

Exp : 12

Implement a Deep convolution GAN to generate complex color image.

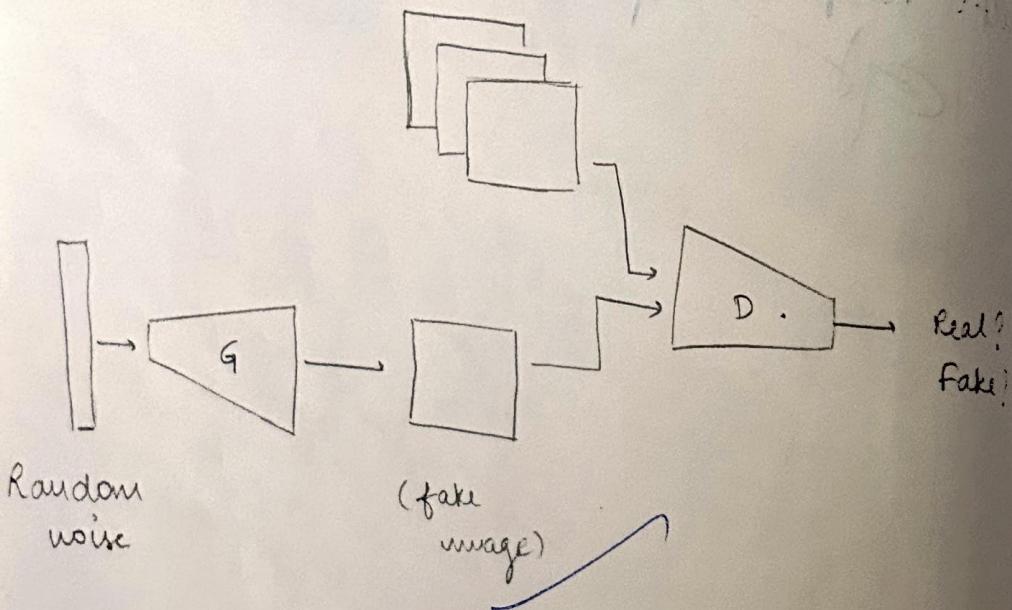
Aim: To implement a Deep convolution GAN in PyTorch for generating realistic color images.

Pseudo code:

1. Import libraries and load dataset (eg: CIFAR-10 or custom image)
2. Normalize image
3. Define the Generator:
 - Input: random noise
 - layers: convTranspose2D + batchNorm + ReLU
 - Output: 3-channel (RGB) image
4. Define the Discriminators:
 - Input: 3-channel image
 - layers: conv2D + BatchNorm + leaky ReLU
 - Outputs: Probability (real / fake)
5. Define loss (Binary cross Entropy) + optimizers
6. Training loop:
 - a. Train ~~discriminators~~ discriminators of real + fake images.
 - b. Train generators to feed the discriminator.
7. Generate & visualize fake color image.

These are called noise & it has
two types:
1. White Gaussian noise &
2. Colored Gaussian noise &
3. Impulse noise &
4. Salt & Pepper noise &
5. Speckle noise &
6. Salt & Pepper noise &
7. Uniform noise &
8. Non-uniform noise

(Real image) \rightarrow $\text{G} \rightarrow$ D \rightarrow Real/Fake



Justification:

* Deep convolution GAN uses convolutional & transposed convolutional layers to learn hierarchical image features directly from data.

~~It's effective for generating complex color images because convolutional filters capture spatial & color patterns efficiently, allowing the generator to create realistic images while the discriminator learns to distinguish real from fake ones.~~

RGJ

Result : Successfully Impl.

loss:

Epoch 1: D: 0.6352

G: 1.9395

Epoch 2: D: 0.1788 G: 0.3006

G: 7.0664 2.2320

Epoch 3: D: 0.0010 G: 0.1788

G: 7.0664

Epoch 4: D: 0.10566

G: 3.1548

Epoch 5: D: 0.0258 G: 0.3800

G: 0.0158

Epoch 6: D: 0.0099 G: 4.4111

G: 0.0099

Epoch 7: D: 0.0099 G: 1.8699

G: 0.0099

Epoch 8: D: 0.2179

G: 2.5012

Epoch 9: D: 0.0610 G: 3.71586

G: 0.0610

Epoch 10: D: 0.5041 G: 1.279

G: 1.279

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torchvision.utils as vutils
import matplotlib.pyplot as plt
import numpy as np
batch_size = 128
image_size = 32
nz = 100
ngf = 64
ndf = 64
nc = 3
num_epochs = 20
lr = 0.0002
beta1 = 0.5

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
transform = transforms.Compose([
    transforms.Resize(image_size),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

dataset = datasets.CIFAR10(root='./data', download=True, transform=transform)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
class Generator(nn.Module):
    def __init__(self):
```

File Edit View Insert Runtime Tools Help

Commands

+ Code

+ Text

> Run all

Untitled8.ipynb - Colab

Deeplearning_RA2311047010012/week7 at main · Aamir06



Connecting



```
        nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
        nn.BatchNorm2d(ndf * 2),
        nn.LeakyReLU(0.2, inplace=True),
        nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
        nn.BatchNorm2d(ndf * 4),
        nn.LeakyReLU(0.2, inplace=True),
        nn.Conv2d(ndf * 4, 1, 4, 1, 0, bias=False),
        nn.Sigmoid()
    )

    def forward(self, x):
        return self.main(x).view(-1, 1).squeeze(1)
netG = Generator().to(device)
netD = Discriminator().to(device)
def weights_init(m):
    if isinstance(m, (nn.Conv2d, nn.ConvTranspose2d, nn.BatchNorm2d)):
        nn.init.normal_(m.weight.data, 0.0, 0.02)
netG.apply(weights_init)
netD.apply(weights_init)
criterion = nn.BCELoss()
optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1, 0.999))
fixed_noise = torch.randn(64, nz, 1, 1, device=device)
G_losses, D_losses = [], []

print("Starting Training DCGAN...")
for epoch in range(num_epochs):
```



```
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.ConvTranspose2d(nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),

            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),

            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.ConvTranspose2d(ngf, nc, 3, 1, 1, bias=False),
            nn.Tanh()
        )

    def forward(self, x):
        return self.main(x)
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
```



+ Code

+ Text | ▶ Run all

```

optimizerG.step()
G_losses.append(lossG.item())
D_losses.append(lossD.item())

print(f"Epoch [{epoch+1}/{num_epochs}] | Loss_D: {lossD.item():.4f} | Loss_G: {lossG.item():.4f}")
with torch.no_grad():
    fake_samples = netG(fixed_noise).detach().cpu()
grid = vutils.make_grid(fake_samples, padding=2, normalize=True)
plt.figure(figsize=(6,6))
plt.axis("off")
plt.title(f"Generated Images - Epoch {epoch+1}")
plt.imshow(np.transpose(grid, (1,2,0)))
plt.show()

plt.figure(figsize=(8,4))
plt.plot(G_losses, label="Generator Loss")
plt.plot(D_losses, label="Discriminator Loss")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.title("DCGAN Training Loss")
plt.legend()
plt.show()

```

100% |████████| 170M/170M [00:02<00:00, 77.3MB/s]



Training DCGAN...

Epoch [1/20] | Loss_D: 1.1088 | Loss_G: 2.3075

Generated Images - Epoch 1





Epoch [2/20] | Loss D: 1.0298 | Loss G: 1.9091

Generated Images - Epoch 2





Epoch [2/20] | Loss_D: 1.0298 | Loss_G: 1.9091

Generated Images - Epoch 2

