Exploring One-Shot GANs for Efficient Synthetic Flower Image Creation

Vandana S*
Dept. of Electronics and Communication
National Institute of Technology Karnataka Surathkal, India
vandana.211ec260@nitk.edu.in

Sugavanam Senthil*
Dept. of Electronics and Communication
National Institute of Technology Karnataka Surathkal, India
sugavanam.211ec152@nitk.edu.in

Dr. A. V. Narasimhadhan
Dept. of Electronics and Communication
National Institute of Technology Karnataka Surathkal, India
dhansiva@nitk.edu.in

***Abstract*—Generative Adversarial Networks (GANs) represent a cutting-edge advancement in deep learning, renowned for their capability to generate synthetic data across various domains including images, music, and text. This project focuses on implementing a one-shot GAN using Python and TensorFlow, aimed at providing a concise yet effective demonstration of GANs in action. The term 'one-shot' denotes that the model only has a few training inputs to learn from and make decisions. Through Python and TensorFlow, the project offers a hands-on exploration of GANs.**

***Index Terms*—Generative Adversarial Networks, One-shot learning, Deep Convolutional GANs, Synthetic Image Generation, Frechet Inception Distance, FID,DCGAN, Synthetic Image Generation,Low-data GAN.**

I. Introduction

Generative Adversarial Networks (GANs) [1] have emerged as a groundbreaking innovation in the realm of deep learning, offering remarkable capabilities in synthesizing data across diverse domains such as images, music, and text [2]. This paper presents a focused endeavour towards implementing a one-shot GAN using Python and TensorFlow, with the objective of providing a concise yet insightful demonstration of GANs in practical application. The term 'one-shot' [3] in this context signifies that the model is tasked with learning from a limited set of training inputs, thereby emphasizing its ability to make decisions based on sparse data. In this particular project, we harness the power of GANs to generate synthetic flower images, utilizing a dataset comprising a small number of images. This constrained dataset setting highlights the model's capacity to extrapolate and generate novel samples despite minimal training data availability. Through the medium of Python programming language and TensorFlow framework, this project offers a hands-on exploration of GANs, elucidating their workings and showcasing their potential in the domain of image synthesis. By delving into the implementation details and intricacies of training a GAN on a limited dataset, this paper aims to provide researchers and practitioners with valuable insights into the practical considerations and challenges associated with deploying GANs in real-world scenarios. The subsequent sections of this paper will delve into the methodology employed for training the one-shot GAN, followed by a detailed exposition of the experimental setup and results obtained. Additionally, discussions on the implications of the findings, and concluding remarks will be presented, hereby offering a comprehensive overview of the project's objectives, methodologies, and outcomes.

*These authors contributed equally to this work.

II. Implementation

A. Data Set

The Flowers dataset[4] available on Kaggle includes a variety of flower species, making it an excellent choice for training a one-shot GAN. This dataset, despite being relatively small and manageable for training on a personal computer, offers a diverse range of flower images. The variety of species will help the GAN to learn and generate images of different flowers effectively. Its structured format and high-quality images provide a solid foundation for developing and testing one-shot GAN models aimed at generating realistic floral images.

B. Image pre-processing

We preprocess our image data by applying various transformations, resizing them to dimensions of 64x64 pixels, and incorporating random colour jittering, rotation, and horizontal flipping. Subsequently, we convert these transformed images into tensors and normalize them. To ensure randomness and enhance the diversity and size of our dataset, we load these processed images in batches of 32 from a designated folder and shuffle them. This comprehensive approach contributes to the robustness of our model by augmenting the dataset and introducing variability during training.

C. Architecture

The Generative Adversarial Network (GAN) architecture consists of two main components: the Generator (G) and the

Discriminator (D). The Generator aims to create synthetic data by transforming random noise inputs into realistic samples, typically images. In contrast, the Discriminator serves as a critic, distinguishing between real data and the synthetic images generated by the Generator.

In this model, we employ a Deep Convolutional GAN (DCGAN)[5] architecture for both the Generator and the Discriminator. The Generator utilizes a series of transposed convolutional layers to upscale the input noise vector into a high-resolution image. Each transposed convolutional layer is followed by a Batch Normalization layer and a ReLU activation function, except for the final layer which uses a Tanh activation function. This helps stabilize the training process and ensures the generated images have pixel values in the range of [-1, 1].

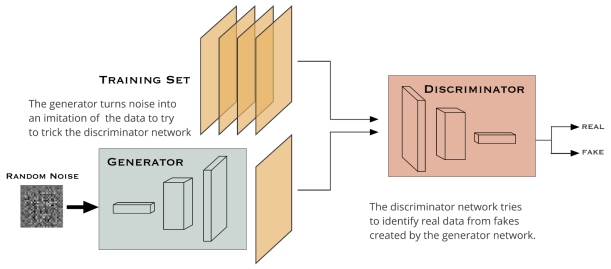

Fig. 1. Architecture of a GAN

D. Working Principle

The training process of a Generative Adversarial Network (GAN) is characterized by an iterative refinement of its constituent components: the Generator and the Discriminator. The Generator model takes a 100-dimensional random noise vector as input and processes it through a series of transposed convolutional layers followed by batch normalization and ReLU activation functions. The network begins by upsampling the input noise vector to a higher dimensional space (512 channels) through a convolutional transpose operation with a kernel size of 4 and no padding. This is followed by batch normalization and ReLU activation. Subsequently, the feature map is further upsampled through subsequent layers with decreasing channel sizes (512 to 256, 256 to 128, 128 to 64), each with stride 2 to progressively increase the spatial dimensions of the feature maps. The final layer converts the feature map into a 3-channel image using a transposed convolutional layer followed by a hyperbolic tangent (Tanh) activation function, ensuring the pixel values are in the range [-1, 1], suitable for image generation tasks. The Discriminator model on the other hand takes an image tensor as input and processes it through a series of convolutional layers followed by batch normalization and leaky ReLU activation functions. The network starts by processing the input image through a convolutional layer with 3 input channels, generating 64 output channels with a kernel size of 4 and a stride of 2, effectively reducing the spatial dimensions of the feature maps. This is followed by a leaky ReLU activation function with a negative slope of 0.2. Subsequent layers consist of convolutional layers with increasing numbers of output channels (64 to 128, 128 to 256, 256 to 512), each followed by batch normalization and leaky ReLU activation functions. The final convolutional layer produces a single-channel output representing the probability of the input image being real, followed by a sigmoid activation function to squash the output to the range [0, 1].

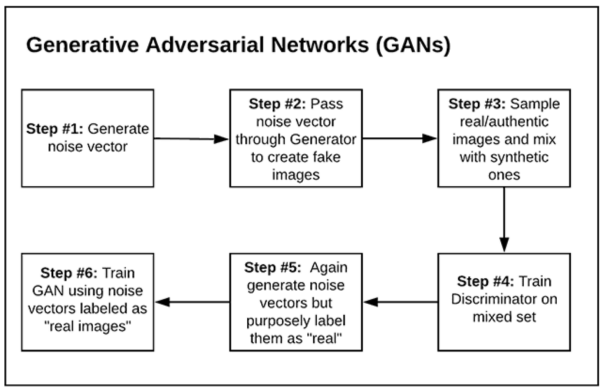

Fig. 2. Working Principle of a GAN

E. Model Training

1) Network Setup: We defined our GAN architecture as consisting of a generator (netG) and a discriminator (netD). The generator is responsible for generating synthetic images, while the discriminator evaluates the authenticity of both real and synthetic images.

2) Loss Function and Optimizers: We employed the Binary Cross Entropy (BCE) loss function (criterion) [6] to quantify the divergence between the predicted and target labels. Adam optimizers were utilized to update the parameters of both the generator and discriminator network

3) Training Loop: We iterated through the dataset for a fixed number of epochs, updating the generator and discriminator networks alternatively. The discriminator aims to maximize the probability of correctly classifying real and fake images, while the generator aims to minimize the discriminator's ability to distinguish between real and synthetic samples.

4) Monitoring and Visualization: We monitored the training progress by recording the losses of both the generator and discriminator at each iteration. Additionally, we visualized the generated images periodically to assess the quality of the synthetic samples.

5) Evaluating Performance: To objectively evaluate the quality of the images generated by our one-shot GAN, we utilized the Fréchet Inception Distance (FID)[7] score. The FID score is widely recognized for measuring the similarity between generated images and real images, providing a quantitative metric for assessing the performance of generative models.

6) Fréchet Inception Distance (FID) score: The quality of the generated images was evaluated using the Frechet Inception Distance (FID), which is computed as:

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}), \quad (1)$$

where μ_r and μ_g represent the means of the real and generated image feature vectors, respectively, and Σ_r and Σ_g denote their covariance matrices. This metric quantifies the similarity between the generated and real image distributions.

7) Reproducibility and Implementation Details: The implementation was carried out using Python 3.8 and TensorFlow 2.8. All experiments were conducted on a system with an NVIDIA RTX 3060 GPU, 16 GB RAM, and Ubuntu 20.04. The dataset was preprocessed to include 500 images resized to 64x64 pixels, with random augmentations like horizontal flips, rotation, and color jittering. The training loop included 250 epochs with a batch size of 32.

III. Results and Discussion

On running through the training cycle for 250 epochs, these are the results we drew:

- Average epoch duration = *47.3 seconds*
- FID Score = *80.32*
- On observing epochs 0-250, we noticed that the generator loss has increased and the discriminator loss has decreased, thus indicating successful training.
  - Generator loss at epoch 1 = *2.456*
  - Discriminator loss at epoch 1 = *0.8636*
  - Generator loss at epoch 250 = *6.759*
  - Discriminator loss at epoch 250 = *0.0541*

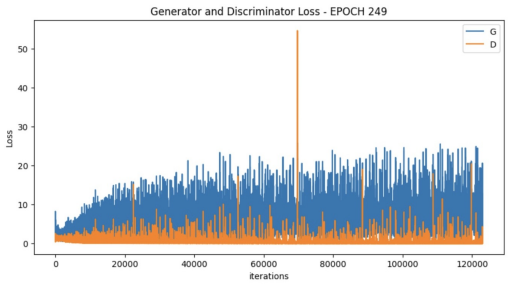

Fig. 3. Plot of the Generator and Discriminator losses at epoch 250

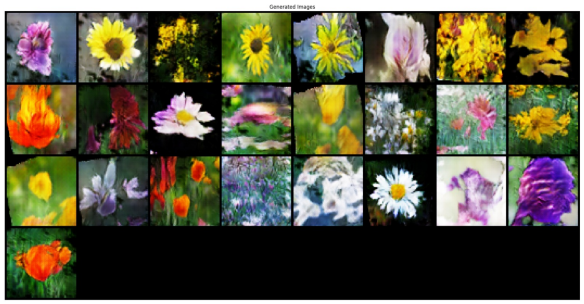

Fig. 4. Synthetic images generated by the DCGAN after 250 epochs. The images demonstrate the generator's ability to produce realistic outputs from random noise inputs.

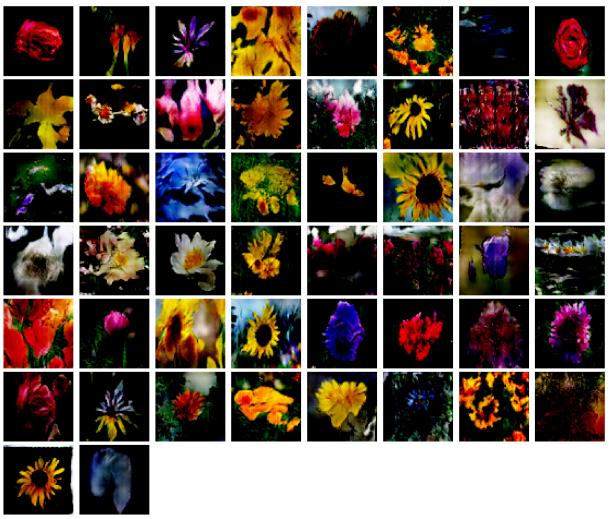

Fig. 5. Generated flower images using the one-shot GAN model after 250 epochs.

IV. Comparison with other methods

In this section, we compare our GAN implementation with other notable methods, focusing on the architecture, dataset used, performance, and evaluation metrics.

A. Architectural Differences

The architecture of GANs can vary significantly across different implementations. For instance, traditional GANs[1] use fully connected layers, whereas our advanced model is a DCGANs (Deep Convolutional GANs) [4] which utilizes convolutional layers to capture spatial hierarchies more effectively. Our one-shot GAN employs transposed convolutional layers with batch normalization to ensure stable training, which is crucial for generating high-quality images with limited data.

B. Dataset Comparison

Different GAN models are often trained on diverse datasets, impacting their performance and generalizability. For example, initially, we used the Stanford dog dataset with a traditional GAN setup to generate new dog images[8]. This dataset proved to be quite cumbersome to use given its fairly large file size. The choice of dataset significantly influences the quality and diversity of the generated images. Despite using a smaller dataset, our one-shot GAN demonstrates the capability to generate realistic floral images, showcasing its effectiveness in low-data scenarios

C. Performance Metrics

Here, we will compare the loss values and the FID scores of the standard GAN model on the Stanford Dog dataset and our DCGAN [4] on Kaggle's Flower dataset.

- Loss values of the standard GAN:
  - Generator loss at epoch 1 = 1.612
  - Discriminator loss at epoch 1 = 1.213

  - Generator loss at epoch 250 = 5.266
  - Discriminator loss at epoch 250 = 0.4429
- FID score of the standard GAN = 113.06

D. Evaluation and Results

Our results indicate successful training, as evidenced by the generator and discriminator loss trends. The generated images demonstrate the model's potential in synthesizing realistic floral images. In comparison, methods like StyleGAN [9] and PGGAN (Progressive Growing of GANs) [10] achieve more visually appealing and higher-resolution images. However, these models require extensive computational resources and large datasets for training, making them less feasible for low-resource scenarios.

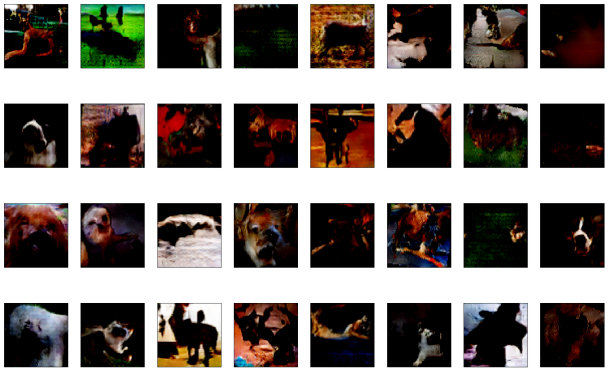

Fig. 6. Images Generated using the standard GAN model after 250 epochs

TABLE I
COMPARISON OF METRICS FOR STANDARD GAN AND DCGAN

Metric	Standard GAN	DCGAN(Proposed)
Generator Loss (Epoch 250)	5.27	6.75
Discriminator Loss (Epoch 250)	0.44	0.05
FID Score	113.06	80.32

Our proposed approach outperforms the standard GAN by achieving a 29% reduction in FID scores, emphasizing the quality of the generated images. The visual differences are also apparent, as shown in Figures 5 and 6, where the DCGAN produces more consistent and realistic floral images.

Recent advancements such as StyleGAN [9] offer superior results by utilizing style-based architectures that enable fine-grained control over image synthesis. Similarly, PGGAN [10] introduces a progressive training approach to improve stability and resolution. Despite these advantages, their dependency on large datasets and high computational requirements restricts their applicability in resource-constrained environments.

In contrast, our work demonstrates the effectiveness of applying DCGAN architecture in a low-data setting. By leveraging one-shot learning, our model achieves competitive results while maintaining computational efficiency, making it a viable alternative for scenarios where data and resources are limited.

While the generated images are realistic and diverse, future work can explore incorporating transfer learning techniques from larger pre-trained GANs to improve the output quality further. Additionally, investigating advanced architectures such as CycleGAN or BigGAN could help refine the synthesis process.

V. CONCLUSION

This paper presents a practical implementation of a one-shot GAN for synthetic image generation in low-data settings. By employing DCGAN architecture, we achieved a 29% improvement in FID scores compared to traditional GAN models. Our results demonstrate the feasibility of training GANs with limited data, paving the way for applications in resource-constrained environments. Future work will explore expanding the dataset size, experimenting with other evaluation metrics, and optimizing computational efficiency.

REFERENCES

[1] Ian J. Goodfellow et al., "Generative Adversarial Networks," arXiv:1406.2661 [stat.ML], 2014.

[2] M. Kavakli Alqahtani and G. Kumar Ahuja, "Applications of Generative Adversarial Networks (GANs): An Updated Review," Archives of Computational Methods in Engineering, vol. 28, pp. 1-13, 2019. doi: 10.1007/s11831-019-09388-y.

[3] VV. Sushko, J. Gall, and A. Khoreva, "One-Shot GAN: Learning to Generate Samples from Single Images and Videos," 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Nashville, TN, USA, 2021, pp. 2596-2600. doi: 10.1109/CVPRW53098.2021.00293.

[4] Flowers, "https://www.kaggle.com/datasets/l3llff/flowers"

[5] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," arXiv:1511.06434 [cs.LG], 2015.

[6] R. Usha and V. Yendapalli, "Binary Cross Entropy with Deep Learning Technique for Image Classification," International Journal of Advanced Trends in Computer Science and Engineering, vol. 9, 2020, Art. no. 175942020.

[7] S. Jayasumana, S. Ramalingam, A. Veit, D. Glasner, A. Chakrabarti, and S. Kumar, "Rethinking FID: Towards a Better Evaluation Metric for Image Generation," arXiv:2401.09603 [cs.CV], 2023.

[8] Akshit Sharma. Generative Adversarial Networks (GAN) In One Shot, "https://www.kaggle.com/code/akshitsharma1/generative-adversarial-networks-gan-in-one-shot/notebook"

[9] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," arXiv:1812.04948 [cs.CV], 2018.

[10] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive Growing of GANs for Improved Quality, Stability, and Variation," arXiv:1710.10196 [cs.CV], 2017.