EX·NO:12
27.10.25

IMPLEMENT A DEEP CONVOLUTIONAL GAN TO GENERATE COMPLEX COLOR IMAGES

AIM

To implement a Deep convolutional gan to generate complex color images.

objective

To understand GAN architecture

To train model on color image dataset

To observe training dynamics and quality of generating images.

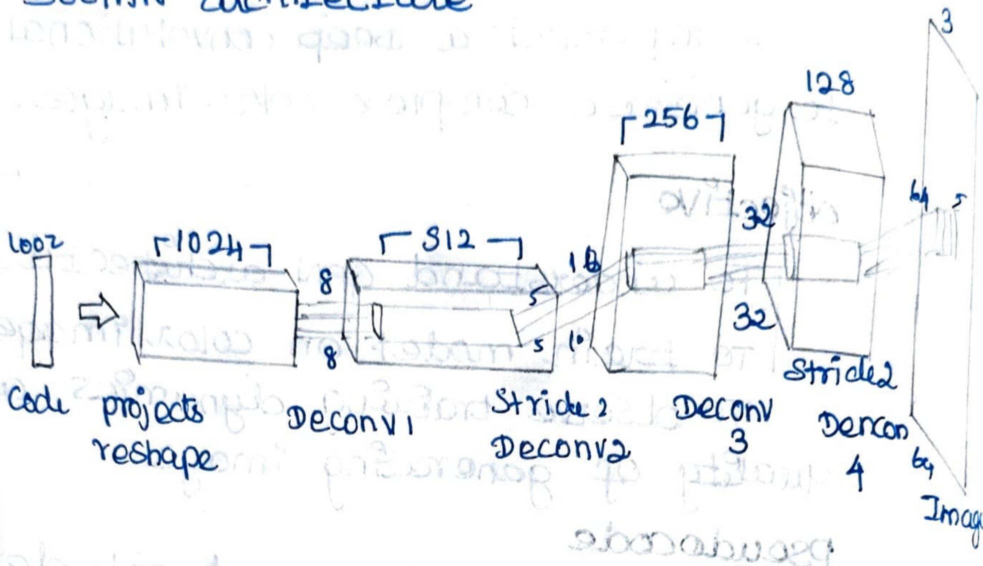
pseudocode

```
import libraries and set device
Load CIFAR-10 dataset, normalize to [-1,1], create dataloader
Define DCGAN generator: series of convtranspose 2D, BatchNorm 2d, Relu
Initialize weights (Normal with mean = 0, std = 0.02)
Define Loss & optimizers
for each epochs
    a) Train Discriminator real images
    b) Train generator to fool Discriminator
    c) save generator batch of generator image for visualization.
Monitor Losses
```

observation

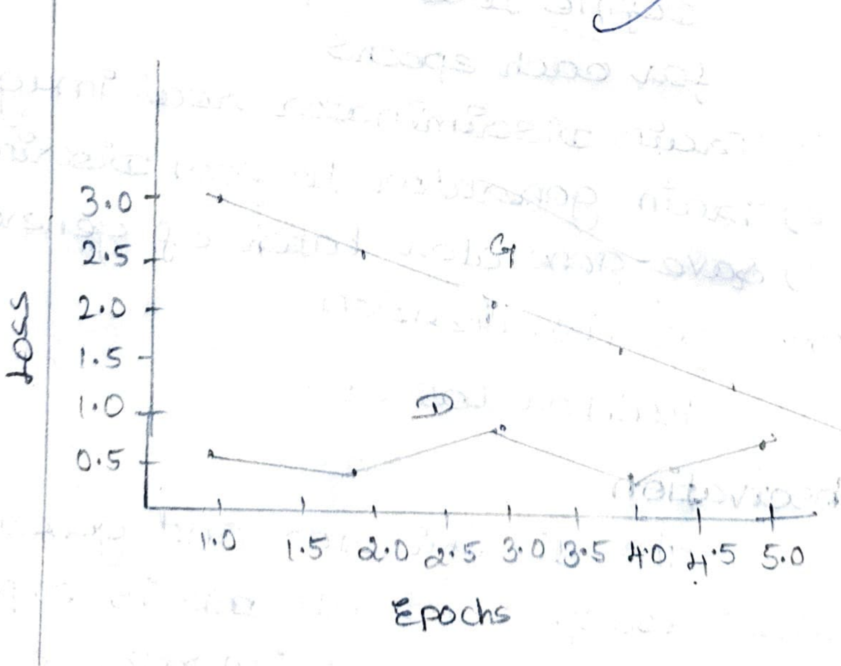
The discriminator and generator losses usally fluctuate this is expected in GAN training as one improves, other responds.

DCGAN architecture



Output

```
Epoch [1/5]  LOSS D : 0.4934  LOSS G : 2.9012
Epoch [2/5]  LOSS D : 0.5894  LOSS G : 266182
Epoch [3/5]  LOSS D : 0.7406  LOSS G : 2.3580
Epoch [4/5]  LOSS D : 0.5590  LOSS G : 2.0222
Epoch (5/5)  LOSS D : 1.3862  LOSS G : 1.5855
```



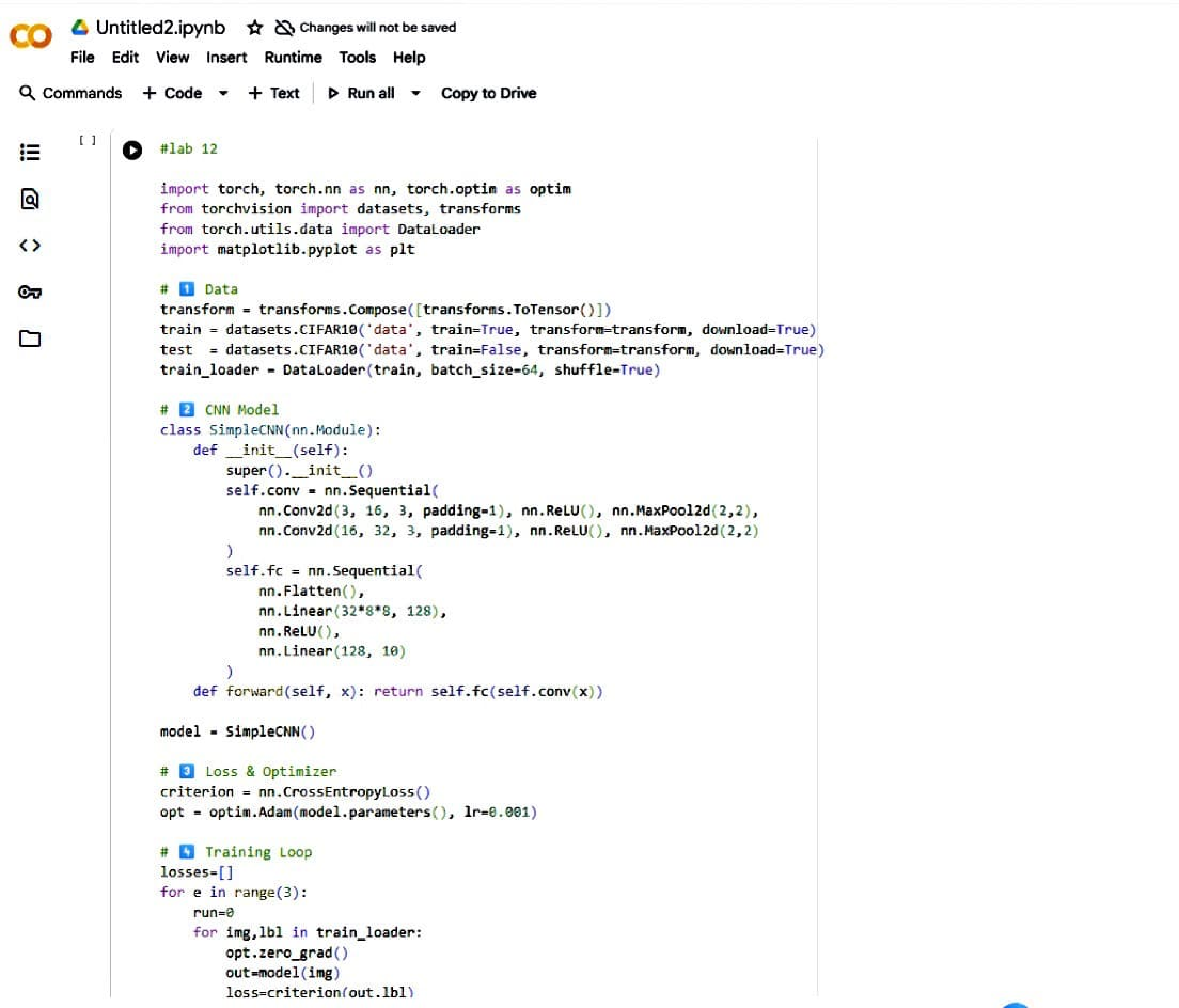
initially generator images are noisy random patterns

over epochs starts showing structure (colors, shape)

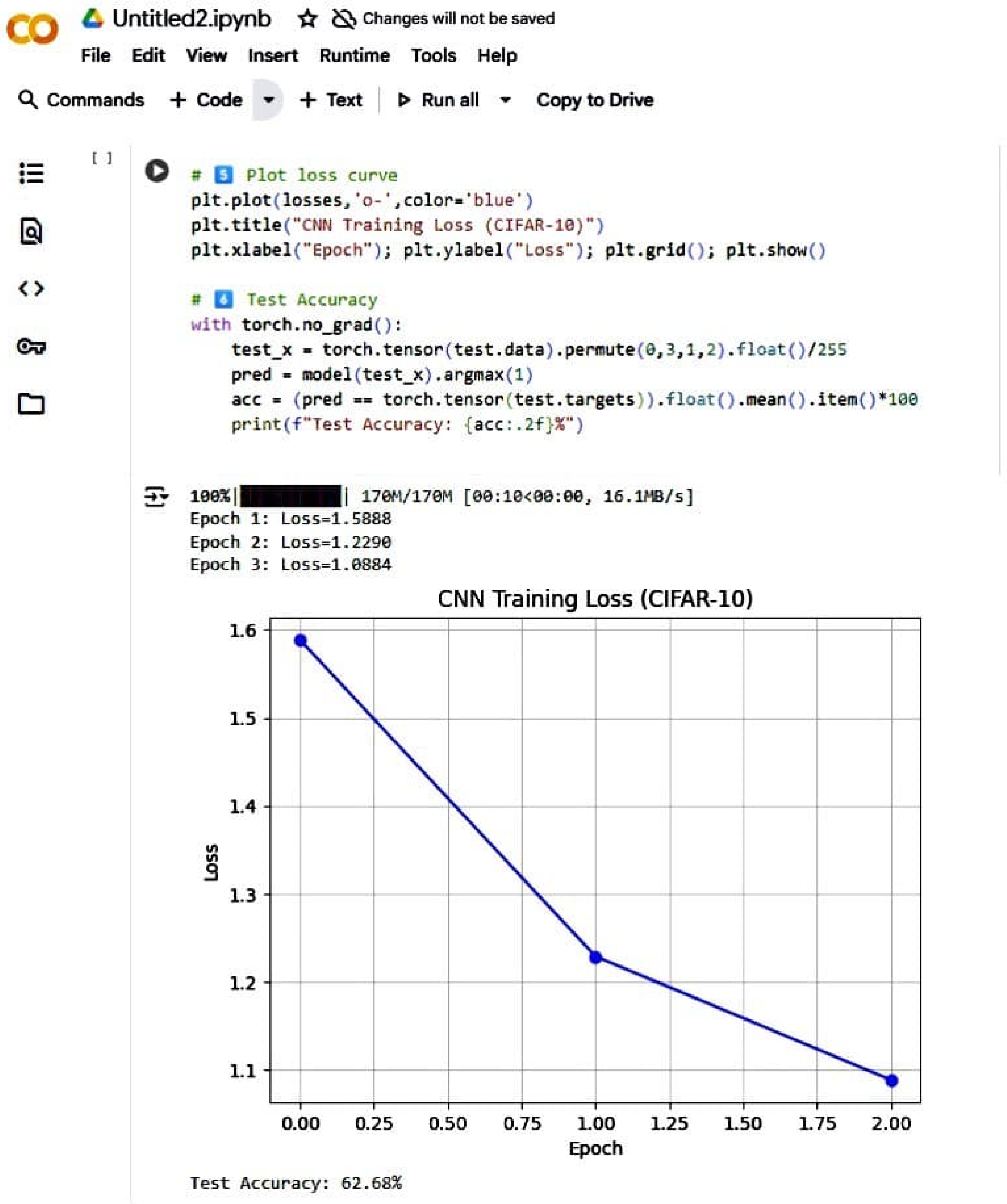
saving images each epoch helps visualize progression from noise → structure → clearer images.

Result

Successfully implemented DCGAN.



```
#lab 12

import torch, torch.nn as nn, torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# 1️⃣ Data
transform = transforms.Compose([transforms.ToTensor()])
train = datasets.CIFAR10('data', train=True, transform=transform, download=True)
test  = datasets.CIFAR10('data', train=False, transform=transform, download=True)
train_loader = DataLoader(train, batch_size=64, shuffle=True)

# 2️⃣ CNN Model
class SimpleCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2,2),
            nn.Conv2d(16, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2,2)
        )
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32*8*8, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )
    def forward(self, x): return self.fc(self.conv(x))

model = SimpleCNN()

# 3️⃣ Loss & Optimizer
criterion = nn.CrossEntropyLoss()
opt = optim.Adam(model.parameters(), lr=0.001)

# 4️⃣ Training Loop
losses=[]
for e in range(3):
    run=0
    for img,lbl in train_loader:
        opt.zero_grad()
        out=model(img)
        loss=criterion(out,lbl)
```



```
# 5️⃣ Plot loss curve
plt.plot(losses,'o-',color='blue')
plt.title("CNN Training Loss (CIFAR-10)")
plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.grid(); plt.show()

# 6️⃣ Test Accuracy
with torch.no_grad():
    test_x = torch.tensor(test.data).permute(0,3,1,2).float()/255
    pred = model(test_x).argmax(1)
    acc = (pred == torch.tensor(test.targets)).float().mean().item()*100
    print(f"Test Accuracy: {acc:.2f}%")
```

```
100%|██████████| 170M/170M [00:10<00:00, 16.1MB/s]
Epoch 1: Loss=1.5888
Epoch 2: Loss=1.2290
Epoch 3: Loss=1.0884
```

```
Test Accuracy: 62.68%
```