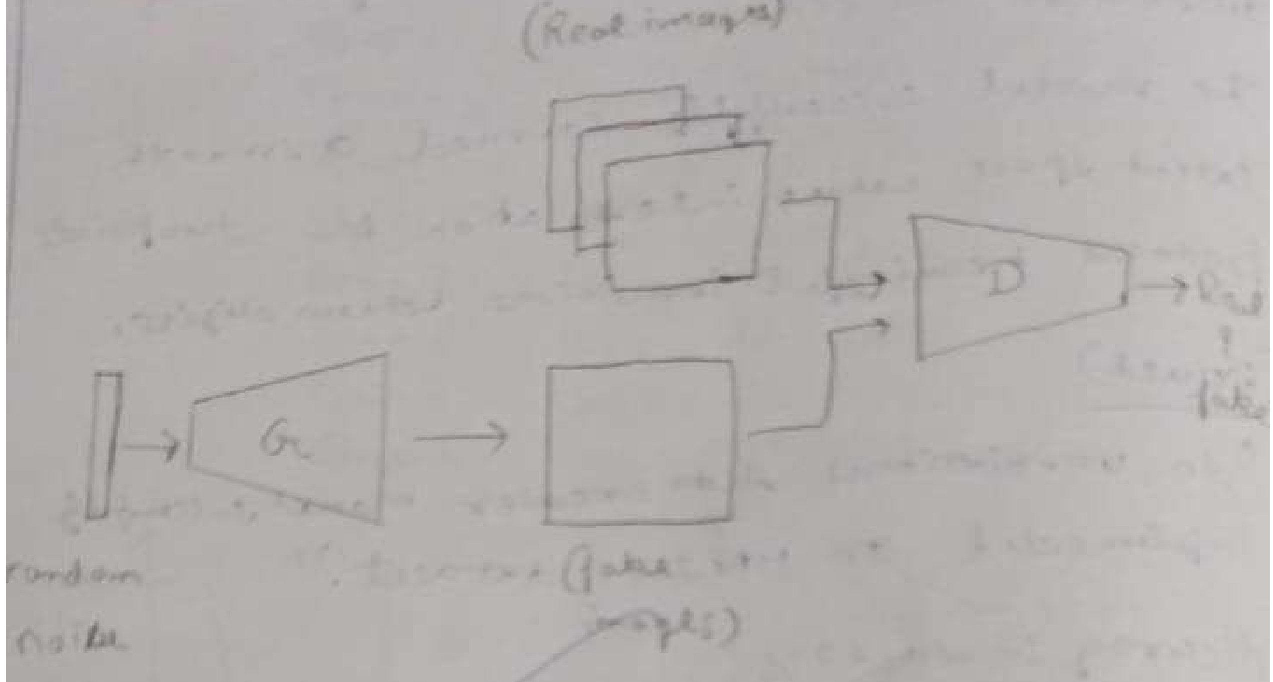
25-10-25

Lab 12:- (Implement a Deep Convolutional GAN to generate Complex Color Images)

Aim:- To impeement a Deep Convolutional Generative Adversarial Network (DCGAN) that can generate complex color images.

Pseudo code:-

- Import libraries:-
  torch, matplotlib.
- Load dataset:-
  use ~~Global~~ label dataset
  Normalize images.
- Define Generator Network:
- Define Discriminator Network
  Input; Real or fake images.
- Initialize both networks and optimizers.
  → Adam optimizer.
- Training loop.
- Visualize.
  - Display generated color images.
  - Compare evolution across epochs.


(Real images)
D
G
random
noise
(fake
images)

(Observation)

1- During the initial epochs, generated images are random noise with no structure.

2- As training progress, the Generator learns color patterns, textures, and shapes resembling real images.

3- The discriminator loss oscillates b/w G and D.

4- After sufficient training, DCGAN produces visually realistic and colorful synthetic imag

5- The quality of generated images depends on dataset complexity, network depth and training stability.

(Result)

Successfully implemented a DCGAN capable of generating complex, realistic color images.

Loss

Output:-

Epoch 1:- D: 0.6353
G: 1.4945

Epoch 2:- D: 0.3806
G: 2.1320

Epoch 3:- D: 0.1758
G: 7.0664

Epoch 4:- D: 0.0366
G: 3.1548

Epoch 5:- D: 0.0241
G: 3.8800

Epoch 6: D: 0.0158
G: 4.4111

Epoch 7: D: 0.0099
G: 4.8699

Epoch 8: D: 0.2179
G: 2.5012

Epoch 9: D: 0.0610
G: 3.7586

Epoch 10: D: 1.3041
G: 1.1274

25-10-25

Lab13:- (Understanding the Architecture of a Pre-trained Model)

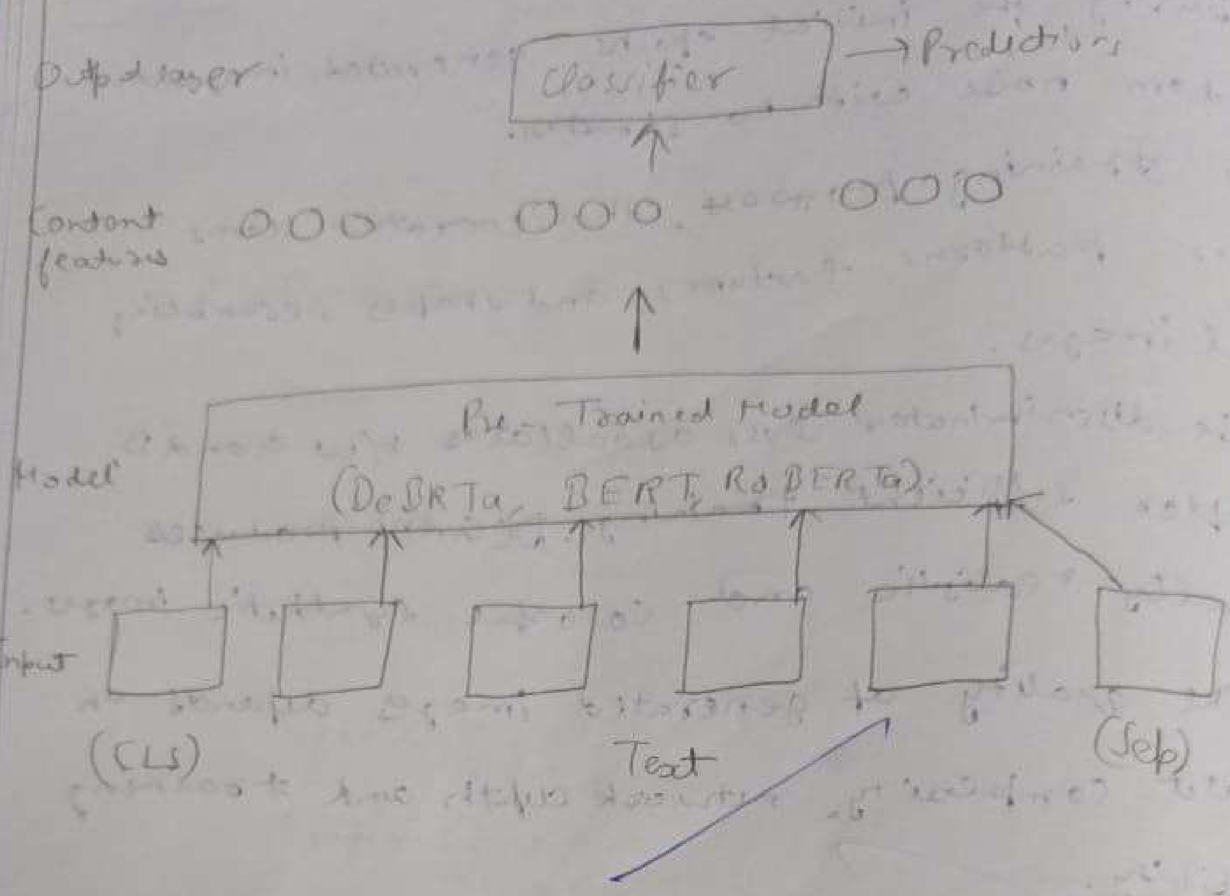
Aim:- To understand and analyze the architecture of a pre-trained deep learning model.

Pseudo code:-

- Import req. libraries.
- Load a pre-trained model from pytorch (torchvision).
- Display the full architecture of the model.
- Count total trainable and non-trainable parameters
- Visualize layer types (Conv, Pooling).
- Optionally pass a sample image through the model to verify dimensions.
- Analyze layer-by-layer flow and parameter size.

(observation)

- The VGG16 model consists of 13 convolutional layers, 3 fully connected layers, and uses ReLU after each convolution.



Output layer
Classifier
Predictions
Content features
Pre-Trained Model
(DeBERTa, BERT, RoBERTa)
Model
Input
(CLS)
Text
(Sep)

. The model ends with a softmax classifier.

. The feature extractor part includes multiple Conv + Max Pool blocks, which progressively reduce spatial dimensions.

. Total parameters are around 138 million.

. Pre-trained weights help in transfer learning.

(Result)

"The architecture and structure of the pre-trained model were successfully analyzed."

(Output)

	Top1 Accuracy	Top 5 Accuracy	Parameter
VGG16	79.0%	94.5%	22.9m

Training	Validation	Testing
86.62%	91.05%	84.97%

Untitled10.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Share

Connect T4

```python
# ------------------------------
# PART 1: Import and Configuration
# ------------------------------

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

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Hyperparameters
image_size = 32
nc = 3           # RGB
nz = 100         # latent vector size
ngf = 64         # generator feature maps
ndf = 64         # discriminator feature maps
batch_size = 128
num_epochs = 10
lr = 0.0002
beta1 = 0.5

OUT_DIR = "./dcgan_outputs"
os.makedirs(OUT_DIR, exist_ok=True)
```

Variables Terminal

Untitled10.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```python
# ------------------------------
# PART 2: Dataset (CIFAR-10)
# ------------------------------

transform = transforms.Compose([
    transforms.Resize(image_size),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5),
                         (0.5, 0.5, 0.5)),  # normalize to [-1,1]
])

dataset = datasets.CIFAR10(root="./data", download=True,
                           transform=transform)

dataloader = DataLoader(dataset, batch_size=batch_size,
                        shuffle=True, num_workers=2)
```

```python
# ------------------------------
# PART 3: Generator & Discriminator
# ------------------------------

# Weight initialization
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)

# Generator
```

Variables Terminal

Untitled10.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Connect T4

```
# -------------------------------
# PART 4: Initialization
# -------------------------------

netG = Generator(nz, ngf, nc).to(device)
netG.apply(weights_init)

netD = Discriminator(nc, ndf).to(device)
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)
real_label = 1.
fake_label = 0.
```

```
print("Training DCGAN for 10 epochs...")
for epoch in range(1, num_epochs + 1):
    for i, (data, _) in enumerate(dataloader, 0):
        # --- Train Discriminator ---
        netD.zero_grad()
        real = data.to(device)
        b_size = real.size(0)
        label = torch.full((b_size,), real_label, dtype=torch.float, device=device)
        output = netD(real)
        lossD_real = criterion(output, label)
        lossD_real.backward()
        D_x = output.mean().item()
```

Variables Terminal

Untitled10.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect T4

Share

```python
        output = netD(fake)
        lossG = criterion(output, label)
        lossG.backward()
        D_G_z2 = output.mean().item()
        optimizerG.step()

        if i % 200 == 0:
            print(f"[{epoch}/{num_epochs}] [{i}/{len(dataloader)}] "
                  f"Loss_D: {lossD.item():.4f} | Loss_G: {lossG.item():.4f}")
```

```
Training DCGAN for 10 epochs...
[1/10] [0/391] Loss_D: 1.4107 | Loss_G: 0.7604
[1/10] [200/391] Loss_D: 0.6353 | Loss_G: 1.4345
[2/10] [0/391] Loss_D: 0.2630 | Loss_G: 2.3100
[2/10] [200/391] Loss_D: 0.3006 | Loss_G: 2.2320
[3/10] [0/391] Loss_D: 0.1905 | Loss_G: 3.1562
[3/10] [200/391] Loss_D: 0.2758 | Loss_G: 2.0664
[4/10] [0/391] Loss_D: 0.0755 | Loss_G: 3.0738
[4/10] [200/391] Loss_D: 0.0566 | Loss_G: 3.1548
[5/10] [0/391] Loss_D: 0.0464 | Loss_G: 3.5390
[5/10] [200/391] Loss_D: 0.0258 | Loss_G: 3.8800
[6/10] [0/391] Loss_D: 0.0195 | Loss_G: 4.1255
[6/10] [200/391] Loss_D: 0.0158 | Loss_G: 4.4111
[7/10] [0/391] Loss_D: 0.0149 | Loss_G: 4.6282
[7/10] [200/391] Loss_D: 0.0099 | Loss_G: 4.8699
[8/10] [0/391] Loss_D: 0.0081 | Loss_G: 5.0301
[8/10] [200/391] Loss_D: 0.2179 | Loss_G: 2.5012
[9/10] [0/391] Loss_D: 0.0809 | Loss_G: 3.2313
[9/10] [200/391] Loss_D: 0.0610 | Loss_G: 3.7586
[10/10] [0/391] Loss_D: 0.2054 | Loss_G: 2.7939
[10/10] [200/391] Loss_D: 1.5041 | Loss_G: 1.1274
```

Variables Terminal

Untitled11.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```python
import torch
import torch.nn as nn
import torchvision.models as models
from collections import OrderedDict
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```python
import torch
from torchvision import models
from torchvision.models import VGG16_Weights, ResNet50_Weights

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model_name = "vgg16"  # or "resnet50"

if model_name.lower() == "vgg16":
    model = models.vgg16(weights=VGG16_Weights.DEFAULT).to(device)
elif model_name.lower() == "resnet50":
    model = models.resnet50(weights=ResNet50_Weights.DEFAULT).to(device)
else:
    raise ValueError("Supported: 'vgg16' or 'resnet50'")

model.eval()
print(f"\n✅ Loaded model: {model_name.upper()} on {device}")
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
100%|██████████| 528M/528M [00:03<00:00, 146MB/s]
```

Variables Terminal

```python
import torch
from torchvision import models
from torchvision.models import VGG16_Weights, ResNet50_Weights

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model_name = "resnet50"  # or "resnet50"

if model_name.lower() == "vgg16":
    model = models.vgg16(weights=VGG16_Weights.DEFAULT).to(device)
elif model_name.lower() == "resnet50":
    model = models.resnet50(weights=ResNet50_Weights.DEFAULT).to(device)
else:
    raise ValueError("Supported: 'vgg16' or 'resnet50'")

model.eval()
print(f"\n✅ Loaded model: {model_name.upper()} on {device}")
```

```
Downloading: "https://download.pytorch.org/models/resnet50-11ad3fa6.pth" to /root/.cache/torch/hub/checkpoints/resnet50-11ad3fa6.pth
100%|██████████| 97.8M/97.8M [00:00<00:00, 144MB/s]

✅ Loaded model: RESNET50 on cuda
```