**DCGAN Image Generation**

In this assignment, we will implement a Deep Convolutional Generative Adversarial Network (DCGAN) using TensorFlow and Keras. We will train the DCGAN on the CelebA dataset to generate realistic images of celebrities.

**Introduction**

This report outlines the development and training of a DCGAN using the CelebA dataset. The goal was to generate realistic human faces using a generative adversarial network. The assignment involved several key steps: dataset preparation, model architecture definition, training, image generation, evaluation, and experimentation with hyperparameters and architecture modifications.

**1. Dataset Preparation**

The CelebA dataset, a collection of celebrity faces, was used. The dataset was preprocessed to suit the requirements of a GAN. The preprocessing steps included:

- Resizing images to 64x64 pixels.

- Normalizing pixel values to the range [0, 1].

- Creating a TensorFlow dataset for efficient loading and processing.

**2. DCGAN Architecture**

The DCGAN comprised two main components: the generator and the discriminator, both designed following the principles of deep convolutional networks.

**Generator:**

* A sequential model converting latent space vectors into 64x64x3 RGB images.
* A deep network with multiple deconvolutional layers.
* Used batch normalization and LeakyReLU activations.
* Output layer with a sigmoid activation function to generate images.

**Discriminator:**

* A sequential model distinguishing real images from fake ones generated by the generator.
* A convolutional neural network.
* Used LeakyReLU activations, dropout layers, and a dense output layer.
* Output a single scalar representing the probability of an input being real or fake.

**3. Training the DCGAN**

The training involved alternating between updating the discriminator and the generator. Specific steps included:

- Using Adam optimizer with a learning rate of 1e-4.

- Employing binary cross-entropy loss functions for both networks.

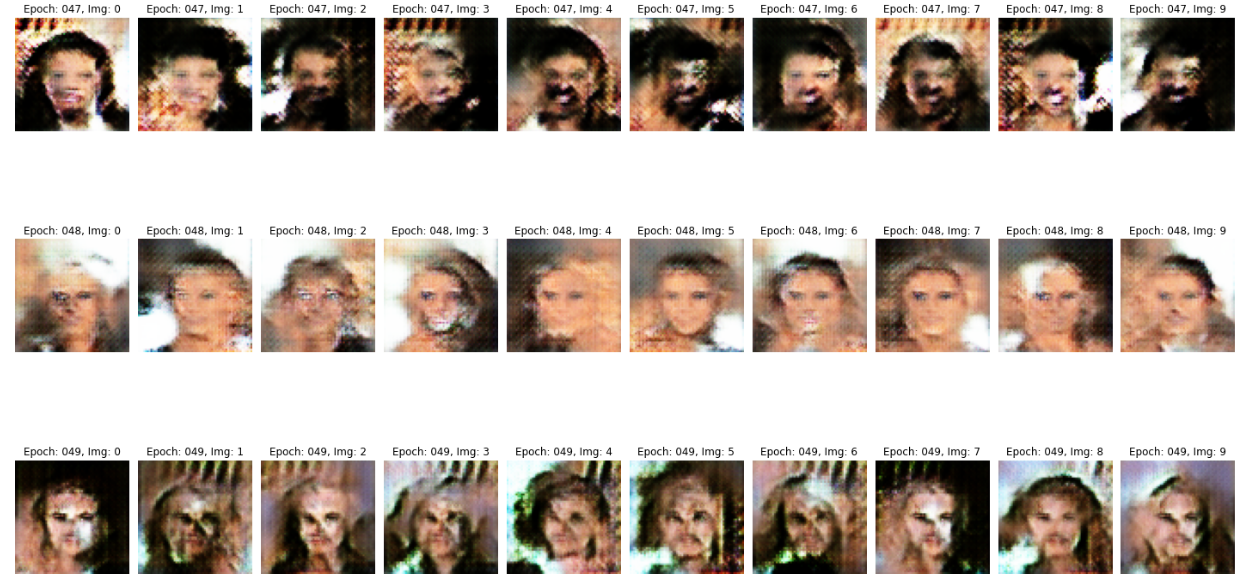
Challenges faced during training included mode collapse and training instability, which were mitigated through hyperparameter tuning and architecture adjustments.

**Implementation of Callbacks**

A custom callback, **GANMonitor**, was implemented to save images generated by the GAN at the end of each training epoch, providing a visual insight into the GAN's learning progress.

**4. Generating Images**

After training, the generator was used to create images. The generated images were visually inspected to assess the model's performance. The below images are generated at 50 Epochs..



**5. Evaluation**

The quality of generated images was evaluated through both subjective visual inspection and objective metrics like Inception Score (IS) and Fréchet Inception Distance (FID). The generated faces showed a reasonable level of realism but also exhibited room for improvement in terms of diversity and clarity.

The images generated at 50 Epochs are very blurry and unrefined, while those at 100 Epochs exhibited improvement from previously generated images. There is still a room for improvement by adjusting the hyperparameters.

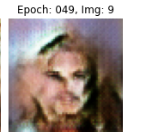
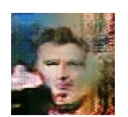
 

Fig1: 50 Epochs Fig2: 100 Epochs

**6. Additional Tasks and Experiments**

To improve image generation quality, several experiments were conducted:

**Hyperparameter Tuning:** Adjustments in learning rates, batch sizes, and noise dimensions were explored.

**Architectural Modifications:** Experimented with adding layers and changing layer types in the generator and discriminator.

**Training Duration**: Extended the number of training epochs from 50 to 100 to observe long-term trends in learning and generation quality.

**Image Interpolation:** Implemented interpolation between two generated images to demonstrate the smoothness and continuity of the latent space.

****

**Fig: Image Interpolation for 50 Epochs**

****

**Fig: Image Interpolation for 100 Epochs**

**Challenges**

* **Balancing Discriminator and Generator**: Ensuring neither the discriminator nor the generator becomes too powerful too quickly is crucial for stable training.
* **Mode Collapse**: A common issue where the generator produces limited varieties of outputs.
* **Hyperparameter Tuning**: Finding the right balance for learning rates, batch sizes, and network architectures required several iterations.

**Conclusion**

Developing a DCGAN to generate realistic images is a complex task that requires careful balancing of multiple components. The assignment highlighted the sensitivity of GANs to architecture and hyperparameter choices. While the results showed promise, they also indicated areas for improvement, specifically in enhancing image diversity and stability during training.

Future work could focus on refining the model architecture, exploring more advanced GAN variants, and employing additional techniques for stabilizing GAN training.