**Image Super-Resolution Using GANs (Generative Adversarial Networks)**

This repository implements an image super-resolution model using a GAN-based architecture. The model takes low-resolution images as input and generates high-resolution images. It uses a Generator-Discriminator architecture inspired by the concept of Generative Adversarial Networks (GANs), along with VGG-based perceptual loss for better quality images. The dataset is prepared by having paired low-resolution and high-resolution images.

**Requirements**

The following libraries are required for the implementation:

* torch
* torchvision
* torchmetrics
* PIL
* matplotlib
* tqdm
* opencv-python
* numpy

You can install the required libraries using the following command:

pip install -r requirements.txt

Or install dependencies individually using:

pip install torch torchvision torchmetrics Pillow matplotlib tqdm opencv-python numpy

**Dataset**

The dataset is structured as follows:

dataset/

├── train/

│ ├── low\_res/

│ └── high\_res/

└── val/

├── low\_res/

└── high\_res/

* **low\_res/**: Directory containing low-resolution images.
* **high\_res/**: Directory containing the corresponding high-resolution images.

Ensure that your images are placed in the respective directories (train/low\_res, train/high\_res, val/low\_res, val/high\_res).

**Architecture**

**Generator**

The generator uses a deep architecture consisting of several residual blocks, each block containing convolutional layers, batch normalization, and PReLU activations. The generator also includes upsampling blocks that help in increasing the resolution of the input low-resolution image.

**Discriminator**

The discriminator is a convolutional network used to classify real and fake images. It employs several convolutional layers with LeakyReLU activations to progressively downsample the image.

**Loss Functions**

* **Adversarial Loss**: Used to train the discriminator and generator.
* **Content Loss (MSE)**: Measures pixel-wise difference between the generated and ground-truth high-resolution images.
* **VGG Loss**: A perceptual loss computed using the VGG19 pre-trained model to measure the perceptual quality of the images.

**Metrics**

* **PSNR (Peak Signal-to-Noise Ratio)**: Measures the quality of the generated images compared to the ground truth.
* **SSIM (Structural Similarity Index Measure)**: Measures the structural similarity between the generated and ground truth images.

**Training the Model**

To train the model, follow these steps:

1. Mount Google Drive (if using Google Colab):
2. from google.colab import drive
3. drive.mount('/content/drive')
4. Set the paths to your dataset:
5. train\_dataset = ImageDataset(root\_dir="/content/drive/MyDrive/dataset/train")
6. val\_dataset = ImageDataset(root\_dir="/content/drive/MyDrive/dataset/val")
7. Initialize the Generator and Discriminator models:
8. generator = Generator().to(DEVICE)
9. discriminator = Discriminator().to(DEVICE)
10. Call the training function:
11. train\_model(train\_loader, val\_loader, generator, discriminator, EPOCHS)

The training loop will run for the specified number of epochs (EPOCHS), and every 10 epochs, a model checkpoint will be saved.

**Saving and Loading Checkpoints**

The model checkpoints are saved every 10 epochs. You can load the saved checkpoints as follows:

checkpoint = torch.load('checkpoint\_epoch\_10.pth')

generator.load\_state\_dict(checkpoint['generator\_state\_dict'])

discriminator.load\_state\_dict(checkpoint['discriminator\_state\_dict'])

optimizer\_G.load\_state\_dict(checkpoint['optimizer\_G\_state\_dict'])

optimizer\_D.load\_state\_dict(checkpoint['optimizer\_D\_state\_dict'])

**Results**

At the end of each epoch, the model will output the following metrics:

* **Generator Loss**
* **Discriminator Loss**
* **PSNR**
* **SSIM**

**Challenges**

While implementing the Image Super-Resolution model, several challenges were encountered and addressed:

1. **Training Stability**: GANs are notoriously difficult to train, with issues such as mode collapse or the generator producing unrealistic images.
   * **Solution**: Careful tuning of hyperparameters, especially learning rates and batch sizes, helped mitigate these issues. Additionally, using a pre-trained VGG network for perceptual loss improved training stability.
2. **Quality of Output**: Initially, the model generated blurry images due to insufficient learning capacity in the Generator.
   * **Solution**: Increased the depth of the Generator and used residual blocks to improve feature learning. Enhanced image quality with the inclusion of perceptual loss based on VGG19.
3. **Training Time**: Training the model from scratch took significant time, especially when the dataset size increased.
   * **Solution**: Utilizing a powerful GPU for training significantly reduced the time. Checkpoints were saved regularly to resume training without losing progress.
4. **Balancing Generator and Discriminator**: A common issue with GANs is the imbalance between the generator and discriminator during training, where one network becomes too powerful compared to the other.
   * **Solution**: This was handled by adjusting the loss weights for the generator and discriminator, ensuring they both improved at a similar rate.
5. **Evaluation Metrics**: Evaluating the quality of generated high-resolution images can be challenging, especially for perceptual quality.
   * **Solution**: Using both PSNR and SSIM along with perceptual loss gave a more comprehensive measure of image quality.
6. **Overfitting**: With a smaller dataset, there was a risk of the model overfitting, especially on the training data.
   * **Solution**: Data augmentation techniques such as flipping, rotating, and cropping were used to introduce more variety into the training dataset.

**Notes**

* Ensure you have access to a GPU for faster training, as this model can take a significant amount of time to train on a CPU.
* You can modify the batch size, learning rate, number of epochs, and other hyperparameters in the configuration section at the start of the notebook.

**License**

This code is available under the MIT License.