Generated Adversarial Network - ECE 285

Mingxuan Li
ECE
A69027684

Abstract

In this project, we aim to implement and analyze the performance of Generative Adversarial Networks (GANs) for generating realistic images. The project includes the design and implementation of the GAN architecture, training on a chosen dataset, and evaluating the quality of the generated images. Additionally, we will explore various enhancements to the basic GAN model, such as Deep Convolutional GAN (DCGAN), to improve stability and performance. We will also test our models on different datasets, starting with simple ones like MNIST and getting deeper with CIFAR-10 and CIFAR-100.

1 Introduction

GANs have emerged as a groundbreaking technique in image generation. The primary motivation behind this project is to harness the power of GANs to generate realistic images that can be used in various applications, such as enhancing training datasets, creating artistic content, and simulating scenarios for research purposes. The ability to generate high-quality images holds immense potential for industries ranging from entertainment to scientific research.

The problem we aim to solve involves training a GAN to generate images that are indistinguishable from real ones. This challenge is non-trivial due to the adversarial nature of GANs. The objective is for the generator to become adept at producing images that the discriminator cannot distinguish from real images, leading to the generation of realistic and high-quality images.

The generator and discriminator have conflicting goals, which can lead to unstable training dynamics. If the discriminator becomes too strong, the generator receives little useful feedback and fails to improve which we tend to face when training embodied by unusual loss increasement and decreasement from two networks. Achieving a balance where both networks improve simultaneously is a key challenge in GAN training.

To address this problem, our approach involves several strategic components. Initially, we implement a basic GAN architecture, comprising a generator with transposed convolutional layers and a discriminator with convolutional layers. Enhancing the basic GAN, we explore advanced techniques such as Deep Convolutional GAN (DCGAN). DCGAN leverages deep convolutional layers and batch normalization to stabilize training and improve the quality of generated images.

Our implementation includes rigorous training protocols and the application of techniques such as label smoothing and gradient penalty to maintain balance between the generator and discriminator. We also experiment with different learning rates and batch sizes to optimize training dynamics. Regular visualization of generated images and quantitative metrics like Inception Score and Frechet Inception Distance are used to evaluate the performance of our models.

The results of our experiments are promising. The basic GAN model demonstrates the ability to generate recognizable images after several epochs of training. With the implementation of DCGAN, we observe significant improvements in the quality and diversity of the generated images. The advanced model not only produce more realistic images but also exhibit greater stability during training.

In summary, this project highlights the potential of GANs in generating high-quality images and the effectiveness of advanced techniques in improving GAN performance.

2 Related Work

Generative Adversarial Networks, introduced by Goodfellow et al. in 2014, have revolutionized the field of generative modeling by employing a game-theoretic approach. GANs consist of two neural networks, the generator and the discriminator, which are trained simultaneously through adversarial processes. The generator aims to produce realistic data, while the discriminator attempts to distinguish between real and fake data. This dynamic interaction drives both networks to improve, resulting in the generation of highly realistic synthetic data. Since their inception, GANs have spurred extensive research and numerous variants aimed at enhancing stability, convergence, and performance.

One notable extension of GANs is the Deep Convolutional GAN (DCGAN), proposed by Radford et al. in 2015. DCGANs integrate convolutional layers into both the generator and discriminator, leveraging the power of convolutional neural networks to capture spatial hierarchies in image data. This architecture has significantly improved the quality of generated images and has become a foundational model for subsequent GAN research. The use of batch normalization and the ReLU activation function in DCGANs has also contributed to more stable training and better performance.

Conditional GANs (cGANs), introduced by Mirza and Osindero in 2014, extend the original GAN framework by conditioning both the generator and discriminator on additional information, such as class labels or data from other modalities. This conditioning enables cGANs to generate data that adheres to specific attributes, making them particularly useful for tasks like image-to-image translation and data augmentation. cGANs have demonstrated remarkable success in generating high-quality, class-specific images and have inspired various domain-specific applications.

The Wasserstein GAN (WGAN), proposed by Arjovsky et al. in 2017, addresses some of the instability issues associated with the original GANs. WGAN replaces the traditional binary cross-entropy loss with the Wasserstein distance, providing a more meaningful gradient for the generator. This modification leads to more stable training dynamics and mitigates issues such as mode collapse. WGAN-GP (with Gradient Penalty), an extension by Gulrajani et al., further enhances WGAN by introducing a gradient penalty term to enforce the Lipschitz constraint, improving both stability and convergence.

In this project, we mainly focus on GAN and DCGAN.

3 Method

In this section, we describe the methodology employed to develop and evaluate a GAN for generating MNIST and CIFAR images. The architecture includes a generator and a discriminator, both of which are designed to perform their roles in the adversarial training process effectively. The generator aims to produce realistic images from random noise, while the discriminator's goal is to distinguish between real images and those generated by the generator. We use basic GAN and DCGAN architecture, enhanced with additional layers to improve the generator's capacity and performance.

3.1 GAN

3.1.1 Generator Architecture

Layers Description:

1. **Input Layer**: The generator receives a 100-dimensional noise vector drawn from a standard normal distribution.
2. **First Fully Connected Layer**: This layer has 256 units followed by a ReLU activation function.
3. **Second Fully Connected Layer**: This layer has 512 units followed by a ReLU activation function.
4. **Third Fully Connected Layer**: This layer has 1024 units followed by a ReLU activation function.

5. **Output Layer**: The final layer has the same number of units as the dimensionality of the output data (e.g., 28x28=784 for MNIST), followed by a Tanh activation function to normalize the output to the range [-1, 1].

The visualized architecture is as shown in figure 1.

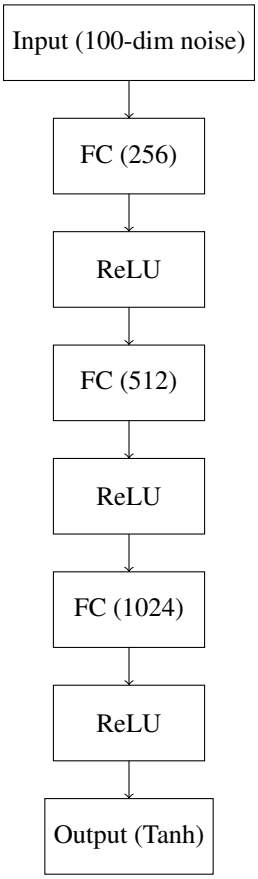

Figure 1: GAN Generator Architecture

3.1.2 Discriminator Architecture

Layers Description:

1. **Input Layer**: The discriminator receives an input data sample (e.g., a flattened 28x28 image for MNIST, resulting in a 784-dimensional input).
2. **First Fully Connected Layer**: This layer has 1024 units followed by a Leaky ReLU activation function.
3. **Second Fully Connected Layer**: This layer has 512 units followed by a Leaky ReLU activation function.
4. **Third Fully Connected Layer**: This layer has 256 units followed by a Leaky ReLU activation function.
5. **Output Layer**: The final layer has a single unit followed by a Sigmoid activation function to output the probability that the input is real.

The visualized architecture is as shown in figure 2.

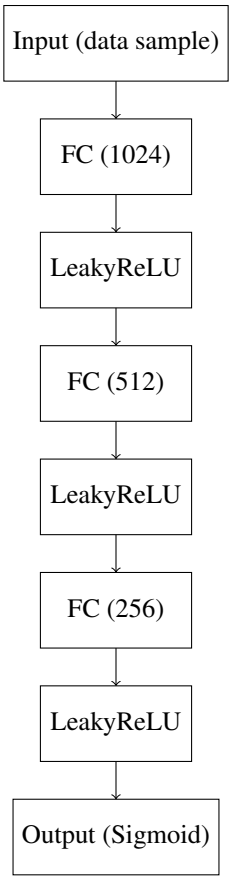

Figure 2: GAN Discriminator Architecture

3.2 DCGAN

3.2.1 Generator Architecture

Layers Description:

1. **Input Layer**: The generator receives a 100-dimensional noise vector drawn from a standard normal distribution.
2. **First Transposed Convolutional Layer**: This layer has 512 filters, a kernel size of 4, a stride of 1, and no padding. It outputs a feature map of size 4x4.
3. **Second Transposed Convolutional Layer**: This layer has 256 filters, a kernel size of 4, a stride of 2, and padding of 1, resulting in an 8x8 feature map.
4. **Third Transposed Convolutional Layer**: This layer uses 128 filters with a kernel size of 4, a stride of 2, and padding of 1, producing a 16x16 feature map.
5. **Fourth Transposed Convolutional Layer**: The final layer has channels filters (1 for MNIST), a kernel size of 5, a stride of 3, and padding of 1, leading to a 28x28 output.

The visualized architecture is as shown in figure 3.

Each 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 to normalize the output to the range [-1, 1].

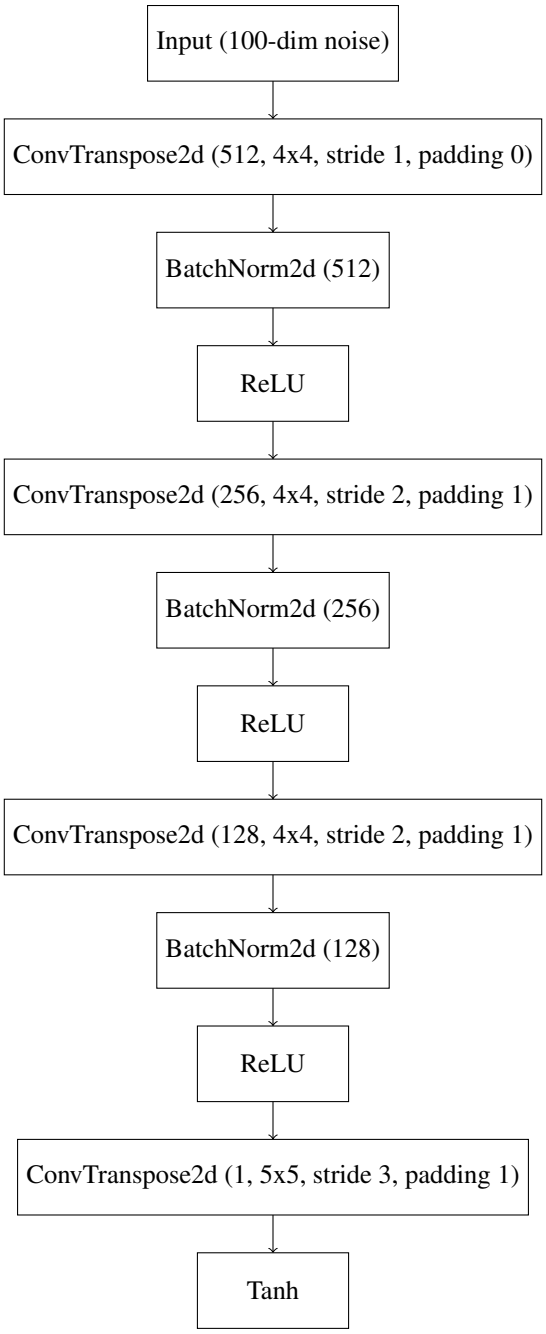

Figure 3: DCGAN Generator Architecture

3.2.2 Discriminator

The DCGAN discriminator is a convolutional neural network that classifies input images as real or fake. It consists of several convolutional layers with Batch Normalization and Leaky ReLU activation functions, followed by a final fully connected layer with a Sigmoid activation function to output a probability score.

Layers Description:

1. **First Convolutional Layer**: This layer uses 64 filters with a kernel size of 4, stride of 2, and padding of 1. Leaky ReLU activation is applied for non-linearity.

2. **Second Convolutional Layer**: This layer uses 128 filters with a kernel size of 4, stride of 2, and padding of 1, followed by Batch Normalization and Leaky ReLU.
3. **Third Convolutional Layer**: This layer uses 256 filters with a kernel size of 4, stride of 2, and padding of 1, followed by Batch Normalization and Leaky ReLU.
4. **Fourth Convolutional Layer**: This layer uses 512 filters with a kernel size of 4, stride of 2, and padding of 1, followed by Batch Normalization and Leaky ReLU.
5. **Fully Connected Layer**: This layer flattens the input and passes it through a fully connected layer with a Sigmoid activation function to output the probability that the input image is real.

The visualized architecture is as shown in figure 4.

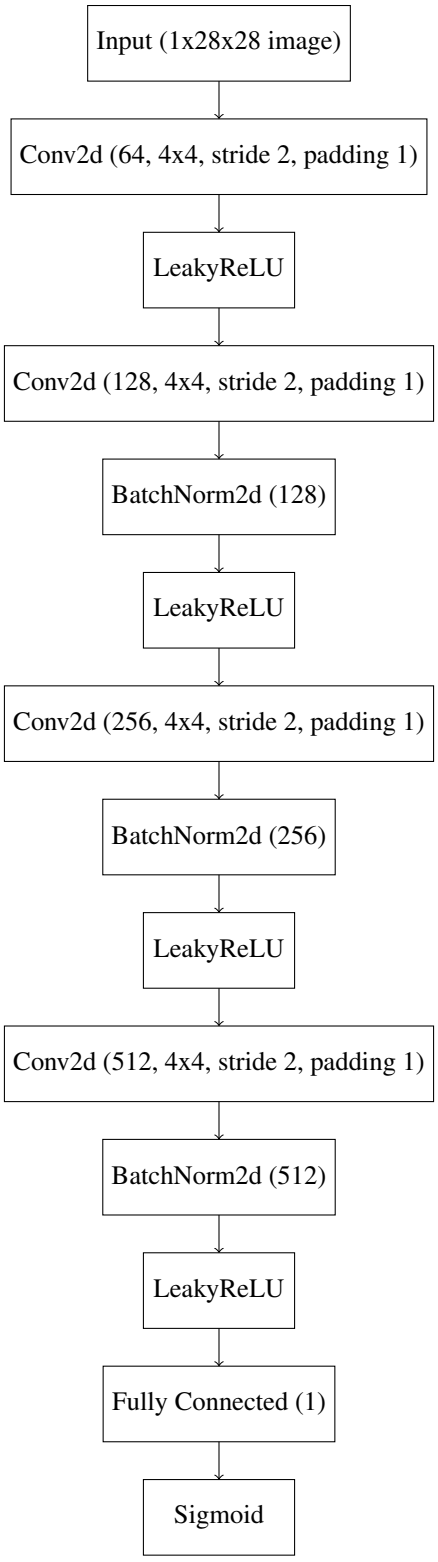

Figure 4: DCGAN Discriminator Architecture

3.3 Training algorithm

The training algorithm involves alternating between training the discriminator and the generator.

Initialize the models, optimizers, and loss function: Both models are initialized, and Adam optimizers are used for both the generator and discriminator. Binary Cross Entropy Loss is used for the loss function.

For each epoch, The discriminator is trained on a batch of real images and a batch of generated images. Each batch compares with its own labels and calculate the loss. The discriminator loss takes both the real and fake images loss. The generator is used to generate fake images based on random noise and then apply the images to the discriminator. After the discriminator is trained, we feed the fake images into discriminator and compares it with the real labels to generate a generator loss. Finally, generated images are periodically visualized, and losses are recorded to monitor training progress.

4 Experiments

4.1 Experimental Setup

Our experiments aim to evaluate the performance of the GAN and DCGAN models on three different datasets: MNIST, CIFAR-10, and CIFAR-100. The goal is to assess the quality of generated images, convergence behavior, and the impact of network depth on performance.

4.1.1 Datasets

- **MNIST**: This dataset consists of 60,000 training images and 10,000 test images of hand-written digits. Each image is 28x28 pixels and grayscale.
- **CIFAR-10**: This dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class.
- **CIFAR-100**: This dataset is similar to CIFAR-10 but with 100 classes containing 600 images each. Each image is 32x32 pixels and in color.

4.1.2 Training Configuration

The generator and discriminator for both GAN and DCGAN models are trained using the Adam optimizer. The learning rate for the generator is set to 0.0002, and for the discriminator, it is set to 0.0002. The models are trained for 50 epochs with a batch size of 64. Label smoothing is applied to stabilize the training process.

4.2 Results

4.2.1 MNIST

For the MNIST dataset, the GAN and DCGAN models were evaluated based on the quality of generated digit images. We visually inspected the images and monitored the loss curves for both the generator and discriminator.

Figure 5: Generated MNIST Images using GAN

Figure 6: Generated MNIST Images using DCGAN

In this part, we could even conclude that basic GAN performs at the same level or even better than DCGAN. This may be due to the simple nature of the images that the convolutional layer fails in some way.

4.2.2 CIFAR-10

For the CIFAR-10 dataset, the models were evaluated based on the quality of generated images for each class.

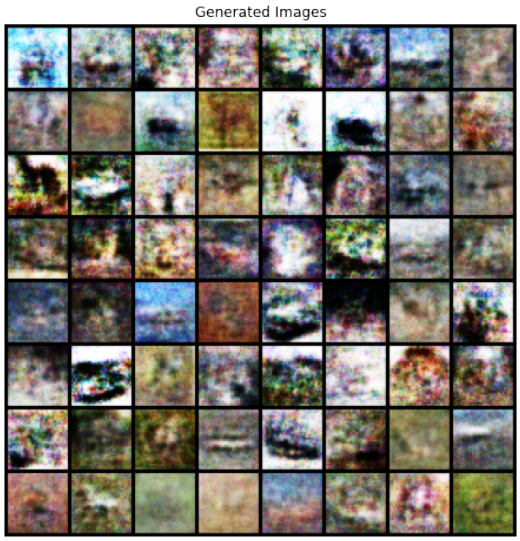

Figure 7: Generated CIFAR10 Images using GAN

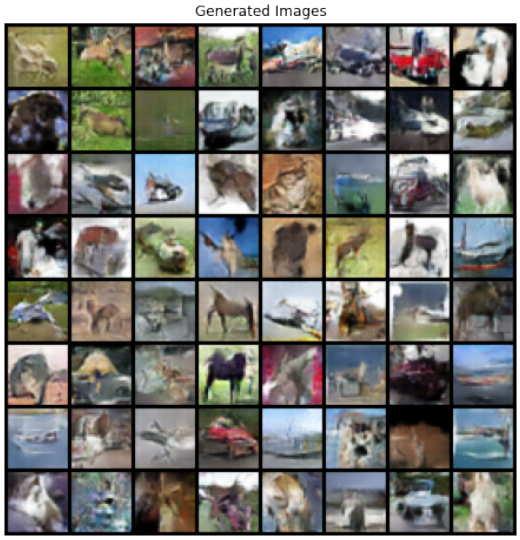

Figure 8: Generated CIFAR10 Images using DCGAN

The DCGAN model demonstrated much better performance in generating diverse and recognizable images across different classes. The images are much clear and much closer to real images.

4.2.3 CIFAR-100

For the CIFAR-100 dataset, the complexity increased due to the higher number of classes. The models were evaluated based on image quality and class diversity.

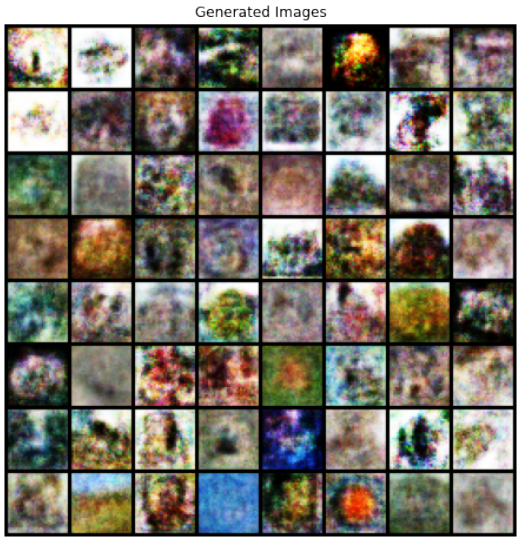

Figure 9: Generated CIFAR100 Images using GAN

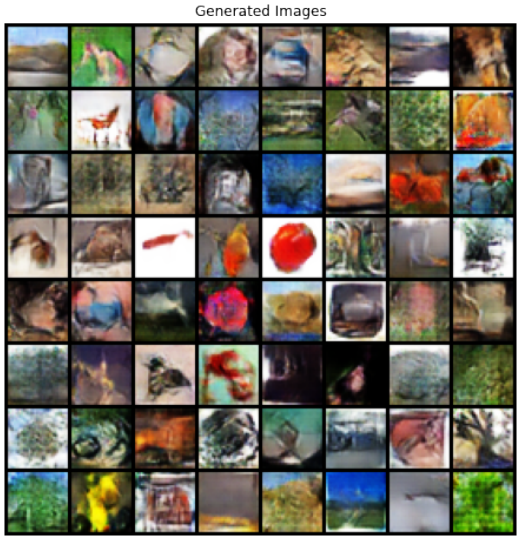

Figure 10: Generated CIFAR100 Images using DCGAN

Even when there are 100 classes, our DCGAN can still perform in a satisfying way, which is our goal of this experiment. GAN, on the contrary, at least with our layer design, does not fit this task of generating complicated images.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, *Generative Adversarial Nets*, Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS), 2014, pp. 2672–2680.

[2] A. Radford, L. Metz, and S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, arXiv preprint arXiv:1511.06434, 2015.

[3] M. Mirza and S. Osindero, *Conditional Generative Adversarial Nets*, arXiv preprint arXiv:1411.1784, 2014.

[4] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein GAN*, arXiv preprint arXiv:1701.07875, 2017.

[5] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, *Improved Training of Wasserstein GANs*, arXiv preprint arXiv:1704.00028, 2017.

5 Implementation

Because most of the work we relate to are built upon tensorflow, and the nature of these networks are quite simple, all of the codes in our project is built from scratch.