1 Introduction

A generative adversarial network (GAN) is a class of machine learning systems invented by Ian Goodfellow and his colleagues in 2014. Two neural networks contest with each other in a game (in the sense of game theory, often but not always in the form of a zero-sum game). Given a training data set, this technique learns to generate new data with the same statistics as the training data set. For instance, a GAN trained on photographs can then generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics. Although originally proposed as a form of generative model for unsupervised learning, GANs have also proven useful for semi-supervised learning, fully supervised learning, and reinforcement learning.

Meanwhile, there are many different types of GANs. In this project we student have to implement two types of Generative Adversarial Networks (GANs): One known as the Deep Convolution GAN (DCGAN); the other known as Conditional Self-Attention Wasserstein GAN (SA-GAN) using CIFAR-10 dataset.

2 DCGAN

Basic DCGAN consists of two neural networks: a discriminator (D) and a generator (G) which compete with each other making each other stronger at the same time. One of the simple variants of GAN in Deep Convolutional GANs (DCGANs) in which G and D are both based on deep convolution neural network.

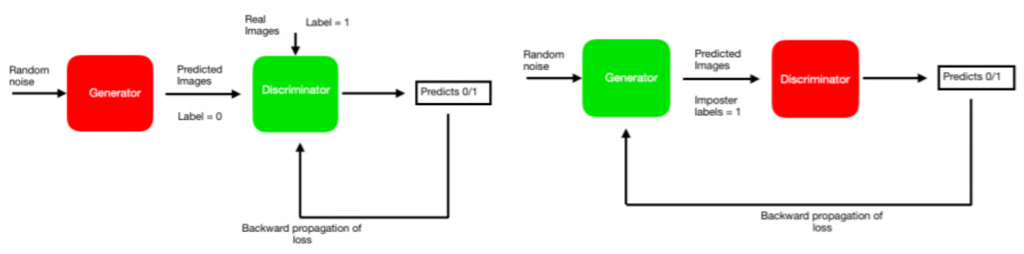

Figure 2-1: Training a basic Generative Adversarial Network. Figure on the left and right shows the training of a Generator and Discriminator respectively.

A DCGAN is a direct extension of the GAN, except that it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively. The discriminator is made up of strided convolution layers, batch norm layers, and LeakyReLU activations. The input is a 3x32x32 input image and the output is a scalar probability that the input is from the real data distribution. The generator is comprised of convolutional-transpose layers, batch norm layers, and ReLU activations. The input is a latent vector-z, that is drawn from a standard normal distribution and the output is a 3x32x32 RGB image. The strided convolutional-transpose layers allow the latent vector to be transformed into a volume with the same shape as an image.

3 SAGAN

3.1 The implementation of the SAGAN model (including the attention component) and the training procedure

Self-Attention Generative Adversarial Network (SAGAN) model introduces a self-attention mechanism into convolutional GANs. The self-attention module is complementary to convolutions and can help with modeling long distance, multi-level dependencies across image regions. Armed with self-attention, the generator can draw images in which fine details at every location are carefully coordinated with fine details in distant portions of the image. Moreover, the discriminator can also more accurately enforce complicated geometric constraints on the global image structure. SAGAN can apply spectral normalization to the GAN generator to improve training dynamics. It can also apply spectral normalization to GAN discriminator by substituting batch norm layers. Therefore, we can use two techniques to stabilize the training procedure of SAGAN. First, use spectral normalization in the generator as well as in the discriminator. Second, use the two-timescale update rule (TTUR) in order for the GAN to converge to a local Nash equilibrium.

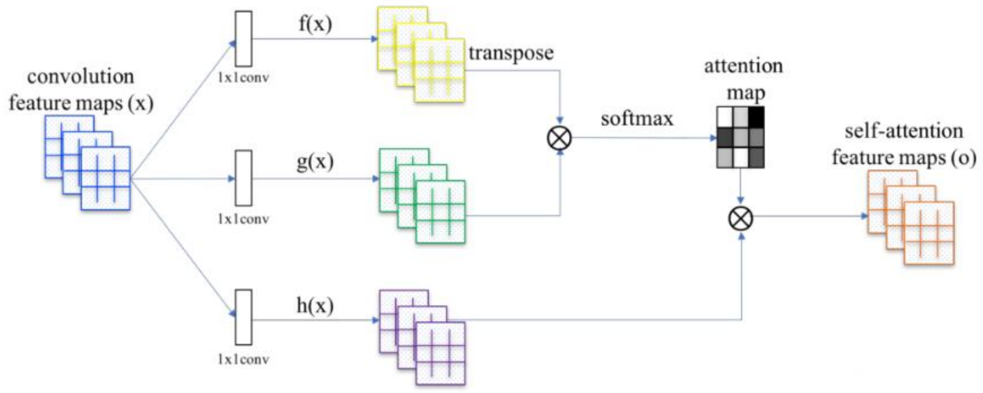

Figure 3-1: Architecture of a Self Attention module. The X denotes matrix multiplication. The softmax operation is performed on each row. This module is stacked after a selected convolution block present in a G and D.

3.2 SAGAN and its difference from DCGAN

Self-Attention Generative Adversarial Network (SAGAN) model introduces a self-attention mechanism into convolutional GANs. Self-Attention Generative Adversarial Network (SAGAN) model allows attention-driven, long-range dependency modeling for image generation tasks. DCGANs generate high-resolution details as a function of only spatially local points in lower-resolution feature maps. Therefore, in SAGAN, details can be generated using cues from all feature locations. Moreover, the discriminator can check that highly detailed features in distant portions of the image are consistent with each other. Because generator conditioning affects GAN performance, SAGAN can apply spectral normalization to the GAN generator to improve training dynamics. It can also apply spectral normalization to GAN discriminator by substituting batch norm layers. Meanwhile, we often set different imbalanced learning rate for generator and discriminator updates in SAGAN for a local Nash equilibrium.

3.3 Spectral Normalization

Spectral normalization is a stabilizer of training of GANs. One of the challenges in the study of generative adversarial networks is the instability of its training. Spectral normalization can stabilize the training of the discriminator and prevent the transformation of each layer from becoming to sensitive in one direction. We can also use spectral normalization to devise a new parametrization for the model. While applying spectral normalization to the GANs on image generation tasks, the generated examples are more diverse than the conventional weight normalization and achieve better or comparative inception scores relative to previous studies. In the absence of complimentary regularization techniques, spectral normalization can improve the sheer quality of the generated images better than weight normalization and gradient penalty.

3.4 Self Attention and why it is beneficial to many machine learning models

Attention mechanisms are an integral part of models that capture global dependencies. A self-attention module computes the response at a position in a sequence (e.g., a sentence) by attending to all positions and taking their weighted average in an embedding space.

For image data, long-distance dependencies are modeled by the large receptive fields formed by deep stacks of convolutional operations. Convolutional and recurrent operations both process a local neighborhood, either in space or time; thus long-range dependencies can only be captured when these operations are applied repeatedly, propagating signals progressively through the data. Repeating local operations has several limitations. First, it is computationally inefficient. Second, it causes optimization difficulties that need to be carefully addressed. Finally, these challenges make multihop dependency modeling difficult.

Self-attention can be considered as a form of the non-local mean. Therefore, it can be used from machine translation to the more general class of non-local filtering operations that are applicable to image and video problems in computer vision.

4 DCGAN Model Performance:

1 epoch = 1000 iterations and trained DCGAN with 50 epochs (50000 iterations).

4.1 Training Curve of FID Score

I calculated FID Scores separately using '10000 generated images and 10000 testing images', '1024 generated images and 1024 testing images' and '128 generated images and 128 testing images' at every 10000 iterations (1 epoch).

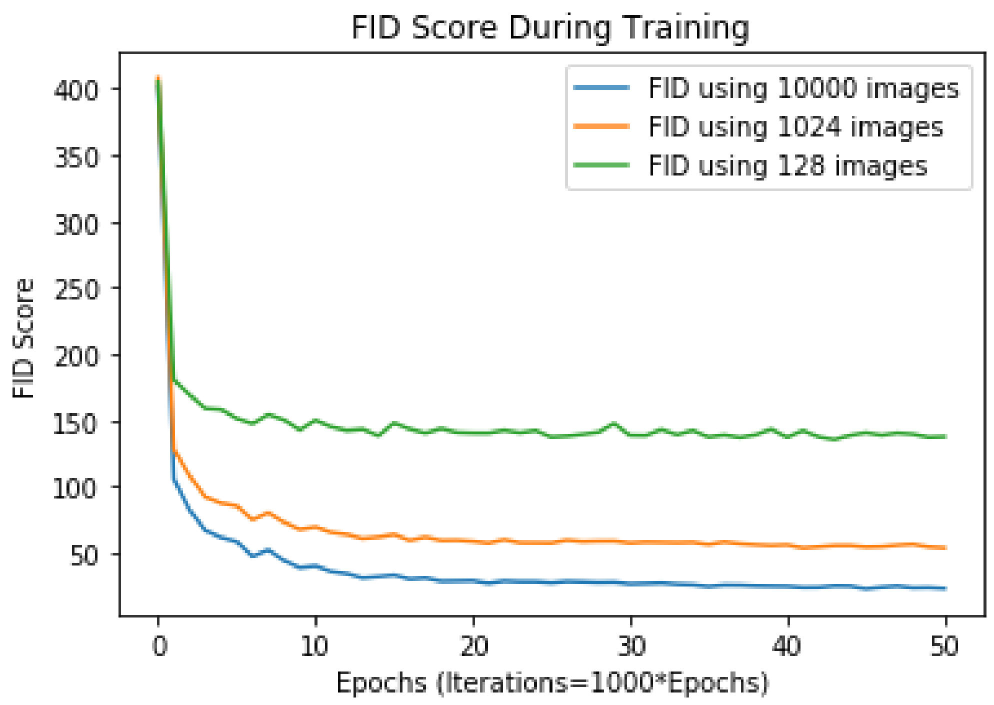

Figure 4-1: FID Scores over the trained epochs

And I also calculated the FID Scores using two real images sets (training images and testing images) to make comparison with FID Scores of 'generated images and testing images' from optimal DCGAN Model. Therefore, I calculated FID Scores separately using '10000 training images and 10000 testing images', '1024 training images and 1024 testing images' and '128 training images and 128 testing images'.

Amounts of Images	The FID Score of Generated images and Testing images	The FID Score of Training images and Testing images
128	140.79	130.18
1024	54.59	39.28
10000	23.19	4.36

4.2 Grid of Generated Images

Because the project1 description requires 8 by 8 image grid and the images become clear fast, I add all images which generated at every 10000 iterations into the report.

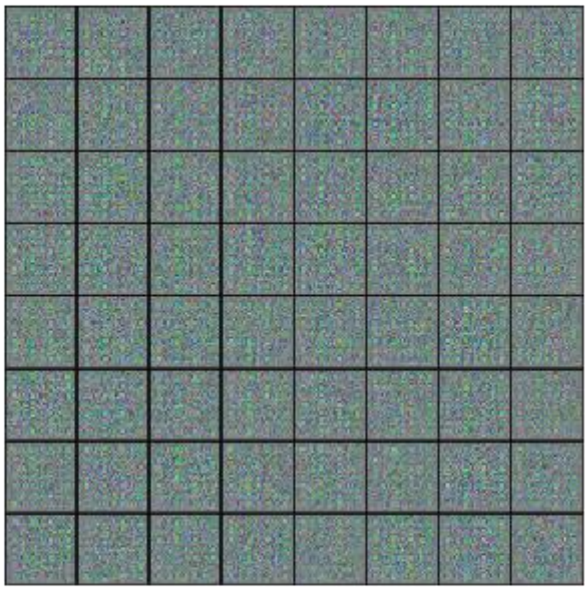

Figure 4-2: Generated Images Grid (Epoch = 0)

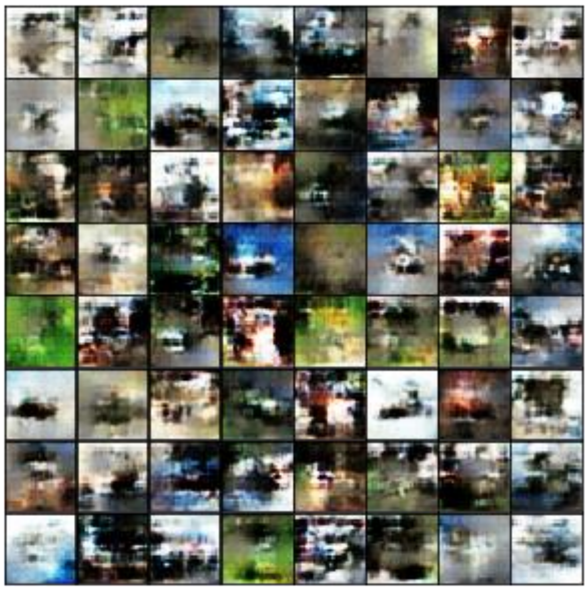

Figure 4-3: Generated Images Grid (Epoch = 1)

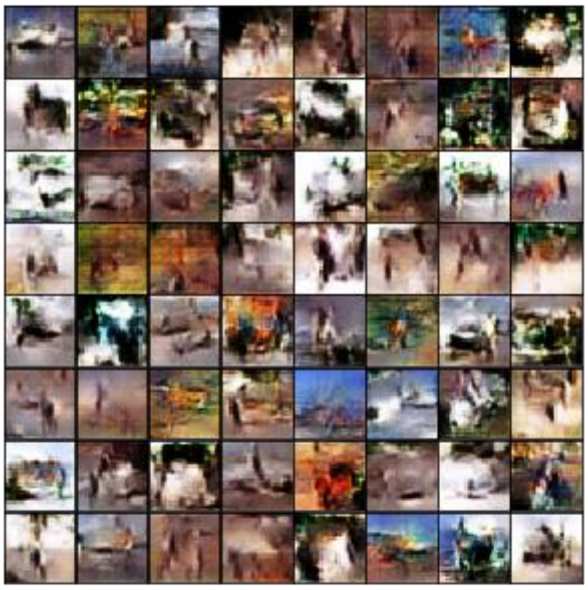

Figure 4-4: Generated Images Grid (Epoch = 2)

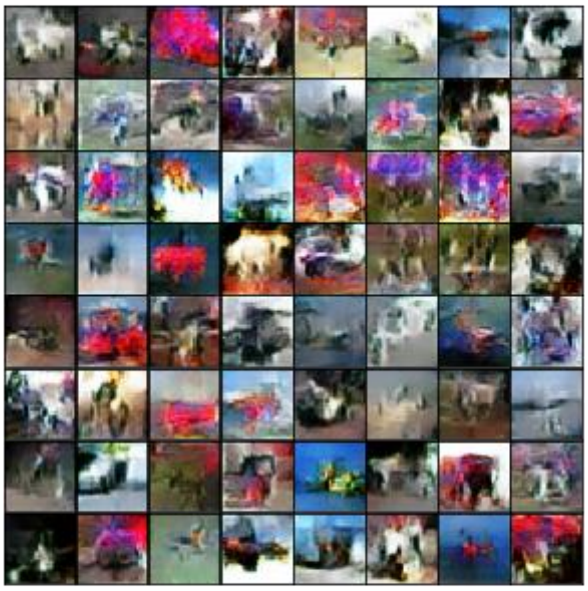

Figure 4-5: Generated Images Grid (Epoch = 3)

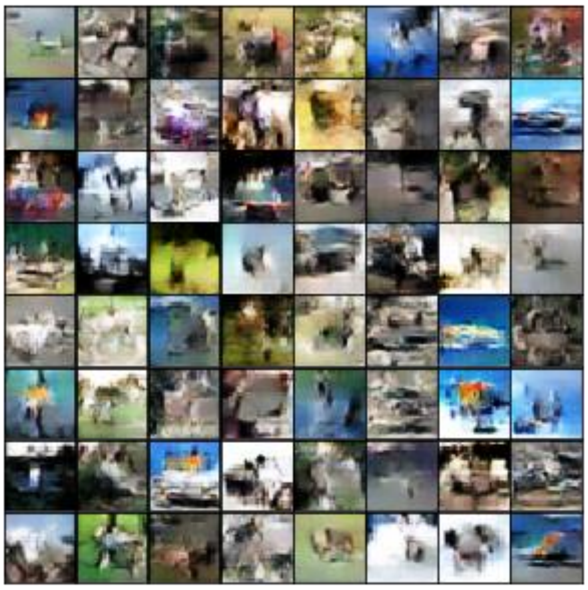

Figure 4-6: Generated Images Grid (Epoch = 4)

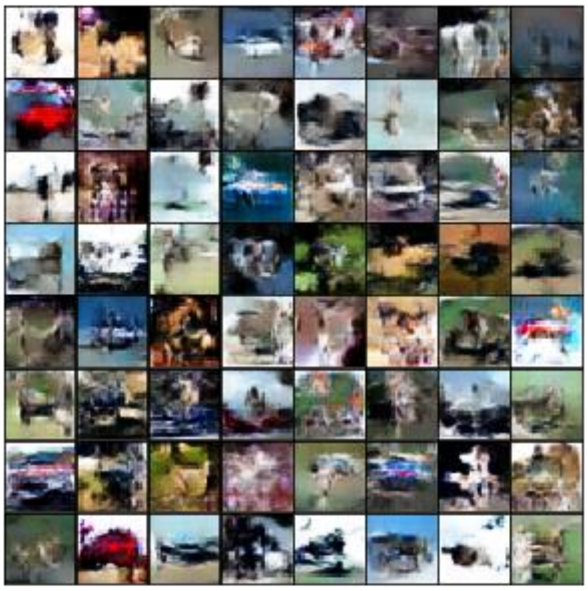

Figure 4-7: Generated Images Grid (Epoch = 5)

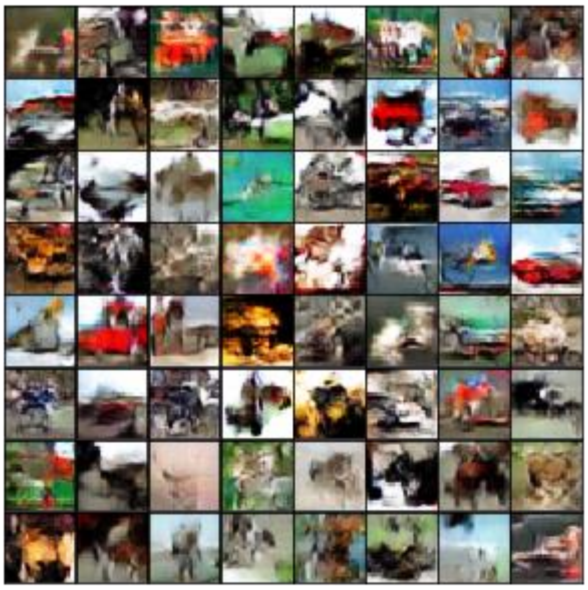
Figure 4-8: Generated Images Grid (Epoch = 6)

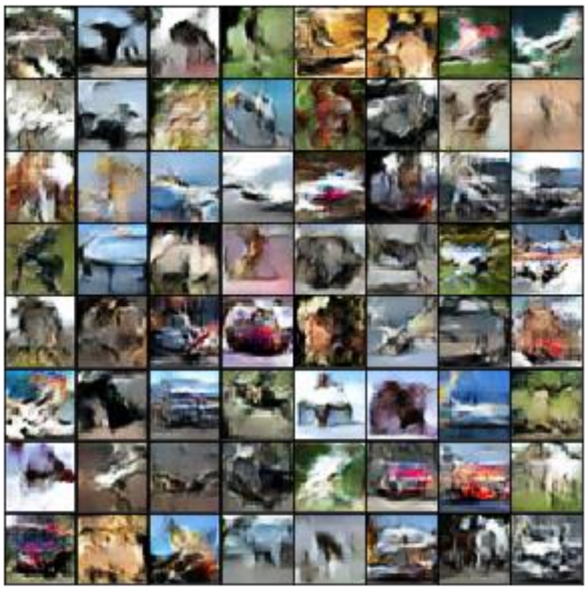
Figure 4-9: Generated Images Grid (Epoch = 7)

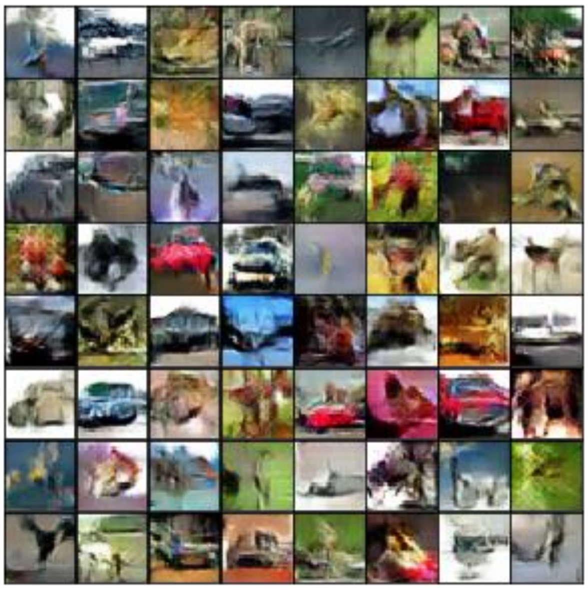

Figure 4-10: Generated Images Grid (Epoch = 8)

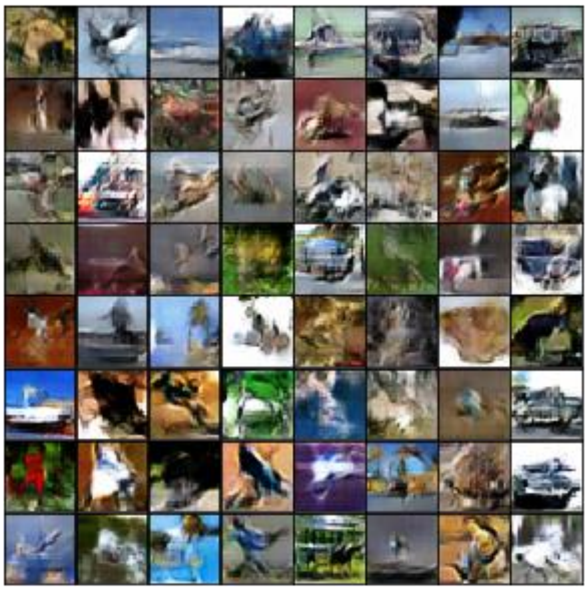

Figure 4-11: Generated Images Grid (Epoch = 9)

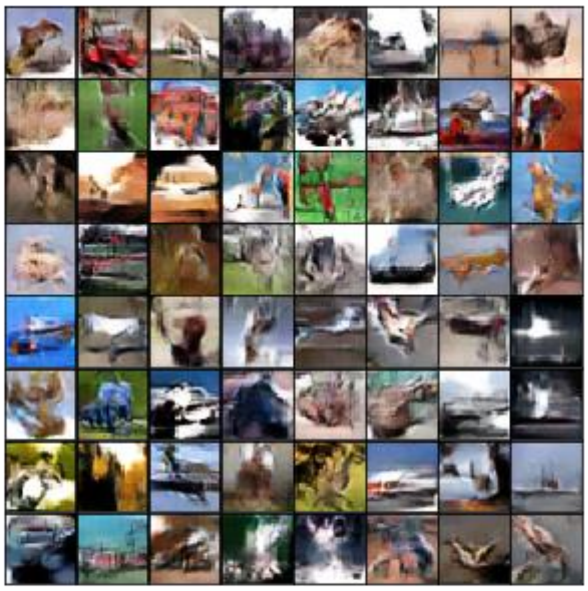
Figure 4-12: Generated Images Grid (Epoch = 10)

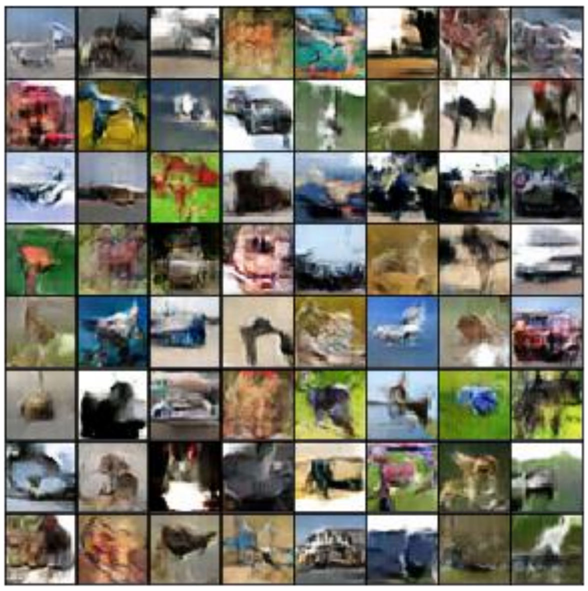
Figure 4-13: Generated Images Grid (Epoch = 11)

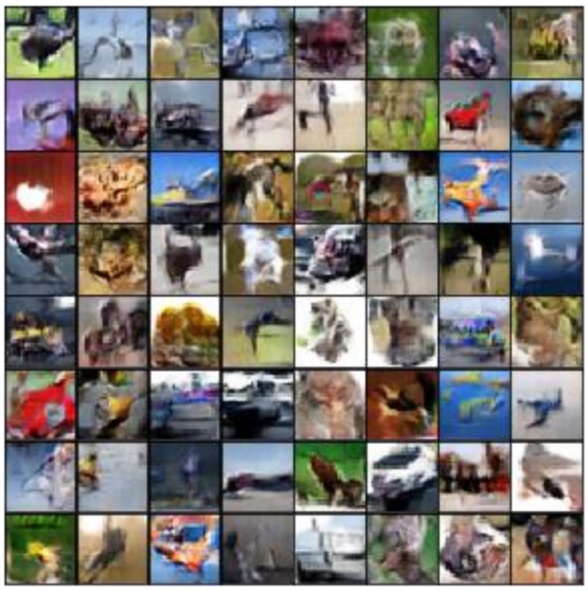
Figure 4-14: Generated Images Grid (Epoch = 12)

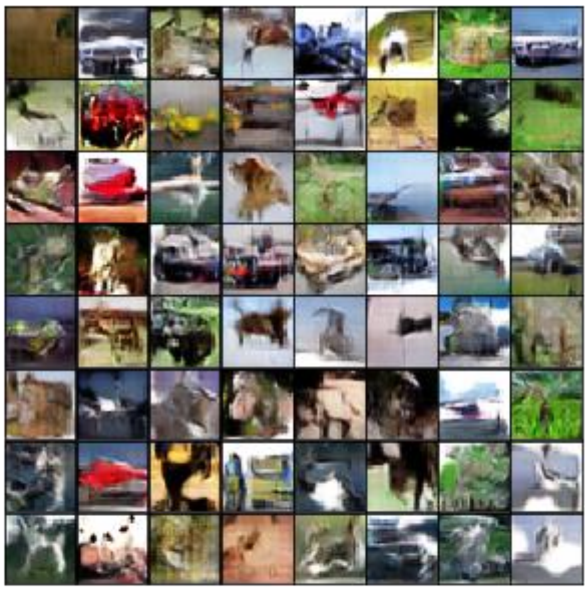
Figure 4-15: Generated Images Grid (Epoch = 13)

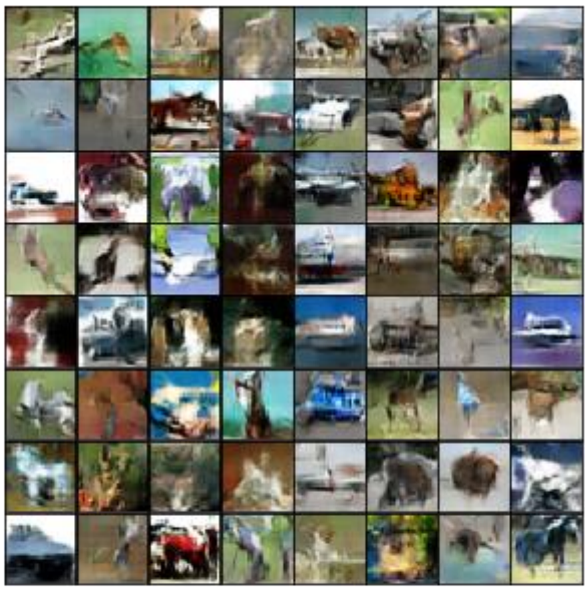

Figure 4-16: Generated Images Grid (Epoch = 14)

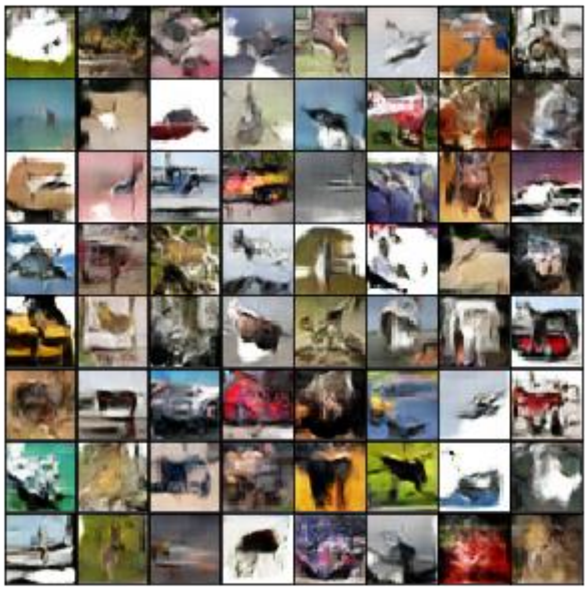

Figure 4-17: Generated Images Grid (Epoch = 15)

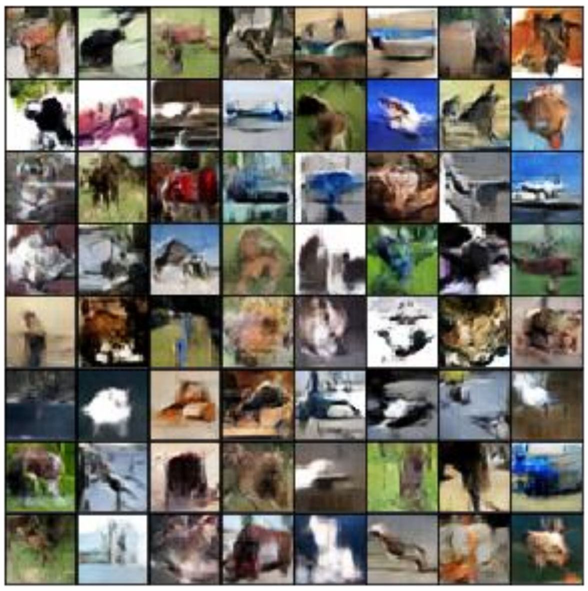
Figure 4-18: Generated Images Grid (Epoch = 16)

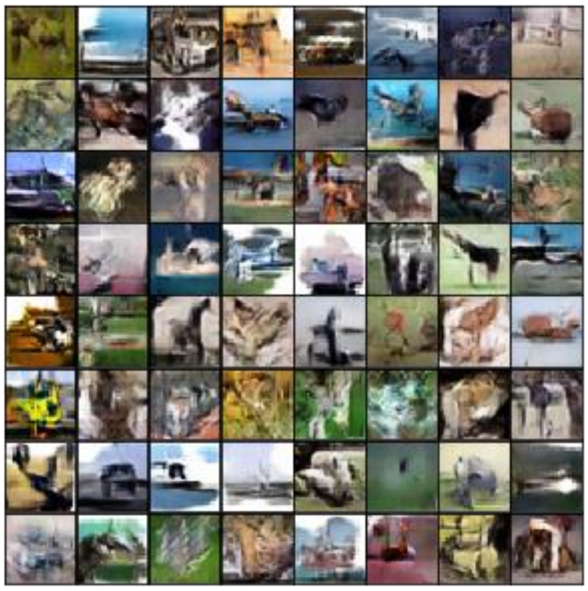
Figure 4-19: Generated Images Grid (Epoch = 17)

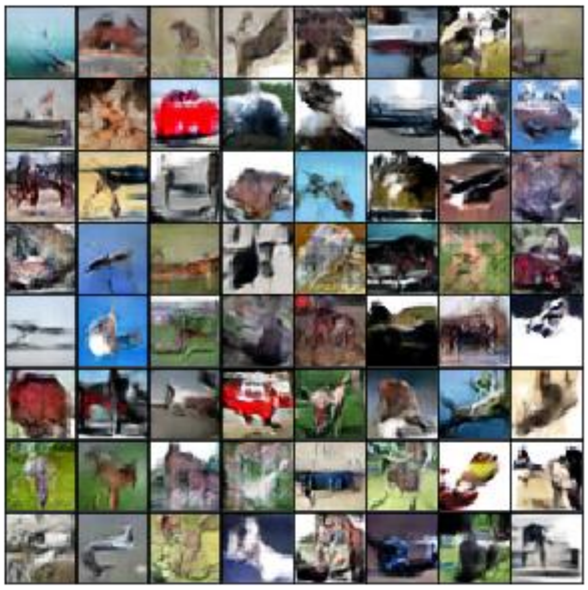
Figure 4-20: Generated Images Grid (Epoch = 18)

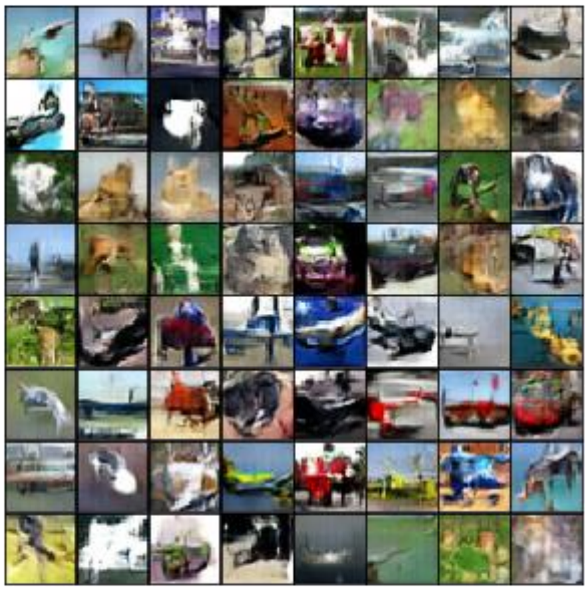
Figure 4-21: Generated Images Grid (Epoch = 19)

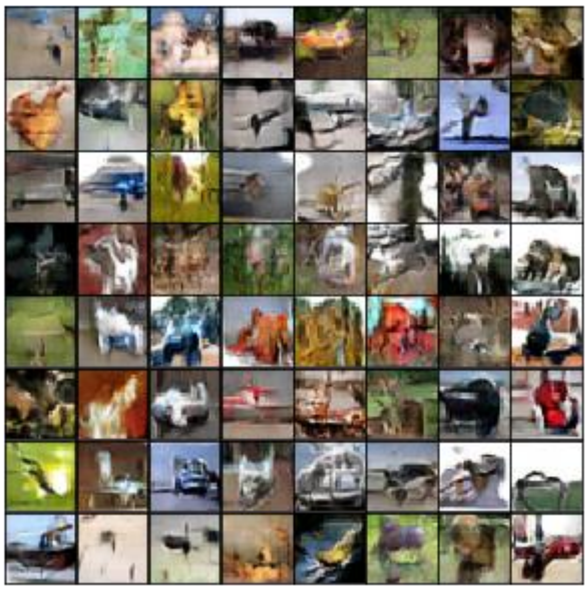
Figure 4-22: Generated Images Grid (Epoch = 20)

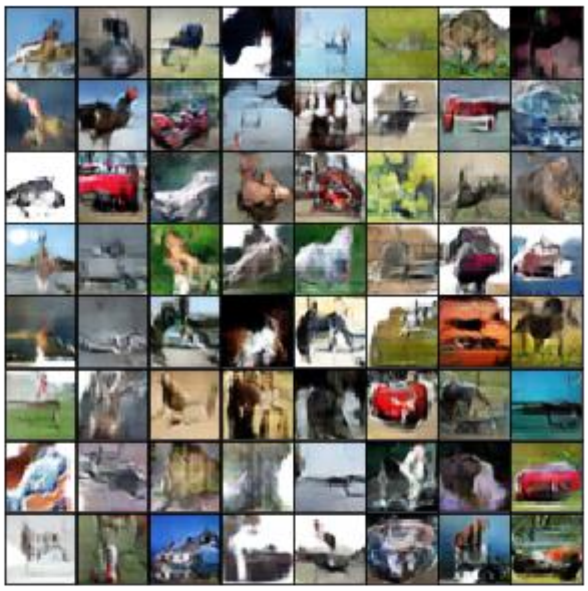
Figure 4-23: Generated Images Grid (Epoch = 21)

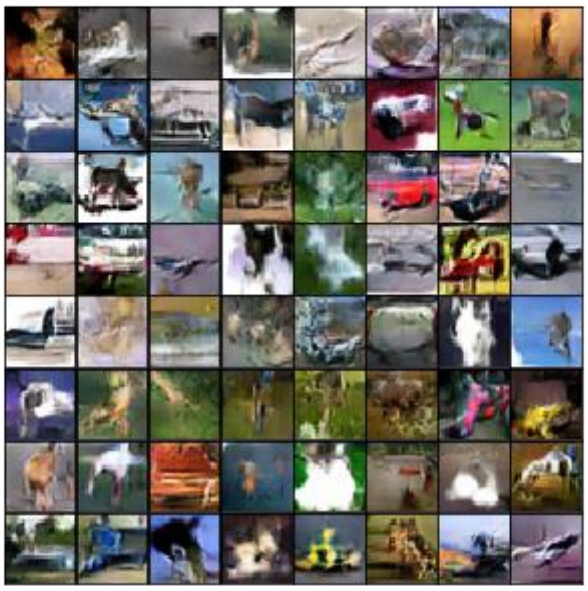
Figure 4-24: Generated Images Grid (Epoch = 22)

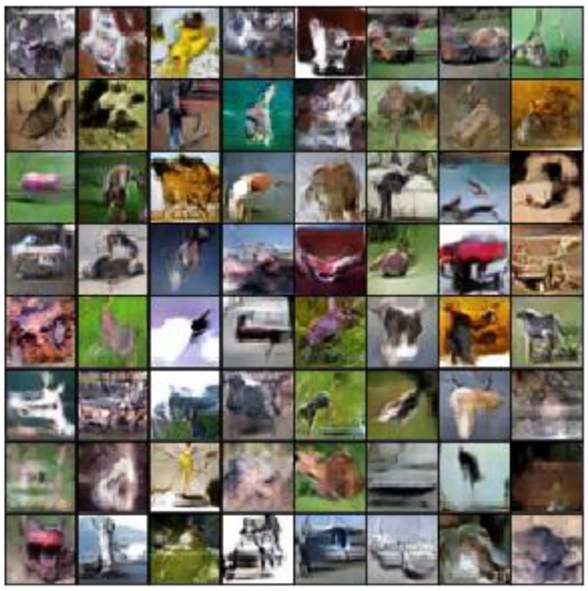
Figure 4-25: Generated Images Grid (Epoch = 23)

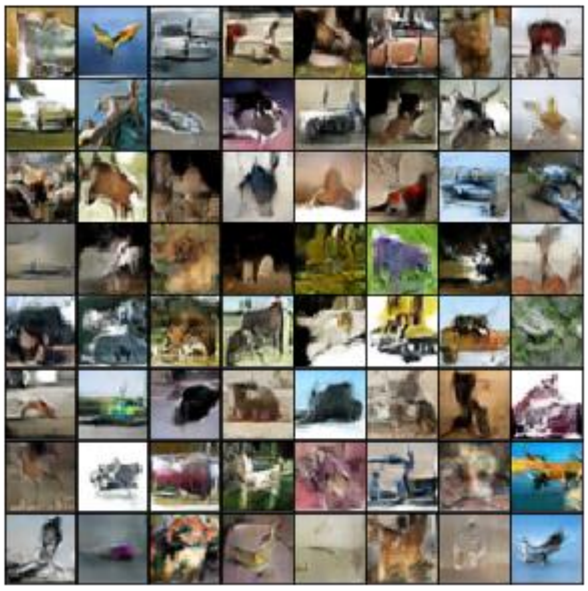
Figure 4-26: Generated Images Grid (Epoch = 24)

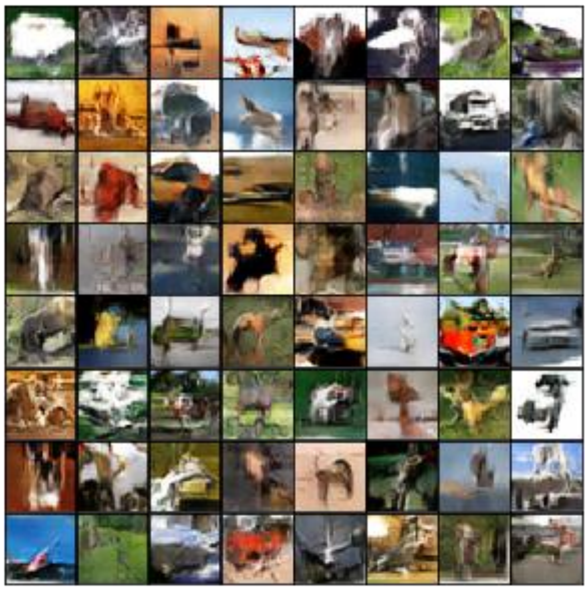
Figure 4-27: Generated Images Grid (Epoch = 25)

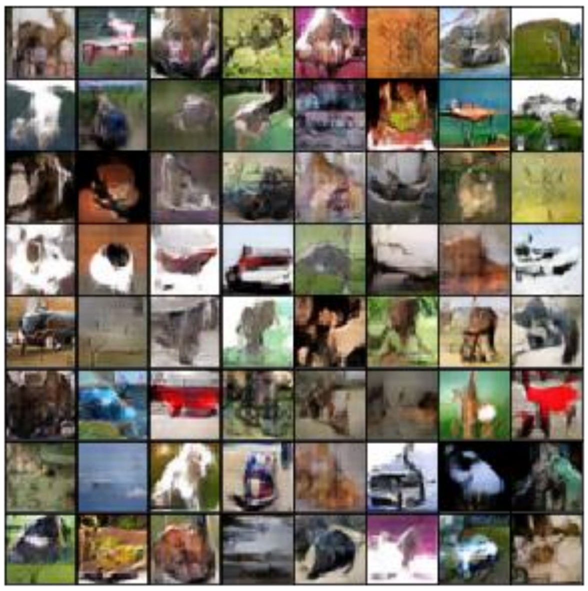
Figure 4-28: Generated Images Grid (Epoch = 26)

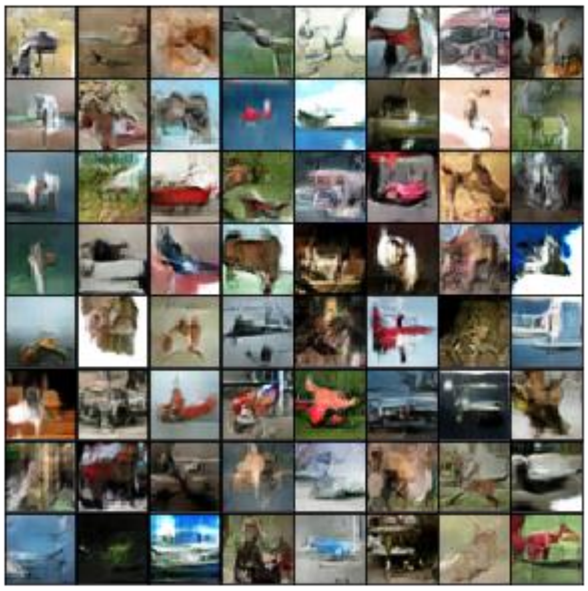
Figure 4-29: Generated Images Grid (Epoch = 27)

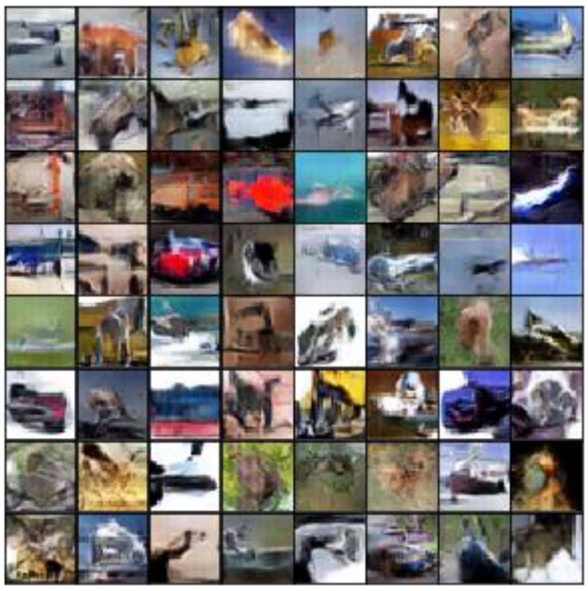
Figure 4-30: Generated Images Grid (Epoch = 28)

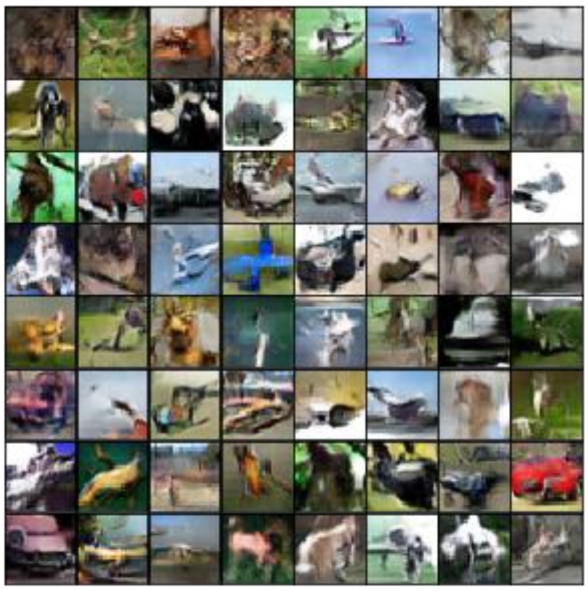
Figure 4-31: Generated Images Grid (Epoch = 29)

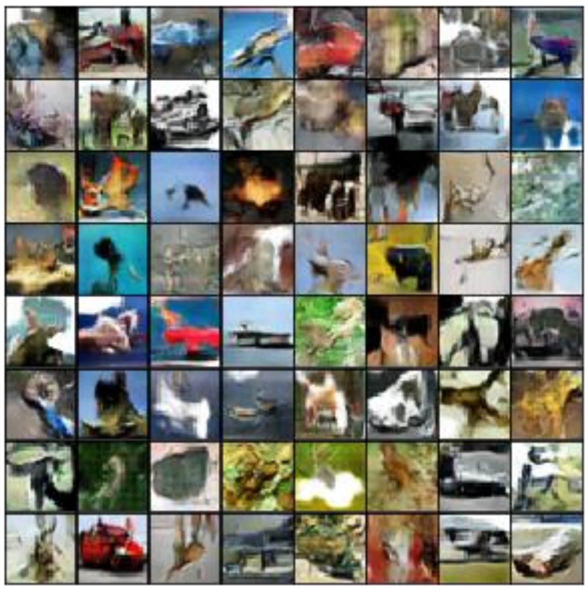
Figure 4-32: Generated Images Grid (Epoch = 30)

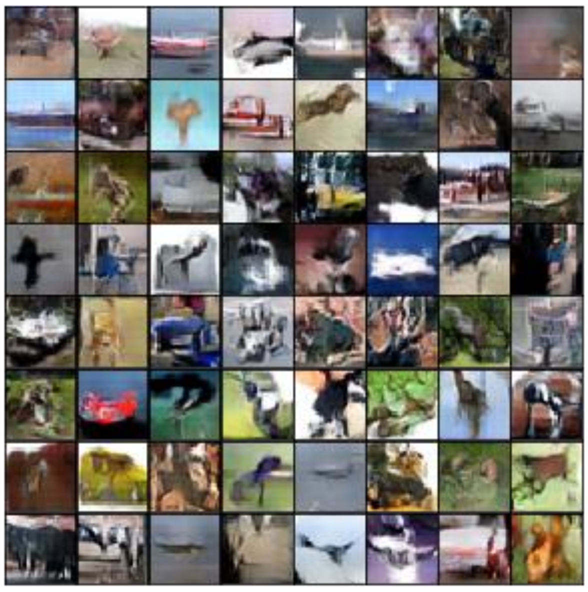
Figure 4-33: Generated Images Grid (Epoch = 31)

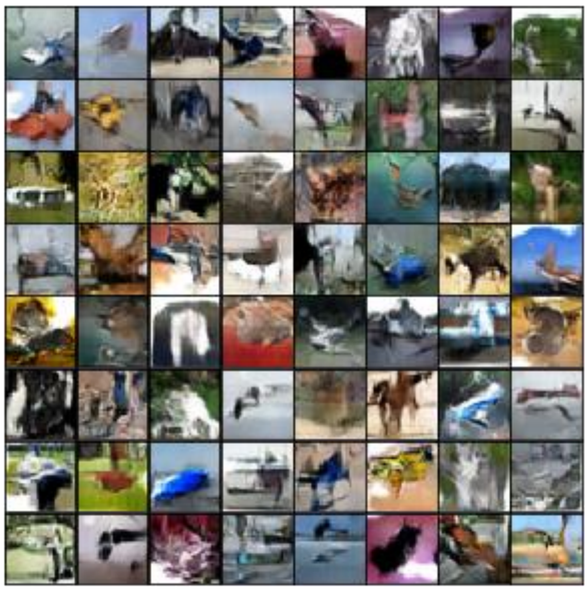
Figure 4-34: Generated Images Grid (Epoch = 32)

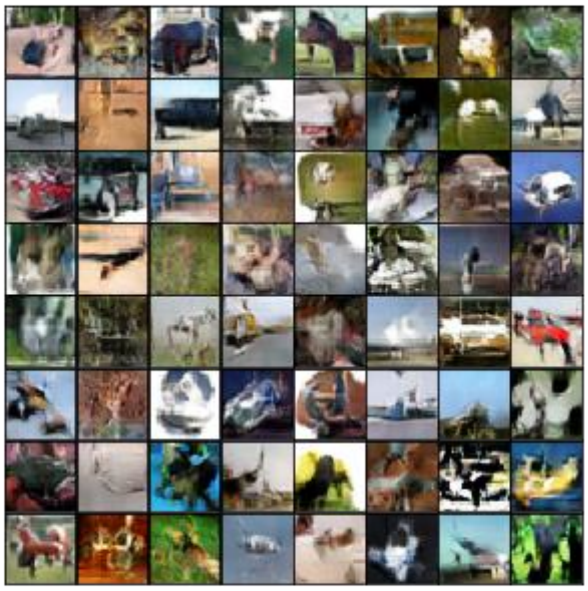
Figure 4-35: Generated Images Grid (Epoch = 33)

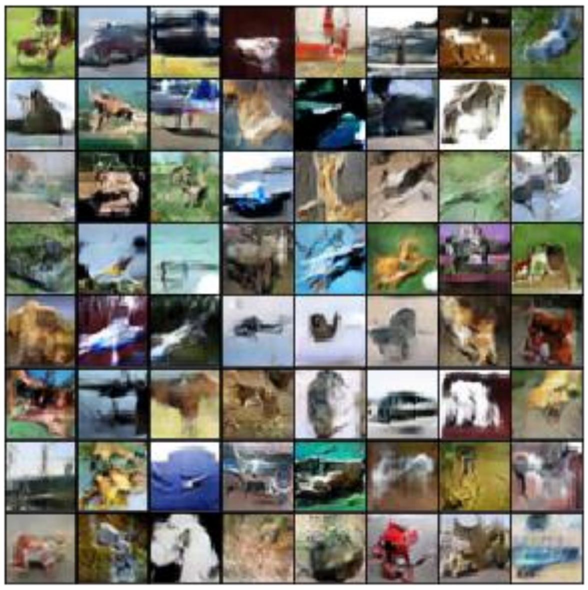
Figure 4-36: Generated Images Grid (Epoch = 34)

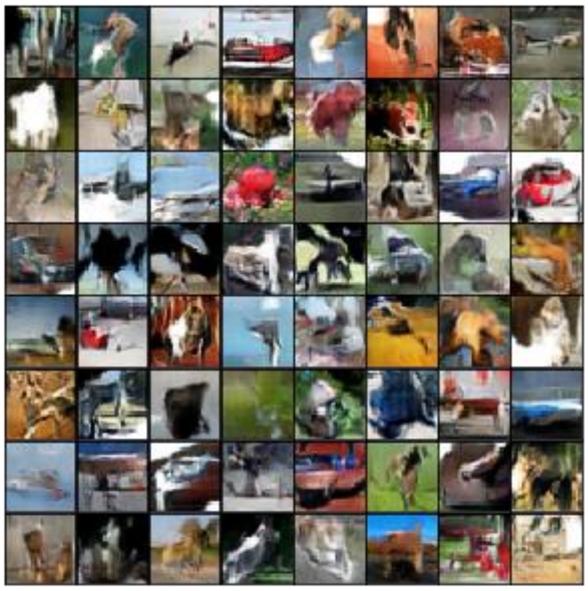
Figure 4-37: Generated Images Grid (Epoch = 35)

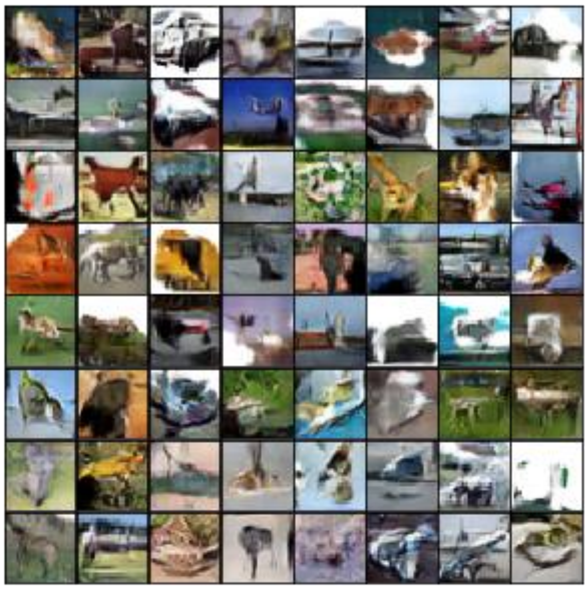
Figure 4-38: Generated Images Grid (Epoch = 36)

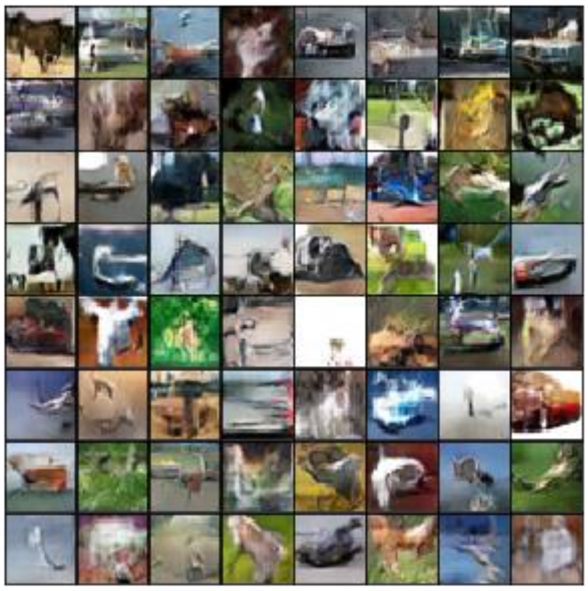
Figure 4-39: Generated Images Grid (Epoch = 37)

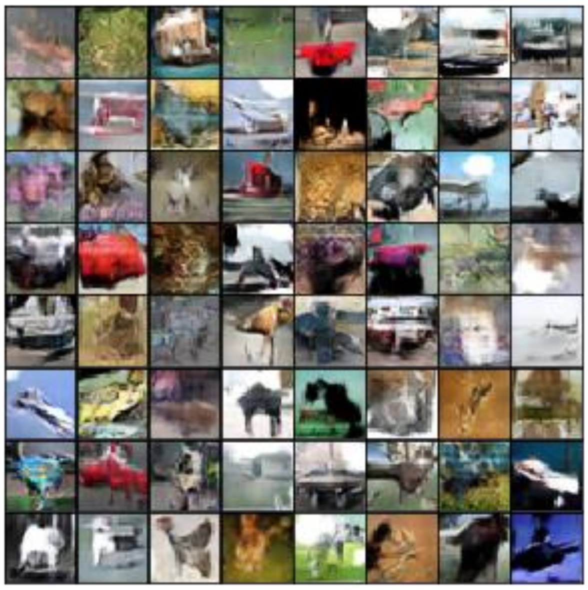
Figure 4-40: Generated Images Grid (Epoch = 38)

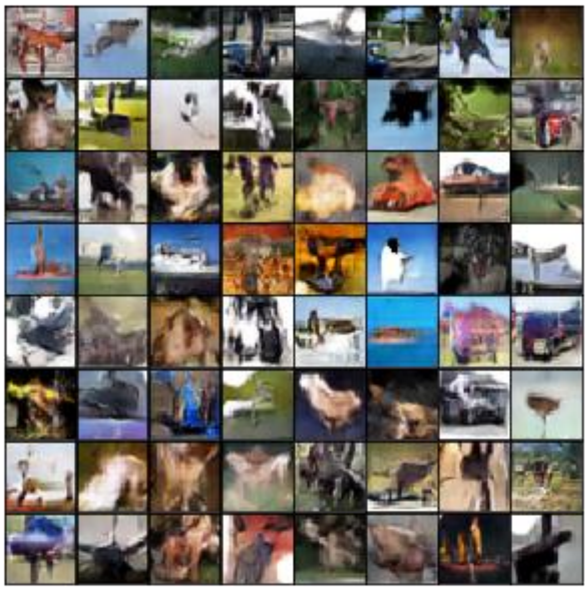
Figure 4-41: Generated Images Grid (Epoch = 39)

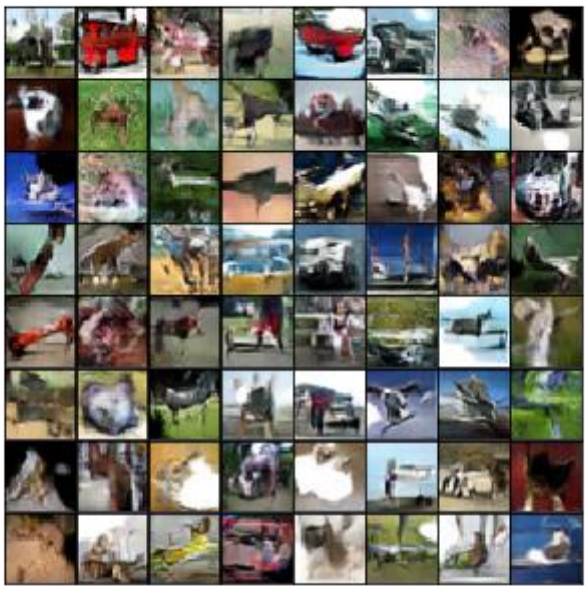
Figure 4-42: Generated Images Grid (Epoch = 40)

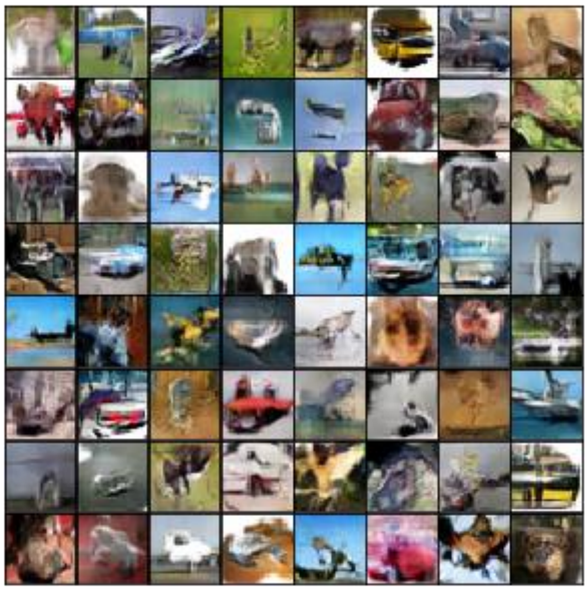
Figure 4-43: Generated Images Grid (Epoch = 41)

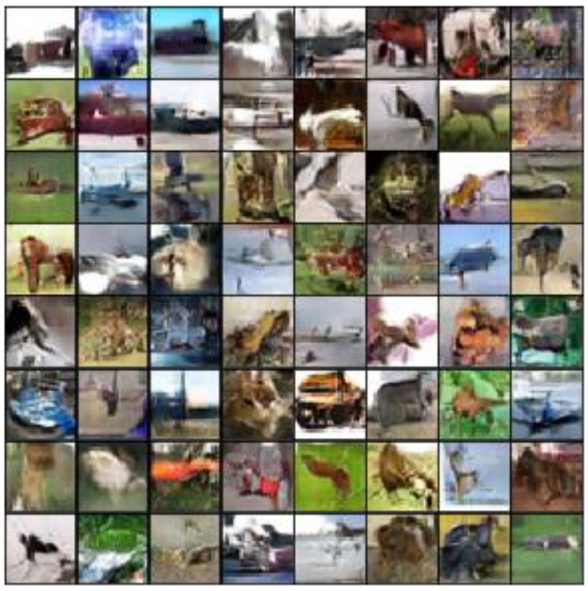
Figure 4-44: Generated Images Grid (Epoch = 42)

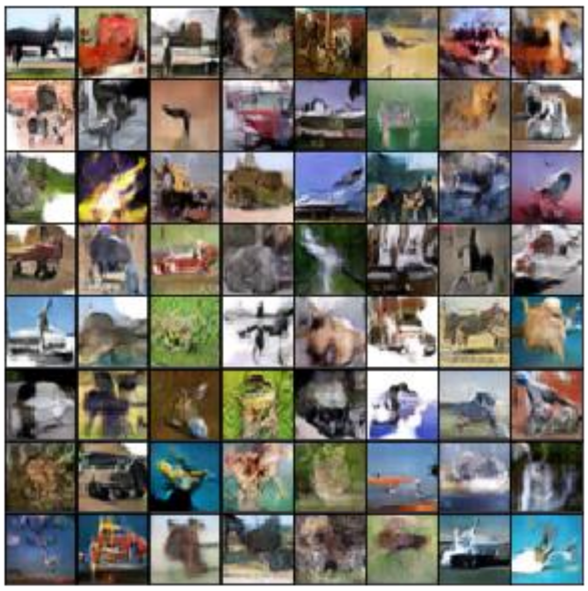
Figure 4-45: Generated Images Grid (Epoch = 43)

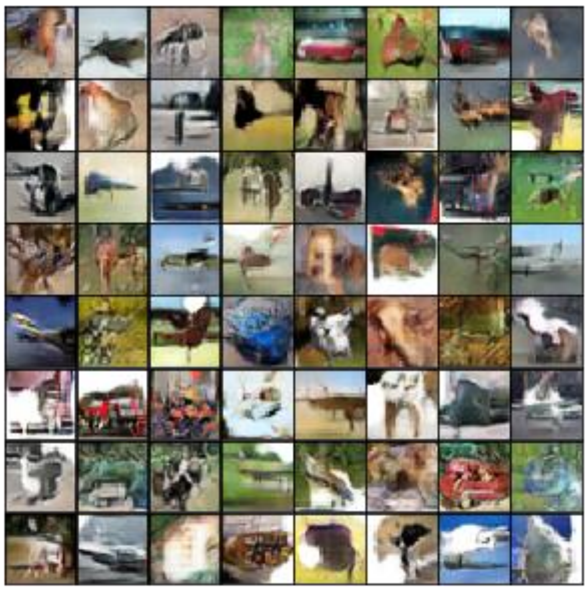
Figure 4-46: Generated Images Grid (Epoch = 44)

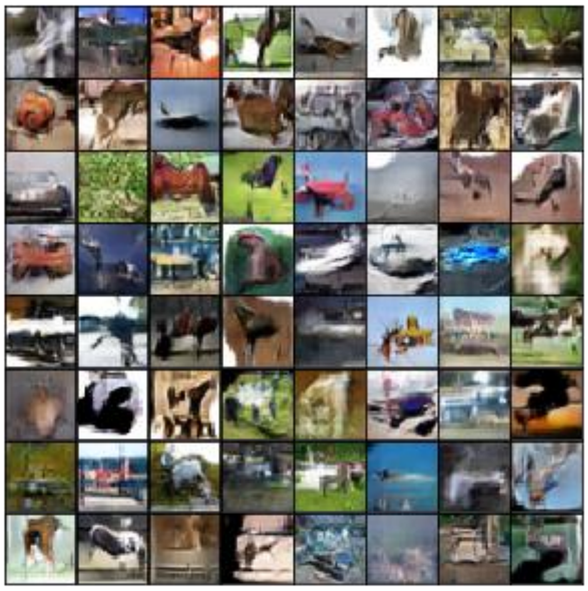
Figure 4-47: Generated Images Grid (Epoch = 45)

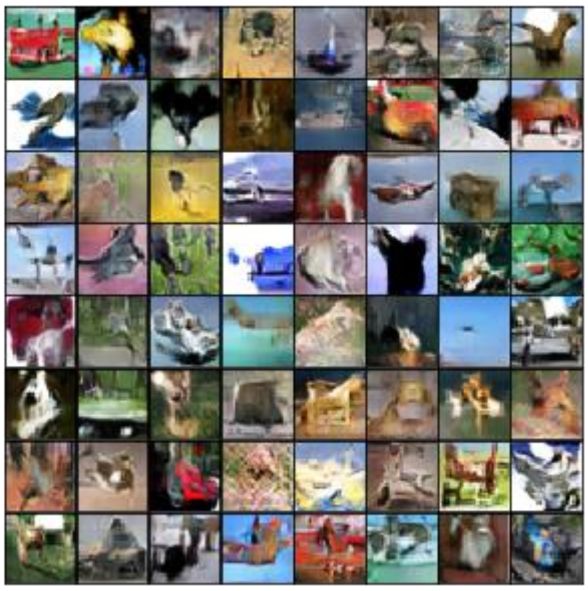
Figure 4-48: Generated Images Grid (Epoch = 46)

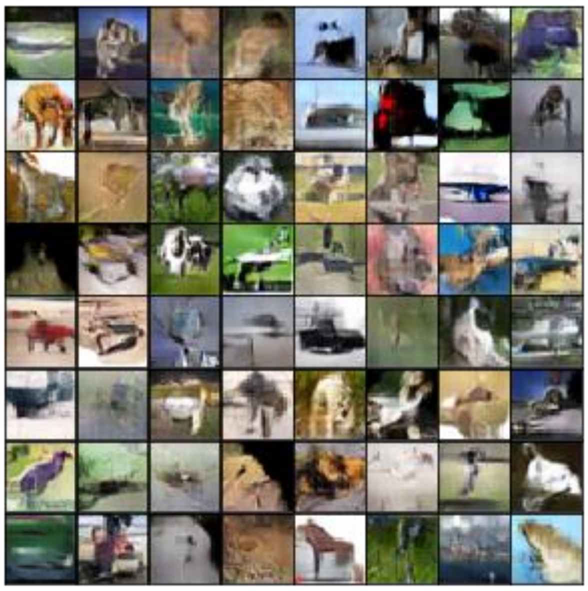
Figure 4-49: Generated Images Grid (Epoch = 47)

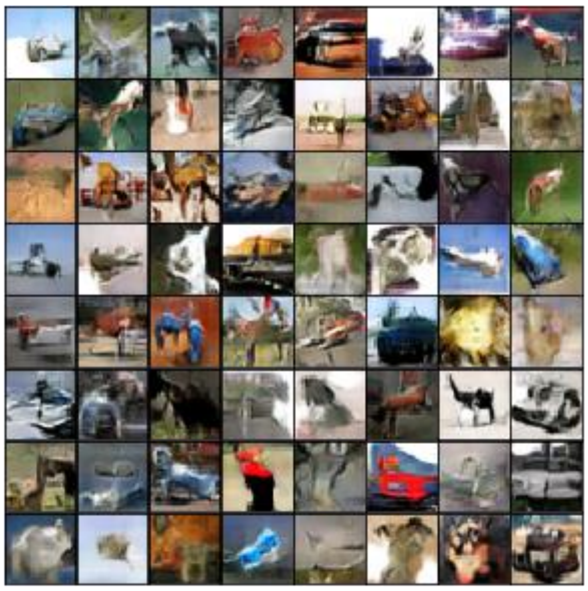
Figure 4-50: Generated Images Grid (Epoch = 48)

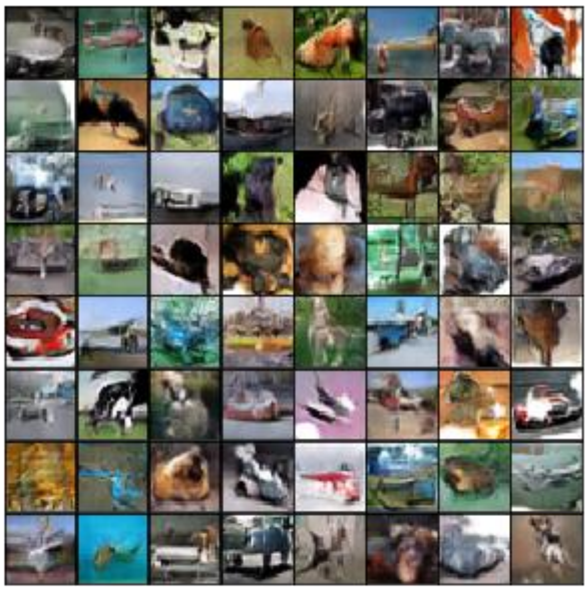
Figure 4-51: Generated Images Grid (Epoch = 49)

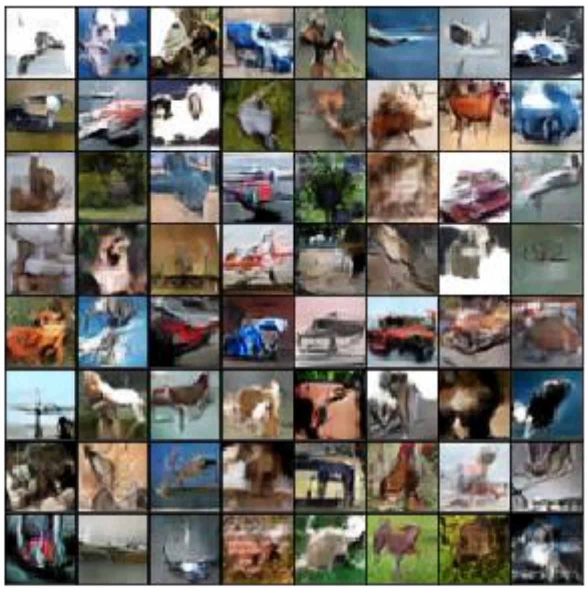

Figure 4-52: Generated Images Grid (Epoch = 50)

4.3 Table of Metrics for Performance (FID Scores)

Iterations	**128 Images**	**1024 Images**	**10000 Images**
0	405.05	408.04	401.65
1000	180.47	128.15	105.73
2000	169.26	108.21	82.43
3000	158.97	92.32	67.1
4000	158.16	87.62	61.85
5000	151.42	85.57	58.68
6000	147.53	75.16	47.63
7000	154.39	80.45	52.45
8000	150.37	73.25	44.48
9000	142.83	67.88	39.12
10000	150.07	69.73	40.37
11000	145.18	65.64	36.22
12000	142.38	64.11	34.57
13000	143.35	61.04	31.35
14000	138.53	62.15	32.32
15000	147.94	64.07	33.45
16000	143.48	59.68	30.52
17000	140.47	62.11	31.28
18000	143.82	59.64	28.91
19000	140.91	59.78	29.08
20000	140.51	58.96	29.27
21000	140.46	57.63	27.33
22000	142.53	60.22	29.04

23000	140.71	57.93	28.57
24000	142.34	57.85	28.54
25000	137.51	57.72	27.66
26000	138.18	59.92	28.73
27000	139.56	58.81	28.36
28000	141.16	59.14	27.93
29000	147.58	59.19	28.12
30000	138.83	57.75	26.8
31000	138.66	58.33	27.11
32000	143.07	58.15	27.5
33000	139.3	57.96	26.61
34000	142.41	58.17	26.23
35000	137.51	56.52	24.94
36000	139.06	58.47	26.05
37000	137.15	57.11	25.94
38000	139.21	56.31	25.47
39000	143.35	55.8	25.26
40000	137.29	56.24	25.11
41000	142.34	53.87	24.26
42000	137.64	54.73	24.34
43000	135.55	55.73	25.44
44000	138.76	55.77	25.29
45000	140.79	54.59	23.19
46000	138.91	54.9	24.45
47000	140.54	56.01	25.32
48000	139.68	56.69	23.98
49000	137.32	54.66	24.16
50000	137.8	53.93	23.47

4.4 Loss

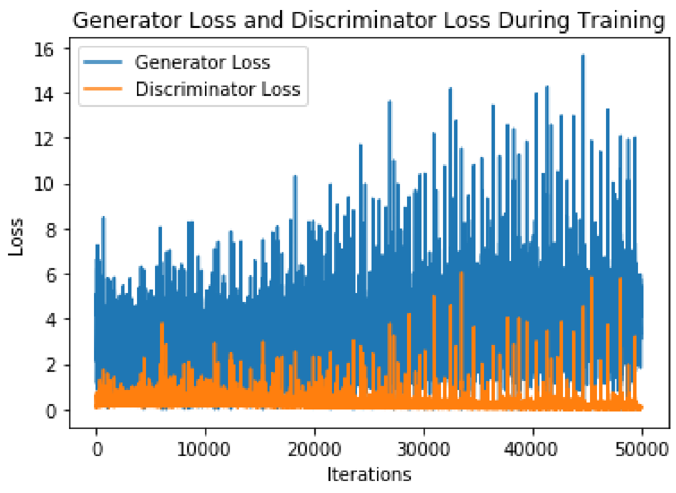

Figure 4-53: Generator Loss and Discriminator Loss

5 C_SA_W_GAN Model Performance:

1 epoch = 1000 iterations and trained C_SA_W_GAN with 50 epochs (50000 iterations).

5.1 Training Curve of FID Score

I calculated FID Scores separately using ‘1024 generated images and 1024 testing images’ and ‘128 generated images and 128 testing images’ at every 10000 iterations (1 epoch).

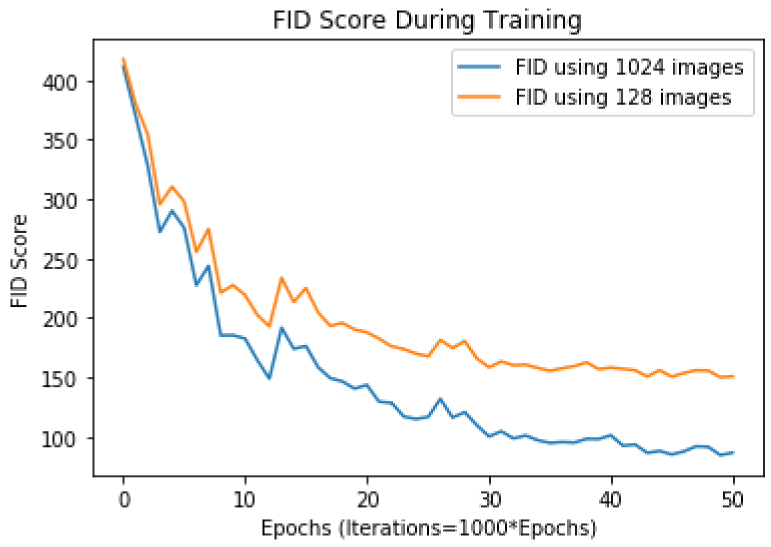

Figure 5-1: FID Scores over the trained epochs

5.2 Grid of Generated Images

Although the project1 description requires 8 by 8 image grid, I think 10 by 10 image grid is appropriate because it is a conditional GAN and the CIFAR10 Dataset has 10 classes. Meanwhile, because these generated images is not good enough, I just select some generated images and add them into the report. (I planned to run 100 epochs in C_SA_W_GAN, but unfortunately, when the programming process went to around 90 epochs, the Google Colab crashed. That best FID Score of 1000 images was around **66** in 80+ epochs.)

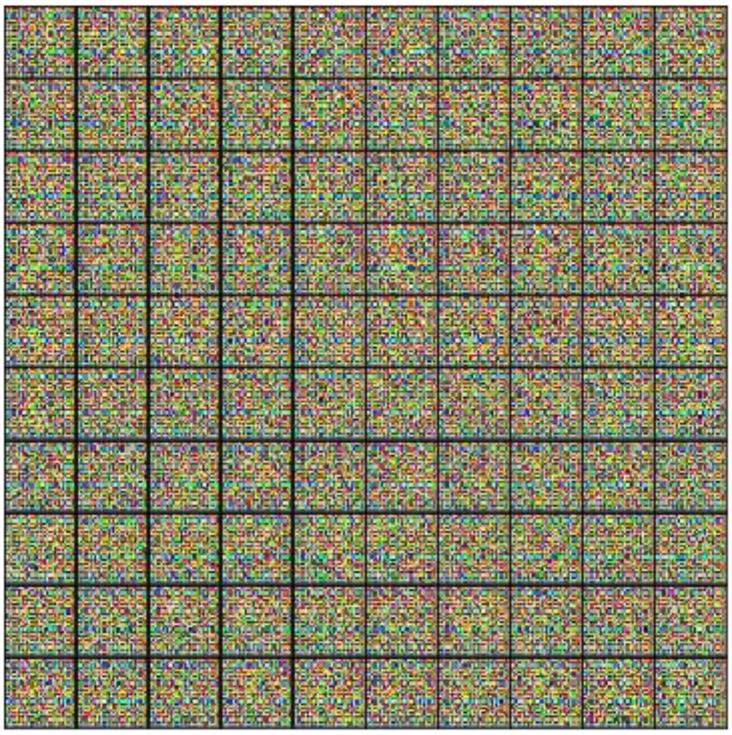

Figure 5-2: Generated Images Grid (Epoch = 0)

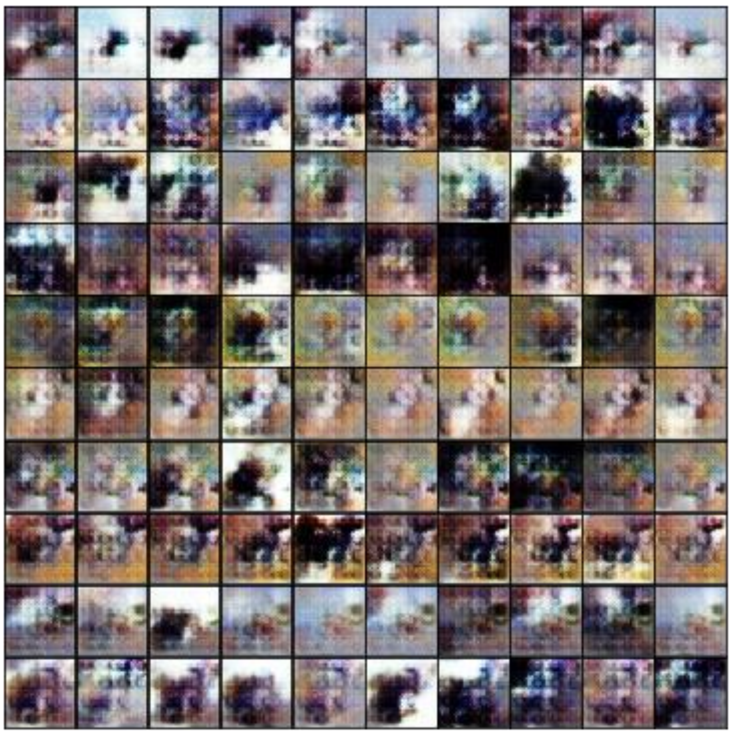

Figure 5-3: Generated Images Grid (Epoch = 10)

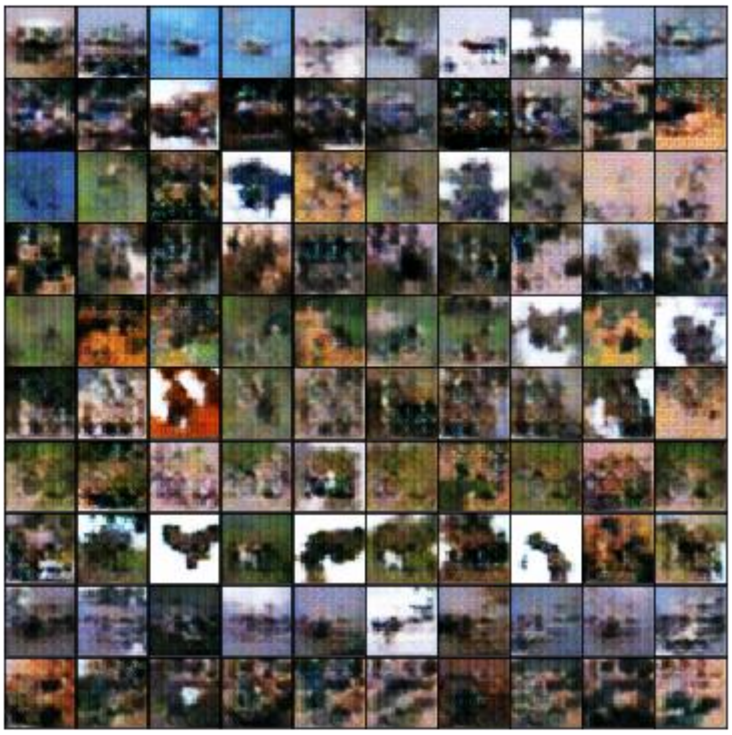
Figure 5-4: Generated Images Grid (Epoch = 20)

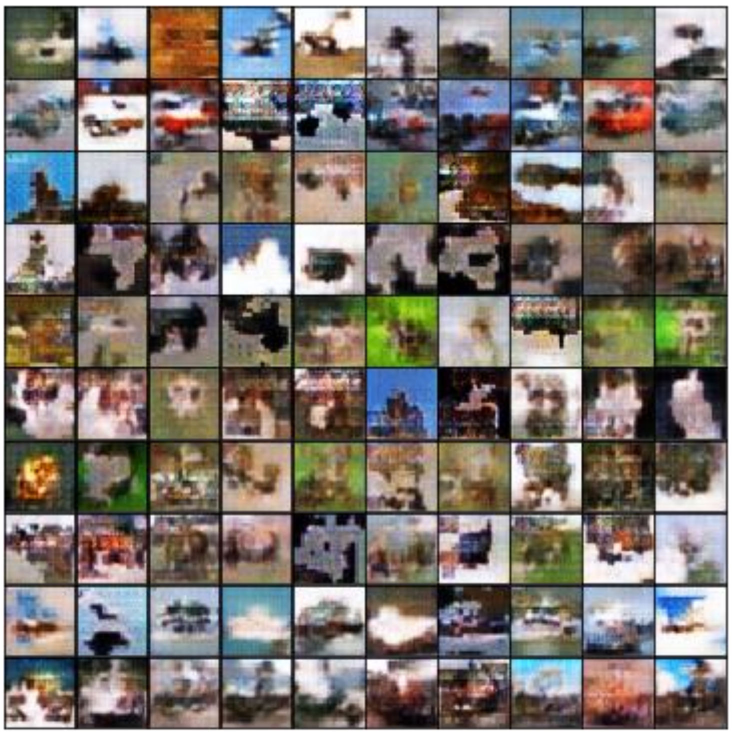
Figure 5-5: Generated Images Grid (Epoch = 30)

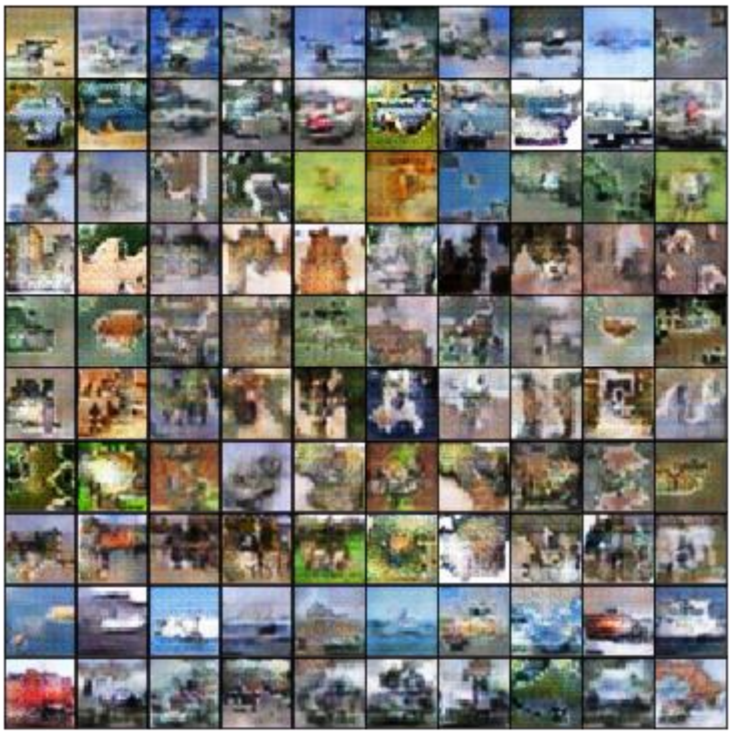

Figure 5-6: Generated Images Grid (Epoch = 40)

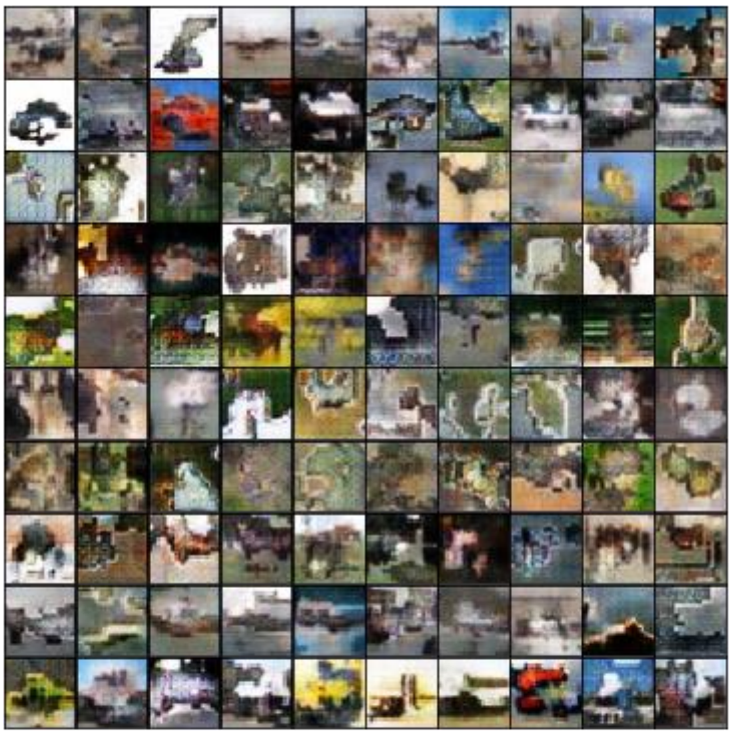

Figure 5-7: Generated Images Grid (Epoch = 50)

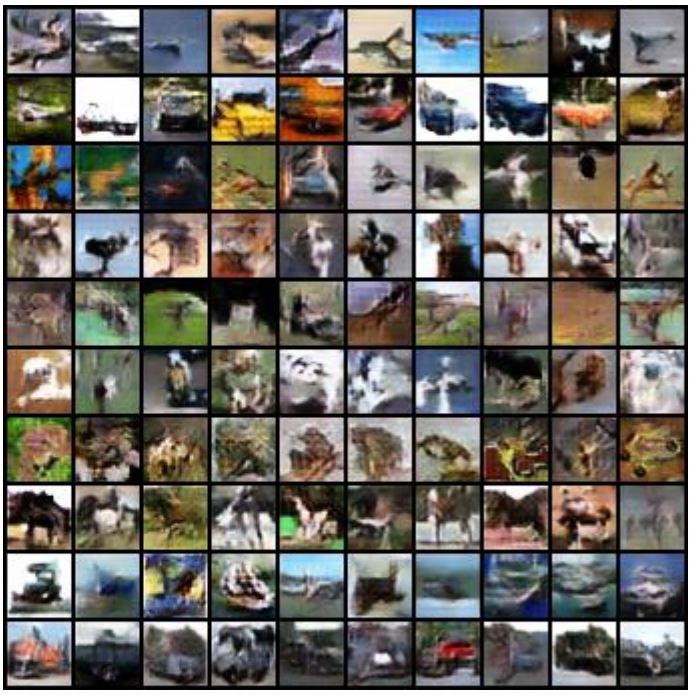
Figure 5-8: Generated Images Grid (Epoch = 67)

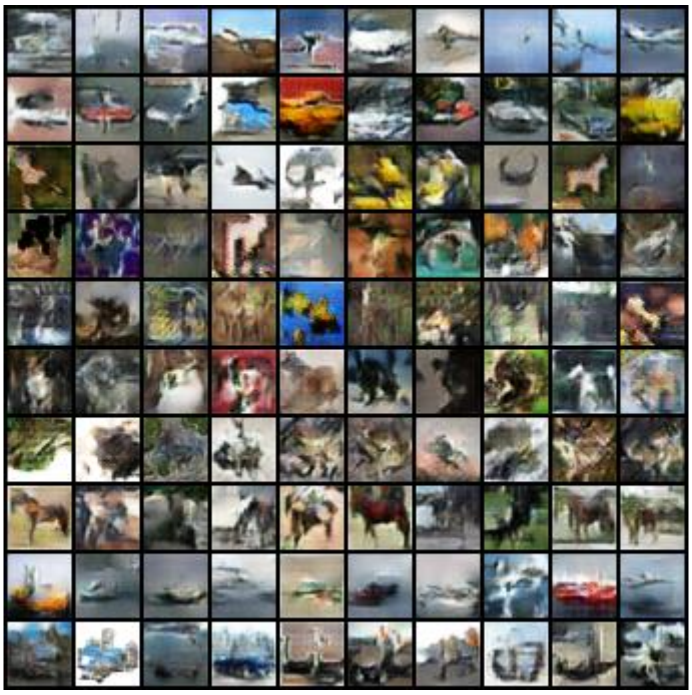
Figure 5-9: Generated Images Grid (Epoch = 84)

These two images of this page have nothing to do with other data or records in the report. They are the results from another program process. (I planned to run 100 epochs in C_SA_W_GAN, but unfortunately, when the programming process went to around 90 epochs, the google colab crashed. That best FID Score of 1000 images was around 66 in 80+ epochs.) Therefore, after enough epochs, we can see that the images become clear and we can recognize many images. **(0-Airplane, 1-Automobile, 2-bird, 3-cat, 4-deer, 5-dog, 6-frog, 7-horse, 8-ship, 9-truck)**

5.3 Table of Metrics for Performance (FID Scores)

Iterations	128 Images	1024 Images
0	417.82	411.29
1000	380.42	370.57
2000	354.78	328.36
3000	295.64	272.35
4000	310.54	290.33
5000	298.05	275.9
6000	255.63	227.56
7000	274.96	243.98
8000	221.29	185.31
9000	227.66	185.53
10000	219.39	182.66
11000	202.72	164.68
12000	192.82	148.88
13000	233.92	191.84
14000	213.37	174.05
15000	225.26	176.42
16000	204.71	158.57
17000	193.37	149.36
18000	195.81	146.56
19000	190.24	140.52
20000	187.96	143.67
21000	182.68	129.6
22000	176.18	128.55
23000	173.8	117.03
24000	169.99	115.0
25000	167.66	116.77
26000	181.51	131.88
27000	174.7	116.24
28000	180.55	120.73
29000	165.94	109.73
30000	158.5	100.29
31000	163.38	104.67
32000	160.21	98.64
33000	160.71	101.15
34000	157.98	97.16
35000	155.54	94.89
36000	157.66	95.65
37000	159.45	95.16
38000	162.61	98.34

39000	156.97	98.14
40000	158.26	101.33
41000	157.19	92.56
42000	155.95	93.53
43000	150.77	86.51
44000	156.05	88.13
45000	150.61	85.16
46000	153.69	87.74
47000	155.99	92.0
48000	155.72	91.6
49000	150.08	84.62
50000	150.9	86.69

5.4 Loss

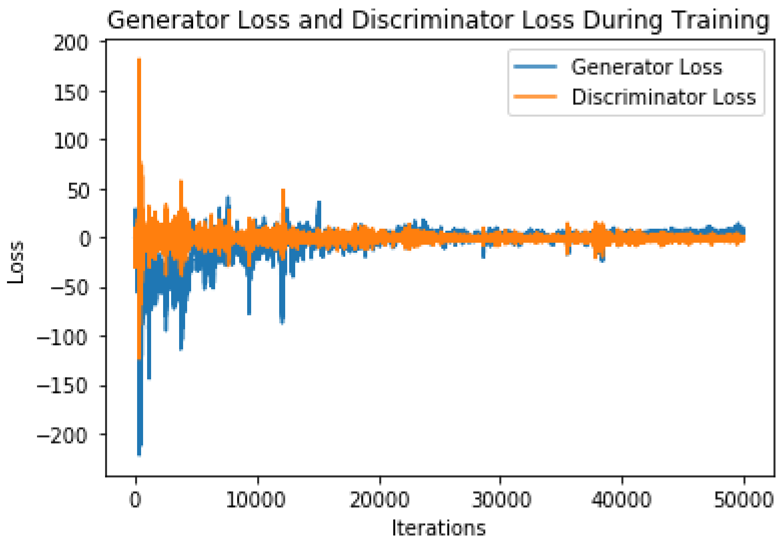

Figure 5-10: Generator Loss and Discriminator Loss

Reference

[1] https://en.wikipedia.org/wiki/Generative_adversarial_network.
[2] https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html.
[3] Han Zhang, Ian Goodfellow, Dimitris Metaxas, Augustus Odena, Self-Attention Generative Adversarial Networks.
[4] Xiaolong Wang, Ross Girshick, Abhinav Gupta, Kaiming He, Non-local Neural Networks, CVPR 2018.
[5] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, Yuichi Yoshida, Spectral Normalization for Generative Adversarial Networks, ICLR 2018.