Completed

D. Khaja Nawaz
RA2311047010037

Deep Learning Techniques (LAB)

Date	Title	Sign
24/07/2025	1. Exploring the deep learning platform	
31/07/25	2. Implement a classifier using open-source dataset	7/8/25
31/07/25	3. Study of the classifiers with respect to statistical parameters	14/8/25
14/08/25	4. Build a simple feed forward neural network to recognize handwritten character	14/8/25
22/08/25	5. Study of Activation functions and their role	9/9
09/09/25	6. Implement gradient descent and backpropagation in deep neural network	
16/09/25	7 Build a CNN model to classify Cat and Dog image	23/9/25
30/09/2025	8. Experiment of LSTM	
30/09/2025	9. Build a Recurrent Neural Network	16/10/25
9/10/2025	10. perform compression on MNIST dataset using autoencoder	
9/10/2025	11. Experiments using variational autoen	
03/11/25	12. Implement a Deep convolutional GAN to generate complex color image	
03/11/25	13. Understanding the architecture of pre-trained model	
03/11/25	14. Implement a pre-trained CNN model as feature Extractor using	
03/11/25	15. Implement a yolo model to detect objects	

03/11/25 LAB: 12

Implement a Deep Convolutional GAN to generate complex color image

*** AIM ÷**

To implement a Deep convolutional ~~convolutional~~ Generative Adversarial Network (DCGAN) of generating complex RGB color image that resemble real-world images.

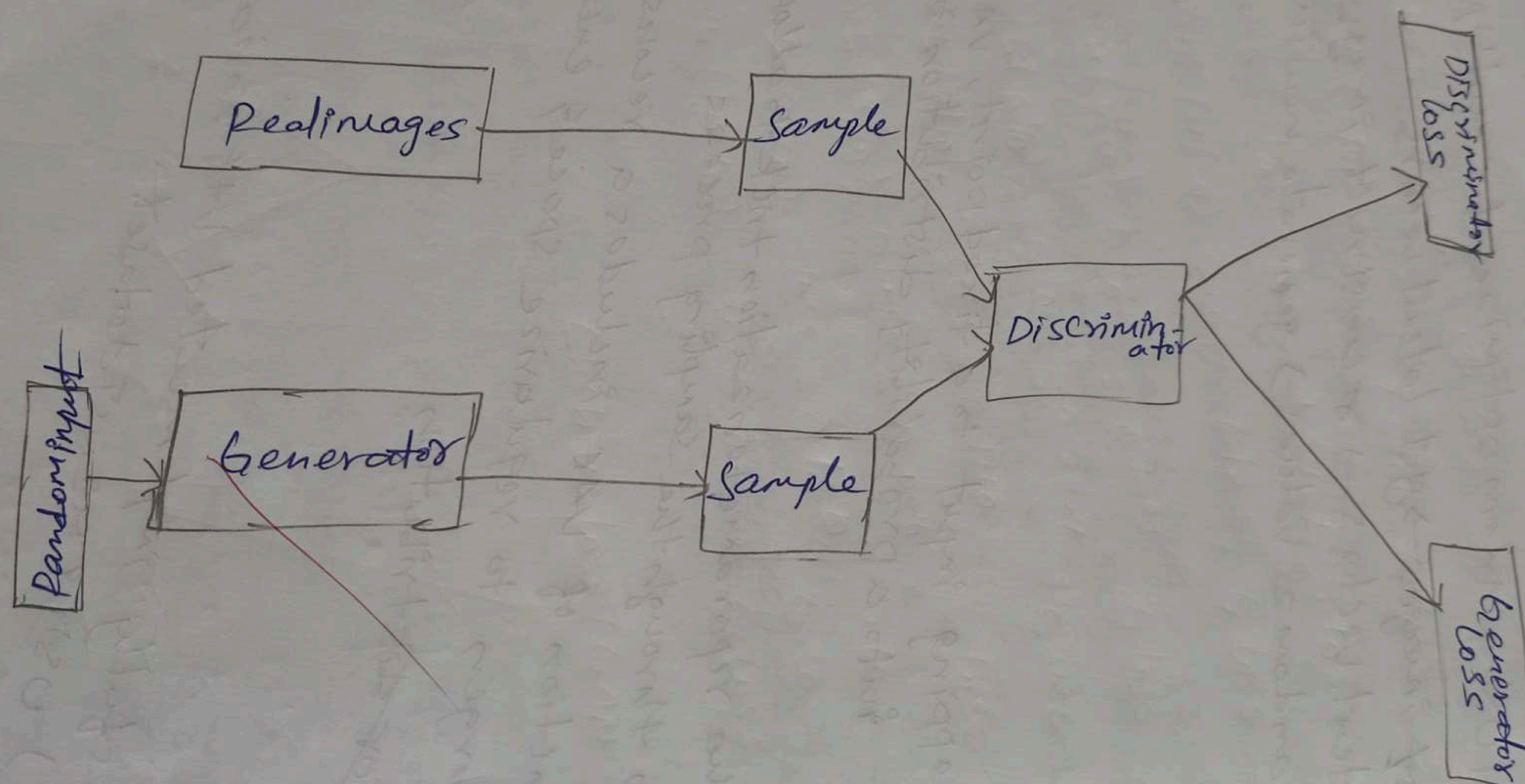
*** objective =**

1. To understand the working of Generative Adversorial Netwoork (GANs)
2. To use convolutional layers for image generation and discrimination.
3. To train the Generator to produce realistic image and the Discriminator to distinguish real from fake
4. To evaluate the quality of generated images visually after training.

*** pseudocode =**

Begin
1. Load dataset of real color images
   → Resize and normalize images to [-1, 1]
2. Define Generator network (G):
   Input: random noise vector (z)
   Layers: series of ConvTranspose2D + BatchNorm + ReLU
   output: RGB image
3. Define discriminator network (D):
   Input: RGB image
   Layers: series of conv2D + BatchNorm + Leakly Relu
   output: single probability

Archeticutre of GAN



output:

Epoch (1/10), D LOSS : 0.0992 , G loss : 0.5678

Epoch (2/10), D LOSS : 0.5860, G LOSS = 2.7113

Epoch (3/10), D LOSS = 1.2431, G LOSS = 3.9827

Epoch (4/10), D LOSS = 0.1285, G LOSS : 4.1865

Epoch (5/10), D LOSS = 0.1988, G LOSS : 3.7253

Epoch (6/10), D LOSS = 0.2754, G LOSS : 4.0325

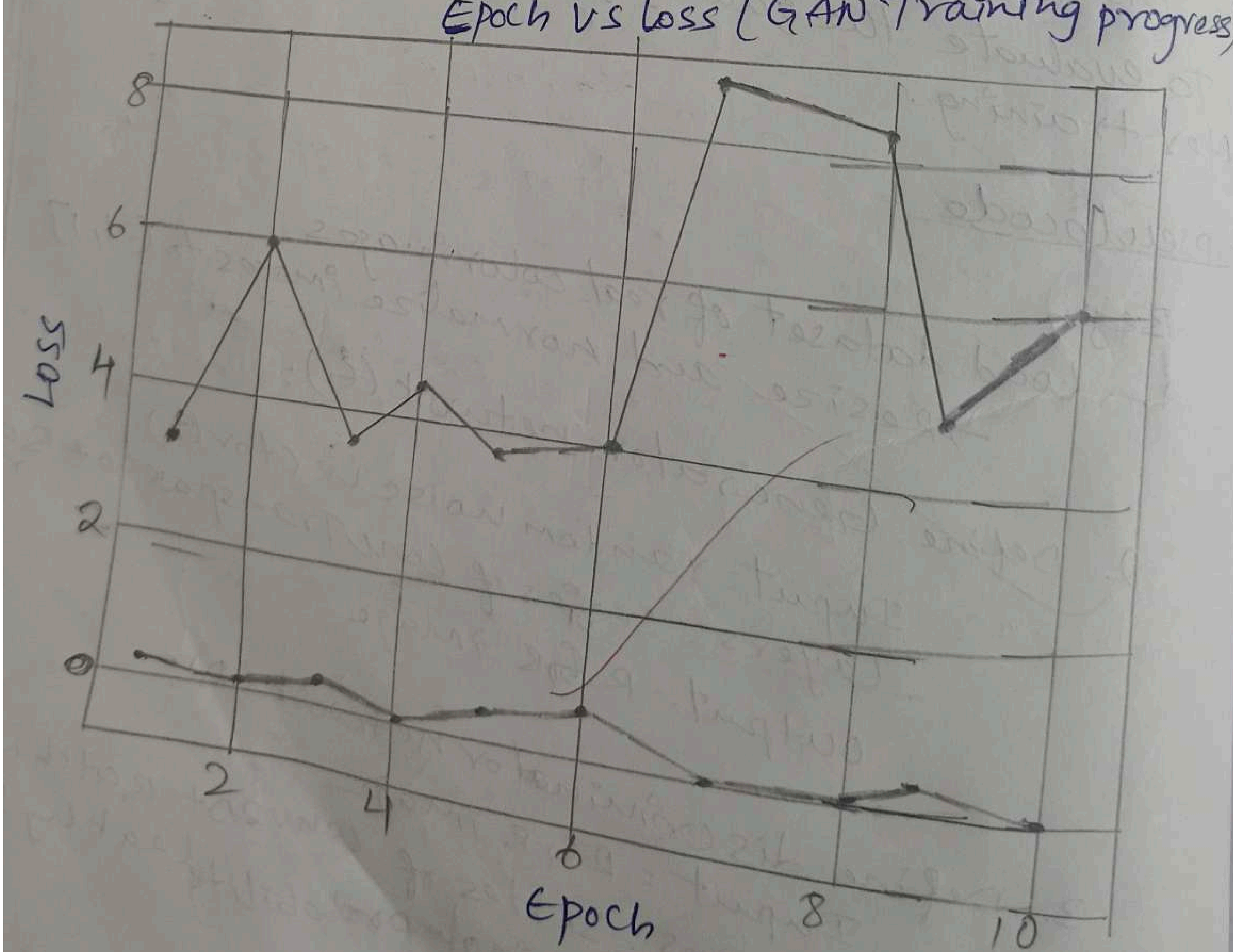
Epoch (7/10), D LOSS = 0.1110, G LOSS : 8.6429

Epoch (8/10), D LOSS = 0.0016, G LOSS : 8.4825

Epoch (9/10), D LOSS = 0.1216, G LOSS : 4.8809

Epoch (10/10), D LOSS : 0.0389, G LOSS : 6.7671

Epoch vs loss (GAN Training progress)



4. SET loss function = Binary cross entropy
SET optimizer = Adam (lr = 0.0002, B_1 = 0.5)

5. For each epoch:
a. Train Discriminator:
→ Feed real image → label = 1
→ Generate fake image from G → label = 0
→ compute loss and update D

b. Train Generator:
→ Generate fake images from random choice
→ label as real
→ compute loss and update G

6. Repeat until generator produces relalistic color images

END

*observation:-
→ The generator gradually learned to produce relistic color images
→ The discriminator improved in distinguishing real and fake images
→ Generated images were visually similar to the training dataset after sufficient epochs
→ Training demonstrated the adversarial learning process of GAN effectively

*Result:-
Succesfully implemented a Deep convolutional GAN to generate complex color images

Lab12.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect Share

```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 torchvision.utils as vutils
import matplotlib.pyplot as plt
import numpy as np

# 1. Hyperparameters
batch_size = 128
latent_dim = 100
epochs = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 2. Data loading (CIFAR-10 color images)
transform = transforms.Compose([
    transforms.Resize(64),
    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)

# 3. Define Generator
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.model = 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.model(x)
```

Lab12.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```python
# 4. Define Discriminator
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.model = 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),
            nn.Conv2d(512, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )
    def forward(self, x):
        return self.model(x).view(-1, 1)

# 5. Initialize models
netG, netD = Generator().to(device), Discriminator().to(device)
criterion = nn.BCELoss()
optimizerG = optim.Adam(netG.parameters(), lr=0.0002, betas=(0.5, 0.999))
optimizerD = optim.Adam(netD.parameters(), lr=0.0002, betas=(0.5, 0.999))

# Lists to store losses
d_losses = []
g_losses = []

# 6. Training loop
for epoch in range(epochs):
    for i, (real_imgs, _) in enumerate(dataloader):
        real_imgs = real_imgs.to(device)
        batch_size = real_imgs.size(0)

        # Real labels = 1, Fake labels = 0
        real_labels = torch.ones(batch_size, 1, device=device)
        fake_labels = torch.zeros(batch_size, 1, device=device)

        # Train Discriminator
        z = torch.randn(batch_size, latent_dim, 1, 1, device=device)
        fake_imgs = netG(z)
        real_loss = criterion(netD(real_imgs), real_labels)
        fake_loss = criterion(netD(fake_imgs.detach()), fake_labels)
        d_loss = real_loss + fake_loss

        optimizerD.zero_grad()
```

```python
    # Store losses
    d_losses.append(d_loss.item())
    g_losses.append(g_loss.item())

    print(f"Epoch [{epoch+1}/{epochs}] D Loss: {d_loss:.4f}, G Loss: {g_loss:.4f}")

# 7. Plot Epoch Graph (Loss vs Epoch)
plt.figure(figsize=(8,5))
plt.plot(range(1, epochs+1), d_losses, label='Discriminator Loss', marker='o')
plt.plot(range(1, epochs+1), g_losses, label='Generator Loss', marker='o')
plt.title("Epoch vs Loss (GAN Training Progress)")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()

# 8. Generate samples
z = torch.randn(64, latent_dim, 1, 1, device=device)
fake_imgs = netG(z)
vutils.save_image(fake_imgs, 'dcgan_generated.png', normalize=True)
plt.figure(figsize=(8,8))
plt.axis("off")
plt.title("Generated Images")
plt.imshow(np.transpose(vutils.make_grid(fake_imgs[:64], padding=2, normalize=True).cpu(), (1,2,0)))
plt.show()
```

```
100%|██████████| 170M/170M [00:08<00:00, 19.3MB/s]
Epoch [1/10] D Loss: 0.3572, G Loss: 3.6326
Epoch [2/10] D Loss: 0.2364, G Loss: 5.8390
Epoch [3/10] D Loss: 0.4392, G Loss: 3.4715
Epoch [4/10] D Loss: 0.1285, G Loss: 4.1865
Epoch [5/10] D Loss: 0.1988, G Loss: 3.7253
Epoch [6/10] D Loss: 0.2754, G Loss: 4.0325
Epoch [7/10] D Loss: 0.1110, G Loss: 8.6499
Epoch [8/10] D Loss: 0.0016, G Loss: 8.4825
Epoch [9/10] D Loss: 0.1210, G Loss: 4.8809
Epoch [10/10] D Loss: 0.0389, G Loss: 6.7671
```