50.035 Computer Vision

Project Report (Spring 2020)

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Research on the benefits of Evolutionary Generative Adversarial Nets

Grand Bhavy Mital 1002945

Andri Setiawan Susanto 1002849

Nigel Chan Terng Tseng 1002027

Zwe Wint Naing 1002791

1. Introduction

1.1 Background and problem framing

Generative adversarial networks (GANs) are two-part AI models consisting of a generator that creates samples and a discriminator that attempts to differentiate between the generated samples and real-world samples [1].

GANs are effective for learning generative models for real-world data. However, existing GANs (and their variants) have a tendency to suffer from training problems such as instability and mode collapse [2].

1.2 Proposed project idea

We proposed to do research into how GANs could be improved with the use of evolutionary computation and the technique of updating training weights via the use of the Genetic Algorithm. As such, we explored the idea of Evolutionary GAN (E-GAN) to find out how this network improved on vanilla GANs.

As a means to see how well these GANs would perform, we decided to look into another traditional GANs, the DC-GAN, where we would compare the performance of the E-GAN against it to observe how much better or worse it performed.

2. Evolutionary Generative Adversarial Nets

2.1 Idea motivation

Unlike existing GANs where the generator and discriminator are trained in an alternate fashion, E-GAN utilizes different adversarial training objectives as mutation operations and evolves a population of generators to adapt to the environment (i.e. the discriminator). Utilizing an evaluation mechanism for measurement of the diversity and quality of generated samples, E-GAN aims to preserve the best offspring (i.e. the generator) for further training [2].

2.2 The Evolutionary Algorithm

The evolutionary algorithm lies at the core of E-GAN. Through the inspiration from natural evolution, the evolutionary algorithm, in essence, aims to equate possible solutions to individuals in a population, producing offspring through variations and selecting the appropriate solutions in accordance with fitness [3]. The evolutionary algorithm was proposed to do a compression of deep learning models through the automatic elimination of convolution filters deemed redundant [4].

2.3 Mechanism

2.3.1 Overview

E-GAN's framework aims to utilize different metrics (Kullback-Leibler divergence, Wasserstein distance, etc) to jointly optimize the generator. This improves on both the generative performance and stability of training.

The E-GAN treats the adversarial training process as an evolutionary problem where the discriminator is seen as the "environment" and the generators are seen as the "population" that evolve in response to the environment. Each individual generator undergoes different "mutations" during the training process to produce offspring that adapt to the environment where different adversarial objective functions aim at minimizing different distances between a real-data distribution and generated distribution.

The given optimal discriminator measures the quality and diversity of the samples generated by the updated offspring and through the principle of "survival of the fittest", aims to preserve only the offspring (generators) that perform well for further training and discards the rest [2].

2.3.2 Usage of evolutionary algorithm

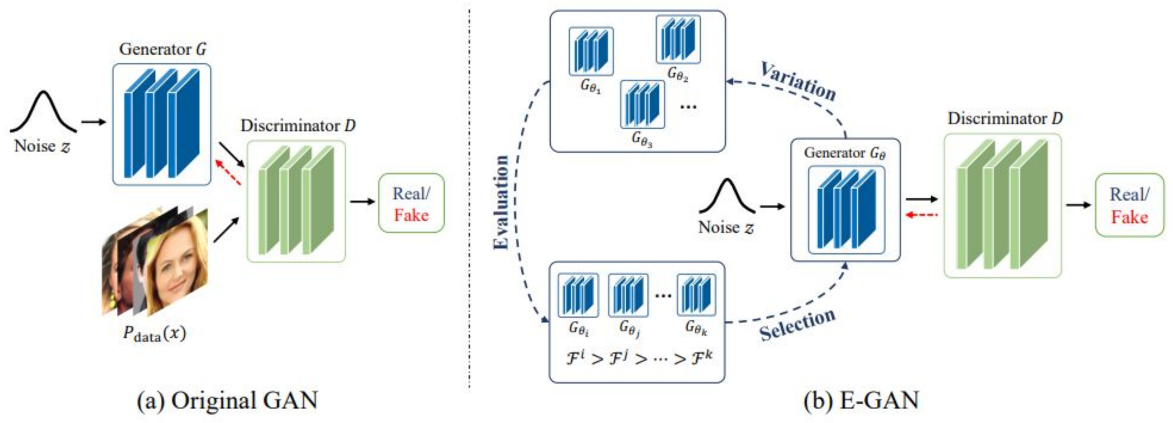


Figure 1: (a) represents vanilla GAN, (b) represents E-GAN [2].

The evolutionary algorithm is devised such that it evolves a population of generator(s), {G}, in a given environment (discriminator D). In the population, each individual represents a possible solution in the parameter space of the generative network G. It is expected that the population gradually adapts to the environment during the evolutionary process. This would lead to the evolved generator(s) generating more realistic samples that are closer to that of real-world data distribution [2].

With reference to Fig. 1 (b), the evolution process consists of three sub-stages in each step:

- **Variation**: Given an individual G_θ in the population, variation operators are used to produce its offspring $\{G_{\theta_1}, G_{\theta_2} \ldots\}$. Specifically, several copies of each individual or parent are created, each modified by a different mutation where each modified copy is considered as a single child.

- **Evaluation**: Each child and its corresponding performance is evaluated by a fitness function that depends on the current environment D.

- **Selection:** Children are selected in accordance with their fitness values and the worst ones are removed (killed). The rest remain alive and evolve to the next iteration.

After each evolutionary step, the discriminative network D is updated to further discern between the real-data samples x and fake generated samples y generated by the evolved generator(s) i.e.,

$$LD = -Ex\text{\textasciitilde}pdata\ [log\ D(x)] - Ey\text{\textasciitilde}pg\ [log(1 - D(y))].$$

In this fashion, D can continually provide the adaptive losses to drive the population {G} to produce better solutions [2].

2.4 Expected results

E-GAN was run through various experiments, one of which included evaluating E-GAN against the LSUN bedroom image dataset [5].

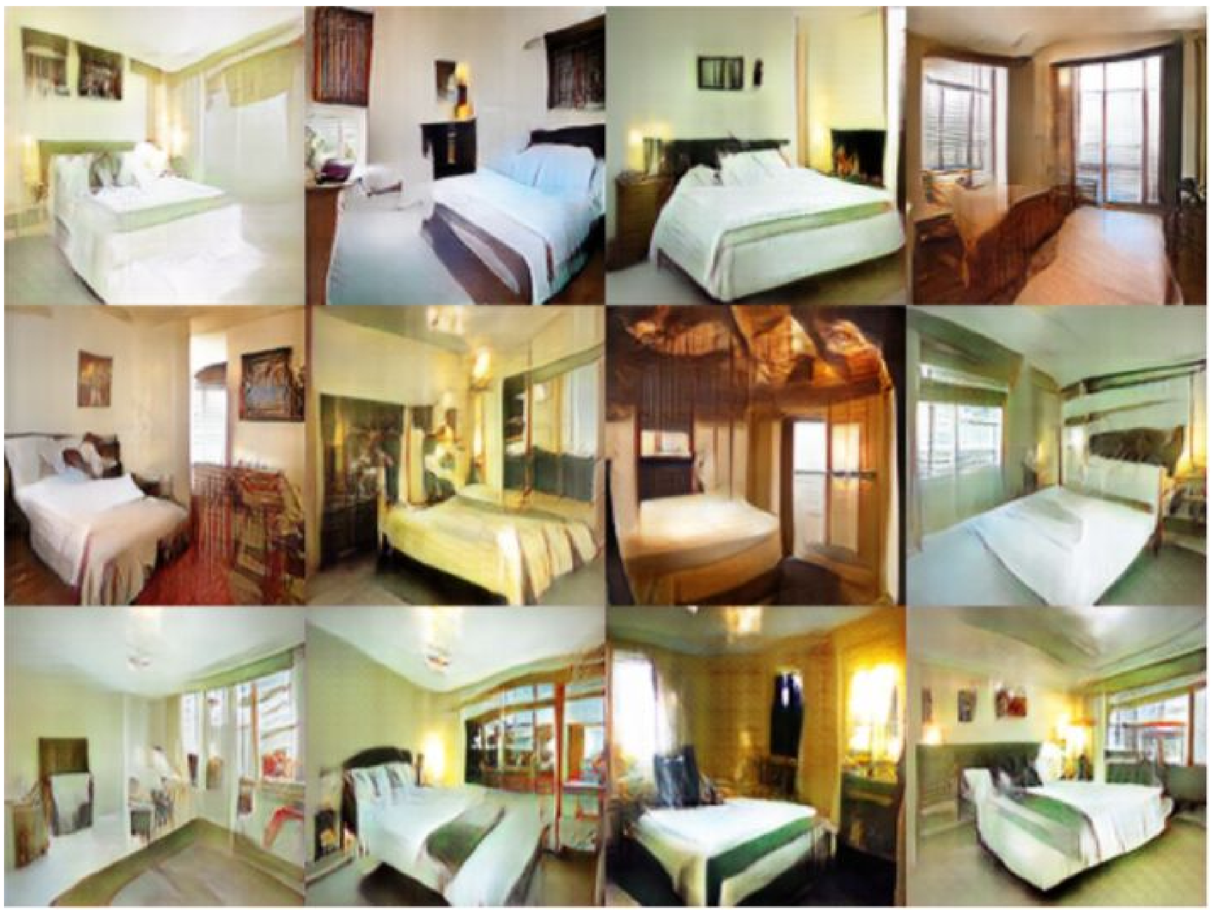


Figure 2: Generated samples on 128 x 128 LSUN bedrooms [2].

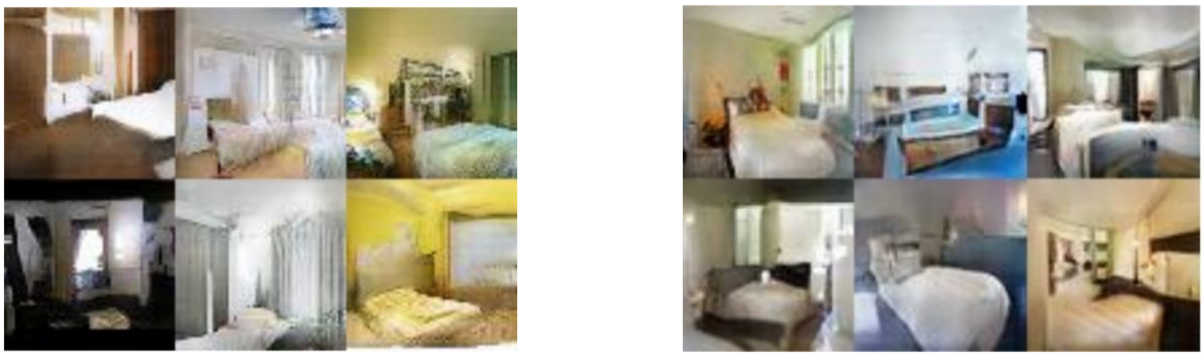


Figure 3: DCGAN vs E-GAN (LSUN bedrooms) [2].

3. Methodology

3.1 Initial run with DCGAN code

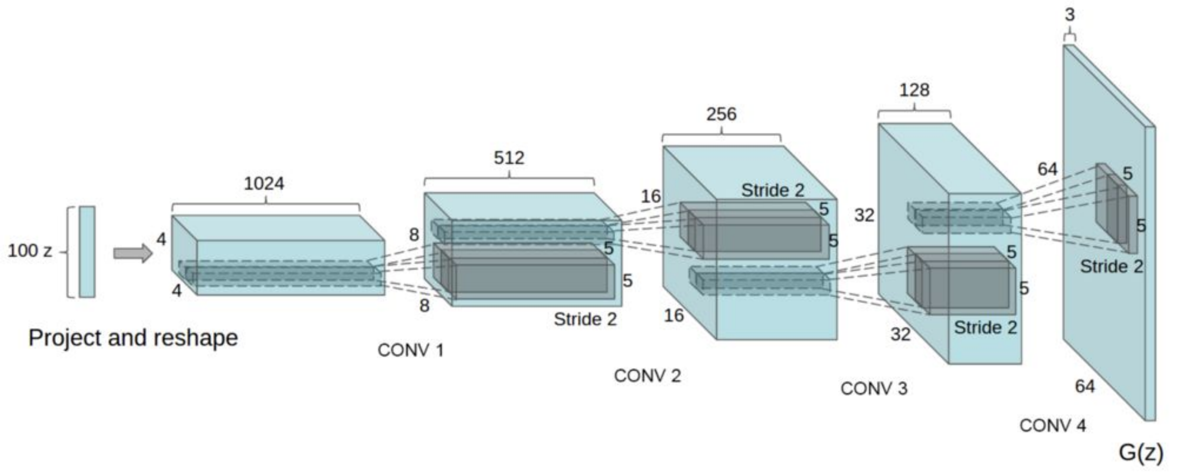


Figure 1: DCGAN Model [9]

Mainly composes of convolution layers without max pooling, or fully-connected layers, DC-GAN uses convolutional stride and transposed convolution for downsampling and upsampling.

The Mechanism of DC-GAN is as follows:

- Replace all max pooling with convolutional strides.
- Use transposed convolution for upsampling.
- Eliminate the fully connected layers.
- Batch normalization except the output layer for the generator and the input layer of the discriminator.
- Using ReLU in the generator (except for the output, which in turn uses tanh).

- Using LeakyReLU in the discriminator.

Our motivation to use DC-GAN is because it's one of the most popular and successful network designs for GAN. The common and successful nature of it means that we are able to use it as a benchmark to test other experimental GANs. In terms of errors and other technical issues, it makes it easy to troubleshoot.

To understand how the DC-GANs model works, we trained the model using the CIFAR10 dataset, which consists of 500,000 images, with 500 images from each class for validation samples, and the remaining images used as training samples.

These codes are trained with random initial values, with absolutely no reliance on any existing parameters.

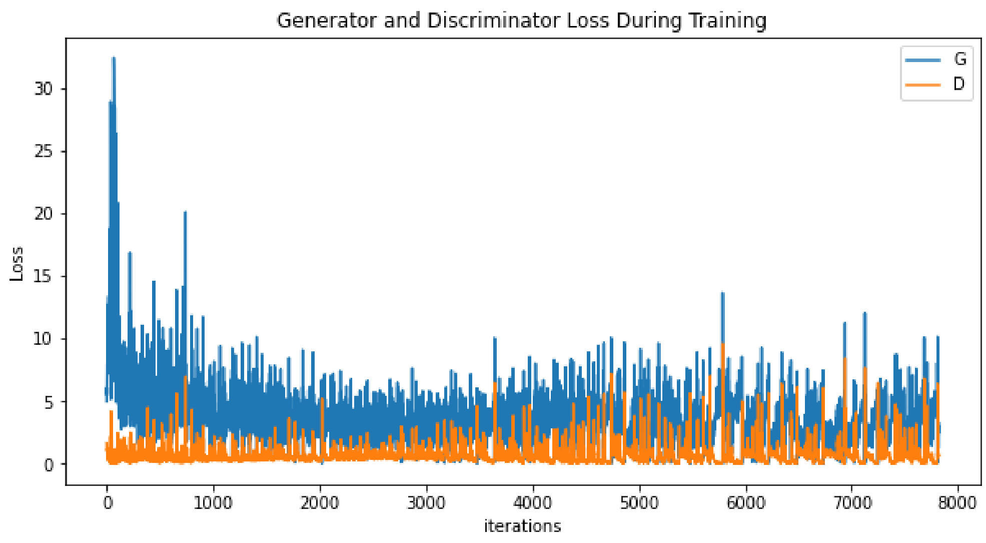
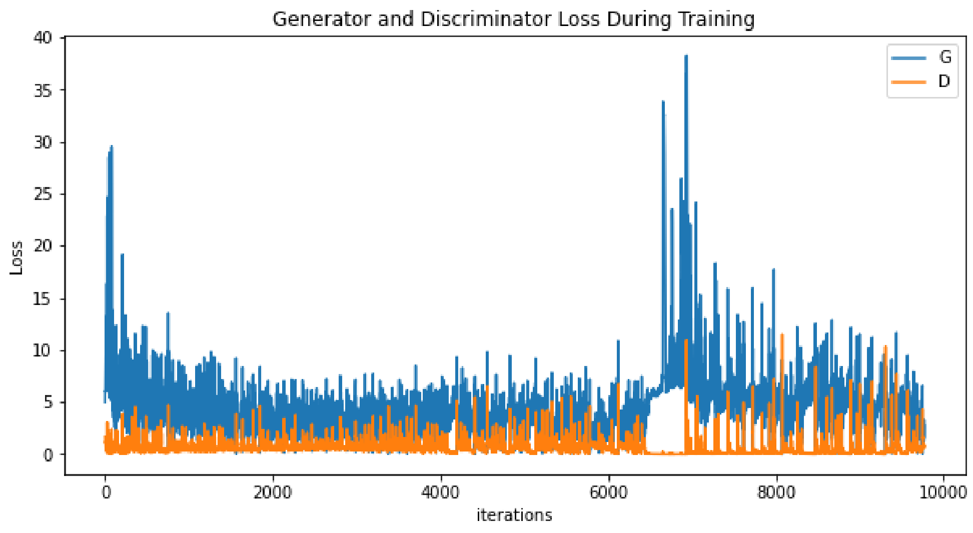
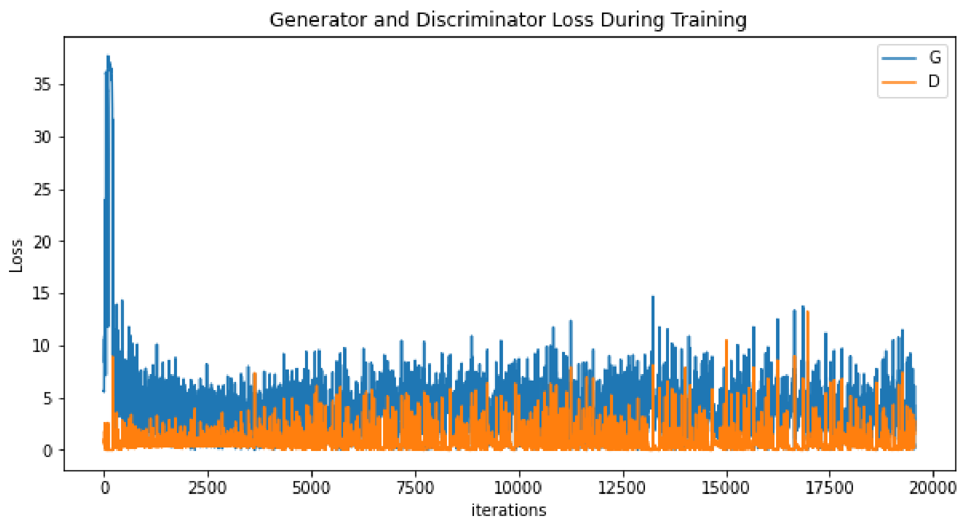
We trained on three separate occasions, on three separate instances – one with 20 epochs, one with 25 epochs, and one with 50 epochs. Each epoch has around 392 iterations, overall making it 7840, 9800, and 19600 iterations respectively.

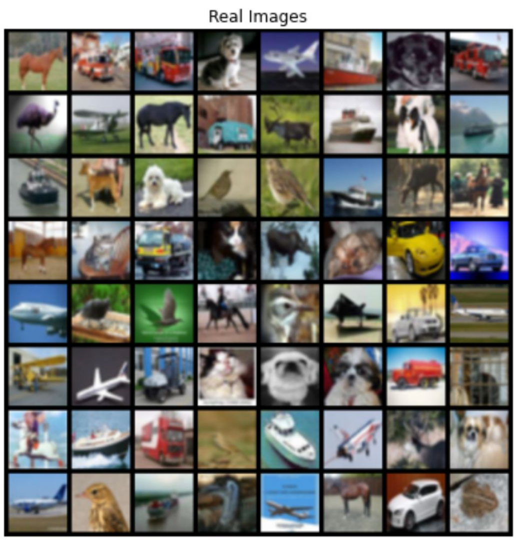
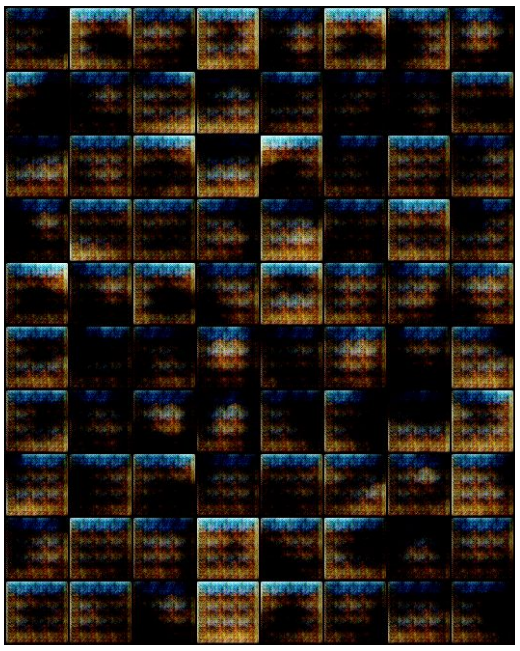
The parameters for all the three instances are consistent: 128 batch size, 64x64 image size, learning rate of 0.0002, β1 = 0.5, and

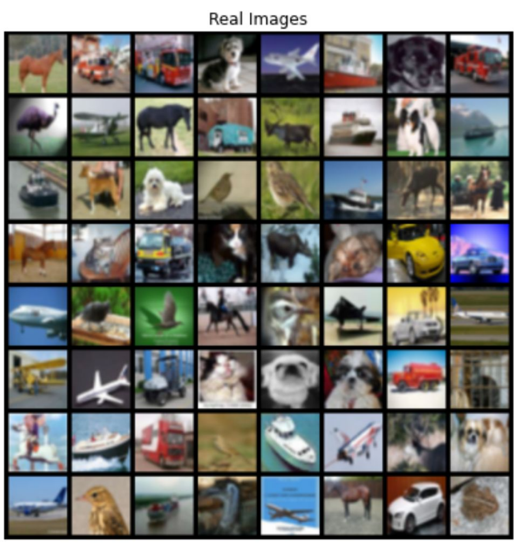
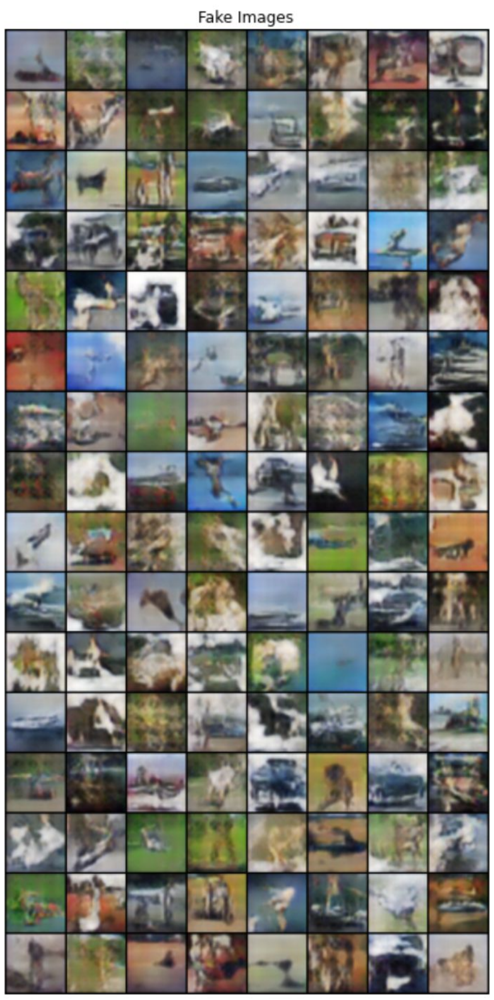
```
optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1, 0.9999))
```

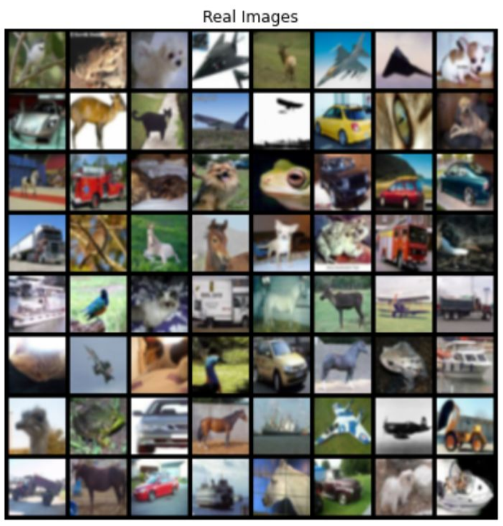
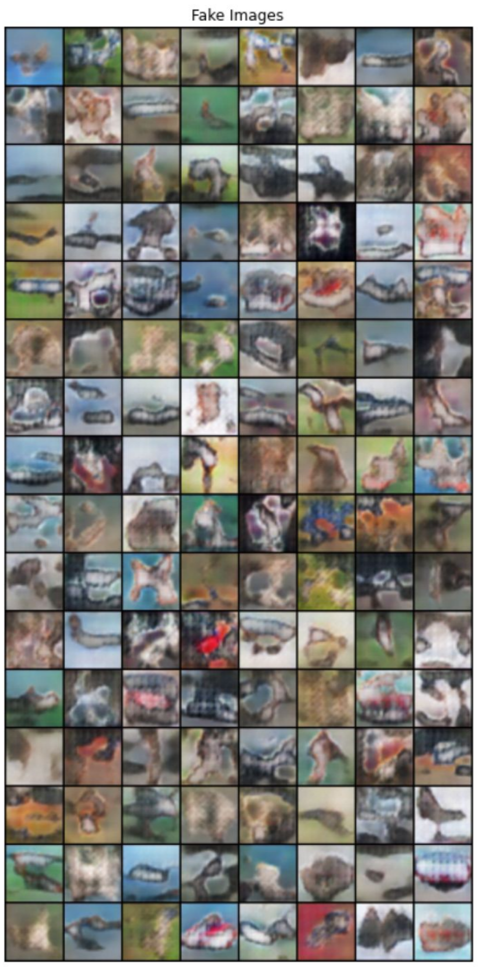
The Generator and Discriminator Loss During Training is consistent, a drop after the first 500 iterations, followed by fluctuations and some variations in the pictures generated, as well as classification by the discriminator. As for epoch=25, there came an anomaly during the 7000th iteration mark whereby the images generated became unsatisfactory, as well as the loss (for both Discriminator and Generator) not recovering and decreasing back.

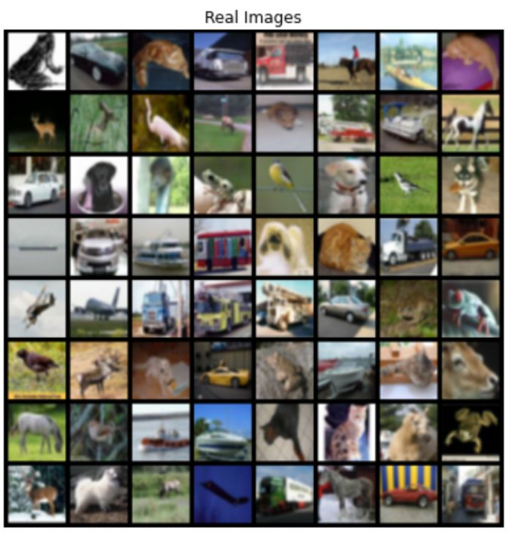
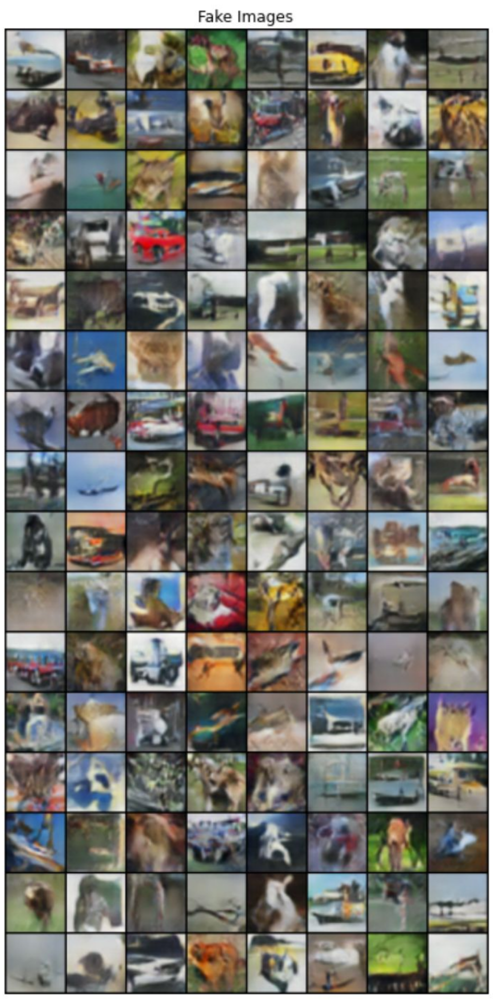
Number of	Loss Graph

Epochs	
Epoch = 20	
Epoch = 25	
Epoch = 50	

Number of Epochs	Real Images	Fake Images (Last sample/epoch)
Epoch = 1		

Epoch = 20 From the 1st Run		

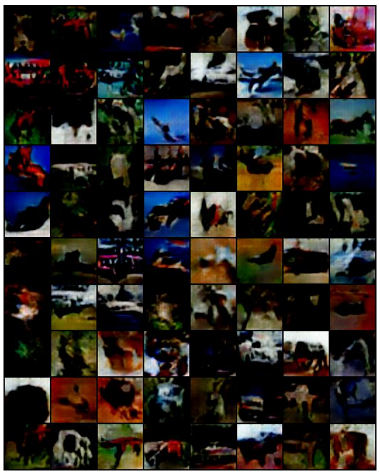
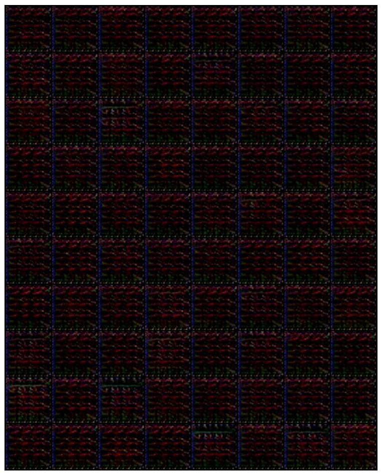
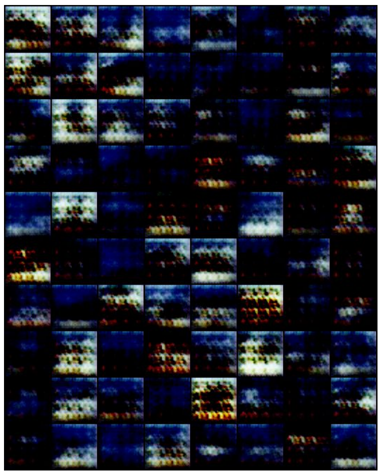
Epoch = 25 From the 2nd Run		

Epoch = 50 From the 3rd Run		

Full-sized images can be seen on this link: https://imgur.com/a/l3dqN1L

In conclusion, for DC-GANs, the model is more or less consistent and on par with what is desired when it comes to generating fake images. However, there were a few occasions whereby the loss for both Generator and Discriminator would spike up yet again, with the parameters retrained.

Sample data for Epoch = 25 / 2nd Run.

16th Epoch	17th Epoch	18th Epoch
		

Above is the sample image from Epoch = 25, from the second run, after the 16th, 17th and 18th epoch respectively. The processed samples and images have an increased amount of contrast on them, as well as the lowered brightness.

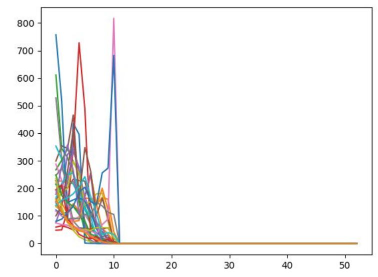
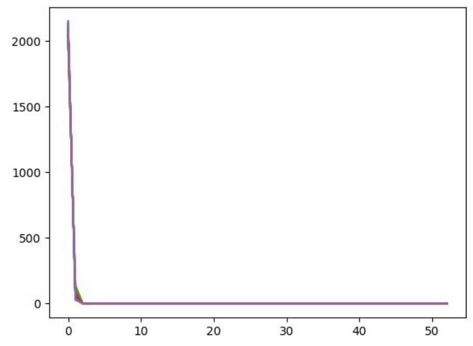
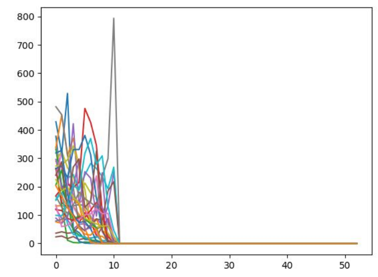
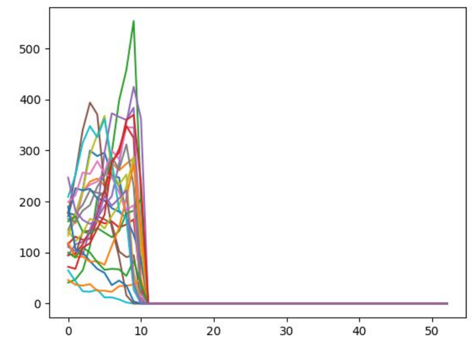
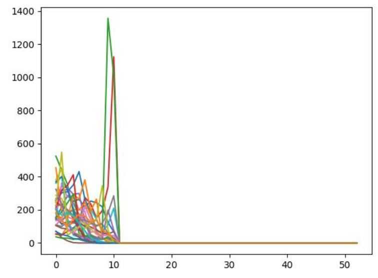
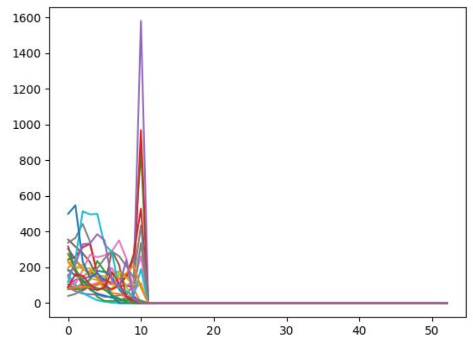
Overlooking random spikes and re-training the data would solve this loss-spike issue when it comes to DC-GAN.

3.2 Initial run with E-GAN code

To understand how E-GAN model works, we trained the model using CIFAR10 [6] dataset that contains 500000 images, with 500 images from each classes for validation samples and the remaining images will be used for training samples.

We trained for 231 epochs (30k iterations), at a learning rate of 0.0002 for both discriminator and generator and with batch size of 32.

To better understand the distribution of the real images and the generated images, we plotted the distribution graph. As shown in Fig. 4, as the number of epochs increases, the distribution from the fake data is getting similar to the distribution from the real data.

Number of Epochs	Real Images Distribution	Fake Images Distribution
1		
10		
50		

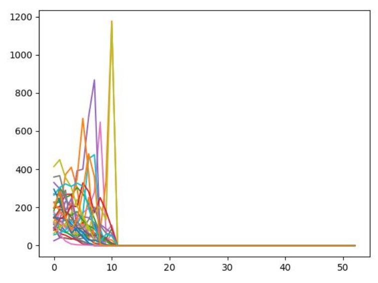
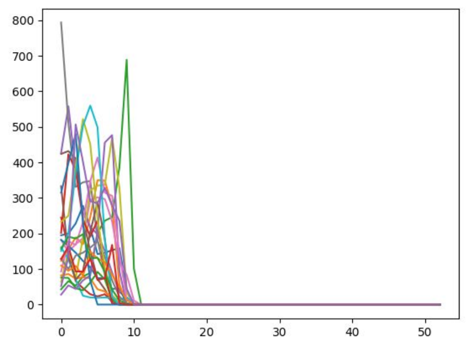
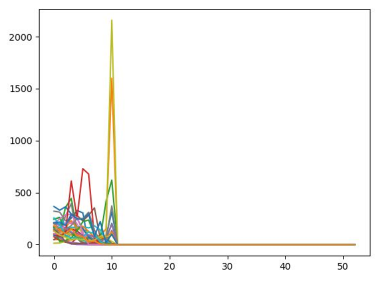
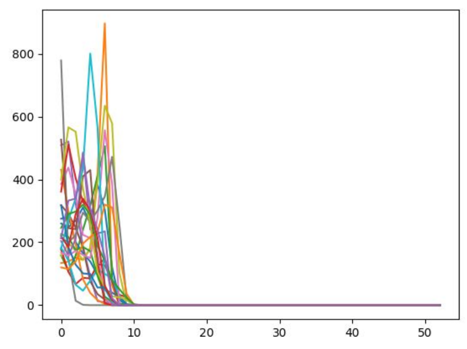
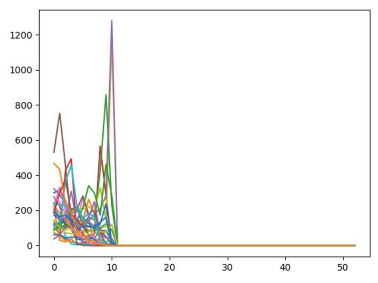
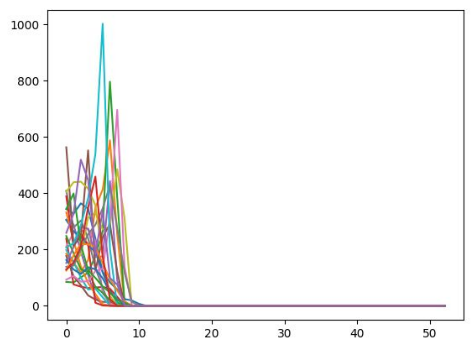
100		
200		
231		

Figure 4: Distribution comparison between real data and fake data

We also plotted the three different loss function:

1. Generator losses (see figure 5): g_fake
2. Discriminator losses (see figure 6): d_real, d_fake
3. Losses (see figure 7): overall generator losses, overall discriminator losses

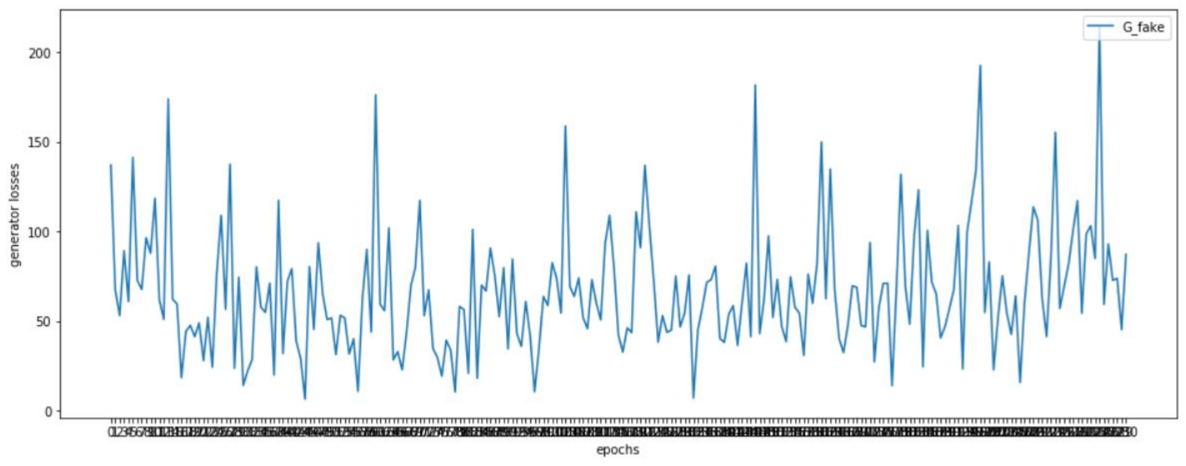


Figure 5: Generator losses breakdown

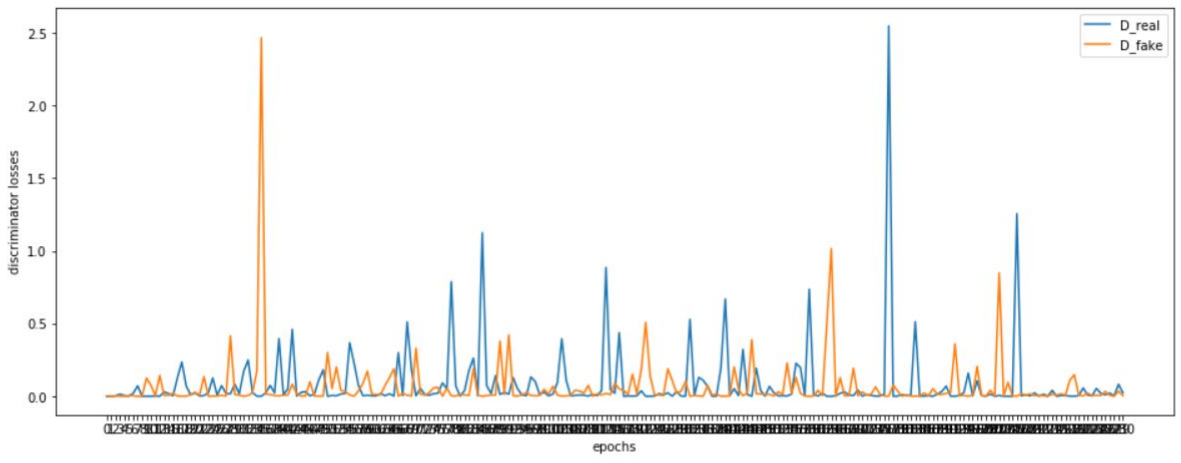


Figure 6: Discriminator losses breakdown

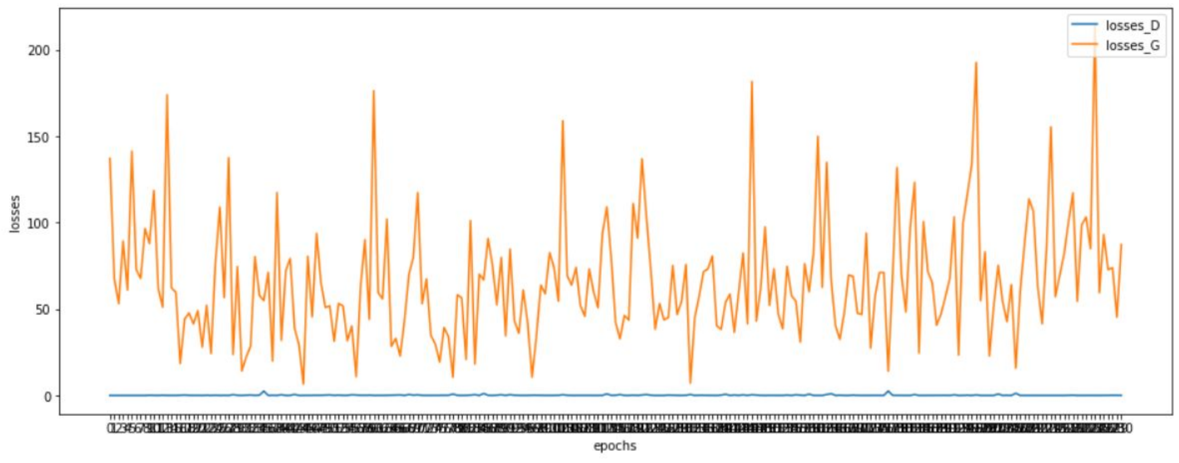
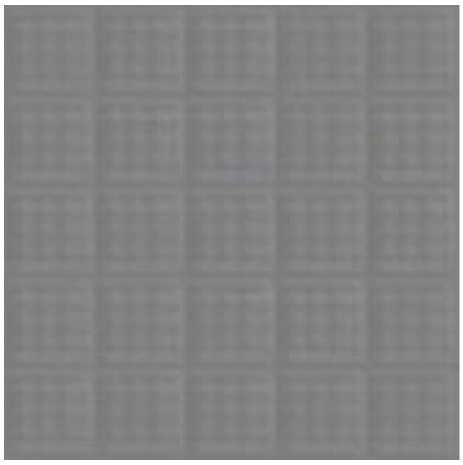
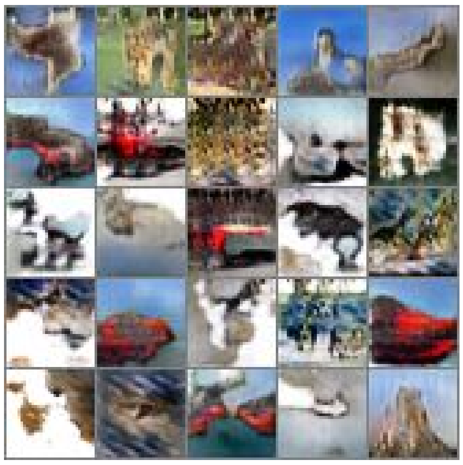
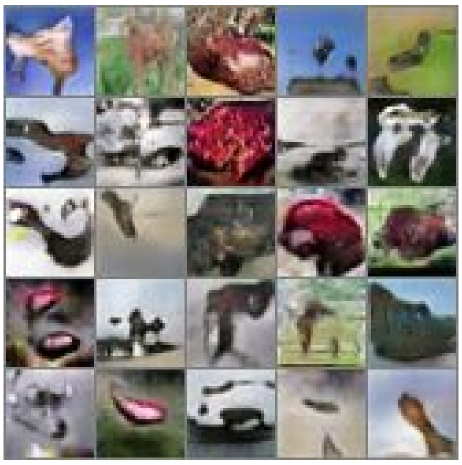
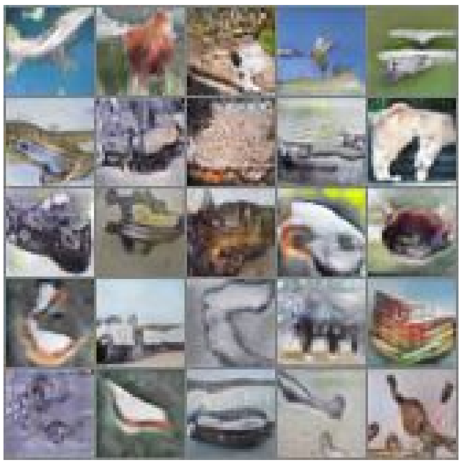
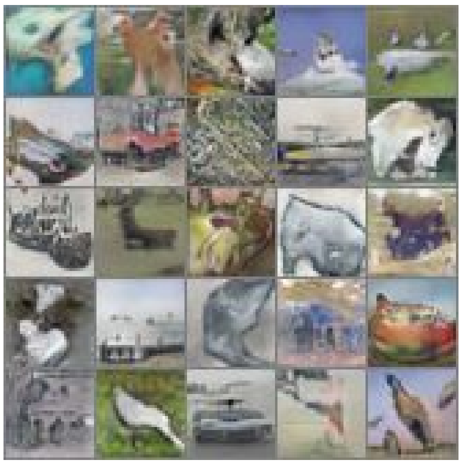
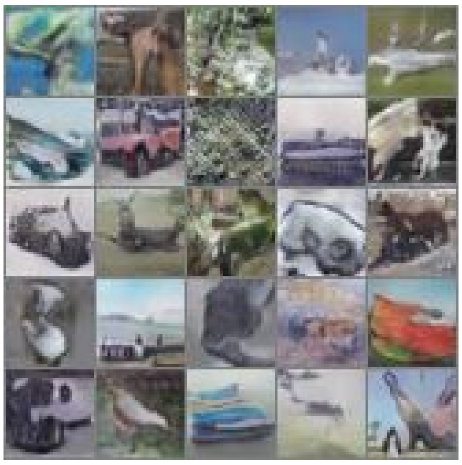


Figure 7: Losses from Generator and Discriminator

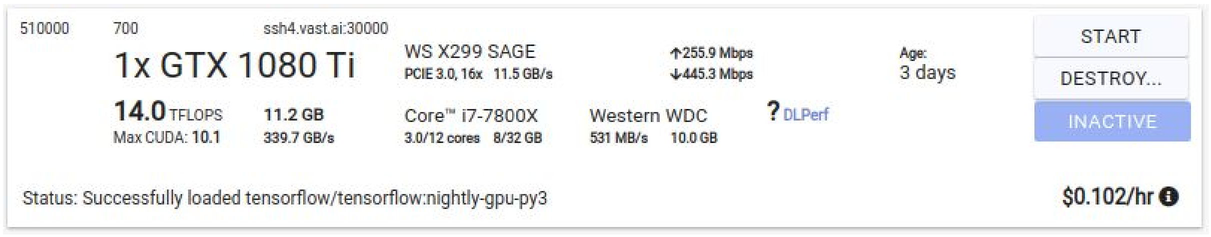
The real images and the generated images are shown below at different epochs.

Number of Epochs	Real Images	Fake Images (Last sample/epoch)
Epoch = 1		
Epoch = 50		
Epoch = 100		

Epoch = 150		
Epoch = 200		
Epoch = 231		

3.3 Comparing E-GAN with DCGAN

We ran both our models on vast.ai. The specs of the testing environment for both DCGAN and E-GAN were pretty similar:



510000 700 ssh4.vast.ai:30000

1x GTX 1080 Ti

WS X299 SAGE
PCIE 3.0, 16x 11.5 GB/s

↑255.9 Mbps
↓445.3 Mbps

Age:
3 days

14.0 TFLOPS
Max CUDA: 10.1

11.2 GB
339.7 GB/s

Core™ i7-7800X
3.0/12 cores 8/32 GB

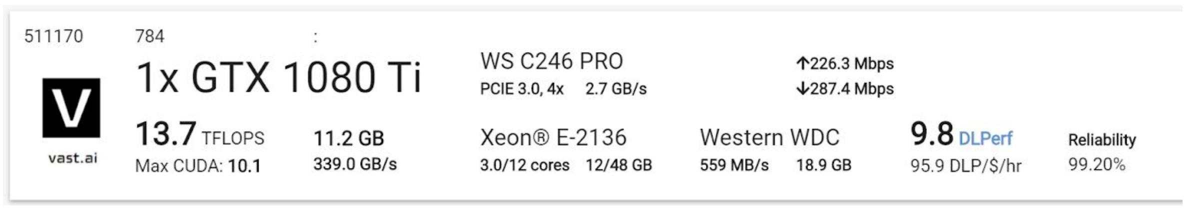
Western WDC
531 MB/s 10.0 GB

? DLPerf

Status: Successfully loaded tensorflow/tensorflow:nightly-gpu-py3

$0.102/hr

Figure 8: Instance used for E-GAN



511170 784 :

1x GTX 1080 Ti

WS C246 PRO
PCIE 3.0, 4x 2.7 GB/s

↑226.3 Mbps
↓287.4 Mbps

13.7 TFLOPS
Max CUDA: 10.1

11.2 GB
339.0 GB/s

Xeon® E-2136
3.0/12 cores 12/48 GB

Western WDC
559 MB/s 18.9 GB

9.8 DLPerf
95.9 DLP/$/hr

Reliability
99.20%

Figure 9: Instance used for DCGAN

As you can see from the results from previous sections, the images generated by both DCGAN and EGAN look very similar but there's a huge difference in the training time. To run 50 epochs in DCGAN it only took around 40 mins but to run 50 epochs in E-GAN, it took around 2 hrs and the results were pretty bad, so we ran E-GAN for a total 231 epochs (30,000 iterations) which took around 9hrs.

There are various reason why the generated images from both E-GAN are not as good as we expected them to be:

1. **Dataset:** It is possible that the CIFAR-10 dataset does not work well with the E-GAN model as it has 10 different classes. In future, we could possibly isolate these classes and train them on E-GAN individually.
2. **Requires further training:** In the E-GAN paper, they trained their model on a single dataset for around 500,000 iteration (roughly around 4,000 epochs). Due to the limited access to powerful systems, we were only able to run the model for 231 epochs (30,000 iterations). In future, if we have 24hr access to a powerful system then we can try running the model for 4,000 epochs.

4. Conclusion

4.1 Summary of objectives learned from this research

E-GAN is a novel approach of generating images and it has a lot of benefits over other GAN models such as DCGAN and LSGAN (as discussed in the previous sections). But currently the performance is not as good as we expected. It requires a lot of computing resources and the training time is also very high yet the size of the generated images is very small. From our tests, DCGAN performed much better as compared to E-GAN in terms of time and computation resources taken and the result was almost similar. Maybe if we would trained the E-GANs for more epochs then it might have performed better. The current code for E-GAN requires to be optimized further and if more research goes into this technique then maybe the performance of E-GAN can be improved further.

4.2 Challenges faced

1. **COVID-19:** The whole COVID-19 situation made it hard for the team to work together on the project as we were unable to meet in-person and it was also a morally challenging situation.
2. **Github code was hard to understand and we had to refactor parts of the code:** The GitHub repository for E-GAN was not very clean and the explanation on the code was unclear. There was a lot of ambiguity in the README file and oftentimes there were certain dependencies that were not included and we had to figure out certain parts by ourselves.
3. **Training takes a long time and requires a lot of resources:** There were more than 20 million parameters in both the Discriminator and Generator in the E-GAN model but the generated images that were produced were only 32x32 pixel big. Due to such a high number of trainable parameters, E-GAN took a long time to train.
4. **Not much research done in this area:** This technique is still relatively new and not much research has been done in light of it. This meant we did not have any other sources to cross-reference.
5. **Dependency Issues:** During our initial runs of E-GAN, we noticed that the original code required both PyTorch and Tensorflow

libraries which caused a lot of dependency issues. Through trial and error, we were able to figure out what we needed to do (e.g. installing Tensorflow in our environment first and then Pythorch allowed). Since this code's repository is not maintained, we also faced issues with the version of scipy library. The code would only work on the older version of scipy but in requirements.txt file, it was mentioned otherwise.

4.3 Future works

For this project we were focused heavily on understanding the E-GAN framework and getting to work the code despite the various problems that it came with (mentioned in section 3.5). As such we were only able to test it out and compare it against DCGAN with the CIFAR10 dataset.

A future work would be to improve on the existing E-GAN model. Firstly, we would research into the different ways of updating weights, and implement E-GAN with the different variations in mind to see if there are any observable improvements. We would then benchmark the performance of the modified E-GAN with vanilla E-GAN, and subsequently benchmark it with other GAN models such as LS-GAN or WGAN.

E-GAN can also be integrated with progressive GAN models and we compare the performance of this proposed pregressive E-GAN model with the vanila pregressive GAN models.

4.4 Project files

Github code for E-GAN:
https://github.com/AndriYang/Evolutionary-GAN-DL

Code referenced for DCGAN:
https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

CIFAR10 dataset:

https://www.cs.toronto.edu/~kriz/cifar.html

LSUN dataset:
https://www.yf.io/p/lsun

5. References

[1] Wiggers, K. (26 December 2019). Generative adversarial networks: What GANs are and how they've evolved. Retrieved from https://venturebeat.com/2019/12/26/gan-generative-adversarial-network-explainer-ai-machine-learning/

[2] Wang, C., Xu, C., Yao, X., & Tao, D. Evolutionary generative adversarial networks. IEEE Transactions on Evolutionary Computation, 23(6), 921-934, 2019.

[3] A. E. Eiben and J. Smith. From evolutionary computation to the evolution of things. Nature, 521(7553):476, 2015.

[4] Y. Wang, C. Xu, J. Qiu, C. Xu, and D. Tao. Towards evolutional compression. arXiv preprint arXiv:1707.08005, 2017.

[5] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. arXiv preprint arXiv:1506.03365, 2015.

[6] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[7] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In the IEEE International Conference on Computer Vision (ICCV), pages 3730-3738, 2015.

[8] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In Proceedings of the International Conference on Learning Representations (ICLR), 2016.

[9] K. Umezawa, "Medium", 18 April 2018. [Online]. Available: https://medium.com/@keisukeumezawa/dcgan-generate-the-images-with-deep-convolutinal-gan-55edf947c34b. [Accessed 22 April 2020].