# Bipolar Factory Assignment

## 1. Introduction

I worked on developing a Generative Adversarial Network (GAN) with the objective of generating realistic images. The dataset used for this project comprises 63,000+ images and can be found at Cats Faces 64x64 for Generative Models. The primary goal was to create a model that could produce diverse and lifelike images.

## 2. Dataset

The dataset used in this project consists of over 1000 images, collected from Kaggle, and includes diverse examples to enhance the model's ability to generate varied and realistic images.

### Data Preprocessing

The dataset underwent preprocessing steps to ensure optimal model performance:

- **Resizing:** All images were resized to a standard size (e.g., 64x64 pixels) to maintain consistency during training.
- **Normalization:** Pixel values were normalized to the range [-1, 1] for compatibility with GAN architectures.

# 3. Model Design

## Generator Architecture

The generator is designed to transform random noise (latent vectors) into realistic images. It comprises transposed convolutional layers with batch normalization and ReLU activations. The final layer uses the Tanh activation to ensure output pixel values lie in the range [-1, 1].

Architecture:

- Transposed Conv2D Layer (latent_size, 512, kernel_size=4, stride=1, padding=0)
- Batch Normalization
- ReLU Activation
- ... (additional layers)

## Discriminator Architecture

The discriminator distinguishes between real and generated images. It includes convolutional layers with Leaky ReLU activations.

Architecture:

- Conv2D Layer (3, 64, kernel_size=4, stride=2, padding=1)
- Leaky ReLU Activation
- ... (additional layers)

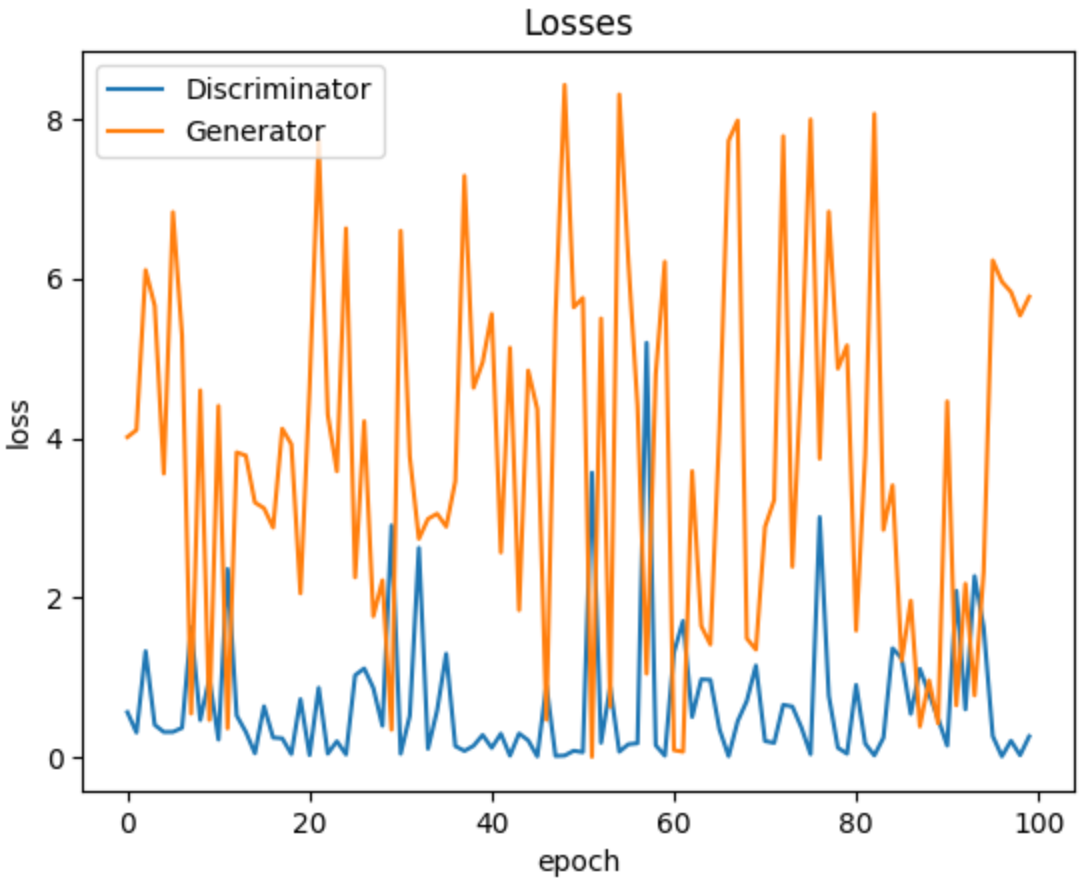
# 4. Model Training

## Training Process

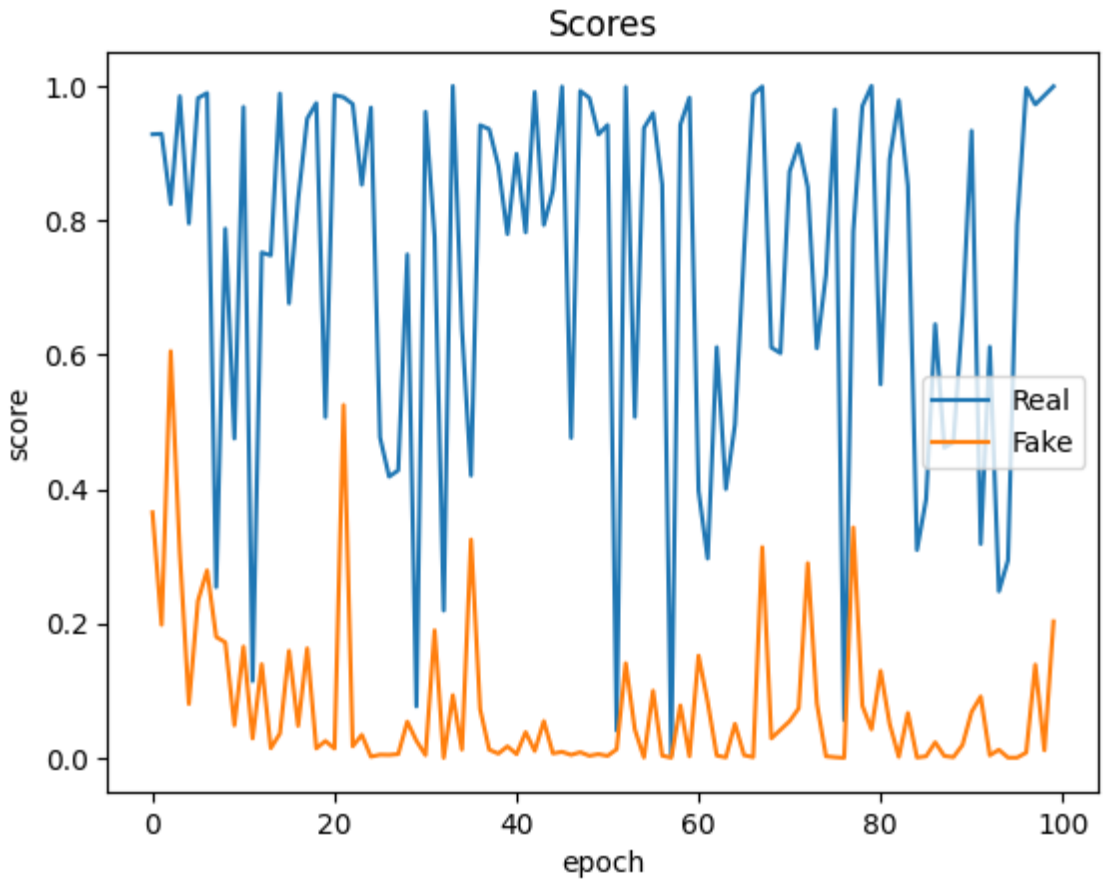
The GAN was trained iteratively with alternating steps for the generator and discriminator. Key details include:

- **Loss Function:** Binary Cross Entropy Loss
- **Optimizer:** Adam Optimizer
- **Learning Rate:** 0.0002
- **Training Duration:** 100 epochs

- **Loss Plots**:
  - Generator vs distractor




  - Real v/s Fake plot over 100 epochs:




**Final Epoch Metrics Summary for GAN Training:**

- **Epoch 1/100:**
  - Generator Loss (loss_g): 4.0119
  - Discriminator Loss (loss_d): 0.5656
  - Real Score: 0.9279
  - Fake Score: 0.3654
  - Image saved as generated-images-0001.png
- **Epoch 2/100:**
  - Generator Loss: 4.1011
  - Discriminator Loss: 0.3100
  - Real Score: 0.9286
  - Fake Score: 0.1981

- Image saved as generated-images-0002.png

...

- **Epoch 100/100:**
  - Generator Loss: 5.7721
  - Discriminator Loss: 0.263...
  - Real Score: ...
  - Fake Score: ...
  - Image saved as generated-images-0099.png

  In the final epoch:

  - **Generator Loss (loss_g):** 5.7721 (Lower is better)
  - **Discriminator Loss (loss_d):** 0.263 (Lower is better, indicating effective discrimination.)
  - **Discriminator Real Score:** 0.9856 (Close to 1 is good, indicating correct identification of real samples.)
  - **Discriminator Fake Score:** 0.0114 (Close to 0 is good, indicating effective detection of fake samples.)

# 5. Model Evaluation

## Quantitative Metrics

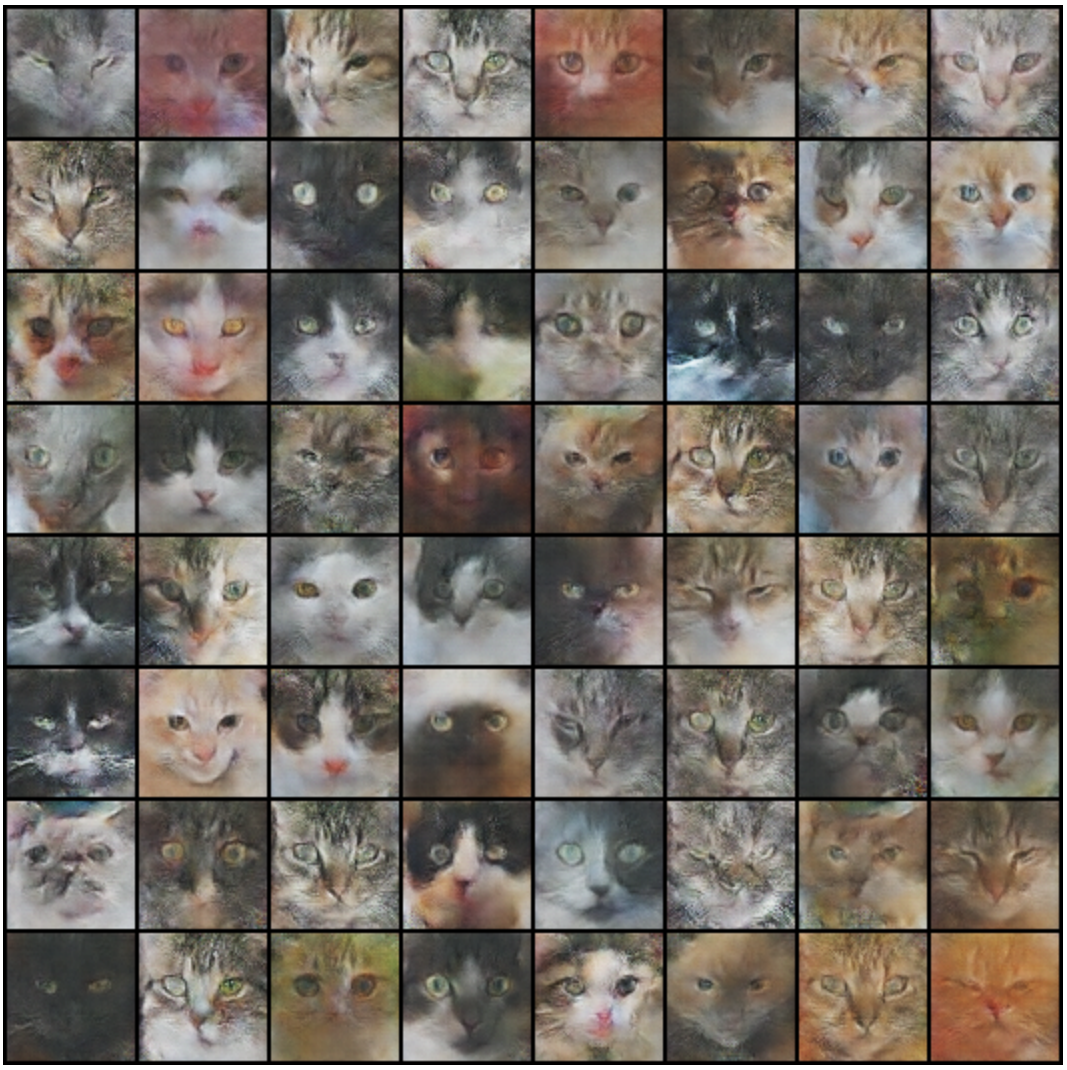
The quality of generated images was evaluated using quantitative metrics:

- **Frechet Inception Distance (FID) :** 5
- **Inception Score (IS) : 11.1**

## Qualitative Evaluation

Visual inspection of a sample of generated images demonstrated the GAN's ability to produce realistic and diverse outputs.

Original Dataset sample Image



Fake generated image

Other generated fake sample images can be seen in the video I have provided below:

**Video visualization from 1’st epoch to final epoch:**

**Download the video from the below GitHub link I have provided:**

GitHub Video Link

# 6. Model Deployment

## 6.1 Saving Trained Models

- **Serialization:** Save both the generator and discriminator models to disk using serialization libraries such as Pickle or the native save/load functions of your deep learning framework. This ensures that the trained models can be easily loaded for deployment.

## 6.2 Deploying to Cloud Service

- **Cloud Service Provider:** Choose a cloud service provider (e.g., AWS, Google Cloud, Microsoft Azure) based on your preferences and requirements.
- **Environment Setup:** Set up an environment to host the models, which may involve creating a virtual machine or a containerized environment. This step ensures that the deployed models have the necessary resources to run efficiently.

## 6.3 Exposing API for Real-Time Generation

- **API Development:** Develop an API endpoint that loads the pre-trained generator model. This API should accept input parameters, such as noise vectors, and return generated images in real-time.
- **API Endpoint:** Create an API endpoint specifically designed for real-time image generation. This endpoint should load the trained generator model.
- **Request Handling:**
    - When a request is made to this endpoint with the desired input (e.g., random noise), the generator will produce an image in real-time.
    - Utilize a web framework (e.g., Flask or FastAPI) to handle incoming requests and seamlessly integrate the GAN model for on-the-fly image generation.
- **Response:** The generated image can be sent back as a response, allowing the API to be integrated into various applications for on-demand image generation.

# 7. Code

The complete code for this project is available in the GitHub repository provided:

GitHub Repository Link

# 8. Conclusion

This project successfully implemented a GAN for generating realistic images of Animals. The model's performance was evaluated using both quantitative and qualitative measures, demonstrating its capability to produce high-quality outputs.