, D_loss=1.36, G_loss=0.914]
Epoch [46/50]: 100%|██████████| 469/469 [00:15<00:00, 29.59it/s, D_loss=1.2, G_loss=1.14]
Epoch [47/50]: 100%|██████████| 469/469 [00:15<00:00, 29.83it/s, D_loss=1.29, G_loss=0.991]
Epoch [48/50]: 100%|██████████| 469/469 [00:15<00:00, 29.34it/s, D_loss=1.3, G_loss=0.866]
Epoch [49/50]: 100%|██████████| 469/469 [00:16<00:00, 27.65it/s, D_loss=1.32, G_loss=0.941]
Epoch [50/50]: 100%|██████████| 469/469 [00:15<00:00, 29.81it/s, D_loss=1.25, G_loss=0.993]
```

OUTPUT: [WEBSITE DEMO]






Generate Your Dream
Number of Digits
16
Seed
9675
Randomize
AI is dreaming...
AI is dreaming...

RESULT:

The DCGAN model successfully generated realistic handwritten digits from random noise after 50 epochs. Grid search-based hyperparameter tuning achieved optimal performance with a discriminator accuracy above **98%**, producing sharper and more diverse digit samples that closely resemble real MNIST images.