Generative Adversarial Networks (GAN) In One Shot

Sreehari Krishnan
Dept. of Electronics and Communication
National Institute of Technology Karnataka Surathkal, India
sreeharikrishnan.211ec150@nitk.edu.in

Divyangana
Dept. of Electronics and Communication
National Institute of Technology Karnataka Surathkal, India
divayangana.211ec216@nitk.edu.in

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

***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 the implementation of 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 few training inputs to learn from and take decisions on. Through Python and TensorFlow, the project offers a hands-on exploration of GANs.**

I. Introduction

Generative Adversarial Networks (GANs) [2] 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[4]. This paper presents a focused endeavor 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” [1] 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 dog images, utilizing a dataset comprising of 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 show- casing 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, thereby offering a comprehensive overview of the project's objectives, methodologies, and outcomes.

II. Implementation

A. Data Set

We will be using Stanford Dogs dataset which contains images of 120 breeds of dogs from all over the world. The Stanford Dogs dataset is a good choice for training a GAN to generate images of dogs. The dataset is relatively small, which means that it can be trained on a personal computer. The dataset also includes a variety of dog breeds, which will help the GAN to learn to generate images of a wide range of dogs.

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. In order 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

A Generative Adversarial Network comprises two primary components: the Generator (G) and the Discriminator (D). The Generator is tasked with creating synthetic data by transforming random noise inputs into realistic samples, often images. On the other hand, the Discriminator acts as a critic, discerning between real data and the synthetic images produced by the Generator. In this case, we modify the architecture of the

Generator and Discriminator in order to help in one shot learning. This adversarial relationship between the Generator and Discriminator forms the crux of the GAN architecture, driving the training process towards generating increasingly convincing synthetic data.

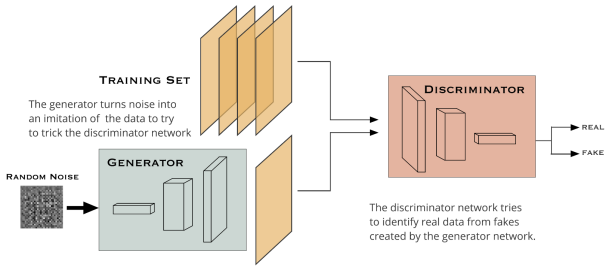

Fig. 1. Architecture of a GAN

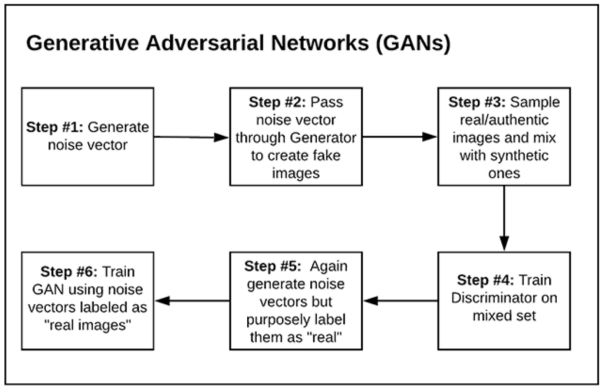

Fig. 2. Working Principle of a GAN

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 convolu- tional 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].

E. Model Tranining

1) Network Setup: We defined our GAN architecture 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) [3] 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

III. Results and Discussion

On running through the training cycle for 100 EPOCHS these are the results we drew:

[1] Average EPOCH duration = *107.023 seconds*

[2] On observing EPOCH 0-99, we noticed that the generator loss has increased and the discriminator loss has decreased, thus indicating successful training.
Generator loss at EPOCH 1 = *1.5102*
Discriminator loss at EPOCH 1 =*1.9724*
Generator loss at EPOCH 99 = *1.4238*
Discriminator loss at EPOCH 99 = *9.8650*

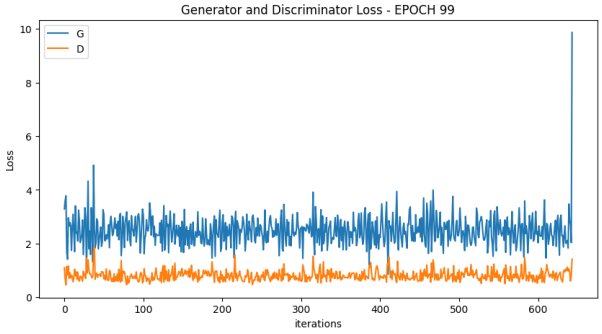

Fig. 3. Plot of Generator and Discriminator losses

Fig. 4. Generated Images

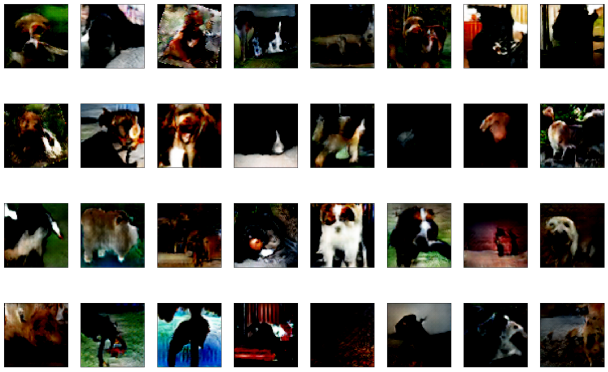

Fig. 5. Images from each class

REFERENCES

[1] 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. keywords: Training;Visualization;Computer vision;Conferences;Computational modeling;Layout;Tools,

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

[3] 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.

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