Evaluation of Custom DCGAN model on CIFAR10 and Monet datasets

Anni Li
a5li@ucsd.edu

Ruoxuan Li
ruli@ucsd.edu

Abstract

With the development of deep learning in recent years, image generation has become one of the most popular topics. The primary goal of our project is to construct a DCGAN (Deep Convolutional Generative Adversarial Network) for image generation and compare the performance of our model on two distinctive datasets (Monet Paintings and CIFAR10). Throughout our experiments, we evaluated the performance of our DCGAN model based on various metrics, including the visual inception of the image quality and the inception score method. We found that our custom DCGAN model performed better on the Monet dataset compared to the CIFAR10 deer dataset. With the same amount of training data, the model was able to produce higher-quality Monet-style images compared to the deer images. The result was interesting, as it revealed the potential challenges we could encounter during training and the potential of DCGAN models for generating vivid images.

1 Literature Review

Generative Adversarial Networks (GANs) were first introduced in the paper "Generative Adversarial Nets" by Ian J. Goodfellow et al. A GAN consists of two models: a generative model G that generates objects of interest, and a discriminative model D that estimates the probability of a data sample to be real rather than generated. The training procedure for G is to maximize the probability of fooling D, so in this adversarial procedure, the quality of generated data can be gradually improved [1]. The image-generation ability of GANs was proved to be further improved in the paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks" (Radford et al, 2015). The authors introduced the DCGAN (Deep Convolutional Generative Adversarial Networks), a promising model that can effectively generate images by applying Convolutional Neural Networks (CNN) layers in the GAN. They also proposed several effective measures to improve the quality of the generated images by generative adversarial models. There are three major changes in the paper, including replacing the pooling functions with strided convolutions, eliminating fully-connected hidden layers for deeper architectures, and using batch normalization in both the generator and the discriminator [2]. Besides the significant changes, they also suggested using LeakyReLU activation for all layers in the discriminator and using Tanh activation in the output layer of the generator. The paper provided valuable insights for us to build our model. In comparison to traditional GAN in image generation tasks, DCGAN is more stable in training and can also learn a hierarchy of features from the images (Radford et al, 2015). For the evaluation of images generated by our GAN, we referred to the method introduced in the paper "Improved Techniques for Training GANs" by Salimans et al [3]. The paper demonstrated that the inception model [4] could be applied to generated images to get the conditional label distribution p(y|x), and the author suggested that images that contain meaningful objects should have a conditional label distribution p(y|x) with low entropy. The author proved that the Inception score correlates well with their subjective judgment of image quality, and collapsed models would have relatively low scores (Salimans et al, 2016). Therefore, by using human recognition and the inception score, we are able to evaluate and compare our model's performance on the two datasets more comprehensively.

2 Dataset

In our project, we mainly used two separate datasets for training. The first dataset is the "Claude Monet pictorial works dataset" [*https://www.kaggle.com/datasets/varnez/claude-monet-pictorial-works-dataset-*

Figure 1: Sample images from the Monet dataset

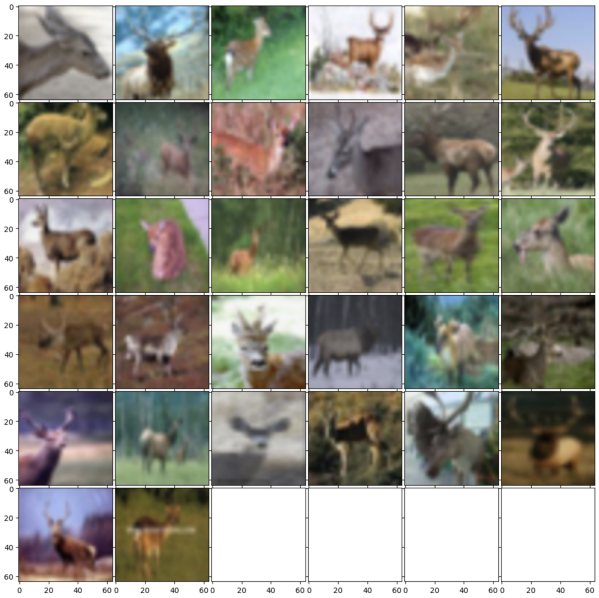

Figure 2: Sample images from the deer dataset

wikiart?resource=download]. The dataset included all Claude Monet's pictorial works with a normalized size of 256x256. This dataset contained 1369 images. To process the data and speed up our training process, we defined a transformer to resize the images to size 64x64 and normalize the pixel values. A sample set of images from the Monet data are shown below in Figure 1. The second dataset is the CIFAR-10 dataset [*https://www.cs.toronto.edu/ kriz/cifar.html*], which consists of 60,000 32x32 color images, with 50,000 images in the training dataset and 10,000 in the testing dataset. For our project purposes, we only selected a subset of the training data that all fit into the "deer" class. We used two approaches to select the data from the CIFAR-10 deer class for training the model. For option 1, we randomly selected 1369 images (same amount of Monet Dataset) from all the deer images and fed them into the model in order to compare the performance on the two datasets more equally. For option 2, we fed all the 5,000 deer images into the model for training in order to get a better result if option 1 does not lead to good outputs. Just like how we processed the Monet dataset, we also rescaled the images to size 64x64 and normalized the pixel values. A sample set of images from the deer data are shown below in Figure 2.

3 Methods

3.1 Model Construction

3.1.1 Discriminator

We referred to Zhong et al, 2023 [5] to construct our DCGAN model, but made some changes in the number of layers of our model. Our discriminator contains 5 convolutional layers. For the first convolutional layer, we define it to take in 64x64 3-channel RGB images inputs and output 64 channels with a 4x4 kernel with a stride of 2 and a padding of 1 on the images. Then we call the LeakyReLU activation function with an alpha value of 0.2. After the activation function, we define another convolutional layer using a 4x4 kernel with the same striding and padding and output 128 channels. We then perform batch normalization on the output channels and called our LeakyReLU activation function again. For our third convolutional layer, we construct it with the same kernel size and the same stride and padding setting as the previous convolutional layers and outputs 256 channels. After the layer, we perform batch normalization on the output and activate our LeakyReLU function. We keep passing the outputs into the next convolutional layer, which is constructed in a similar manner as the previous layers and outputs 512 channels. After performing batch normalization and calling the LeakyReLU activation function on the output channels, we perform the last convolution operation which has a 4x4 kernel with a stride of 1 and 0 padding and outputs 1 channel. Then, we flatten

the channels and call our Sigmoid activation function. The detail of the discriminator model can be found in Figure 3.

3.1.2 Generator

Our generator consists of 5 transposed convolutional layers, upscaling a small section of the image from a latent space to the desired 64x64 generated image outputs. The generator mirrors a similar structure to the construction of the discriminator model. We define the latent space to contain 128 points. Our first transposed convolutional layer takes in the latent space and performs an upscaling of kernel size 4 with a stride of 1 and 0 padding, outputting 512 channels. Then batch normalization is performed on the outputted channels and the ReLU activation function is called. After the activation function, the channels are fed into the next transposed convolutional layer. In this layer, a 4x4 kernel with a stride of 2 and padding of 1 is used to perform the transposed convolution operation and outputs 256 channels. As in the previous layer, batch normalization and ReLU activation functions are performed again. In the next transposed convolutional layers, a 4x4 kernel with a stride of 2 and a padding of 1 is called on the input channels and outputs 128 channels, following batch normalization and ReLU activation function. The next transposed convolutional layer is constructed in a similar manner with 64 output channels, followed by batch normalization and ReLU activation function. In the last layer, a transposed convolutional layer with the same set of kernel parameters outputs 3 channels. Lastly, the Tanh activation function is called and the desired images are returned. The detail of the discriminator model can be found in Figure 4. The complete model structure is shown in Figure 5.

3.2 Training

In our training, we labeled the real image samples from the datasets with 1 and the generated images with 0. We defined the batch size to be 32 and the number of epochs to be 200. We used Binary Cross Entropy (BCE) as the loss function, noise_dim=128 for the generator, and used Adam as the optimizer for both G and D in our training. The hyperparameters we chose to tune are the learning rates and beta_1. For the Monet dataset, we tried {lr=0.0002 beta1=0.5}, {lr=0.0004 beta1=0.5}, and {lr=0.0002 beta1=0.8}. By human recognition and inception score, it turned out that the combination

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 32, 32]	3,072
LeakyReLU-2	[-1, 64, 32, 32]	0
Conv2d-3	[-1, 128, 16, 16]	131,072
BatchNorm2d-4	[-1, 128, 16, 16]	256
LeakyReLU-5	[-1, 128, 16, 16]	0
Conv2d-6	[-1, 256, 8, 8]	524,288
BatchNorm2d-7	[-1, 256, 8, 8]	512
LeakyReLU-8	[-1, 256, 8, 8]	0
Conv2d-9	[-1, 512, 4, 4]	2,097,152
BatchNorm2d-10	[-1, 512, 4, 4]	1,024
LeakyReLU-11	[-1, 512, 4, 4]	0
Conv2d-12	[-1, 1, 1, 1]	8,192
Flatten-13	[-1, 1]	0
Sigmoid-14	[-1, 1]	0

Total params: 2,765,568
Trainable params: 2,765,568
Non-trainable params: 0

Input size (MB): 0.05
Forward/backward pass size (MB): 2.31
Params size (MB): 10.55
Estimated Total Size (MB): 12.91

Figure 3: Composition of the discriminator model

Layer (type)	Output Shape	Param #
ConvTranspose2d-1	[-1, 512, 4, 4]	1,048,576
BatchNorm2d-2	[-1, 512, 4, 4]	1,024
ReLU-3	[-1, 512, 4, 4]	0
ConvTranspose2d-4	[-1, 256, 8, 8]	2,097,152
BatchNorm2d-5	[-1, 256, 8, 8]	512
ReLU-6	[-1, 256, 8, 8]	0
ConvTranspose2d-7	[-1, 128, 16, 16]	524,288
BatchNorm2d-8	[-1, 128, 16, 16]	256
ReLU-9	[-1, 128, 16, 16]	0
ConvTranspose2d-10	[-1, 64, 32, 32]	131,072
BatchNorm2d-11	[-1, 64, 32, 32]	128
ReLU-12	[-1, 64, 32, 32]	0
ConvTranspose2d-13	[-1, 3, 64, 64]	3,072
Tanh-14	[-1, 3, 64, 64]	0

Total params: 3,806,080
Trainable params: 3,806,080
Non-trainable params: 0

Input size (MB): 0.00
Forward/backward pass size (MB): 3.00
Params size (MB): 14.52
Estimated Total Size (MB): 17.52

Figure 4: Composition of the generator model

of params lr=0.0002 beta1=0.5 worked better for the Monet Dataset. The best-generated batch of images by our model is shown below in Figure 6 and the training loss curve is shown in Figure 7. For training parameters {lr=0.0002, beta1=0.8} on the Monet dataset, however, the corruption of the final general image quality as shown in Figure 8 is observed. For the CIFAR-10 deer dataset, we initially used {lr=0.0002, beta1=0.5} for training on the selected 1,369 images and we experienced model corruption as shown in Figure 9. During the fine-tuning phase, we fed all the deer images (5000) into our model and tested both different beta1 and lr combinations including {lr=0.0002, beta1=0.5}, {lr=0.0002, beta1=0.8}, {lr=0.0004, beta1=0.8}, {lr=0.0001, beta=0.6} and achieved the best image quality with lr=0.0004 and beta1=0.8 as shown in Figure 10. The training loss curve is shown in Figure 11.

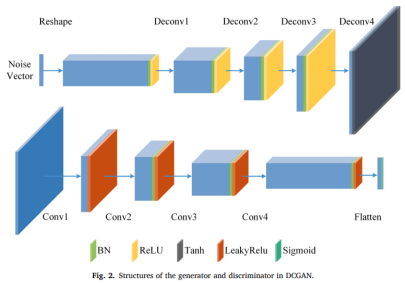

Figure 5: The structure of the DCGAN model for our reference.

3.3 Evaluation Metrics

In addition to human recognition, We used the inception score to evaluate the effectiveness of our models on the two datasets and compare the performances [3]. The inception score algorithm we used is to compute the entropy of the probabilities evaluated by the VGG16 model on our generated images. We imported the pre-trained image-recognition model VGG16 and split all generated images into 10 images a group and calculated the means and standard deviations for the inception score among all the groups. For the pre-trained model of our choice, the lowest inception score we can achieve is 1.0, and the highest possible inception score we could achieve is 1000. However, it is important to notice that the model used in the original paper to calculate the inception score is InceptionV3 pre-trained on the ImageNet Dataset. InceptionV3 requires the input image size to be 299x299, which is much higher than our generated image size, and resizing our image from 64x64 to 299x299 would cause a low score that lacks the value of reference. Therefore, we changed the evaluation model to VGG16, which accepts input sizes as low as 32x32. This change caused our final inception scores not comparable to the original paper, but could only be used to compare the performance of our single model on different datasets.

4 Results

The 200-epoch training using our model is comparably fast and can be finished in approximately 12 minutes, and the generated images look similar, at least have the same style, as the images in the original dataset by human evaluation. By calculating the inception scores of the generated images from the two datasets, we evaluated our model's performance more quantitatively and reliably. The best inception score of the generated Monet image batch is 346.23 ± 1.22, and the best inception score of the generated CIFAR10 deer image batch is 254.06 ± 1.23. As a higher inception score indicates a better-generated image quality, this result indicates that our model performs better on the Monet dataset compared to the CIFAR10 deer dataset. Meanwhile, we observed model corruption in both datasets during the fine-tuning phase, which corresponds to the observation in the original paper that models trained longer sometimes collapse a subset of filters to a single oscillating mode [2].

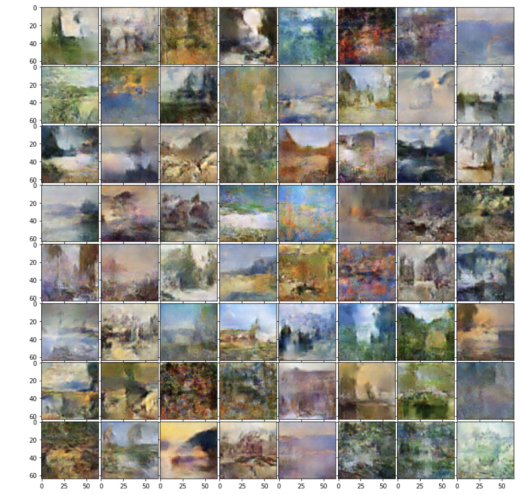

Figure 6: Best generated Monet image batch after training with lr=0.0002 and beta1=0.5.

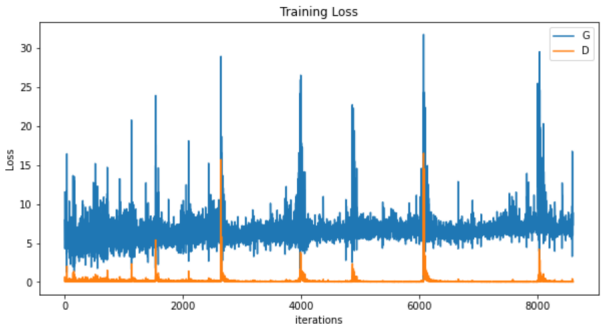

Figure 7: The training loss curve of the best quality training parameters for the Monet dataset

In addition, it is interesting to notice that the model performed better on the Monet dataset even using a smaller training set than on the CIFAR-10 dataset. It could probably be because the quality of the original images in the Monet dataset is better, or because of the nature of these two distinctive image domains. Monet's impressionism artworks are inherently more abstract in their visual representation, thereby having a greater tolerance for

Figure 8: The generated corrupted images after training with lr=0.0004 and beta1=0.8 on the Monet dataset.

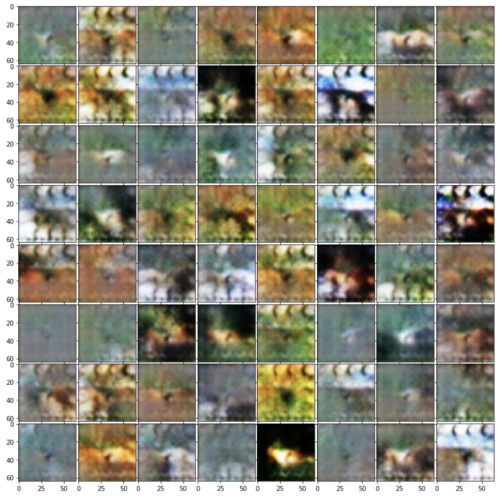

Figure 9: The final generated deer image batch after training with lr=0.0002 and beta1=0.5 on 1,139 images only. As shown in the figure, the generated images are corrupted.

variations and abstractions in generated images. In contrast, images of deer are of real-world objects, which are realistic and strict in detail. As a result, it is harder for our DCGAN model to generate deer images that meet these stringent standards, leading to a comparatively lower Inception Score.

5 Limitation

We must acknowledge that due to the hardware constraints, we were not able to train the model on a larger dataset using more epochs and higher-quality images. Additionally, fine-tuning a DCGAN model often requires extensive computational power to optimize various hyper-parameters. Thus, we were

Figure 10: Best generated deer image batch after training with lr=0.0004 and beta1=0.8.

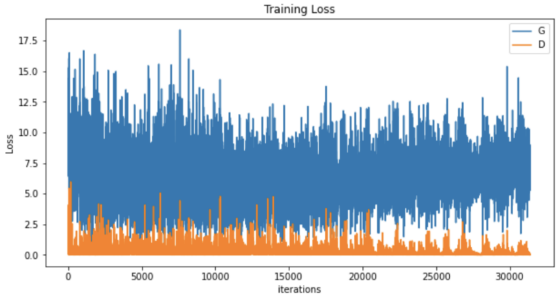

Figure 11: The training loss curve of the best quality training parameters for the deer dataset.

not able to fine-tune the model more thoroughly due to the limited resources and time. Despite these limitations, we believe that our project still provides valuable insights into the performance of DCGAN models on the Monet and CIFAR10 datasets. While our hardware constraints restricted the extent of fine-tuning, our findings offer a foundational understanding of the capabilities and potential of DCGAN models in generating visually appealing images.

6 Conclusion

In our project, we constructed, trained, and tuned a DCGAN model for image generation on two datasets, a Monet painting dataset and the CIFAR-10 dataset, and we used the inception score to evaluate the performance of our DCGAN. Based on the result of our experiment, we can safely conclude that our model performs better on the Monet dataset compared to the CIFAR10 dataset. We observed that with the same number of images and a same number of epochs, CIFAR10 dataset per-

formed worse compared to the Monet dataset, with the model corrupted at a certain point of training. Several possible causes could explain the difference in the performances of the model. First, it could be that when we preprocessed the images, the images from the Monet data size were shrunken to a smaller size of 64x64 from 256x256 while the images from the CIFAR10 dataset were enlarged from 32x32 to 64x64. When we enlarged the size of the CIFAR10 dataset, the images inevitably became blurry and had fewer details compared to the original images and to the images in the downsized Monet dataset. This could potentially bring unnecessary noise while training the model. Another possible explanation for this difference could be that the "deer" class from the CIFAR10 dataset contains more complexity compared to the Monet dataset. The unique features of the Monet dataset mainly lie in the color arrangements, while the deer images in CIFAR10 contain more diverse angles and shapes, possibly causing more challenges for the model to learn the pattern.

7 Contribution

Both authors actively collaborated and communicated throughout the project, including discussing project outlines, searching for paper references, understanding the model, debugging codes, and writing the report paper.
Anni Li: Construction of the model, data processing and training on Monet Dataset
Ruoxuan Li: Data processing and training on CIFAR-10 Dataset, Evaluation of the model, Inception Score insertion

References

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Nets. In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14), 2672–2680.

[2] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. https://arxiv.org/abs/1511.06434

[3] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training GANs. In Neural Information Processing Systems (Vol. 29, pp. 2234–2242). http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf

[4] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. arXiv preprint arXiv:1512.00567, 2015.

[5] Zhong, H., Fernando, T., Trinh, H., Lv, Y., Yuan, R., & Wang, Y. (2023). Fine-tuning transfer learning based on DCGAN integrated with self-attention and spectral normalization for bearing fault diagnosis. Measurement, 210, 112421. https://doi.org/10.1016/j.measurement.2022.112421