Synthesizing Representations: Deep Learning for Image Generation with Deep Convolutional Generative Adversarial Networks"

Nathaniel Greenberg
Computer Engineering
A16674776

Preston Le
Electrical Engineering
A18100872

Abstract

Deep Convolutional Generative Adversarial Networks (DCGANs) are commonly used to generate high-quality images. In our project, we studied and reimplemented the original DCGAN model developed by Radford, evaluating its performance on three datasets: USPS (grayscale handwritten digits), SVT (street view text), and Synthetic Digits (digits embedded on random backgrounds). We assessed performance using metrics such as Inception Score (IS) to show the accuracy of our DCGAN model and visual inspection. Our findings demonstrate that while DCGAN performs effectively on the simpler datasets such as USPS and Synthetic Digits, it exhibits limitations when handling the more complex datasets such as SVT, as it resulted in generated images that lack clarity and consistency.

1 Introduction

DCGANs by Radford (2015), was a critical step forward in image generation. Compared to other GAN models that relied on fully connected layers, DCGANs use convolutional layers to better capture spatial details in images. The design also uses batch normalization and strided convolutions instead of pooling layers which increased image quality and made training more stable.

Motivation
Although DCGANs work well for tasks like generating handwritten digits and scene text, issues such as mode collapse, unstable training, and poor performance on complex datasets are still relevant. In this project, we analyze these problems and work to improve the networks.

Background
Our project evaluates DCGAN's effectiveness on three datasets with distinct characteristics: USPS (grayscale handwritten digits), Synthetic Digits (synthetically generated digits on random backgrounds), and SVT (street view text). These datasets introduce varying levels of complexity, from simple digit structures in USPS to complex text images in Synthetic Digits and SVT. Testing our DCGAN across these datasets will display its generalization ability and limitations.

Approach
We reimplemented DCGAN and trained it on each dataset, measuring performance with the Inception Score (IS), and visual inspection, while also considering training stability, style variation, and image sharpness.

Results Summary
DCGAN performs well on the low-noise USPS dataset, generating clear and recognizable digits. However, its performance diminishes on more complex datasets like Synthetic Digits and SVT, indicating that improvements, such as adaptive normalization, are necessary for

handling noisy, real-world data. Additionally, when applied to images with intricate and scenic backgrounds, the generated results were often unclear and hard to identify. Overall, our project provides valuable insights into how DCGANs work, shedding light on their strengths while also highlighting areas for improvement.

2 Related Work

We began our research on DCGANs by first exploring their predecessor, Generative Adversarial Networks (GANs), introduced by Ian Goodfellow (2014). These early models use two neural networks; a generator and a discriminator trained in tandem to produce realistic images. One shortcoming of these models was their reliance on fully connected layers, which limited their ability to capture fine-grained spatial details. Radford et al.'s DCGAN (2015) improved upon this by replacing fully connected layers with convolutional ones, enabling hierarchical feature learning and more stable training through batch normalization and tailored activations (ReLU/LeakyReLU).

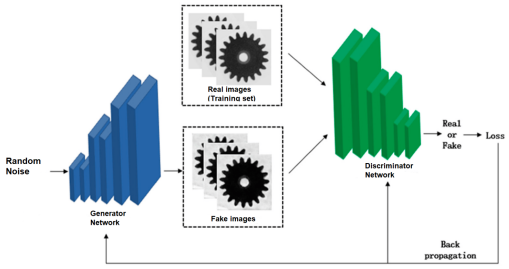

Figure 1: DCGANs Training Process

Seeking further advancements, we read StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks (Zhang et al., 2017). While DCGANs utilize convolutional layers to enhance feature learning and training stability, their single-stage architecture often struggles with producing high-resolution images. StackGAN addresses this limitation by employing a two-stage approach: the first stage generates low-resolution images capturing the basic shape and color based on the input text description, and the second stage refines these images to a higher resolution. This hierarchical structure allows StackGAN to produce more detailed and realistic images compared to the single-stage design of DCGANs.

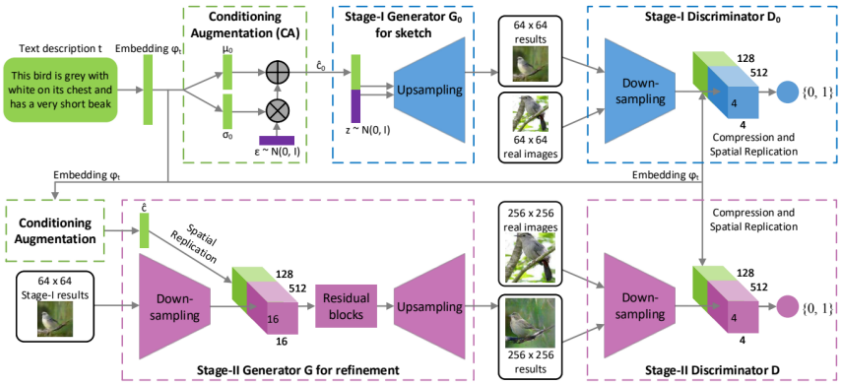

Figure 2: StackGAN Architecture

While StackGAN's two-stage approach offers advancements in generating high-resolution images from text descriptions, it has limitations such as producing unintelligible images and susceptibility to mode collapse. To address these issues, we adapted Radford et al.'s DCGAN implementation to our datasets: USPS, Street View Text (SVT), and Synthetic Digits. This adaptation enabled us to evaluate the model's performance across diverse image types.

3 Method

Overview
In our reimplementation, we had the following procedure: Data preparation, proper DCGAN architectural design, flexible training steps, and finally a performance summary.

Data Preparation
For our 3 datasets, each image was resized to 64x64 pixels to maintain consistency. Normalizing the pixel values to range between -1 and 1 ensured uniformity across the dataset. Finally, we converted the images into PyTorch tensors, preparing them for seamless integration into our model.

Model Architecture Design
With the data preprocessed, we then constructed the generator and discriminator models, referencing the DCGAN networks. The generator was designed with transposed convolutional layers, each followed by batch normalization and ReLU activation functions. This configuration resulted in a Tanh activation function in the final layer, producing images with pixel values spanning from -1 to 1. Adjacently, the discriminator was built using convolutional layers, each accompanied by batch normalization and Leaky ReLU activation functions. The final layer employed a Sigmoid activation function, outputting the probability of an image being real or generated.

Training Algorithm
Next, we began our training procedure. The generator aimed to create realistic images from random noise, while the discriminator aimed to distinguish between real and generated images. Both networks utilized the Binary Cross Entropy loss function and the Adam optimizer to facilitate efficient learning. Throughout training, we recorded and monitored the loss values of each network to assess stability and performance.

Image Generation and Performance Evaluation
After completing the training phase, we generated a batch of images to evaluate the efficacy of the model. These fake images were compared to our real dataset images to assess quality and realism. Additionally, we calculated the discriminator's accuracy in differentiating between real and fake images, which provided us a simple accuracy percentage for evaluating the DCGAN's performance.

4 Experiments

For our project, we used three unique datasets: United States Postal Service (USPS), Street View Text (SVT), and Synthetic Digits, to assess and train our DCGAN Model.

USPS Dataset
This dataset contains grayscale images of handwritten digits (0–9) which each measure 16x16 pixels.

Street View Text (SVT) Dataset
This dataset consists of images extracted from Google Street View, that focus on text found in natural scenes with images varying in sizes and aspect ratios.

Synthetic Digits Dataset
This dataset consists of synthetic images of handwritten digits that are generated through a rendering pipeline that depicts various writing styles and backgrounds. Each image is 32x32 pixels and paired with a corresponding label.

As mentioned in methods, we prepared each dataset making sure every image is 64x64 pixels. We then designed both the generator and discriminator networks following Radford's guidelines.

Generator Architecture

The generator starts with a 100-dimensional noise vector from the latent space. It then processes this input through a series of transposed convolutional layers. The first layer transforms the noise vector into a 4x4x512 tensor using a 4x4 kernel with a stride of 1 and no padding, followed by batch normalization and a ReLU activation function. The

subsequent layers upsample the tensor: the second layer outputs a 8x8x256 tensor, the third produces a 16x16x128 tensor, and the fourth generates a 32x32x64 tensor, each employing a 4x4 kernel with a stride of 2 and padding of 1, accompanied by batch normalization and ReLU activations. The final layer outputs a 64x64x3 RGB image, utilizing a 4x4 kernel with a stride of 2 and padding of 1, followed by a Tanh activation function to ensure the pixel values are within the range of -1 to 1.

Discriminator Architecture

The discriminator takes a 64x64 RGB image as input and processes it through a series of convolutional layers. The first layer applies a 4x4 kernel with a stride of 2 and padding of 1, converting the input into a 32x32x64 tensor, followed by a LeakyReLU activation function with a slope of 0.2. The second layer transforms this tensor into a 16x16x128 representation, the third into an 8x8x256 tensor, and the fourth into a 4x4x512 tensor, each using a 4x4 kernel with a stride of 2 and padding of 1, accompanied by batch normalization and LeakyReLU activations. The final layer applies a 4x4 kernel with a stride of 1 and no padding, reducing the tensor to a single scalar value, which is then passed through a Sigmoid activation function to output the probability of the input image being real or fake.

Additional Convolutional Layer

While testing our reimplementation, we noted its limitations in handling complex image details. As an improvement to our model, we added an extra convolutional layer to both the generator and discriminator. This allows the model to capture more detailed patterns by giving it additional processing steps. We monitored it's training stability after this change, making sure the layer wouldn't cause imbalances between the two networks. This resulted in more realistic generated images while maintaining reliable convergence.

The discriminator loss quantifies how poorly the discriminator performs by measuring its mistakes when identifying real images and rejecting fake ones. In contrast, the generator loss encourages the generator to create images so convincing that the discriminator mistakes them for real images.

Discriminator Loss:

$$L_D = -\mathbb{E}\left[\log(D(x))\right] - \mathbb{E}\left[\log(1 - D(G(z)))\right]$$

Generator Loss:

$$L_G = -\mathbb{E}\left[\log(D(G(z)))\right]$$

To assess the performance of both the Discriminator and Generator in our DCGAN, we use several key metrics. These metrics evaluate how well the Discriminator distinguishes real from fake images, and how effectively the Generator creates realistic images that can deceive the Discriminator.

Discriminator Loss
Measures the discriminator's ability to distinguish real vs. fake images. Lower values indicate better performance.

Generator Loss
Reflects the success of the generator in fooling the discriminator. Lower values are preferable.

Accuracy
Evaluates the percentage of real and fake images correctly classified by the Discriminator. Higher accuracy suggests better model performance.

USPS Results

The generated images from the USPS dataset look surprisingly close to handwritten digits, showing that the DCGAN is doing a good job of capturing the structure of the data. When we take a closer look, the digits are clear and well-defined, which means the model is learning effectively from the simple grayscale images. In terms of performance, the Discriminator Loss is fairly low, which suggests it's doing a solid job of telling apart real from fake images. However, the Generator Loss

is still on the higher side, indicating that the Generator has some difficulty creating digits that can fool the Discriminator. Despite this, the model is performing pretty well at replicating the USPS dataset visually. With an accuracy of 73.9%, it's clearly distinguishing real from generated images quite effectively. Still, there's room for improvement in the Generator to produce even more realistic samples.

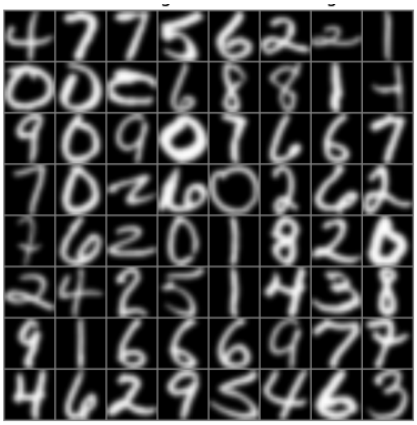

Figure 3: Real Images

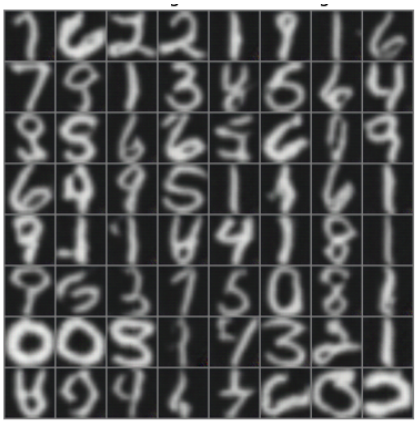

Figure 4: Generated Images

Synthetic Results

The generated images from the synthetic dataset show patterns that resemble the training data, although there are a few inconsistencies. The model does a good job of capturing the basic textures and structures, but it struggles with finer details, resulting in slightly blurry images. Despite these minor flaws, the DCGAN is still able to generate synthetic images that are fairly coherent. Looking at the performance metrics, the Discriminator Loss is quite low, which shows it's doing a good job of telling apart real and generated images. On the other hand, the Generator Loss is moderate, which suggests the Generator still has some work to do in fully fooling the Discriminator. Overall, the generated images are promising, but with a few tweaks to the Generator, the model could improve the clarity and structure of the outputs. The DCGAN model achieved an accuracy of 81.35%, demonstrating a solid ability to distinguish real from fake images.

Figure 5: Real Images

Figure 6: Generated Images

SVT Results

The generated images from the SVT dataset show very little resemblance to real text structures, with most of the images appearing blurry and distorted. The model struggled significantly to capture even basic character shapes, and the text often looked fragmented or unclear. Among all the datasets, SVT was by far the most challenging for the DCGAN, and the results were the weakest. Despite these challenges, the model still managed to generate text-like images, but the output quality was far from satisfactory. Performance metrics indicate that the Discriminator Loss is relatively low, meaning the Discriminator was good at distinguishing real from fake images. However, the Generator Loss was still high, suggesting that the Generator had a lot of difficulty in fooling the Discriminator. The model achieved an accuracy of 25.93%, showing that it had significant issues in replicating realistic text. To improve results, further refinements such as attention mechanisms or higher-resolution models would be necessary.

Figure 7: Real Images

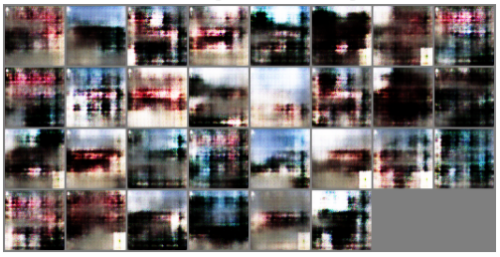

Figure 8: Generated Images

Final Analysis

Dataset	Accuracy	Epochs
USPS	73.90%	35
Synthetic Digits	81.35%	35
SVT	25.93%	100

Figure 9: DCGAN Accuracy

Overall, our findings show that DCGAN works well with simpler datasets like USPS and Synthetic Digits, where it captures the basic structures and generates clear images. However, when it comes to more complex datasets like SVT, the generated images are often unclear and inconsistent. The Discriminator does a good job of distinguishing real from fake images across all datasets, but the Generator struggles to create realistic samples. This suggests that while DCGAN is good for generating images with clear structures, it could benefit from some improvements. For instance, adding more convolution layers could help the model capture finer details and improve image quality. Additionally, running the model for more epochs might allow it to refine its outputs. Using attention mechanisms or higher-resolution models could also help the network focus on important features

and generate more realistic images. With these adjustments, the model could produce better-quality images for a wider variety of datasets.

5 Supplementary Material

Video Presentation is located in our zip file.

References

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

[2] Goodfellow, Ian, et al. "Generative Adversarial Nets." In *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680. https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[3] Wang, Xiaogang, et al. "StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020, https://arxiv.org/abs/1710.10916.

[4] "DCGAN Architecture: Concepts and Real-World Examples." *Vitalflux*, https://vitalflux.com/dcgan-architecture-concepts-real-world-examples/.

[5] "Understanding DCGANs." *YouTube*, https://www.youtube.com/watch?v=MG8UxEy2Ut8.

[6] "Deep Convolutional Generative Adversarial Network." *TensorFlow*, https://www.tensorflow.org/tutorials/generative/dcgan.

[7] "A Comprehensive Collection of Datasets for Deep Learning." *Papers With Code*, https://paperswithcode.com/datasets.

[8] "Implementing Custom Layers in PyTorch." *PyTorch Tutorials*, https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html.

[9] "USPS Handwritten Digit Dataset." *USPS Database*, http://www.science.smith.edu/~jc/USPS/. Accessed 2023.

[10] "Synthetic Handwritten Digits Dataset." *Kaggle*, https://www.kaggle.com/datasets/ahmedmohamed/synthetic-handwritten-digits. Accessed 2023.