**Assignment Task:**

Design and Implement a Generative Adversarial Network (GAN) to Generate New Images.

**Objective:**

The objective of this project is to create and deploy a sophisticated deep learning model capable of generating new images belonging to a specific category, such as "paintings," "cars," or "animals," utilizing the power of a Generative Adversarial Network (GAN).

**Requirements:**

**Dataset:**

**Data Preprocessing:**

Before feeding the data into the GAN model, it is crucial to perform essential preprocessing steps to enhance model performance. This involves activities like image resizing, normalization to bring pixel values within a consistent range, and data augmentation to diversify the dataset.

Code:

dataset = dset.ImageFolder(root=dataroot, transform=transforms.Compose([ transforms.Resize(image\_size),

transforms.CenterCrop(image\_size),

transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)), ]))

Explaination:

1. transforms.Resize(image\_size): This transformation resizes the input image to a specified size. In the code, image\_size is a variable that defines the target size for resizing the image. This step ensures that all images in the dataset have the same dimensions, which is required for neural network input.
2. transforms.CenterCrop(image\_size): This transformation crops the center of the image to the specified size. If the original image's dimensions are larger than image\_size, this step helps extract a square region from the center of the image. This is important for maintaining the aspect ratio and relevant features of the image.
3. transforms.ToTensor(): This transformation converts the image from its original format (JPG image) to a PyTorch tensor. Neural networks in PyTorch work with tensors as input data. This step also normalizes the pixel values of the image from the range [0, 255] to the range [0, 1].
4. transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)): This transformation normalizes the tensor image by subtracting the mean and dividing by the standard deviation. The provided means and standard deviations (in tuples) are used to normalize each channel of the image. In this case, the code is performing a normalization that transforms the pixel values from the [0, 1] range to the [-1, 1] range. This normalization is used to center the data and stabilize the learning process of neural networks.

**Model Design:**

DCGANs (Deep Convolutional Generative Adversarial Networks) are a specialized form of GAN architecture designed for stable and effective training. They employ deep convolutional neural networks (CNNs) for both the generator and discriminator models. DCGANs address challenges like mode collapse and vanishing gradients, ensuring stable convergence. By utilizing hierarchical features and spatial invariance, DCGANs excel in generating high-resolution images with intricate details. Their flexibility extends to various applications, including image synthesis, translation, and transfer learning, underpinning their pivotal role in advancing generative modeling.

The GAN will consist of two key components: a **Generator** and a **Discriminator**.

**Generator:**

Generator(

(main): Sequential(

(0): ConvTranspose2d(100, 512, kernel\_size=(4, 4), stride=(1, 1), bias=False)

(1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): ConvTranspose2d(512, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(5): ReLU(inplace=True)

(6): ConvTranspose2d(256, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(8): ReLU(inplace=True)

(9): ConvTranspose2d(128, 64, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(11): ReLU(inplace=True)

(12): ConvTranspose2d(64, 3, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(13): Tanh()

)

)

**Discriminator:**

The Discriminator acts as a critic, discerning between real and generated images. Similar to the Generator, it comprises multiple layers. Its design is equally significant to promote effective adversarial training.

Discriminator(

(main): Sequential(

(0): Conv2d(3, 64, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(1): LeakyReLU(negative\_slope=0.2, inplace=True)

(2): Conv2d(64, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(4): LeakyReLU(negative\_slope=0.2, inplace=True)

(5): Conv2d(128, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(7): LeakyReLU(negative\_slope=0.2, inplace=True)

(8): Conv2d(256, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(10): LeakyReLU(negative\_slope=0.2, inplace=True)

(11): Conv2d(512, 1, kernel\_size=(4, 4), stride=(1, 1), bias=False)

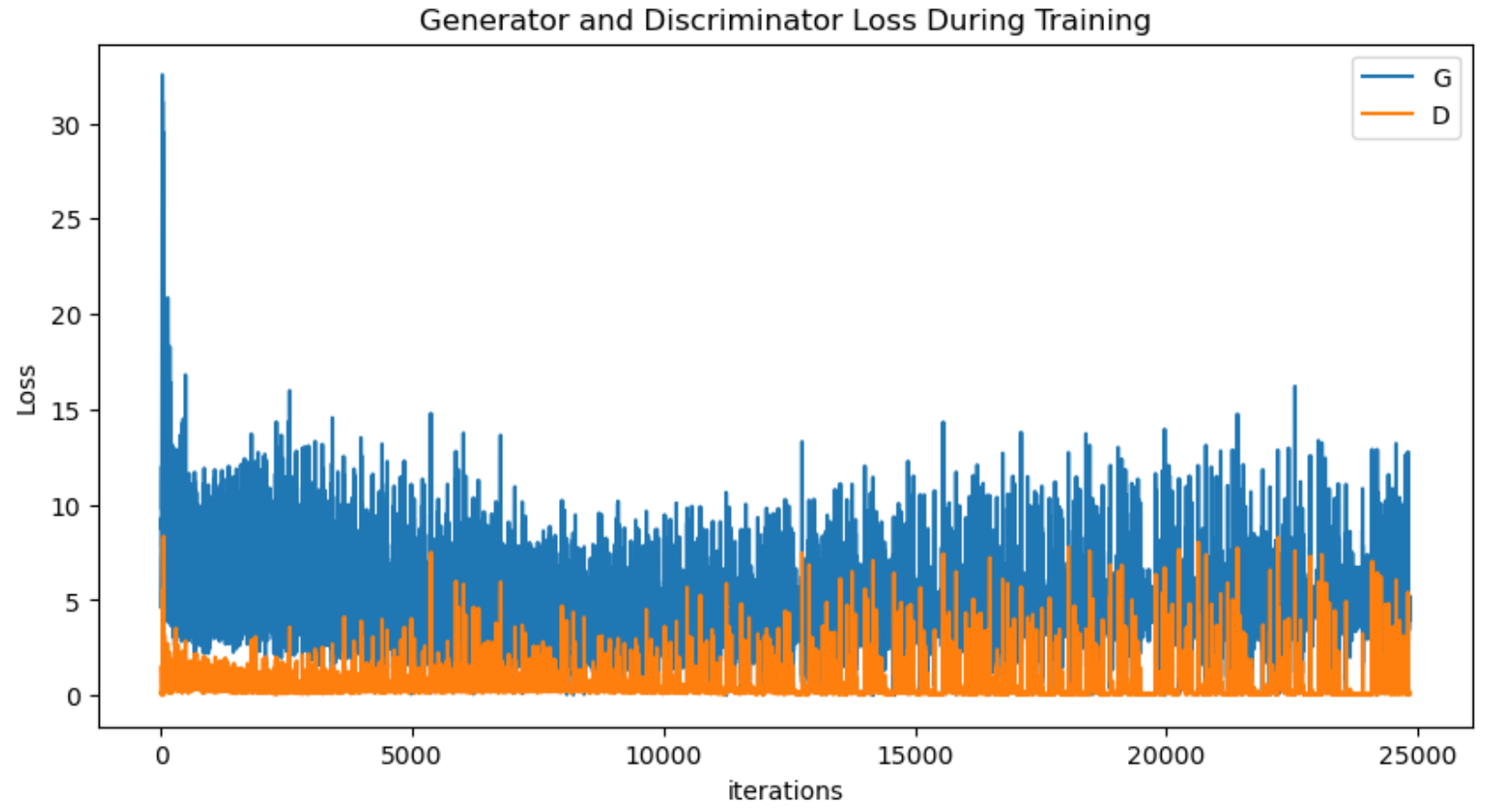
(12): Sigmoid()

)

)

**Model Training:**

The GAN model will be trained using the prepared dataset. The core of GAN training is the interplay between the Generator and Discriminator. While the specifics of training can be complex, crucial aspects include:

* **Loss Function**: GANs involve a binary classification problem, where the discriminator aims to distinguish between real and fake samples. BCELoss is well-suited for binary classification tasks, making it a natural choice for GANs.
* **Optimizer**: Adam combines the concepts of momentum and RMSProp, making it effective in handling sparse gradients and noisy data. This is beneficial in training GANs, which can encounter such challenges
* **Learning Rate** . A learning rate that is too high can lead to rapid convergence, but it might cause instability, oscillations, or even divergence in the training process. On the other hand, a learning rate that is too low can slow down convergence. A value around 0.0002 strikes a balance between these factors, promoting relatively stable and gradual convergence.The training process will be documented meticulously, including the chosen parameters and the reasoning behind each selection.
* To monitor progress, a plot depicting the changes in Generator and Discriminator losses over time is provided.

**Model Evaluation:**

The quality of generated images is assessed quantitatively and qualitatively.

**Quantitative Evaluation:**

Quantitative metrics like the **Frechet Inception Distance (FID)** and **Inception Score (IS)** are computed. FID measures the similarity between real and generated images, while IS quantifies the diversity and quality of generated images.

FID is 110.8811 is good for a DCGAN.

**Qualitative Evaluation:**

Visual inspection remains indispensable. A selection of generated images will be presented for qualitative assessment. This visual evaluation provides insight into the GAN's performance and its ability to capture intricate details.

Real Fake

**Model Deployment:**

The final model, is deployed to a cloud-based service. This deployment will enable real-time image generation, ensuring seamless accessibility and utilization. Microsoft Azure is considered for this purpose. Azure app platform is used and github repository is used to deploy.The app.py is uses streamlit to build the webapp.