Individual Final Report

Introduction

In recent years, the field of artificial intelligence has witnessed significant advancements, particularly in the domain of image generation. Deep Convolutional Generative Adversarial Networks (DCGANs) have emerged as a potent tool for synthesizing highly realistic images, revolutionizing the way we approach visual data synthesis. This individual project report delves into the development and refinement of a DCGAN model designed specifically for generating lifelike images of dogs.

Individual Work

1. EDA(Exploratory Descriptive Analysis)
2. Construction of DCGAN Model
3. Fine-tuning DCGAN Model

DCGAN

Discriminator

The purpose of the discriminator is to distinguish all fake images. A discriminator works like a classifier with the sole task of determining if an image is real or fake. It is essential to incorporate downsampling methods into Discriminator.

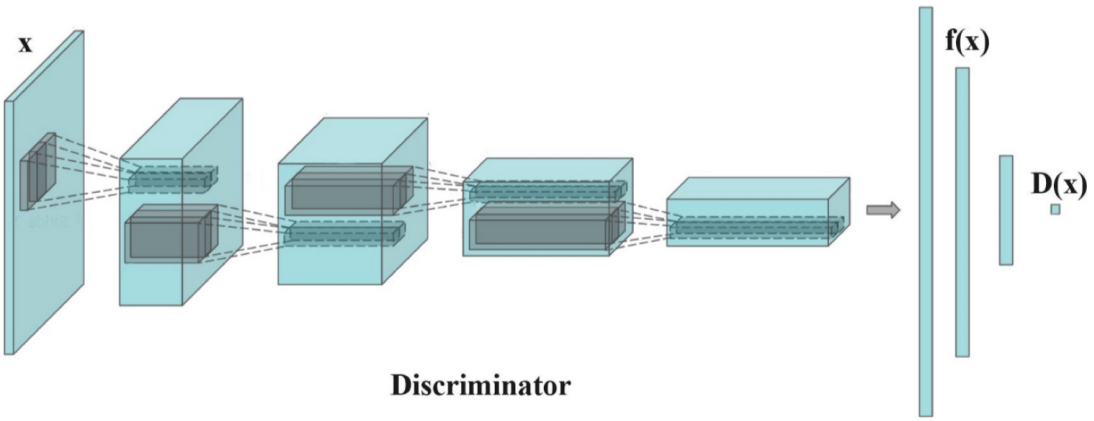

Fig-1: Example Architecture of Discriminator

Generator

The purpose of the generator is to generate realistic images from a random noise vector. In our project, it is designed to generate dog images. From a vector to an image, upsampling methods are always applied to the Generator.

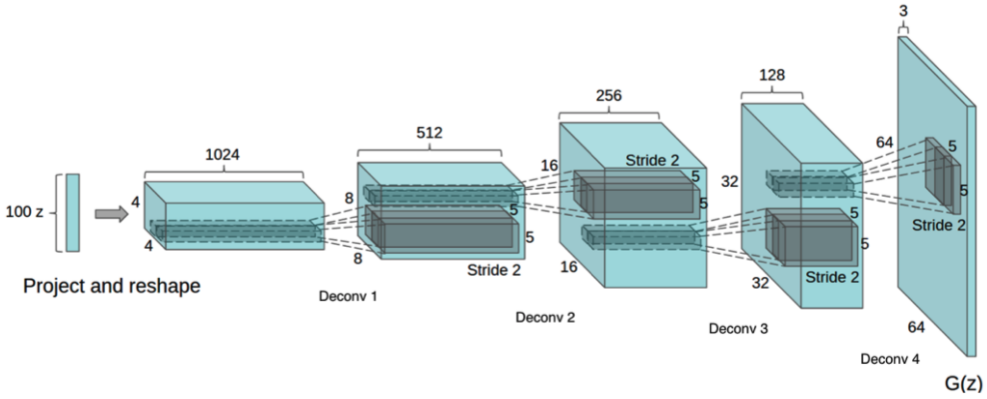

Fig-2: Example Architecture of Generator

Training Discriminator

In training the Discriminator for a DCGAN, the process involves several key steps. All weights within this network are initialized. Following this, the initialized Generator is utilized to generate a batch of fake images from random noise inputs. Next, real images, randomly selected from the original dataset, are labeled as class 1, while the fake images produced by the Generator are

labeled as class 0. Lastly, the Discriminator is trained to differentiate between these real and fake images, during which its weights are updated accordingly.

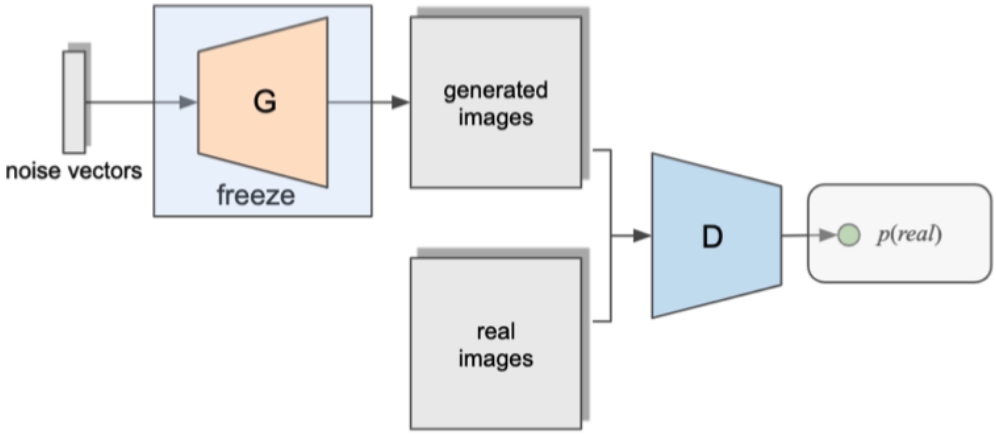

Fig-3: Training Discriminator

Training Generator

To train the Generator in a DCGAN, first, ensure the Discriminator's weights remain fixed. Begin by inputting a batch of random noise into the Generator to produce fake images. Then, feed these fake images into the already-trained Discriminator to evaluate the likelihood of them being perceived as real. During the training of the Generator, the goal is to have the input noise classified as class 1 (real). Therefore, use the target label 1 to compute the loss and gradients, which will in turn update the Generator's weights.

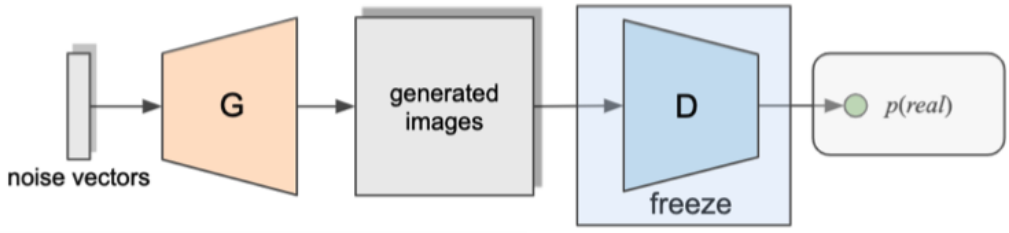

Fig-4: Training Generator

EDA

Training a Generative Adversarial Network (GAN) effectively necessitates a substantial dataset. To circumvent the laborious task of assembling a well-balanced collection of dog images ourselves, we opted for the Stanford Dogs Dataset. This publicly accessible dataset consists of 20,580 annotated images spanning over 120 dog breeds.

Despite its comprehensive classification, we encountered notable limitations in this dataset. A significant inconsistency was observed in the portrayal of dogs across the images. In some cases, dogs were prominently placed at the center, occupying a large part of the frame. In contrast, other images featured dogs positioned to the side or appearing distant.

This lack of uniformity, coupled with the variable and often noisy backgrounds - characterized by stark color differences even within the same breed - presented a substantial challenge. These factors contributed to a considerable degree of intra-class variation, complicating the task for the GAN. Specifically, it became more difficult for the generator component of the GAN to focus and accurately capture the specific features of the dog breeds that we aimed to highlight.

Overview of the DataSet

Our dataset contains 20579 images belonging to 120 dog breeds from all over the world. The number of each breeds' images mainly focuses on the range from 150 to 250.

	Dog Breed	Number of Observations
0	n02085620-Chihuahua	152
1	n02085782-Japanese_spaniel	185
2	n02085936-Maltese_dog	252
3	n02086079-Pekinese	149
4	n02086240-Shih-Tzu	214
.	.	.
.	.	.
.	.	.
116	n02113978-Mexican_hairless	155
117	n02115641-dingo	156
118	n02115913-dhole	150
119	n02116738-African_hunting_dog	169

Fig-5: Dog Images Description

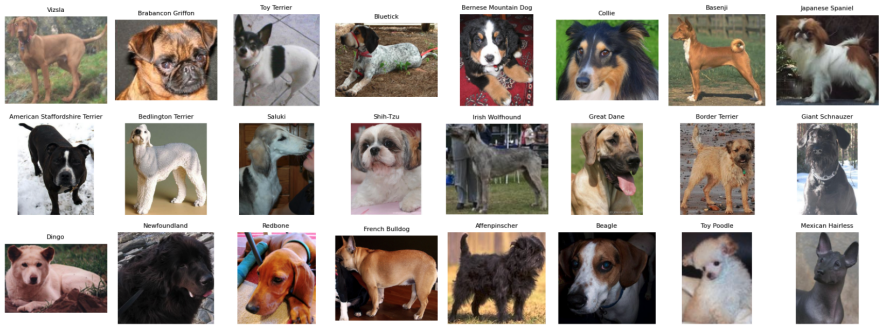

Fig-6: Dog Images

By checking the real Dog Images, their sizes vary and the presence of dogs is inconsistent. Their complex background would be distracting elements for our research, since some images are dominated by complex backgrounds rather than the dogs themes. Our goal is to create images predominantly featuring dogs. How to identify the dog features and eliminate the background influence would be the main challenge for us.

EDA(Exploratory Descriptive Analysis)

In the EDA section, we analyzed the distribution of images across different breeds to understand the dataset's composition better. Additionally, we explored the impact of various image augmentation techniques to enhance our dataset's diversity and robustness. This involved manipulating images in different ways, such as resizing, rotating, or altering brightness, to create a more comprehensive training set for our model. We also investigated the color distribution within the RGB channels to gain insights into the color patterns prevalent in the dataset.

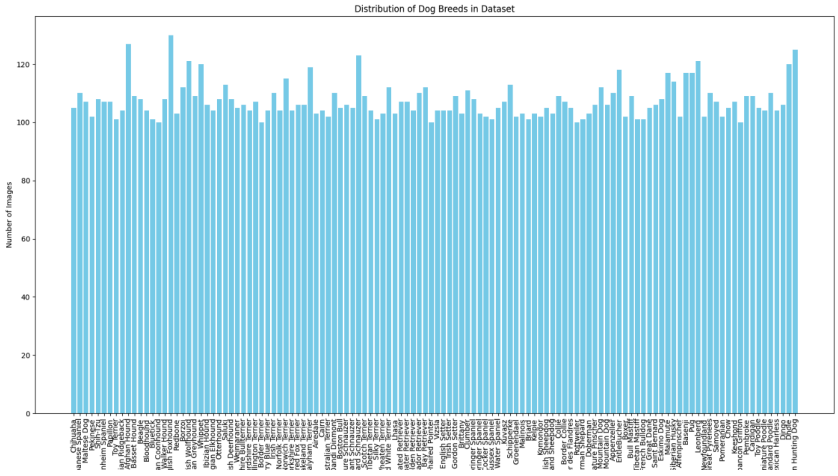

Fig-7: Dog Images Distribution

Fig-8: Augmented Dog Images

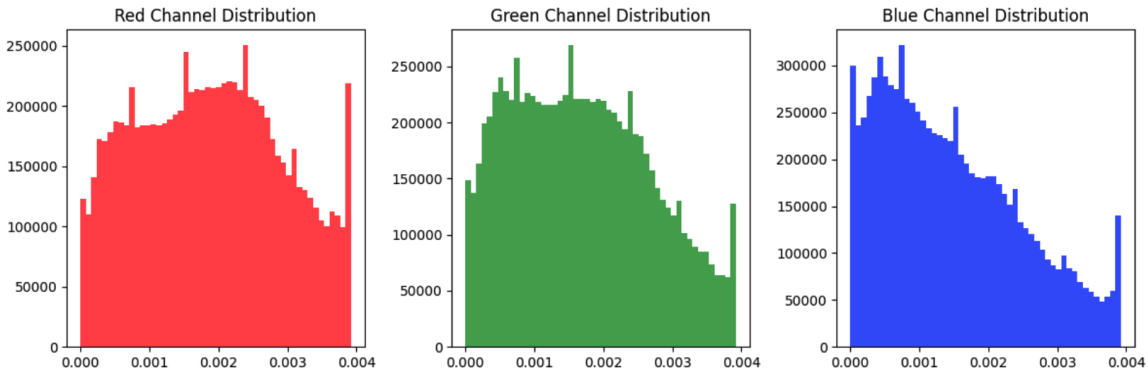

Fig-9: RGB Channels

NC = 3 (Number of Channels)
NZ = 100 (Size of z latent vector)
NDF = 64 (Size of feature maps in discriminator)
NGF = 64 (Size of feature maps in generator)

Generator

Layer (Type)	**Input Shape**	**Output Shape**	**Details**
ConvTranspose2d	(NZ, 1, 1)	(NGF * 8, 4, 4)	4x4 kernel, stride 1, padding 0, no bias
BatchNorm2d			Applies Batch Normalization
ReLU			Applies ReLU activation function
Dropout			p = 0.2
ConvTranspose2d	(NGF * 8, 4, 4)	(NGF * 4, 8, 8)	4x4 kernel, stride 2, padding 1, no bias
BatchNorm2d			Applies Batch Normalization
ReLU			Applies ReLU activation function
Dropout			p = 0.2
ConvTranspose2d	(NGF * 4, 8, 8)	(NGF * 2, 16, 16)	4x4 kernel, stride 2, padding 1, no bias
BatchNorm2d			Applies Batch Normalization

ReLU			Applies ReLU activation function
Dropout			p = 0.2
ConvTranspose2d	(NGF * 2, 16, 16)	(NGF, 32, 32)	4x4 kernel, stride 2, padding 1, no bias
BatchNorm2d			Applies Batch Normalization
ReLU			Applies ReLU activation function

Discriminator

Layer (Type)	**Input Shape**	**Output Shape**	**Details**
Conv2d	(NC, 64, 64)	(NDF, 32, 32)	4x4 kernel, stride 2, padding 1, no bias
LeakyReLU			negative_slope=0.3
Conv2d	(NDF, 32, 32)	(NDF * 2, 16, 16)	4x4 kernel, stride 2, padding 1, no bias
BatchNorm2d			Applies Batch Normalization
LeakyReLU			negative_slope=0.3
Dropout			p = 0.2
Conv2d	(NDF * 2, 16, 16)	(NDF * 4, 8, 8)	4x4 kernel, stride 2, padding 1, no bias
BatchNorm2d			Applies Batch Normalization
LeakyReLU			negative_slope=0.3
Dropout			p = 0.2

Conv2d	(NDF * 4, 8, 8)	(NDF * 8, 4, 4)	4x4 kernel, stride 2, padding 1, no bias
BatchNorm2d			Applies Batch Normalization
LeakyReLU			negative_slope=0.3
Conv2d	(NDF * 8, 4, 4)	(1, 1, 1)	4x4 kernel, stride 1, padding 0, no bias
Sigmoid			Applies Sigmoid activation function

- 20% Dropout on Generator's three hidden layers and Discriminator's two hidden layers and 40 Degree Random Rotation

Reason : Dropout is a regularization technique used to prevent overfitting; it can encourage the generator to explore a wider range of outputs, potentially leading to more diverse and realistic image generation. Also A 20% dropout rate in two hidden layers of the discriminator can help the discriminator from becoming too powerful compared to the generator, especially in the early phases of training. By weakening the discriminator slightly, the generator has a better chance of improving its image generation capabilities without being overwhelmed by the discriminator's accuracy. And 40-degree rotation can also add diversity to the dataset without introducing unrealistic or overly distorted images

Fig-10.1: Fine-tuning Results 1

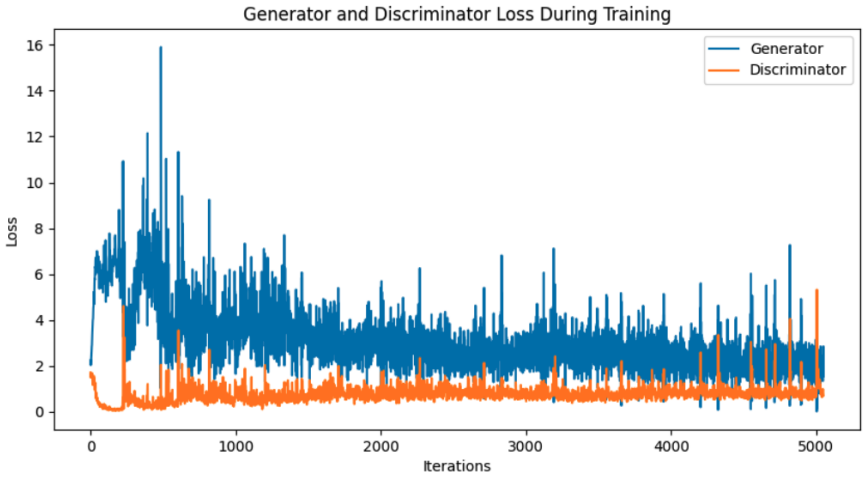

Fig-10.2: Fine-tuning Results 1

Interpretation: From the fake image showcase, it shows that our model is able to capture those colorful information and some dogs' shapes, but fails to present the dog with detailed information. The loss plot shows that there is some fluctuation in the first 1000 iterations, which indicates some model instability.

- Add LayerNorm in Discriminator

Reason: It would normalize the activations of the previous layer for each data sample, making the training less sensitive to the scale of weights and the initial values. It may improve the training stability, make the model less sensitive to choice of hyperparameters.

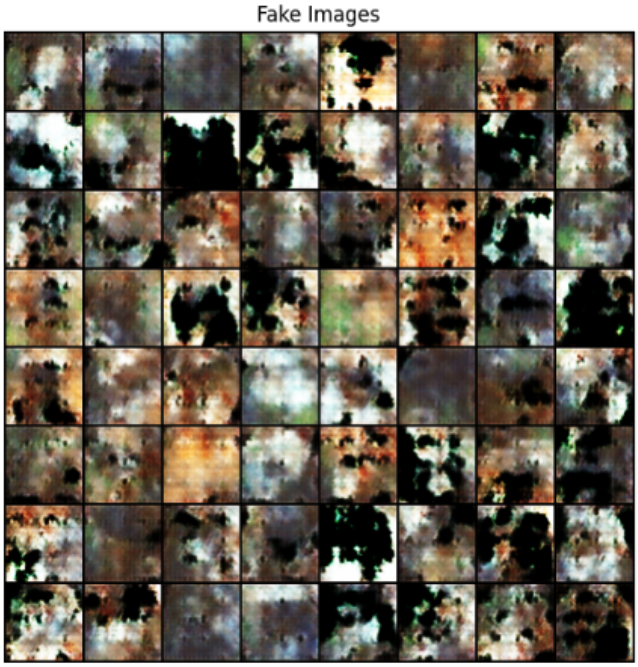

Fig-11.1: Fine-tuning Results 2

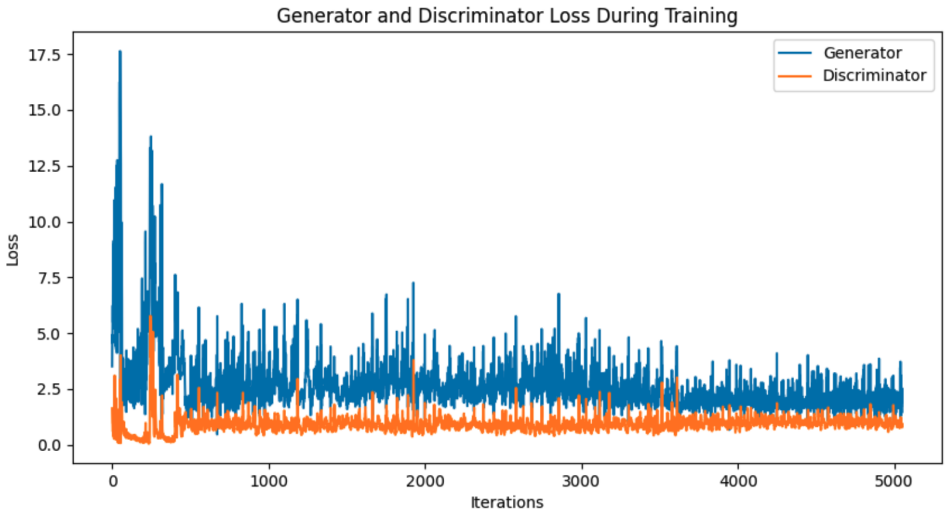

Fig-11.2: Fine-tuning Results 2

Interpretation: After this fine-tuning techniques, it is clear to see that the outcome of fake images are not descent, even though from the loss plot, the model tends to be stable after around 500 iterations.

- Apply LeakyReLu with 0.2 both on Generator and Discriminator

Reason: The "leak" in LeakyReLU is defined by a small slope with 0.2, which means that LeakyReLU will output some small negative value when the input is negative.The implementation in Generator can help Generator maintaining diversity in generated images, as it

prevents the loss of gradient information throughout the Layers. Similarly for Discriminator, LeakyReLU ensures that all neurons remain active and contribute to distinguishing between real and fake images, which is vital for effectively training the generator.

Fig-12.1: Fine-tuning Results 3

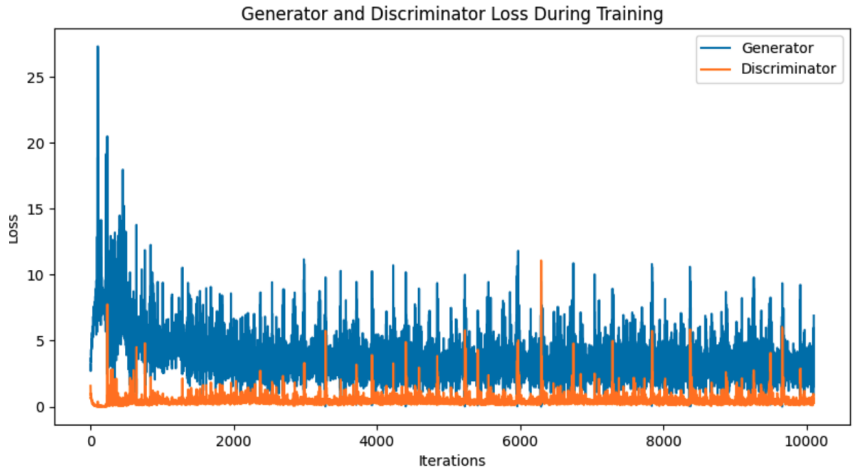

Fig-12.2: Fine-tuning Results 3

Interpretation: Compared with previous fine-tuning, this result shows a much better performance. It is able to capture the shape features but still fails to capture the detailed information.

- BETA = 0.8, Giving higher weights on recent gradient

Reason: By setting Beta1 to 0.8, we are trying to place more emphasis on the more recent gradients as opposed to the historical average. This means the optimizer will be more responsive to recent changes in the gradient, which can lead to faster adjustments in the model weights.

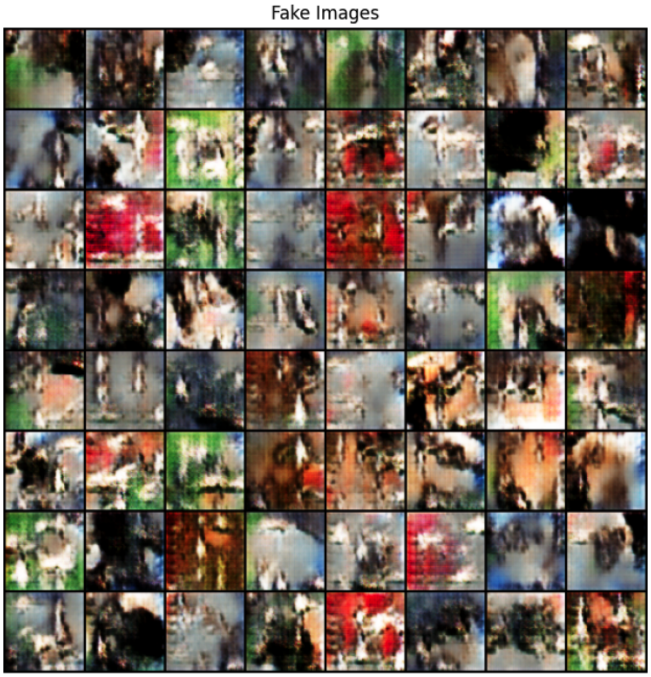

Fig-13.1: Fine-tuning Results 4

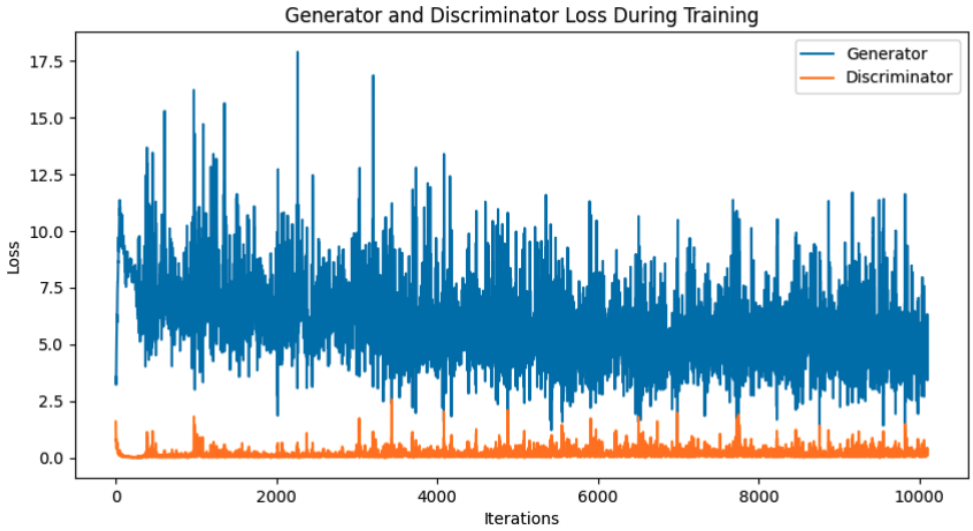

Fig-13.1: Fine-tuning Results 4

Interpretation: By adjusting the BETA value, the model turns worse. And from the loss plot, the Discriminator is overpowering the Generator, which means that it is difficult for Generator to improve.

Summary and Conclusion

Throughout the course of this endeavor, I gained invaluable experience in performing Exploratory Data Analysis (EDA) tailored specifically for the realm of image generation tasks. I became well-versed in the structure and construction of Deep Convolutional Generative Adversarial Networks (DCGANs), from their foundational principles to the intricate process of assembling and fine-tuning such a model.

The fine-tuning process proved to be particularly challenging, yet enlightening. I adhered to the tuning recommendations provided in our coursework materials, which guided me through the meticulous adjustments required to refine the model's performance. Although the final iteration of my model did not achieve the level of precision necessary to create highly realistic images of dogs, the progress it made was noticeable.

Despite the fact that the end result fell short of generating high-fidelity canine portraits, the exploration into DCGANs was a compelling and educational journey. It highlighted the complexities and nuances of machine learning and deep learning models, providing a clear perspective on both their capabilities and limitations. This project served as a practical testament to the iterative nature of AI model development, where each iteration contributes to a deeper understanding and sets the stage for future advancements.

The lessons learned from this project extend beyond technical skills, touching on the patience and perseverance required when venturing into the cutting-edge field of artificial intelligence. This experience has laid a solid foundation for my ongoing development in the field and has sparked a keen interest in overcoming the challenges presented by DCGANs and similar advanced generative models.

Reference

Avinash H. (2017). The GAN Zoo. GitHub. https://github.com/hindupuravinash/thegan-zoo
Jonathan H. (2018). GAN — Ways to improve GAN performance. Towards Data Science. https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b
Jonathan H. (2018). GAN — Why it is so hard to train Generative Adversarial Networks! Medium. https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-traingenerative-advisory-networks-819a86b3750b
Jonathan H. (2018). GAN — DCGAN (Deep convolutional generative adversarial networks) . Medium. https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutionalgenerative-adversarial-networks-df855c438f