Exploring Image Generation with Deep Learning: Deep Convolutional Generative Adversarial Networks

Kenneth Casimiro
Electrical
Computer Engineering
A17132441

Steven Sharp
Electrical
Computer Engineering
A13232191

Abstract

One of the pioneering techniques for generating high-quality images is the Deep Convolutional Generative Adversarial Network (DCGAN). In this project, we revisit the original DCGAN study and apply its methods to diverse datasets to explore the technique's effectiveness. We use three diverse datasets for our experiments: EMNIST, which contains handwritten digits; Celeb-A, a collection of celebrity faces; and CIFAR-10, a collection of 10 image classes. Our objective is to understand the inner workings of DCGAN, identify its strengths, and pinpoint areas where it falls short.

1 Introduction

In recent years, deep learning has become a powerful tool for solving complex problems in various fields, including computer vision, natural language processing, and more. One exciting area where deep learning has made significant progress is image generation - the creation of high-quality, realistic images. These images have a wide range of applications, from art and entertainment to scientific simulations and enhancing existing datasets. One of the techniques in this domain is the Deep Convolutional Generative Adversarial Network (DCGAN), which has shown results in generating lifelike images.

1.1 Motivation

The motivation for our project is to revisit the original DCGAN study by Alec Radford, Luke Metz, and Soumith Chintala [2]. We will be recreating their methods over different image datasets and exploring how effective the technique is. We believe that by reconstructing the experiments and redefining the DCGAN approach, we can learn the architecture, identify its strength, and pinpoint where it falls short.

1.2 Project Problem

Our goal is to use DCGANs to generate high-quality, realistic images by training them on a variety of diverse datasets. Specifically, we will focus on datasets containing handwritten digits, celebrity faces, airplanes from the image collection. We chose these datasets because they are varied and complex, which will help us evaluate strength and weaknesses of a DCGAN.

1.3 Our Approach

To tackle this problem, we plan to re-implement the DCGAN architecture and apply it to the selected datasets. We will train the DCGANs and assess their performance based on the quality and diversity of the images they generate. Additionally, we will analyze the architecture of the DCGAN, looking at the roles of the generator and discriminator networks and how various factors affect the overall performance over our datasets.

1.4 Summary

In conclusion, our results show that the DCGAN approach works well with images that focus on a specific object like a human face or a handwritten digit. We found that images that had a scenic environment caused the generated images to be insufficient to point where the image cannot be identified easily. Overall, our project provides insights into how DCGANs work and identify the strengths and weaknesses of their performance.

2 Related Work

Generative Adversarial Networks (GANs) were first introduced by Ian Goodfellow and his colleagues in 2014, consisting of two neural networks, a generator and a discriminator, that are trained simultaneously [1]. The generator generates fake images from random noise, while the discriminator evaluates the generated images and real images, learning to distinguish between the two. The generator tries to produce images that can fool the discriminator, leading to an adversarial process that eventually results in the generator creating realistic images.

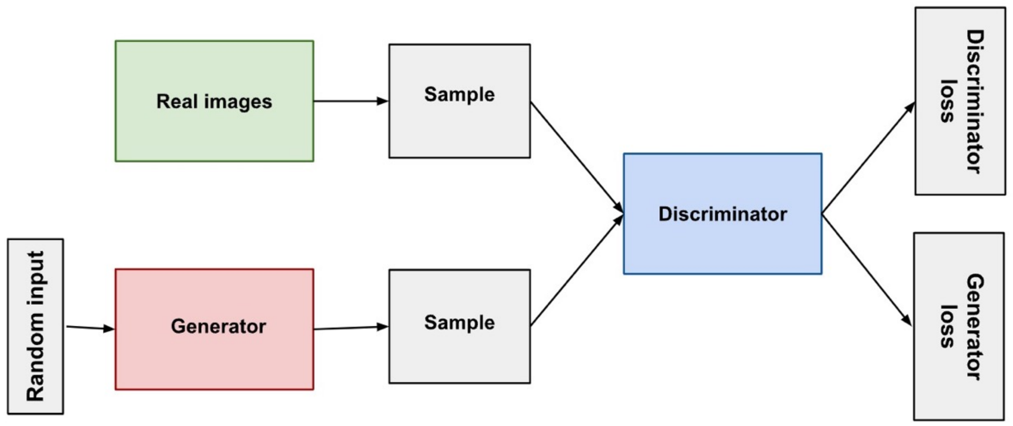

Figure 1: GAN Architecture [4]

One approach to build upon the GAN framework is DCGANs. We reference the research paper titled "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks" (2015, [2]). The authors propose several modifications, such as using convolutional layers instead of fully connected layers, employing batch normalization, and using specific activation functions (ReLU for the generator and Leaky ReLU for the discriminator). These changes improve the stability and performance of GANs, allowing the generation of higher-quality images. The authors experimented on three different datasets, which are LSUN, Imagenet-1k, and a custom dataset of faces to demonstrate that their architecture can generate realistic images. Therefore in our project, we recreated the DCGAN framework and created the experiments with different datasets and determined if we can generate realistic images from our datasets, being EMNIST, Celeb-A, and CIFAR-10.

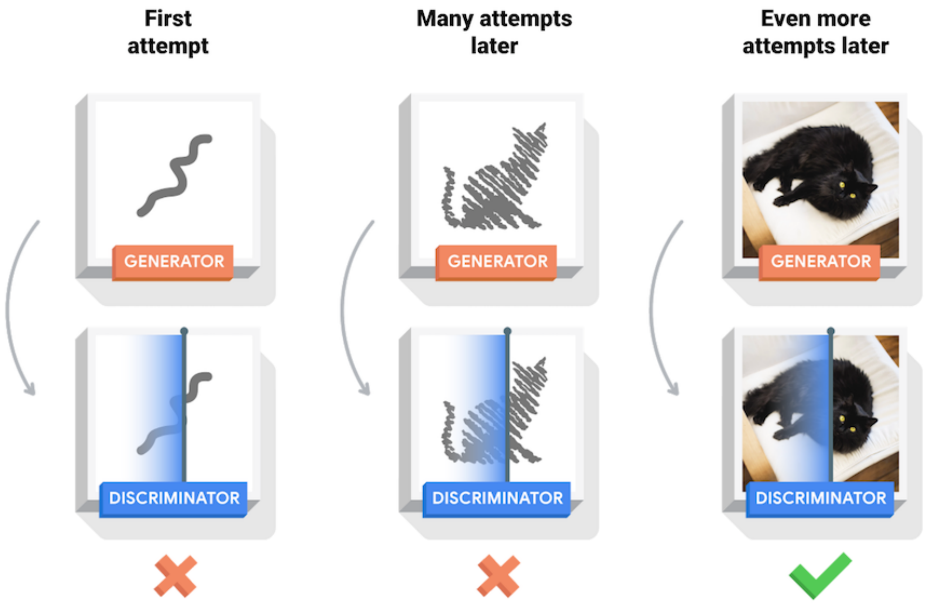

Figure 2: DCGAN Training Diagram [5]

3 Method

There will be four essential steps to our method for recreating the DCGAN framework. The first will be loading and preprocessing the dataset for our experiment. The second will be creating the generator and discriminator architecture. The second will be training and analyzing our results. The fourth and final step will be generating and comparing a batch of images from the dataset to the trained result to determine realism.

3.1 Data Loading and Preprocessing

The first step is loading and preprocessing the dataset. For each dataset, we will download and organize our data and then resize the images to a consistent size (64x64), normalize the pixel values, and convert the images to PyTorch tensors. This step will ensure our images will be uniformed for training.

3.2 Generator and Discriminator Architecture

The second step is following the DCGAN architecture guidelines proposed in the research paper. When creating the generator, we used a series of transposed convolutional layers with batch normalization and ReLU activation functions with a final layer that uses a Tanh activation function to generate images with pixel values in the range of -1 to 1. With the discriminator, we used convolutional layers with batch normalization and Leaky ReLU activation functions with a final layer that uses a Sigmoid activation function to output the probability of an image being real or fake.

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Figure 3: Architecture guidelines for Deep Convolutional GANs [2]

3.3 Training and Loss Analysis

The third step is to deploy the adversarial process. We will train the generator and discriminator simultaneously. The generator will train to create realistic images from random noise while the discriminator is trained to distinguish between real and fake images. We then will use a Binary Cross Entropy loss function and the Adam optimizer for both networks. During the training, it will output the loss from each network respectively to determine stability and the discriminator output (D(x) and D(G(z)) probabilities for real and fake images [3].

```
[9/10][2800/3166]       Loss_D: 0.1270  Loss_G: 6.9989  D(x): 0.9127    D(G(z)): 0.0117 / 0.0036
[9/10][2850/3166]       Loss_D: 0.1473  Loss_G: 5.7211  D(x): 0.9611    D(G(z)): 0.0866 / 0.0079
[9/10][2900/3166]       Loss_D: 0.0800  Loss_G: 4.4613  D(x): 0.9724    D(G(z)): 0.0468 / 0.0201
[9/10][2950/3166]       Loss_D: 0.0919  Loss_G: 4.9013  D(x): 0.9352    D(G(z)): 0.0168 / 0.0219
[9/10][3000/3166]       Loss_D: 0.0559  Loss_G: 5.3102  D(x): 0.9578    D(G(z)): 0.0088 / 0.0115
[9/10][3050/3166]       Loss_D: 0.0426  Loss_G: 5.3165  D(x): 0.9767    D(G(z)): 0.0173 / 0.0108
[9/10][3100/3166]       Loss_D: 0.0479  Loss_G: 5.7214  D(x): 0.9764    D(G(z)): 0.0218 / 0.0081
[9/10][3150/3166]       Loss_D: 0.0713  Loss_G: 5.1483  D(x): 0.9479    D(G(z)): 0.0121 / 0.0186
```

Figure 4: Generator and Discriminator Loss & Probability

3.4 Image Generation Analysis

The final and fourth step will be analyzing a batch of images before training as a baseline for comparison. After training, we will generate another batch of images to assess quality and realism compared to our baseline. This will finalize our re-implementation to determine if we can create high-quality realistic images with DCGAN.

4 Experiments

We trained our DCGAN on three different datasets. These being EMNIST, Celeb-A, and CIFAR-10. The EMNIST dataset is an extension of the MNIST dataset, containing handwritten digits and characters [6]. It consists of 28x28 grayscale images. The Celeb-A dataset is a large-scale collection of celebrity faces, comprising over 200,000 RGB images [7]. Each image is 64x64 pixels in size. The CIFAR-10 dataset consists of 60,000 32x32 color images, with 10 different object classes [8]. For our experiment, we focused on the airplane class. With each dataset, we pre-processed the data and created a generator and discriminator with the following layers:

Generator:

- Transposed convolution layer with 100 input channels, 512 output channels, 4x4 kernel size, 1 stride, and 0 padding.
- Transposed convolution layer with 512 input channels, 256 output channels, 4x4 kernel size, 2 stride, and 1 padding.
- Transposed convolution layer with 256 input channels, 128 output channels, 4x4 kernel size, 2 stride, and 1 padding.

- Transposed convolution layer with 128 input channels, 3 output channels (for RGB images), 4x4 kernel size, 2 stride, and 1 padding.

Discriminator:

- Convolution layer with 3 input channels (for RGB images), 128 output channels, 4x4 kernel size, 2 stride, and 1 padding.
- Convolution layer with 128 input channels, 256 output channels, 4x4 kernel size, 2 stride, and 1 padding.
- Convolution layer with 256 input channels, 512 output channels, 4x4 kernel size, 2 stride, and 1 padding.
- Convolution layer with 512 input channels, 1 output channel, 4x4 kernel size, 1 stride, and 0 padding.

Each transposed convolution layer in the Generator is followed by a batch normalization layer and a ReLU activation function, except for the last one, which uses a Tanh activation function. In the Discriminator, the convolution layers are followed by batch normalization layers (except for the first one) and LeakyReLU activation functions. The final output of the Discriminator is processed using a sigmoid activation function to obtain probabilities. This is because the sigmoid is projecting confidence probability to determine real and fake images.

With each training, we will be able to analyze the following variables to determine if our performance and accuracy has improved or not.

- Loss D: This is the Discriminator Loss. It represents how well the Discriminator can distinguish between real and generated images during each iteration. Lower values indicate better performance.
- Loss G: This is the Generator Loss. It represents how well the Generator can create realistic images that can "fool" the Discriminator. Lower values indicate better performance.
- D(x) [3]: This is the average output probability of the Discriminator for real images (x). It represents how well the Discriminator can identify real images. Values closer to 1 indicate better performance.
- D(G(z)) [3]: This value has two parts: the numerator is the average output probability of the Discriminator for generated images before the Generator update. Lower values indicate that the Discriminator is better at identifying fake images. The denominator is the average output probability of the Discriminator for generated images after the Generator update. Higher values indicate that the Generator is better at creating realistic images that can fool the Discriminator.

Finally with each dataset at the end, we will be plotting batches of images from the real dataset and the generated images after training to determine similarities and differences through our naked eye.

4.1 EMNIST - Result

Figure 5 and 6 showcase impressive results for the EMNIST dataset using our DCGAN. A comparison between real and generated images reveals realistic handwritten digits. This leads us to conclude that DCGAN performs best with datasets where a distinct contrast exists between the background and the object within the image.

Figure 7 shows that the Discriminator Loss is generally quite low, indicating that the Discriminator excels at distinguishing between real and fake images. On the other hand, the Generator Loss is relatively high, suggesting difficulty in generating realistic images capable of fooling the Discriminator. D(x) being close to 1 implies that the Discriminator successfully recognizes real images, while the low D(G(z)) numerator and denominator indicate that the Discriminator can easily identify fake images, and the Generator struggles to create realistic images that deceive the Discriminator. Therefore this contradicts our visual assessment, as the two batches of images appear nearly identical. This discrepancy could be due to errors in our metrics, or the possibility that we have a few lucky samples generated by chance which requires further investigation. We suspect that it overfits, resulting in the network to memorize the real dataset inputs.

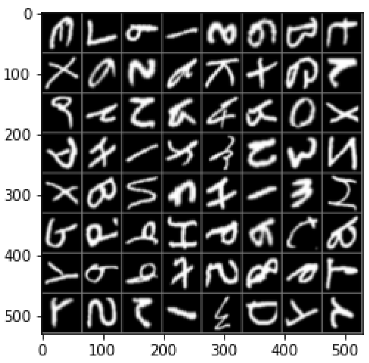

Figure 5: Left Image (Real)

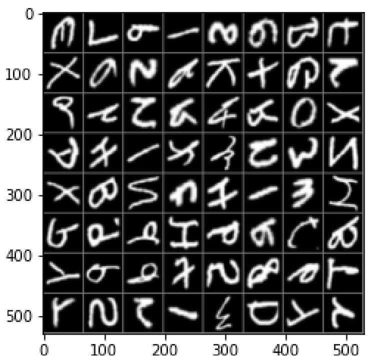

Figure 6: Right Image (Generated)

```
[10/10] [700/882] Loss_D: 0.0001        Loss_G: 9.8182  D(x): 1.0000    D(G(z)): 0.0001 / 0.0001
[10/10] [750/882] Loss_D: 0.0000        Loss_G: 12.1525 D(x): 1.0000    D(G(z)): 0.0000 / 0.0000
[10/10] [800/882] Loss_D: 0.0001        Loss_G: 9.9660  D(x): 0.9999    D(G(z)): 0.0001 / 0.0000
[10/10] [850/882] Loss_D: 0.0003        Loss_G: 9.1757  D(x): 0.9998    D(G(z)): 0.0001 / 0.0001
```

Figure 7: EMNIST Loss Probability

4.2 Celeb-A - Result

In our experiments with the Celeb-A dataset, our goal was to generate realistic human faces using DCGAN. Upon comparing batches of real images (Figure 8) to the generated images (Figure 9), it becomes evident that there are noticeable differences between the two. The generated faces appear disfigured, suggesting that the DCGAN needs further improvements to create more realistic images.

Examining the performance metrics in Figure 10, we observe the following: The Discriminator Loss is low, which indicates better performance. However, the Generator Loss is relatively high, pointing to lower performance. The D(x) value is close to 1, suggesting that the Discriminator performs well on average in terms of output probability. The D(G(z)) numerator is low, indicating that the Discriminator is effective at identifying fake images. Meanwhile, the D(G(z)) denominator is also low, which implies that the Generator is not producing realistic images that can deceive the Discriminator. Overall, these results show that further refinements are necessary for the DCGAN model to generate more convincing human faces. We suggest that it requires more epochs or data to be trained with.

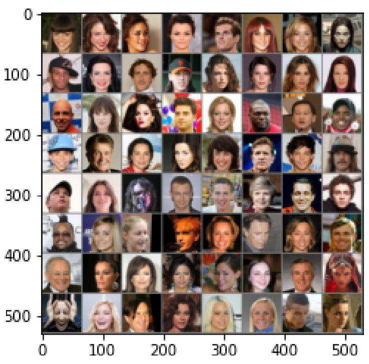

Figure 8: Left Image (Real)

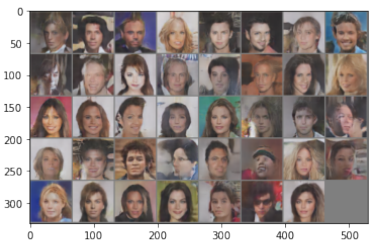

Figure 9: Right Image (Generated)

```
[9/10][2800/3166]      Loss_D: 0.1270  Loss_G: 6.9989  D(x): 0.9127    D(G(z)): 0.0117 / 0.0036
[9/10][2850/3166]      Loss_D: 0.1473  Loss_G: 5.7211  D(x): 0.9611    D(G(z)): 0.0866 / 0.0079
[9/10][2900/3166]      Loss_D: 0.0800  Loss_G: 4.4613  D(x): 0.9724    D(G(z)): 0.0468 / 0.0201
[9/10][2950/3166]      Loss_D: 0.0919  Loss_G: 4.9013  D(x): 0.9352    D(G(z)): 0.0168 / 0.0219
[9/10][3000/3166]      Loss_D: 0.0559  Loss_G: 5.3102  D(x): 0.9578    D(G(z)): 0.0088 / 0.0115
[9/10][3050/3166]      Loss_D: 0.0426  Loss_G: 5.3165  D(x): 0.9767    D(G(z)): 0.0173 / 0.0108
[9/10][3100/3166]      Loss_D: 0.0479  Loss_G: 5.7214  D(x): 0.9764    D(G(z)): 0.0218 / 0.0081
[9/10][3150/3166]      Loss_D: 0.0713  Loss_G: 5.1483  D(x): 0.9479    D(G(z)): 0.0121 / 0.0186
```

Figure 10: Celeb-A Loss Probability

4.3 CIFAR-10 - Result

In our experiments with the CIFAR-10 dataset, we observed that DCGAN struggled to generate convincing images when the input contained a complex, scenic background. As seen in Figure 12, the generated images lack a discernable airplane and appear smudgy. This suggests that DCGAN performs better with datasets featuring prominent objects in their images with a distinct contrast between the background and the object.

Analyzing the performance metrics in Figure 13, we note the following: The Discriminator Loss is low, indicating good performance. Conversely, the Generator Loss is higher, suggesting that it is not performing as well. The D(x) value is not close to 1, which implies that, on average, the Discriminator's performance is suboptimal. The D(G(z)) numerator reveals that the Discriminator can identify fake images, but the denominator demonstrates that the Generator is unable to create realistic images that can fool the Discriminator. Based on these findings, further improvements to the DCGAN model are necessary for better results with the CIFAR-10 dataset.

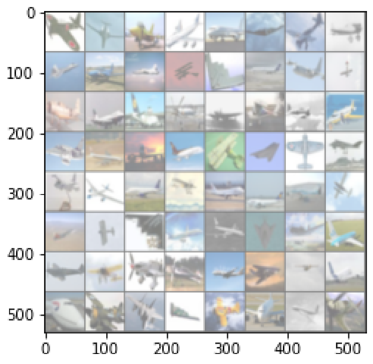

Figure 11: Left Image (Real)

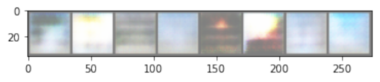

Figure 12: Right Image (Generated)

```
[5/10][50/79]   Loss_D: 0.4552  Loss_G: 1.8231  D(x): 0.7636    D(G(z)): 0.1608 / 0.1669
[6/10][0/79]    Loss_D: 0.4143  Loss_G: 1.9662  D(x): 0.8073    D(G(z)): 0.1712 / 0.1778
[6/10][50/79]   Loss_D: 0.3774  Loss_G: 1.8868  D(x): 0.8286    D(G(z)): 0.1701 / 0.1548
[7/10][0/79]    Loss_D: 0.3880  Loss_G: 2.0402  D(x): 0.8468    D(G(z)): 0.1869 / 0.1593
```

Figure 13: CIFAR-10 Loss Probability

5 Conclusion

In conclusion, we were able to re-implement the DCGAN from the research paper. Although we did not have the best results compared to what was found on the research paper, we were able to identify what requires further investigation with our experiments to improve our network. Our experiment using DCGAN with three different datasets – EMNIST, Celeb-A, and CIFAR-10 – provided valuable insights into the performance and capabilities of the model. We discovered that DCGAN performs exceptionally well with datasets featuring a distinct contrast between the background and the object within the image, as demonstrated by the realistic handwritten digits generated with the EMNIST dataset. This may have occurred due to overfitting where the network has memorized the input of the real images. However, the model struggled to generate convincing human faces with the Celeb-A dataset and failed to generate recognizable airplanes in the case of CIFAR-10. These occurred because these datasets were more contrast and higher dimensional compared to EMNIST.

While the Discriminator consistently performed well across all datasets, the Generator experienced difficulties creating realistic images capable of fooling the Discriminator. We determine that further training, model adjustments, or exploring alternative GAN architecture may be necessary to improve the Generator's performance. Without changing the DCGAN architecture, we would try to employ a more strategic updating paradigm with generator's updates much more frequently than the discriminator's, as this seems to be the biggest issue with our results. For alternative GAN architecture, contrasting learning techniques such as SimCLR or MoCo, should drastically enhance both the Discriminator and Generator networks to parse through different structures of the images. Additionally, having more data will also increase performance.

An additional observation of note is that both networks utilize primarily convolutional layers which direct the DCGAN's generator capabilities to focus mostly over contrast within the images than other structural features. This may simply be a core weakness of DCGAN architecture itself, even if the results can be improved through other means.

6 Supplementary Material

Presentation can be found here on Google Slides.

Video presentation can be found here on YouTube and on the GradeScope submission.

References

[1] Goodfellow, Ian, "Generative adversarial networks." Communications of the ACM 63.11 (2020): 139-144. https://dl.acm.org/doi/pdf/10.1145/3422622

[2] Radford Alec, Metz Luke, and Chintala Soumith. 2015. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." arXiv:1511.06434. https://arxiv.org/abs/1511.06434.

[3] (n.d.). GAN Loss. Google - Machine Learning. Retrieved March 17, 2023,
`https://developers.google.com/machine-learning/gan/loss`

[4] Overview of GAN Structure; Google - Machine Learning. Retrieved March 17, 2023,
`https://developers.google.com/machine-learning/gan/gan_structure`

[5] Deep convolutional generative Adversarial Network - Tensorflow. Retrieved March 17, 2023.
`https://www.tensorflow.org/tutorials/generative/dcgan`

[6] Cohen, G., Afshar, S., Tapson, J., van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters.
`https://www.nist.gov/itl/products-and-services/emnist-dataset`

[7] Liu, Ziwei and Luo, Ping and Wang, Xiaogang and Tang, Xiaoou. 2015. "Deep Learning Face Attributes in the Wild". Proceedings of International Conference on Computer Vision (ICCV).
`https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html`

[8] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.
`https://www.cs.toronto.edu/~kriz/cifar.html`