Generate Fake Dog images With GAN

DATS 6303 Group 7

Cody Yu, Kismat Khatri, Jeffrey Hu, Ei Tanaka

Outlines

- Introduction of GAN
- EDA
- Model Description
- Experimental Setup
- Results
- Challenges

Introduction of GAN

Image Generator

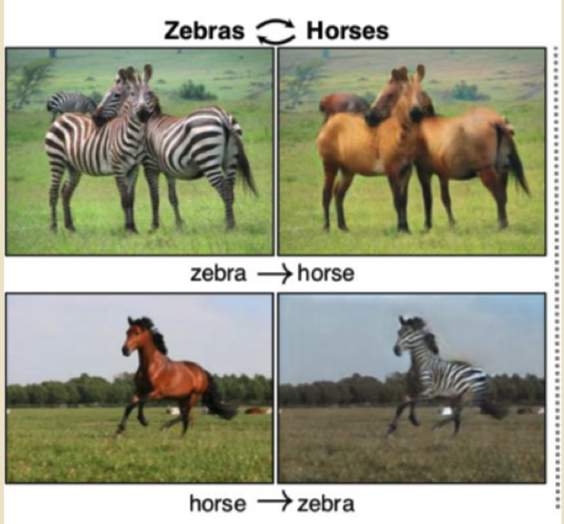
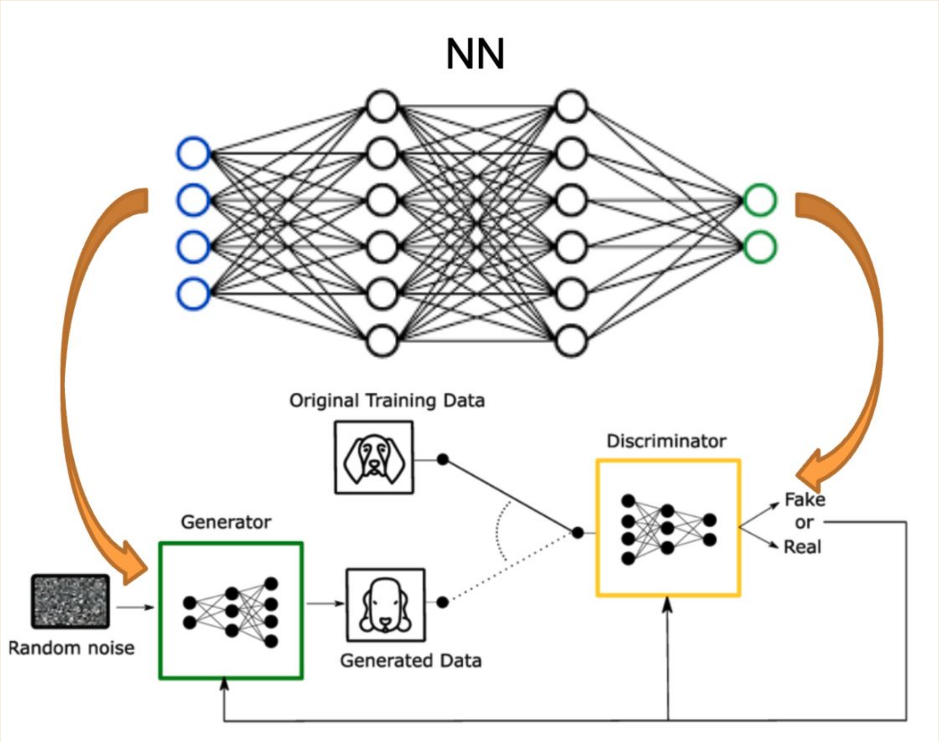


Image-to-Image translation

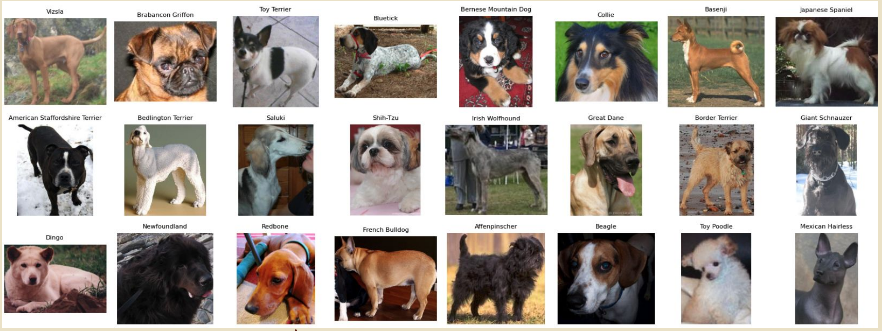
Text-to-Image translation

Research Target

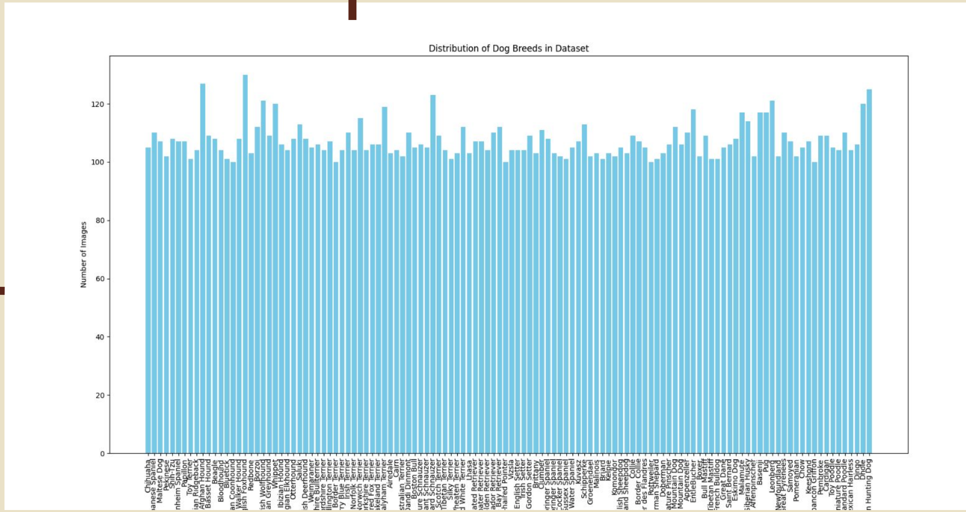
The choice of this problem is driven by the complexity and diversity of dog features, which poses an exciting challenge in synthetic image generation



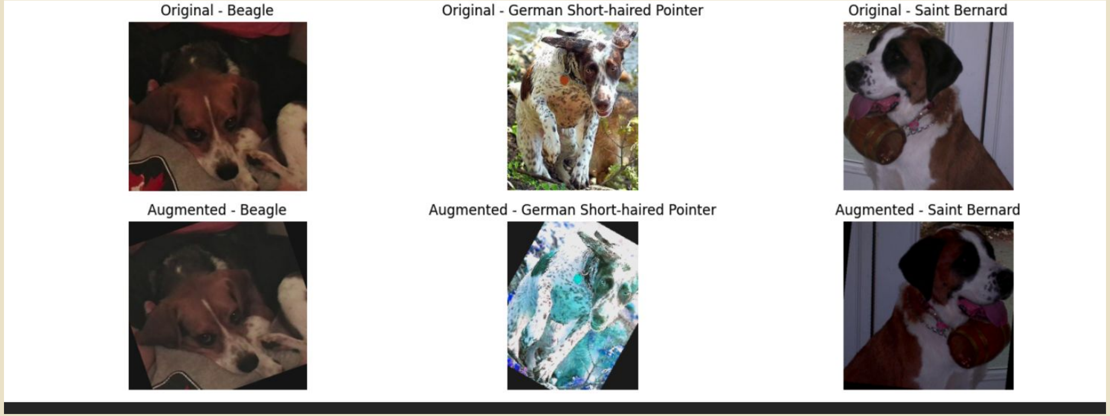
EDA



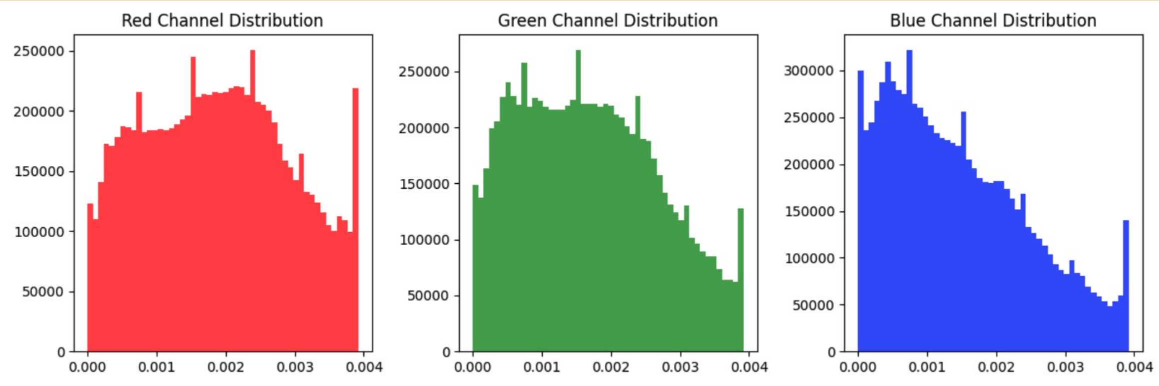
	Dog Breed	Number of Observations
0	n02085620-Chihuahua	152
1	n02085782-Japanese_spaniel	185
2	n02085936-Maltese_dog	252
3	n02086079-Pekinese	149
4	n02086240-Shih-Tzu	214
.	.	.
.	.	.
.	.	.
116	n02113978-Mexican_hairless	155
117	n02115641-dingo	156
118	n02115913-dhole	150
119	n02116738-African_hunting_dog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



EDA



Augmented Images



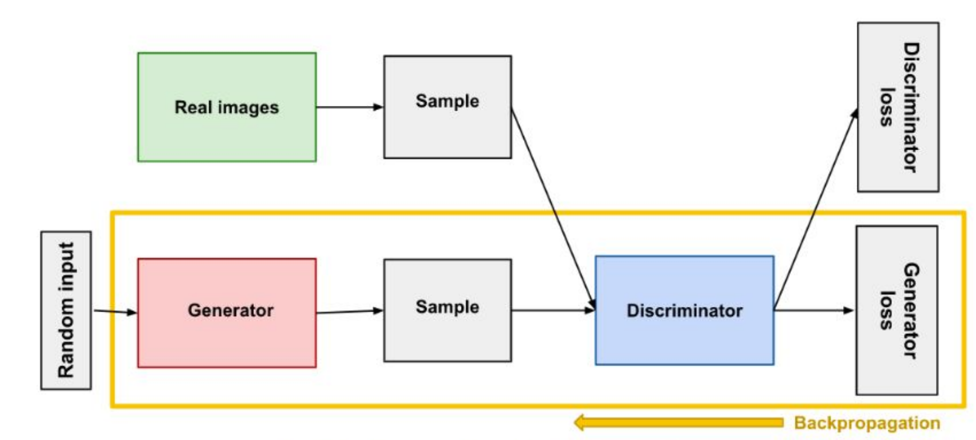
Color Distribution

Model Architecture

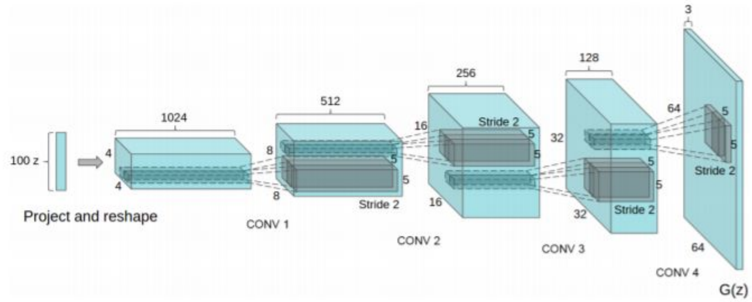
- DCGAN (Deep Convolutional Generative Adversarial Network)
- UNet2D Diffuser Model

Model Description - DCGAN

- Convolutional Layers
- Eliminating Fully Connected Layers
- No Pooling Layers
- Batch Normalization
- ReLu / Leaky ReLu
- Stridden Convolutions and Transposed Convolutions



Source: https://developers.google.com/machine-learning/gan/generator



Source: From the paper "Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks"

Model Description - DCGAN

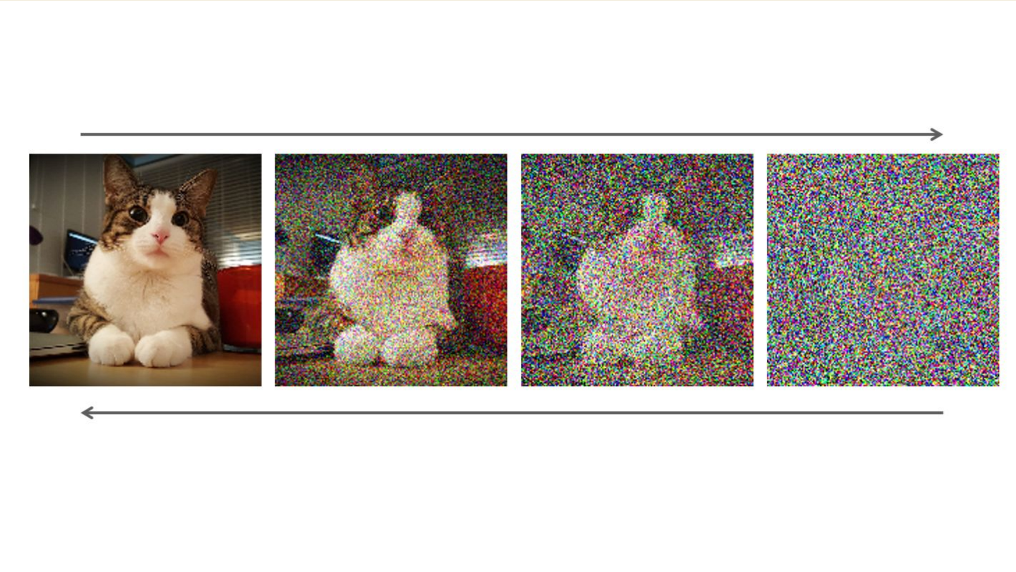
```python
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # Input is Z, going into a convolution
            nn.ConvTranspose2d(NZ, NGF*8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(NGF*8),
            nn.ReLU(True),
            # State size: (NGF*8) x 4 x 4
            nn.ConvTranspose2d(NGF*8, NGF*4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(NGF*4),
            nn.ReLU(True),
            # State size: (NGF*4) x 8 x 8
            nn.ConvTranspose2d(NGF*4, NGF*2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(NGF*2),
            nn.ReLU(True),
            # State size: (NGF*2) x 16 x 16
            nn.ConvTranspose2d(NGF*2, NGF, 4, 2, 1, bias=False),
            nn.BatchNorm2d(NGF),
            nn.ReLU(True),
            # State size: NGF x 32 x 32
            nn.ConvTranspose2d(NGF, NC, 4, 2, 1, bias=False),
            nn.Tanh()
            # State size: NC x 64 x 64
        )
```

```python
class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main= nn.Sequential(
            # Input is (NC) x 64 x 64
            nn.Conv2d(NC, NDF, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # State size: NDF x 32 x 32
            nn.Conv2d(NDF, NDF*2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(NDF*2),
            nn.LeakyReLU(0.2, inplace=True),
            # State size: (NDF*2) x 16 x 16
            nn.Conv2d(NDF*2, NDF*4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(NDF*4),
            nn.LeakyReLU(0.2, inplace=True),
            # State size: (NDF*4) x 8 x 8
            nn.Conv2d(NDF*4, NDF*8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(NDF*8),
            nn.LeakyReLU(0.2, inplace=True),
            # State size: (NDF*8) x 4 x 4
            nn.Conv2d(NDF*8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

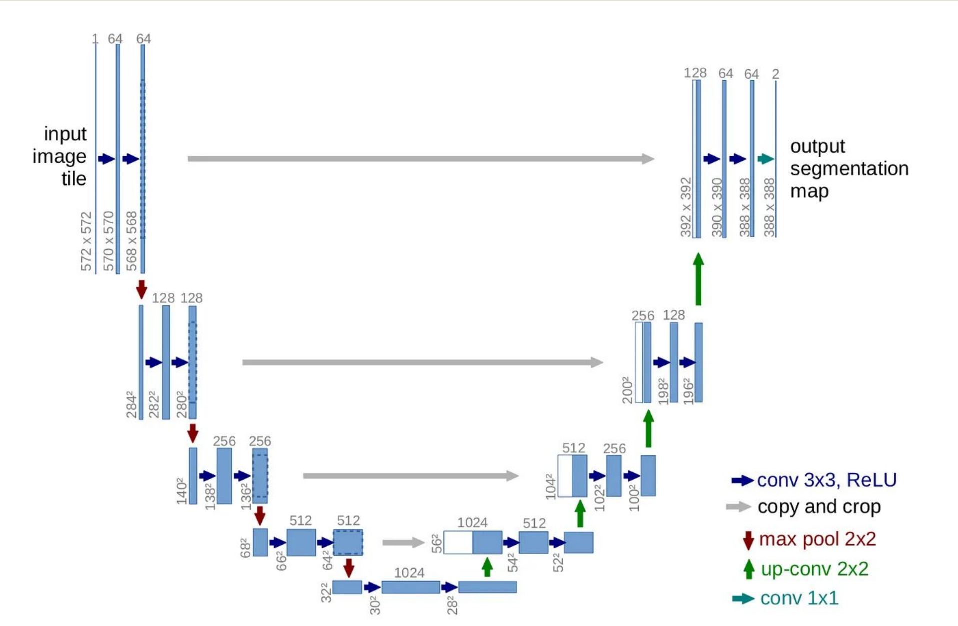
Model Description - Diffusion Model

- Model History and Application
- Dall-E, Stable Diffusion
- Diffusion vs GAN Models
- Noise Introduction



Model Description - UNet2D

- Img2Img
- UNet2D
- Model Architecture
- Noise Scheduler DDPM
- Variations and Current Applications

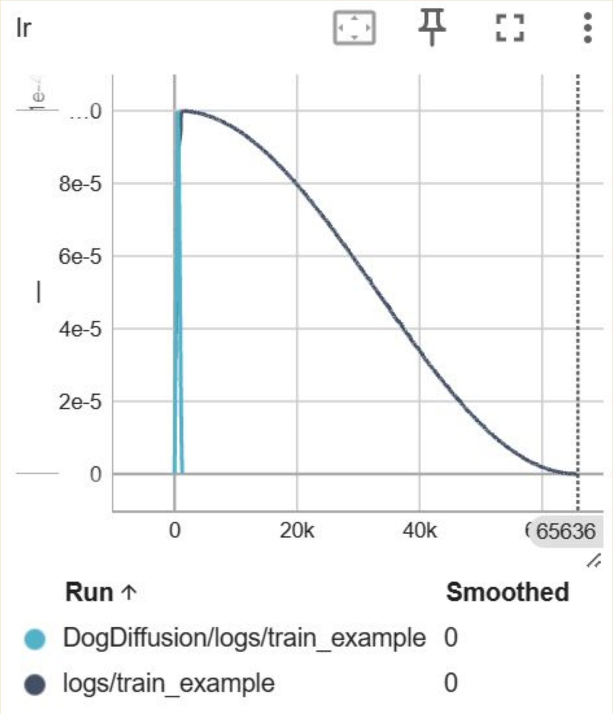


Train Details - DCGANS

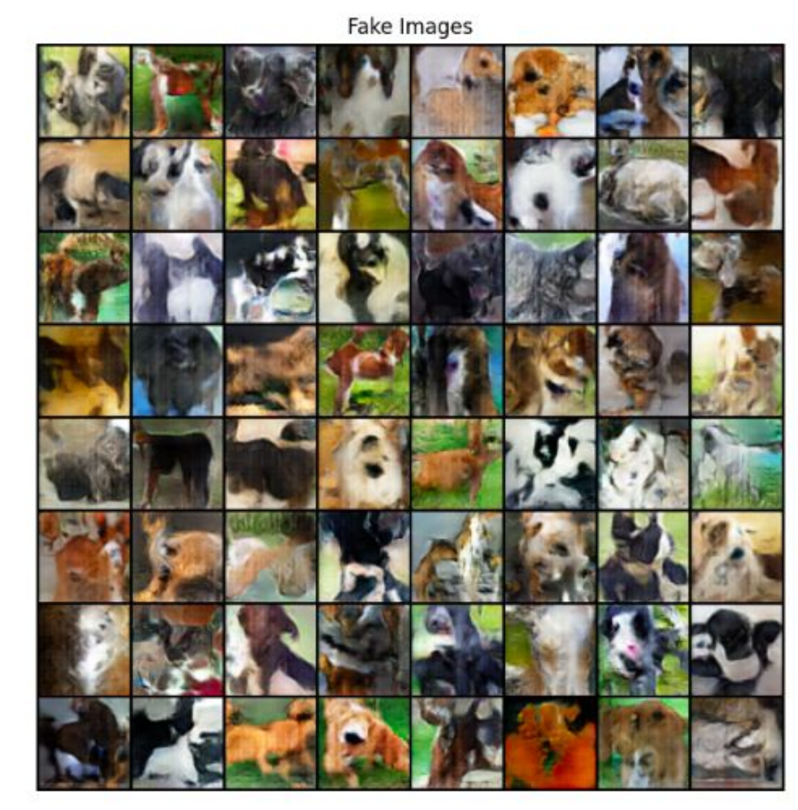
- Preprocessing, scaled to range of tanh activation [-1, 1]
- Batch_Size = 64
- Image_size = 64
- Num_Epochs = 50
- Adam Optimizer Learning Rate = 0.001 with BETA = 0.5
- The weights of Convolution layers : A normal distribution with a mean of 0.0 and a standard deviation of 0.02.

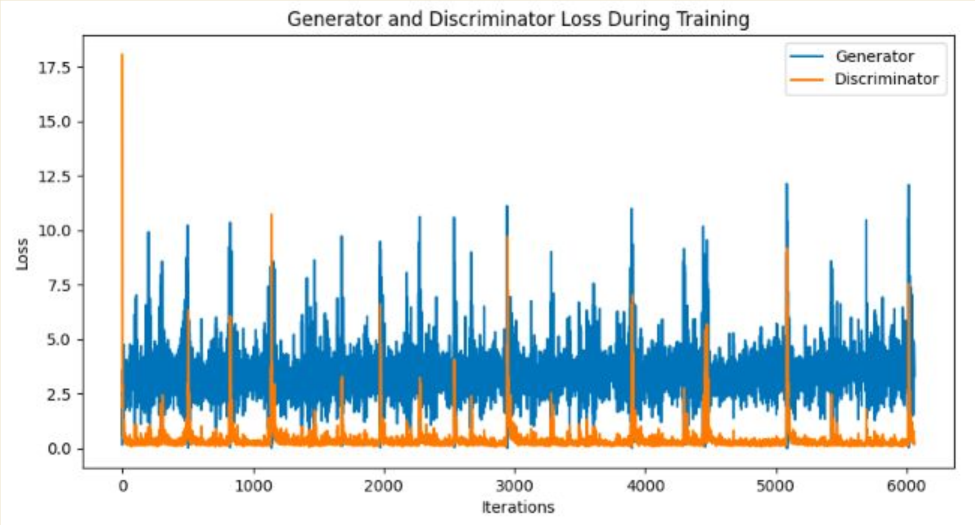
Train Details - Unet2D Diffuser

- Preprocessing, Image Resizing, Random Horizontal flips
- Batch_Size = 16
- Image_size = 128
- Num_Epochs = 50
- AdamW Optimizer Learning Rate = 0.001



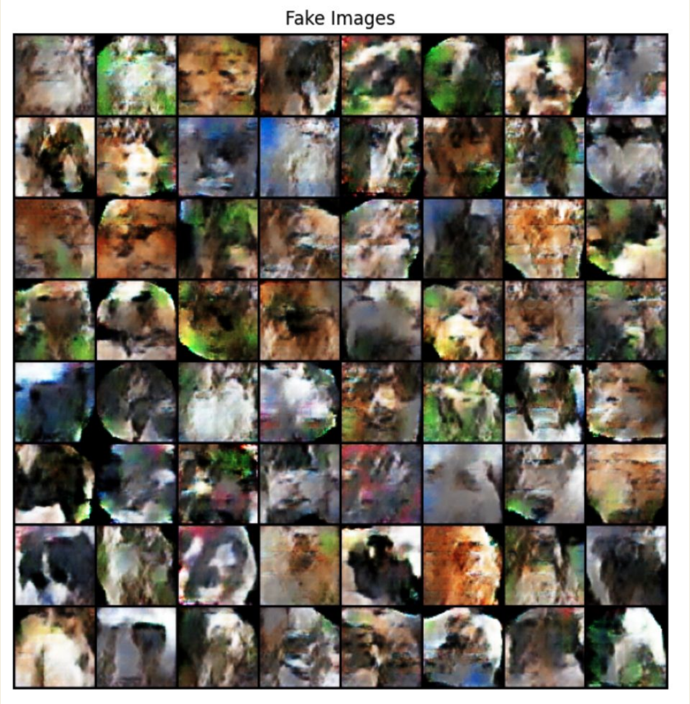
Results - DCGAN

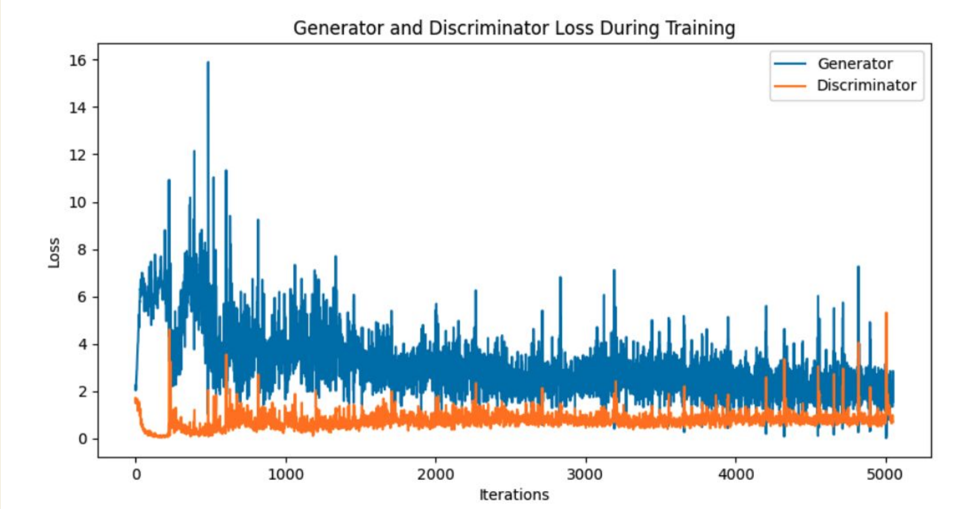




Fine-Tuning Results

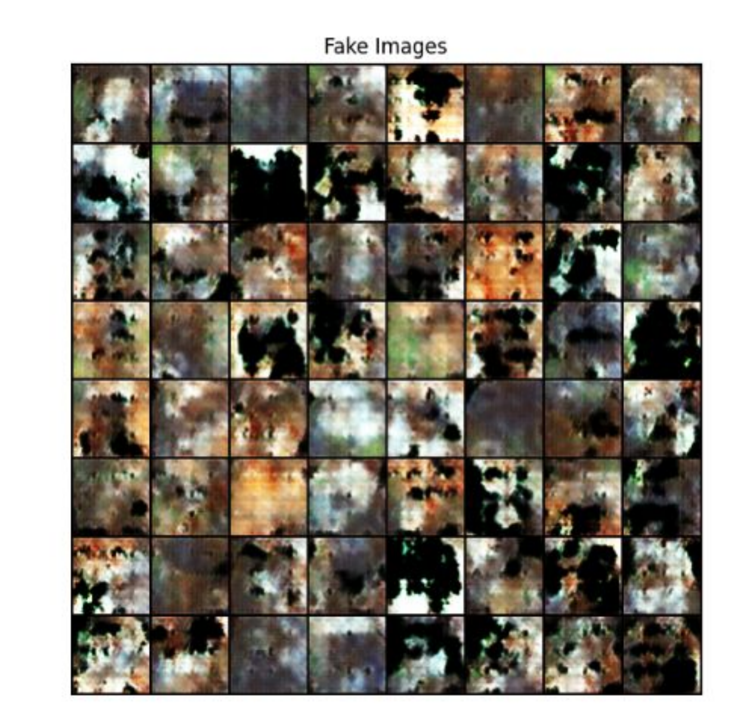
- Add 20% Dropout in G and D and 40 Degree Random Rotation

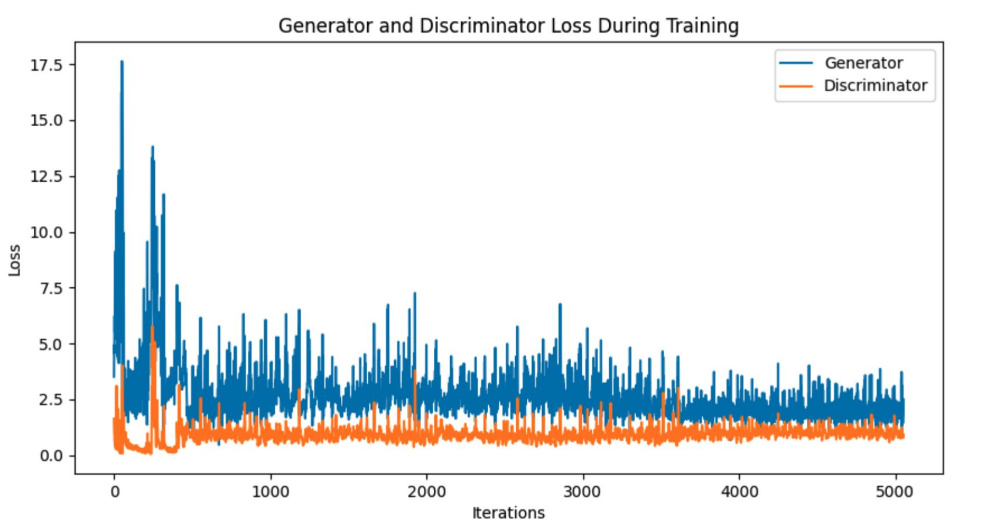




Fine-Tuning Results

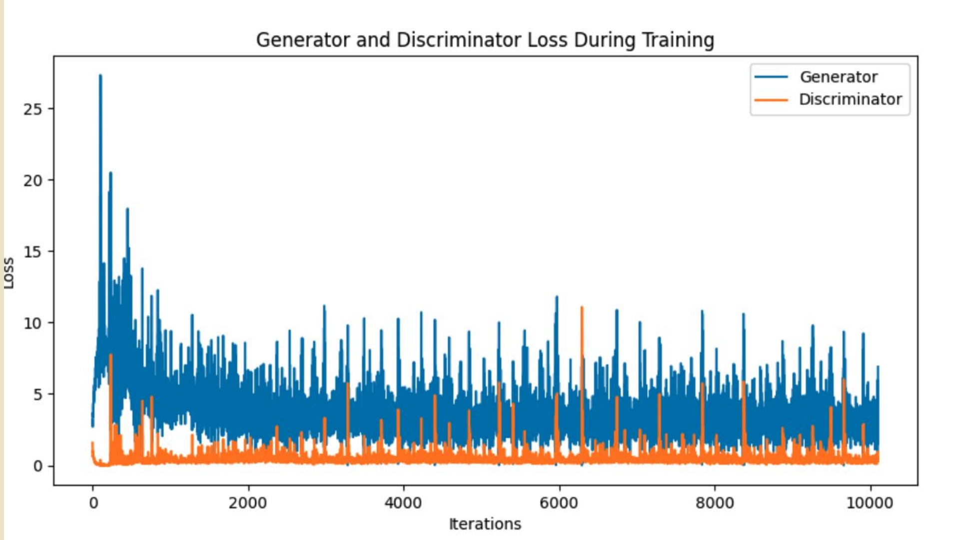
- Add LayerNorm in Discriminator





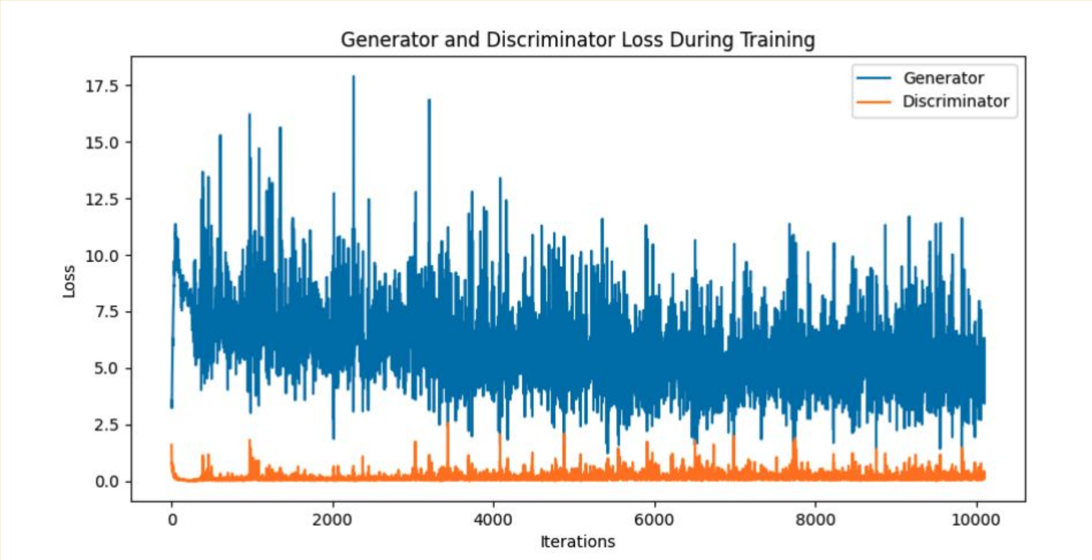
Fine-Tuning Results

- Apply LeakyReLu with 0.2 both on G and D



Fine-Tuning Results

- BETA = 0.8, Giving higher weights on recent gradient



Challenges

- Imbalanced capacity between G and D
- No Evaluation Metrics
- More Hyperparameter training
- Different Types GAN : BEGAN, StyleGANs
- Different Types of Diffusers : Img2Img vs Text2Img
- Image Resolution and Model Constraints