```
# Deep Convolutional GAN (DCGAN)
# Generates complex color images (e.g. CIFAR-10)

import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torchvision.utils import save_image

# ----------------------------
# 1. Data Loading
# ----------------------------
transform = transforms.Compose([
    transforms.Resize(64),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

dataset = datasets.CIFAR10(root='./data', download=True, transform=transform)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=128, shuffle=True)

# ----------------------------
# 2. Generator
# ----------------------------
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.ConvTranspose2d(100, 512, 4, 1, 0, bias=False),
            nn.BatchNorm2d(512),
            nn.ReLU(True),

            nn.ConvTranspose2d(512, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.ReLU(True),

            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),

            nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),

            nn.ConvTranspose2d(64, 3, 4, 2, 1, bias=False),
            nn.Tanh()
        )

    def forward(self, x):
        return self.main(x)

# ----------------------------
# 3. Discriminator (FIXED)
# ----------------------------
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Conv2d(3, 64, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(128, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(256, 512, 4, 2, 1, bias=False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),

            # Make sure output → [batch, 1, 1, 1]
            nn.Conv2d(512, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):
        # flatten from [batch,1,1,1] → [batch]
        return self.main(x).view(-1)
```

```
        return self.main(x).view(-1)

# ----------------------------
# 4. Setup
# ----------------------------
device = 'cuda' if torch.cuda.is_available() else 'cpu'
G = Generator().to(device)
D = Discriminator().to(device)

criterion = nn.BCELoss()
optimizerG = optim.Adam(G.parameters(), lr=0.0002, betas=(0.5, 0.999))
optimizerD = optim.Adam(D.parameters(), lr=0.0002, betas=(0.5, 0.999))

# ----------------------------
# 5. Training Loop
# ----------------------------
for epoch in range(3):  # demo epochs
    for imgs, _ in dataloader:
        real = imgs.to(device)
        b_size = real.size(0)

        # Labels
        real_label = torch.ones(b_size, device=device)
        fake_label = torch.zeros(b_size, device=device)

        # --- Train D ---
        noise = torch.randn(b_size, 100, 1, 1, device=device)
        fake = G(noise)

        D_real = D(real)
        D_fake = D(fake.detach())

        lossD_real = criterion(D_real, real_label)
        lossD_fake = criterion(D_fake, fake_label)
        lossD = lossD_real + lossD_fake

        optimizerD.zero_grad()
        lossD.backward()
        optimizerD.step()

        # --- Train G ---
        D_fake = D(fake)
        lossG = criterion(D_fake, real_label)

        optimizerG.zero_grad()
        lossG.backward()
        optimizerG.step()

    print(f"Epoch [{epoch+1}/3] | Loss_D: {lossD:.4f} | Loss_G: {lossG:.4f}")

    # Save generated images
    with torch.no_grad():
        sample = G(torch.randn(16, 100, 1, 1, device=device)).detach().cpu()
    save_image(sample, f"dcgan_generated_epoch_{epoch+1}.png", normalize=True)

print("✅ Training complete! Generated images saved.")
```

```
Epoch [1/3] | Loss_D: 0.3889 | Loss_G: 4.4132
Epoch [2/3] | Loss_D: 0.4372 | Loss_G: 5.1061
Epoch [3/3] | Loss_D: 0.3033 | Loss_G: 3.3544
✅ Training complete! Generated images saved.
```

Start coding or generate with AI.