**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:**

The foundation of our image generation lies in the GAN architecture. For this purpose, we will design a GAN using a powerful deep learning framework such as TensorFlow or PyTorch. The GAN will consist of two key components: a **Generator** and a **Discriminator**.

**Generator:**

class Generator(nn.Module):

def \_\_init\_\_(self, ngpu):

super(Generator, self)

\_\_init\_\_():

self.ngpu = ngpu self.main = nn.Sequential( # input is Z, going into a convolution nn.ConvTranspose2d( nz, ngf \* 8, 4, 1, 0, bias=False), nn.BatchNorm2d(ngf \* 8), nn.ReLU(True), # state size. (ngf\*8) x 4 x 4 nn.ConvTranspose2d(ngf \* 8, ngf \* 4, 4, 2, 1, bias=False), nn.BatchNorm2d(ngf \* 4), nn.ReLU(True), # state size. (ngf\*4) x 8 x 8 nn.ConvTranspose2d( ngf \* 4, ngf \* 2, 4, 2, 1, bias=False), nn.BatchNorm2d(ngf \* 2), nn.ReLU(True), # state size. (ngf\*2) x 16 x 16 nn.ConvTranspose2d( ngf \* 2, ngf, 4, 2, 1, bias=False), nn.BatchNorm2d(ngf), nn.ReLU(True), # state size. (ngf) x 32 x 32 nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False), nn.Tanh() # state size. (nc) x 64 x 64 ) def forward(self, input): return self.main(input) **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.

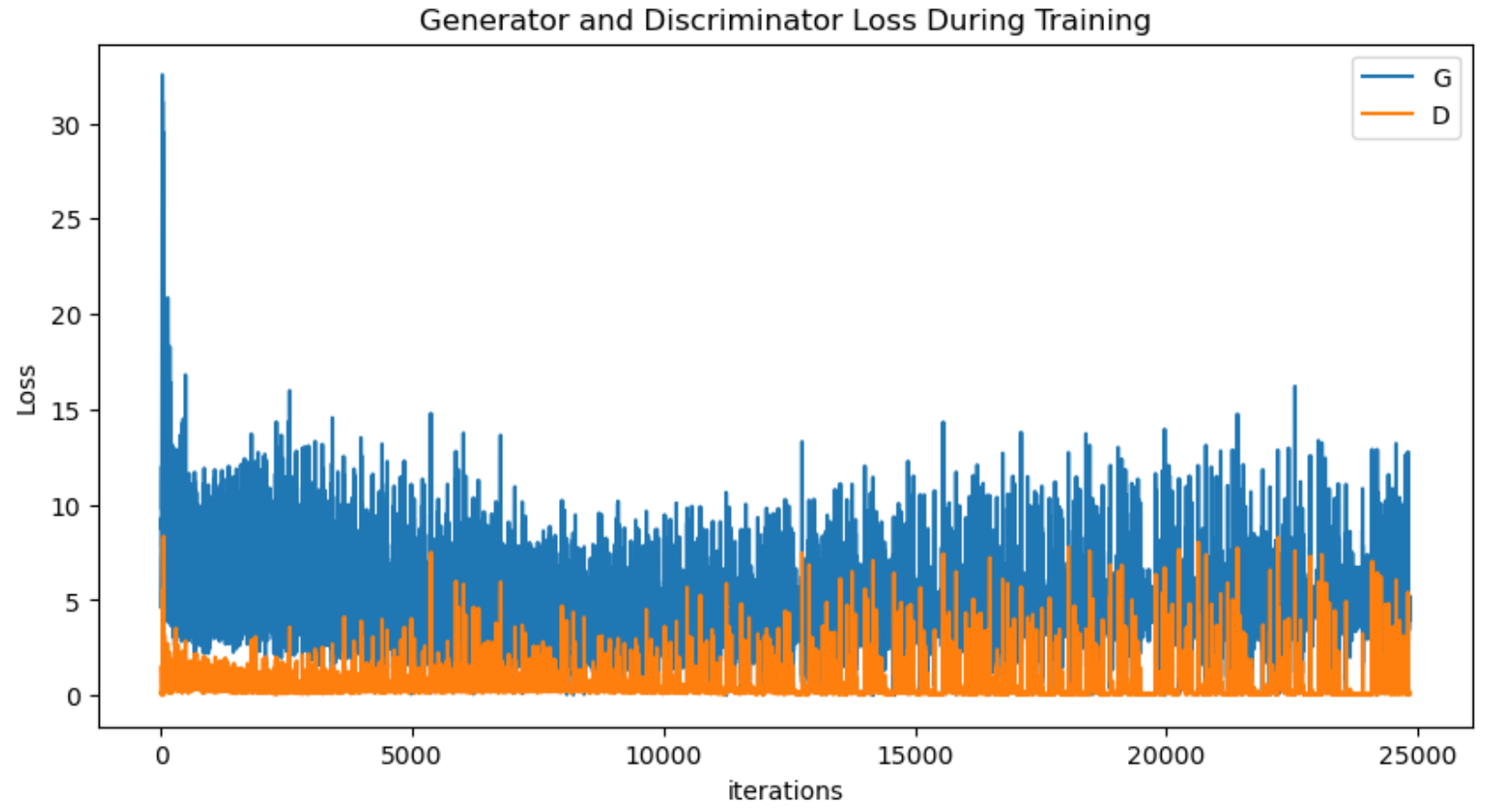
**Model Training:**

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**: Selection of an appropriate loss function that captures the GAN's objectives, often involving terms like binary cross-entropy.
* **Optimizer**: Choice of an optimizer, such as Adam, RMSprop, or SGD, to drive the model's weight updates.
* **Learning Rate**: Determination of an optimal learning rate, ensuring stable and effective training.

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 will be provided.

**Model Evaluation:**

The quality of generated images will be assessed quantitatively and qualitatively. 

**Quantitative Evaluation:**

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

**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, once trained and evaluated, will be deployed to a cloud-based service. This deployment will enable real-time image generation via an API, ensuring seamless accessibility and utilization. Prominent cloud platforms like AWS, Google Cloud, or Microsoft Azure will be considered for this purpose.