Using a DC-GAN for Animal Image Synthesis

Ike Pawsat, Katie Baek, Kayla Imbriale, and Riley Byrnes
Boston College
pawsat@bc.edu, baekka@bc.edu, imbrialk@bc.edu, byrnesri@bc.edu

Abstract

One of the main issues in image generation is that it is slow, difficult, and expensive. To do it cheaply requires simple and easy datasets to train on. In this paper, we attempt to explore different image generation pathways. Ultimately, we used a DC-GAN to create images of Dogs, Cats, Lions, Elephants, and Horses. We experimentally confirmed that cheaper image generation is possible; however, improvements can still be made.

1. Introduction

Our project involves using and improving upon a DC-GAN (Deep Convolutional Generative Adversarial Networks) to generate images of various animals. We closely followed the architecture set out in the paper "Unsupervised Learning with Deep Convolutional GANS", then added some improvements such as Wasserstein loss and W-loss with GP to make an improved DC-GAN. Furthermore, through this process we discovered some future work that could be done to vastly improve model performance while remaining a cost effective option.

Image synthesis has become a very hot topic in the last year and we wanted to see if we could create an image synthesizer with a GAN structure, which was something we never attempted in class. We knew that training GANs was slow, difficult, and expensive, so the goal was not perfection, but rather a working model on a more complex dataset than what is typically shown and used in research papers.

2. Related work

Throughout our research and preparation, we found four main pieces of related work that influenced the architecture and direction of our model. With the oldest piece of work we analyzed being from 2016. While not every proposed method we read and researched was the perfect fit for our model, it was interesting to learn about the various thought processes and steps that go into creating GANs in this context. Figuring out what did not work for us was just as helpul as figuring out what did.

2.1. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks [4]

This paper proposes an improvement upon the traditional GAN by getting rid of the fully-connected linear layers and using deep convolutional layers instead (DC-GAN). Additionally, batch-norm is utilized in both the generator and discriminator. Rather than pooling in-between layers, strided convolutions are implemented. The ReLU activation function remains in all generator layers except for the output, which uses the TanH function. In the discriminator, the Leaky ReLU function is used.

2.2. Image Augmentations for GAN Training [7]

A large focus in this piece of work is questioning whether the addition of image augmentations during the GAN training process results in significant model improvement. Previously, adding image augmentation was not desirable since the discriminator would learn the augmentations as part of the image distribution and guide the generator accordingly. This paper found that only adding the augmentations for the generator had no impact, whereas adding the augmentations for both the generator and the discriminator resulted in notable improvement.

2.3. An Improved GAN using Distance Restraints [5]

This paper introduces the idea of constraining the generator using an auto-encoder(AE), and then reformulating the outputs from the AE as "real" input for the discriminator. This couples the convergence of the AE to that of the discriminator and slows its convergence, reducing the problem of gradient vanishing. Two novel distance constraints were also proposed to improve the generator. While this may seem like a viable option for increased performance, this requires a completely different model than the one we utilized and would require a significant amount of time to implement correctly. It also comes with the risk of more complexity without providing substantial enough improvements.

2.4. Simple Yet Effective Way of Improving the Performance of GAN [6]

The method proposed here only works for a fully connected linear GAN. One issue that GANs have is that the discriminator fails to guide the generator because it distinguishes between real and generated features using "silly or non-robust features". In order to combat this, a cascading rejection model is suggested. This fix can be applied to existing frameworks with little work, which made this a viable enhancement for our model.

3. Method

Our model proposes a DC-GAN with 3x128x128 input channels. We experimented with grayscale image generation (1x128x128) but decided on color generation so as to not sacrifice quality for cost.

3.1. Training

The training process for our model is framed as a "two-player game" between the generator and the discriminator, which are both initialized with random weights from a normal distribution before training begins. In each batch, a random latent vector is passed through the generator to create a fake image. Both this fake image and a real image will serve as inputs to the discriminator. First, the generator is trained by inputting the fake image into the discriminator. The output is then used to update the generator weights using backpropagation. Next, the discriminator is trained by taking the real image as input. This output is them used to update the weights of the discriminator through backpropagation. This process is repeated for each batch in each epoch of our training loop, allowing both models to improve based on feedback from the discriminator.

3.2. Generator

The generator aims to create realistic images to fool the discriminator. It takes a random noise vector (latent vector) as input and outputs a "fake" image of 128x128 pixels. This is achieved through sequential up-sampling over a series of layers, each consisting of a transposed convolution (with 4x4 kernels), a batch-norm, and a ReLU activation. The transposed convolution increases the spatial dimensions of the vector while also refining the features of the final image. The batch-norm helps to stabilize training and the ReLU activation introduces nonlinearity, allowing the generator to learn complex data distributions. No batch-norm or ReLU is used after our final transposed convolution layer. In place of this, a tanh activation is used to ensure that the pixel values of the output image are within the range [-1, 1], marching the normalized pixel values of the training images.

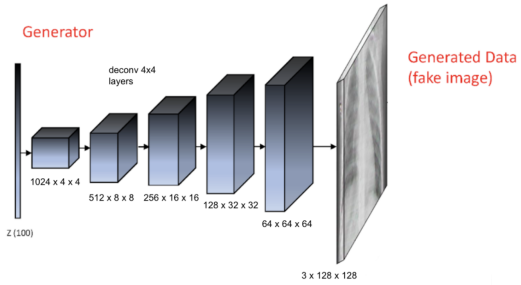

Figure 1. **Note:** Though the image generated looks black and white, this is the structure we used for colored images. Only deconvolutional layers are represented in this diagram, with batch-norm and activation functions excluded.

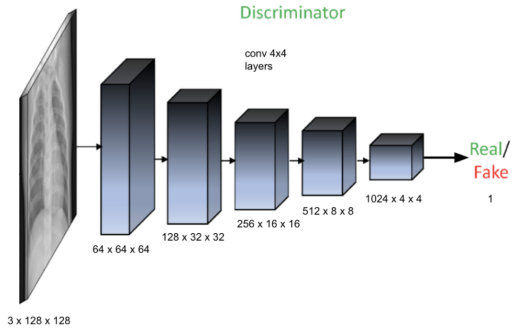

Figure 2. **Note:** Though the image generated looks black and white, this is the structure we used for colored images. Only convolutional layers are represented in this diagram, with batch-norm and activation functions excluded.

3.3. Discriminator

The goal of the discriminator is to accurately distinguish between real image tensors and fake ones created by the generator. Taking a 128x128 pixel image as input, it outputs a single probability representing how confident the discriminator is that the image is real, with 0 being not confident at all and 1 being extremely confident. Unlike the generator, this is achieved through sequential down-sampling over a series of layers, each consisting of a convolution (with 4x4 kernels), a batch-norm, and a leaky ReLU activation. The convolutions reduce the spatial dimensions of the image vector, while extracting important patterns and details that allow it to determine real from fake. The batch-norm is used again for stable training. A leaky ReLU activation is used to ensure that all inputs have a gradient, even if they are negative. This helps to avoid the "dying neuron" problem seen in the standard ReLU. No batch-norm or leaky ReLU is used after our final convolution layer. Instead, a sigmoid activation function is used to output a probability value between 0 and 1.

4. Experiments

Hyperparameters: For the experiments, we used the following set of hyperparameters:

- Learning rate: 2e-4
- Batch size: 128
- Epochs: Minimum 500, Maximum 1000
- Optimizer: Adam

These hyperparameters were chosen based on preliminary experiments, where we observed that these values provided the best trade-off between performance and training time.

4.1. Data

For our data, we used a Kaggle dataset called "Animals" by Anto Beneditti. It contains about 15,000 images of Cats, Dogs, Lions, Horses, and Elephants. Each of these animals has about 3,000 photos for each of them, and the dataset has them separated into train and val folders. The splits are about 2,700 for train and 300 for val. Each image was RGB and processed to be 512 by 512 pixels. Since our goal was to decrease training expenses, we further reduced these to 128 by 128 pixels.

When processing the data, we initially decided to also gray-scale the images to further reduce their dimensionality in the hope that this would improve training costs; however, we later decided that this strayed from the goal of the project of generating quality images at a cheaper cost.

4.2. Evaluation Metrics

Since it is image generation, while developing the model we were looking at the quality of the photos generated. We called this the eye-test, and would adjust hyper parameters and model structure during development until we felt the most satisfied. For these tests, we trained on and generated images of Dogs.

In the end, for metric testing, we used two different measurements, FID (Fréchet Inception Distance) score and KID (Kernel Inception Distance) score. We started with FID score but got very high results, even for cats, which came out very well. We discovered that this was because FID score performs poorly on validation sets of less than 1,000 data points. The reason for this is because it compares the means and covariances of the last pooling layer in the Inception v3 model after the generated and validation images are passed through. Since our validation set only had three hundred images, there is not enough information to capture the true mean and covariance of the true data distribution. Compariteviy, the KID score [2] is more suitable for smaller datasets, as it does not depend on characteristics like the mean and covariance to calculate its score.

4.3. Results

In this section, we present the results of our experiments on different animals. The chart below depicts the FID and KID Loss results for each metric, along with our personal ranking of their results.

Animal	FID Loss	KID Loss	Ranking
Cats	54.7	0.06 ± 3.17e-7	1
Dogs	88.7	0.13 ± 4.22e-7	2
Horses	81.0	0.04 ± 0.01	3
Elephants	59.0	0.05 ± 0.001	4
Lions	47.2	0.10 ± 0.02	5

A good FID score is one considered to be below 5 and above 15 is considered awful. As you can see, even our arguably best model, the cats, is nowhere near this. Once again, this is because the size of our validation set was too small. We could have done some data augmentation to increase the size of it; however, we decided to move on to a different loss function, which was the KID loss. The KID loss reflected the quality of the results much better. A perfect KID loss is a value of 0 and a good loss is within the range of 0.01 to 0.05. While we hovered barely outside of this range, this was expected, because we were attempting to make decent quality images at a cost-effective price. In addition, the standard deviation better reflects the variation in images generated through the different datasets. In the cat and dog images (Figures 3 and 4), all of the generated images center on the cat and dog face, so there is very little variation in that manner. Whereas in the horse, elephant, and lion images (Figures 5, 6, and 7), the animals are more in the background and there is less uniformity. Additionally, we have included our personal rankings of the images, with cats being the best and lions ranking the worst.

Figure 3. Generated cat images using DC-GAN with 500 epochs.

5. Conclusion

As you can see from the images, some animals turned out much better than others. For example, the cats and dogs look very good, however, elephants, horses, and lions did

Figure 4. Generated dog images using DC-GAN with 390 epochs.

Figure 5. Generated horse images using DC-GAN with 520 epochs.

Figure 6. Generated elephant images using DC-GAN with 590 epochs.

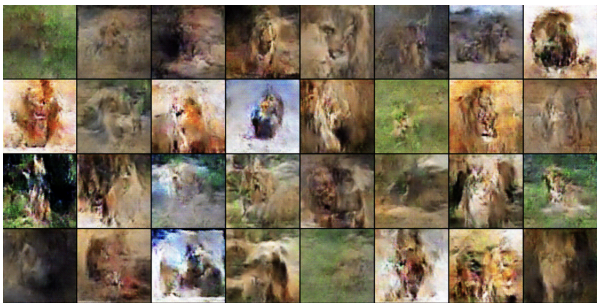

Figure 7. Generated lion images using DC-GAN with 400 epochs.

not turn out very well. We believe this is because of the more chaotic backgrounds. For example, the cats and dog datasets had much closer photos with less background visible and they were facing the camera. The other three, being significantly less domesticated, had much more variable backgrounds. Elephants had water, trees, and savanna backgrounds while the lions dataset was also interesting, as it had some cartoon photos of lions and a dolphin in it.

5.1. Problems

As mentioned above, one of the believed issues was the difficult backgrounds and variable framing of the non-domesticated animals, like lions and elephants, which made it much more difficult for the model to train on them. A solution to this is pretraining on the CIFAR-10 dataset, which is discussed in Future Work (5.3). Another issue we discovered was that the model had a very poor understanding of the latent spaces; for example, the cat's latent space is included here in Figure 8. As you can see, it is very dark and contains some large changes. To try to improve the quality of the latent space, we tried adding the Wasserstein loss function [1], both with and without gradient penalty as a way to enforce the Lipschitz continuity. However, the Wasserstein without gradient penalty model did not work at all as the loss for the Generator quickly went to 0. W-GAN with the gradient penalty [3]ended up working, however the training of this model was significantly slower, and Colab did not like this so we were unable to train until the image quality was sufficient. As a result, we were only able to train on the dog dataset, and only for around 300 epochs. The latent space visualization (Figure 9) for the model that we were able to partly train looked to be a bit better than the DCGAN, however the quality of the images generated was not as good. We believe that with more training, the WGAN with gradient penalty could result in a better latent space visualization, and thus better quality images.

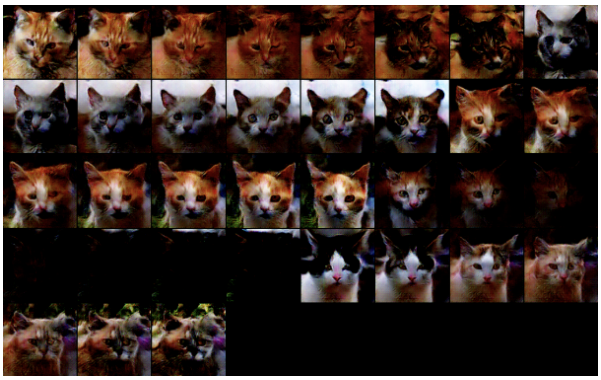

Figure 8. The cat's latent space.

5.2. Reproducibility

None of us have GPUs so we did our model training on Google Colab. To do this we used the Kaggle dataset men-

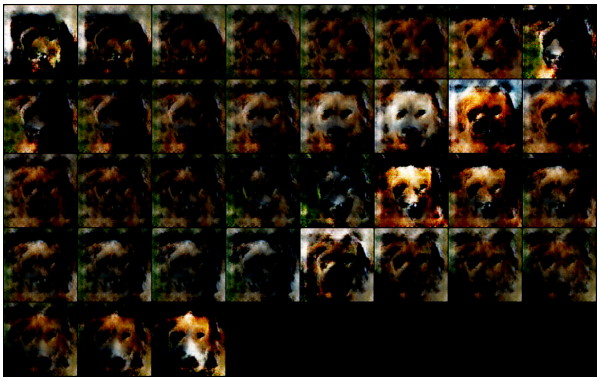

Figure 9. Dog latent space with WGAN-GP, trained on 250 epochs.

tioned which can be found in the references. Additionally, the GitHub link can be found at the bottom of the Contributions section (6) below. To run the code, access the dataset and the model in the GitHub. The results above were completed using the GAN, not the WGAN. You can download the model and use Colab to run it; the generator.pth files will automatically save to your Google Drive and same with the images every 10 epochs. Each animal ran for a variable amount of epochs, but each had a minimum of 500. To get the FID/KID scores as well as visualize the latent space, there is a different notebook that is available to run in Colab called *evaluate_model.ipynb*

5.3. Future Work

To improve the results further, while remaining cost-friendly and effective, we would use a model pretrained on the CIFAR-10 dataset so that it would not have to learn what a tree or body of water was while also trying to learn how to generate these images. Once again, since we were attempting to train on a difficult dataset and as cost effectively as possible, a larger or more clean dataset may not be possible, but this would definitely help model success. Overall, the biggest change that should be made would be model pretraining on the CIFAR-10 dataset.

We have pretrained our model with classifying the CIFAR-10 dataset and below are the results of the training. While we did not have enough time train this model and apply it to our dataset, this was a proof of concept. We used ResNet50 pretrained on ImageNet and then upsized the CIFAR-10 dataset to 128x128. We know this causes issues; however, we did not want to change our model architecture this late down to 32x32. Moreover, while the model is pretrained, it does need to be pretrained again because we upsized the images, but the current model, ResNet50, helps speed this up. Below are the results of this retraining.

Figure 10. Resizing from 32 to 128 degraded image quality as seen, all predictions were correct

6. Contribution

The contributions of each author to the work presented in this paper are as follows:

Ike Pawsat - Responsible for: Introduction (1), Data (4.1), Experiments (4), Training (3.1), Evaluation (4.2), Problems (5.1), Reproducibility (5.2), and Future Work (5.3).

Katie Baek - Responsible for: Evaluation Metrics (4.2), Results (4.3), Problems (5.1), Reproducibility (5.2), References

Kayla Imbriale - Responsible for: Method (3), Discriminator (3.3), and Generator (3.2)

Riley Byrnes - Responsible for: Literature Review and Related Work (2), Method (3), and References

GitHub: https://github.com/baekka1/hyperparamhackers

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017. 4

[2] Mikołaj Bińkowski, Danica J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans, 2021. 3

[3] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017. 4

[4] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016. 1

[5] Ngoc-Trung Tran, Tuan-Anh Bui, and Ngai-Man Cheung. Dist-gan: An improved gan using distance constraints, 2018. 1

[6] Yoon-Jae Yeo, Yong-Goo Shin, Seung Park, and Sung-Jea Ko. Simple yet effective way for improving the performance of gan. *IEEE Transactions on Neural Networks and Learning Systems*, 33(4):1811–1818, 2022. 2

[7] Zhengli Zhao, Zizhao Zhang, Ting Chen, Sameer Singh, and Han Zhang. Image augmentations for gan training, 2020. 1