Midterm report on Generative Adversarial Networks An in-depth research and enhancing through Convolutional Neural Networks

Nguyen Nguyen-Binh
VNU-UET
21020526@vnu.edu.vn

Hoang Tran-Ba
VNU-UET
21020631@vnu.edu.vn

***Abstract*—This report is about the research and experiments on Generative Adversarial Networks for the Midterm Project of Computer Vision class INT3412 20 in Fall 2023. We report on the original Generative Adversarial Networks paper [1], giving our opinions and carrying out experiments on the GAN architecture. We also experimented with an updated version of GAN that we came up with. Our source code is available at: https://github.com/hoangbros03/GAN-implement.**

I. Introduction

Generative Adversarial Networks (GAN) was introduced in 2014 as a new framework for estimating generative models via an adversarial process. It was a breakthrough in the field of generative models. GAN has many real-world applications such as generating examples for image datasets, text-to-image translations, face aging, etc. In this report, we provide an in-depth research into the original GAN paper and provide an enhanced version of GAN through Convolutional Neural Networks.

II. Generative Adversarial Networks

A. Idea

In the Generative Adversarial Networks architecture, two models are trained simultaneously: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. This framework corresponds to a minimax two-player game. To make it easy to understand, the generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency.

B. Model

In the original GAN paper, the authors explored the special case when the generative model generates samples by passing random noise through a multilayer perceptron, and the discriminative model is also a multilayer perceptron. The authors could train both models using only the highly successful backpropagation and dropout algorithms and sample from the generative model using only forward propagation.

We define some variables as follows:

- p_g: generator's distribution
- $p_z(z)$: prior on input noise variables.
- $G(z; \theta_g)$: mapping to data space
- $D(x)$: probability that x came from the data rather than p_g
- $D(x; \theta_d)$: multilayer perceptron that outputs a single scalar.

The goal of training is to:

- Train D to maximize the probability of assigning the correct label to both training examples and sample G.
- Train G to minimize $\log(1 - D(G(z)))$

Therefore, we can say that D and G play the two-player minimax game with value function $V(D, G)$ as follows:

$$V = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z))]$$

In the paper, the authors alternated between k steps of optimizing D and one step of optimizing G. This results in D being maintained near its optimal solution, so long as G changes slowly. The authors also stated that early in training, instead of training G to minimize $\log(1 - D(G(z))$, they trained G to maximize $\log D(G(z))$, as it provided much more gradient in early training.

C. Theoretical theorems and proofs

There are two things we need to prove to conclude that the adversarial process works as expected:

- The global optimality of the training criterion is $p_g = d_{data}$.
- The training algorithm of GAN makes p_g converge to p_{data} at the end.

These claims are proved with the following propositions and theorems.

Proposition 1. For G fixed, the optimal discriminator D is

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

The proof of this proposition was included in the original paper. Now, the minimax game can be reformulated as:

$$\begin{aligned} C(G) &= \max_V(G, D) \\ &= \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_G^*(x)\right] + \mathbb{E}_{x\sim p_g}\left[\log(1 - D_G^*(x)\right] \\ &= \mathbb{E}_{x\sim p_{data}(x)}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right] \\ &\quad + \mathbb{E}_{x\sim p_g}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right] \end{aligned}$$

Theorem 1. The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-\log 4$.

Proof. From the previous equation, we can obtain:

$$\begin{aligned} (G) = &- \log 4 + KL\left(p_{data} \| \frac{p_{data} + p_g}{2}\right) \\ &+ KL\left(p_g \| \frac{p_{data} + p_g}{2}\right) \end{aligned}$$

In this equation, KL is the Kullback-Leibler divergence. This equation can be re-written with the Jensen-Shannon divergence as follows:

$$C(G) = -\log 4 + 2JSD(p_{data} \| p_g)$$

The Jensen-Shannon between two distributions is always non-negative and zero only when they are equal. Therefore, we have shown that $C^* = -\log 4$ is the global minimum of $C(G)$ and that the only solution is $p_g = p_{data}$. This means the generative model perfectly replicates the data generating process.

Proposition 2. If G and D have enough capacity, and at each step of the training algorithm, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion

$$\mathbb{E}_{x\sim p_{data}(x)}\left[\log D_G^*(x)\right] + \mathbb{E}_{x\sim p_g}\left[\log(1 - D_G^*(x)\right]$$

then p_g converges to p_{data}.

This proposition is also proved in the original paper. Now, we can claim that in theory, in Generative Adversarial Networks, the generator G can successfully replicates the generation process.

III. Our opinion on GAN

We examined some of the methods for image generation tasks and saw that GAN is one of the main methods to handle this task. Despite the age of GAN (since 2014), this method proved efficient in replicating similar images which are indistinguishable from the real images from the training dataset. In terms of speed, GAN can generate a hard-to-classify image in a much shorter time than diffusion models, whose technique is slowly removing noises from Gaussian noise images. Also, GAN showed reliable results on other related challenges, such as Image-to-Image Translation or Semantic-Image-to-Photo Translation.

However, the generator and discriminator in GAN have to be learned on a proper scale to avoid one model being superior to another. If it happens, either the generator model can pool the discriminator model with non-sense images or the discriminator model can classify too well and the generator model has no idea how to improve itself. As a result, model architectures have to be considered with caution to avoid model collapse behavior. Moreover, the losses measured in both models do not provide critical insights for us. Even when the model is corrupted, the losses are still stable with low numbers. On the other hand, GAN is inferior compared to other methods in terms of creativity as experimented by others.

Fig. 1. Example of GAN performance on Image-to-Image Translation task.

IV. Experiment with original GAN

We first tried to implement the GAN using the information gathered from the original paper. Due to the fact that the architecture details of both the generator model and discriminator are not provided, we have to figure out on ourselves using the combinations of Linear and Activation function layers. The algorithm and the dataset (MNIST) are re-implemented with no change. However, the results provided are not ideal. After 200 epochs, the generated images are mostly the same but they look like the corrupted QR codes, while the loss of generator and discriminator models is somewhat low and stable. Interestingly, we observed the images at intervals of every 10 epochs, and this pattern persisted from the 20th epoch until the conclusion of the training process. Our inference is that our model experienced a collapse. Further insights were gained by consulting articles on this issue, which asserted that such behavior is a common occurrence and easily encountered in GAN.

In order to resolve the problem, we have tried to do the following adjustments:

- Change the Adam's learning rate: We tested with several learning rate number, from 0.0001 to 0.001.
- Change the dimension of the latent space: We change between the rate of 100 and 256 as recommended in online guides.

- Increase the epoch number: We let the model train up to 2000 epochs, which takes nearly 10 hours to finish the process.
- Make the ground truth output dynamic: The ground truth values for "fake" were adjusted to a range of [0, 0.3], while "real" was set to [0.7, 1], introducing a nuanced perspective to the adversarial training process.
- Change the proportion between generator and discriminator model in training process: In the initial GAN paper, the discriminator was trained at every step, with the generator trained only once per epoch. Our experimentation involved equalizing their training opportunities or establishing a fixed proportion between the generator and discriminator training frequencies.
- Adjust the layers inside models: We added and removed some batch normalization layers and dropout layers with different configurations. In terms of activation function, we also switch between ReLU, LeakyReLU, Sigmoid, and Tanh function.

After the tremendous efforts on tweaking the hyperparameters and methods, we finally can produce a usable generator model. The architecture of both models are displayed in tables I and II.

TABLE I
GENERATOR ORIGINAL GAN ARCHITECTURE

Layer (type:depth-idx)	Output Shape	Param #
Generator	[1, 1, 28, 28]	4,021,648
Sequential: 1-1	[1, 784]	–
Linear: 2-1	[1, 256]	25,856
LeakyReLU: 2-2	[1, 256]	–
Linear: 2-3	[1, 512]	131,584
LeakyReLU: 2-4	[1, 512]	–
Linear: 2-5	[1, 1024]	525,312
LeakyReLU: 2-6	[1, 1024]	–
Linear: 2-7	[1, 784]	803,600
Tanh: 2-8	[1, 784]	–

TABLE II
DISCRIMINATOR ORIGINAL GAN ARCHITECTURE

Layer (type:depth-idx)	Output Shape	Param #
Discriminator	[1, 1]	–
Sequential: 1-1	[1, 1]	–
Linear: 2-1	[1, 1024]	803,840
LeakyReLU: 2-2	[1, 1024]	–
Dropout: 2-3	[1, 1024]	–
Linear: 2-4	[1, 512]	524,800
LeakyReLU: 2-5	[1, 512]	–
Dropout: 2-6	[1, 512]	–
Linear: 2-7	[1, 256]	131,328
LeakyReLU: 2-8	[1, 256]	–
Dropout: 2-9	[1, 256]	–
Linear: 2-10	[1, 1]	257
Sigmoid: 2-11	[1, 1]	–

V. OUR PROPOSED UPDATED GAN

In the original GAN paper, one thing we noticed from the beginning is the use of multilayer perceptron in both the generator G and discriminator D. We concluded that we can improve the model by using more complex architectures in the generator and discriminator. This makes both these components perform better in their tasks, hence make the generated images look better. Therefore, we used Convolutional Neural Networks (CNN) architecture [2] in both the generator and discriminator to make then newly updated GAN model. Notably, in generator, we used transposed convolution layers. The architecture and tensor size we used to train with MNIST dataset is described in figure 2 and 3. Between each

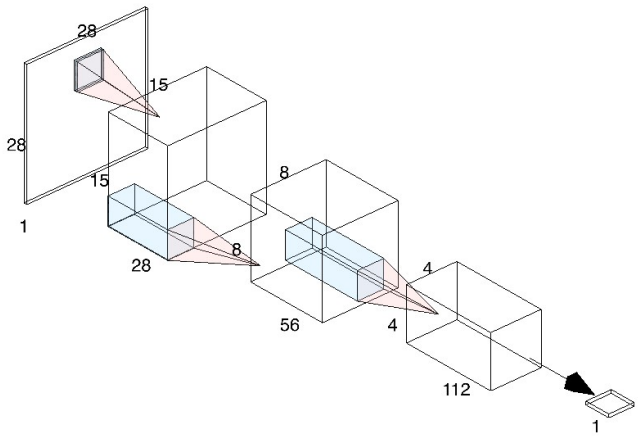

Fig. 2. Discriminator convolutional architecture for MNIST dataset.

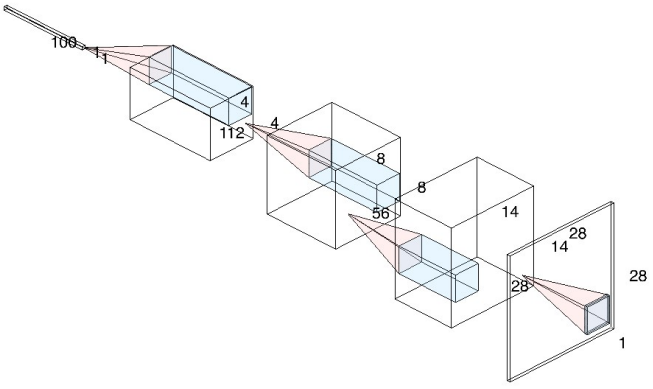

Fig. 3. Generator transposed convolutional architecture for MNIST dataset.

convolutional layer in discriminator and generator, 2D batch normalization layers and LeakyReLU activation are used. At the final layer of the discriminator, Sigmoid function is used to create the probability prediction. We trained the model with the Adam optimizer.

When training for MNIST dataset and CelebA dataset, we had to use different sizes for convolutional layers due to different input size of images. However, the whole architecture is maintained throughout these two experiments.

One notable detail in our implementation is the proportion of steps between optimizing D and optimizing G. We observed

that optimizing D once per each time of optimizing G did not yield good results. We found that we needed k steps of optimizing G for each step of optimizing D. This showed much better performance in both datasets. The opposite was true for the original paper, where the authors optimized k steps of D for each step of optimizing G. The detailed performance of this architecture is reported in section VI.

VI. Experimental evaluation

A. Dataset and Metrics

We use the MNIST Database and the CelebA dataset in our experiment respectively. For evaluation, we use both manual evaluation as well as the Frechet Inception Distance.

1) Handwritten digit MNIST Database: MNIST, short for the Modified National Institute of Standards and Technology database, is a widely used dataset in the field of machine learning and computer vision. It consists of a vast collection of 28x28 pixel grayscale images, each depicting handwritten digits (0 through 9). MNIST has an important role in the development and evaluation of various image classification algorithms. It serves as a foundational dataset for researchers and practitioners alike, providing a common ground for comparing and advancing image processing techniques.

2) CelebA dataset: The CelebFaces Attributes Dataset (CelebA) [3] is an extensive collection of face attributes encompassing over 200,000 images of celebrities. This dataset includes images with diverse poses and background complexities. We exclusively use this dataset with the enhanced GAN, while the original GAN, which contains predominantly composed of multi-layer perceptron layers, is too simplistic to yield reliable results.

3) Frechet Inception Distance: The Fréchet inception distance (FID) [4], which was introduced in 2017, is a standard metric used to assess the quality of images created by a generative model, especially generative adversarial network (GAN). This metric assesses the likeness between generated and real images by computing the distance between their respective feature vectors. In our evaluation process, we meticulously selected 128 stable samples from the ground truth and juxtaposed them with a batch of 128 images generated by the GAN generator.

4) Kernel Inception Distance: Kernel Inception Distance (KID) [5] is a metric used to evaluate the quality of generated images in the context of generative models. Introduced as an improvement over Fréchet Inception Distance (FID), KID focuses on comparing the statistical similarity of feature representations between real and generated images. It utilizes the Maximum Mean Discrepancy (MMD) to measure the difference in distributions of feature embeddings from an Inception Network. A lower KID score indicates higher similarity between the generated and real images, signifying better image quality. This metric provides a more nuanced understanding of the performance of generative models by considering the distributional differences in feature space.

B. Original GAN

Our "original GAN" was trained with both models have equal training times, while other configurations are similar as discussed. Since both models are relatively small, we used our personal laptop equipped with a GTX1060 GPU for the training process. Despite we decided to take 200 epochs, the generator model can produce some meaningful samples from epoch 40th. After the training process, we can see some noise pixels are removed, and the quality of generated images is not different than the real images at all.

Fig. 4. Images generated from original GAN in the epoch 40th.

Fig. 5. Images generated from original GAN in the epoch 200th.

We also recorded the loss from both models in every epochs and made a line chart in image 6 with the support from Wandb tool. We can easily observe that they are not fluctuate significantly, which is a good sight for training process.

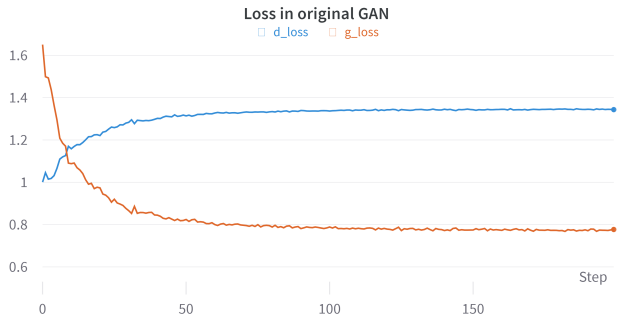

Fig. 6. Loss in original GAN.

After that, we chose to load model checkpoints at several epochs, obtain the generated samples, and use metric discussed above to evaluate. Since the FID only works with the RGB images, we have to convert the images to the ideal shape. The result is displayed in the table III. Our model gained the lowest FID and KID scores on epoch 100th, and kept the reasonable range of scores when continue the training process.

TABLE III
FID AND KID SCORES AT DIFFERENT EPOCHS

Epoch	FID	KID
10	1.1595	(mean: 1.0729, std: 0.0569)
20	0.3653	(mean: 0.3210, std: 0.0292)
30	0.2598	(mean: 0.2436, std: 0.0237)
40	0.2331	(mean: 0.1720, std: 0.0223)
50	0.4441	(mean: 0.3089, std: 0.0348)
60	0.1591	(mean: 0.1055, std: 0.0148)
70	0.1849	(mean: 0.1255, std: 0.0199)
80	0.1367	(mean: 0.1087, std: 0.0143)
90	0.1126	(mean: 0.0796, std: 0.0099)
100	**0.0769**	**(mean: 0.0501, std: 0.0088)**
110	0.1333	(mean: 0.0874, std: 0.0139)
120	0.1228	(mean: 0.0883, std: 0.0124)
130	0.1053	(mean: 0.0764, std: 0.0105)
140	0.0993	(mean: 0.0581, std: 0.0104)
150	0.1654	(mean: 0.1112, std: 0.0141)
160	0.1522	(mean: 0.0914, std: 0.0163)
170	0.1578	(mean: 0.0992, std: 0.0177)
180	0.2129	(mean: 0.1322, std: 0.0219)
190	0.1422	(mean: 0.0982, std: 0.0138)
200	0.1287	(mean: 0.0862, std: 0.0118)

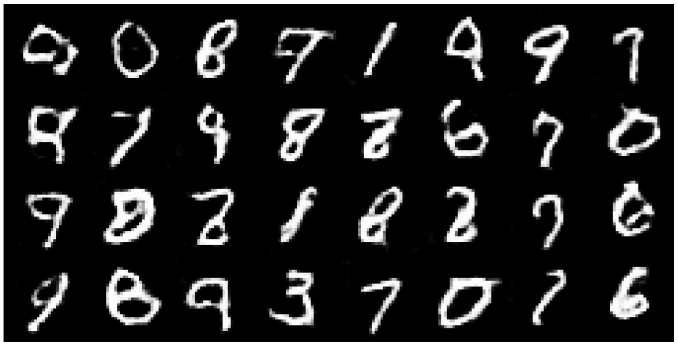

Fig. 7. MNIST generating results with the enhanced GAN.

C. Enhanced GAN

Images of the enhanced GAN architecture trained on the MNIST dataset are shown in figure 7. As we can see, the model successfully generated photos of hand-written numbers in the MNIST dataset. The generated data are very similar to the original MNIST dataset, and totally interpretable by human.

In figures 8, 9, 10 are images of the enhanced GAN architecture trained on the CelebA dataset throughout training iterations in sequential order. As we observe, the enhanced GAN architecture successfully recreated human faces authentically. Throughout iterations, the network improved gradually in generating human faces. Although the performance can still be improved, we argue that this output shows that the GAN architecture worked as expected and produced better results than then original GAN architecture.

In figure 11 is the loss of discriminator and generator throughout iterations in the CelebA dataset. Note that each iteration is a batch of 32 images in the dataset. In the first iterations, the generator's loss drops dramatically. After that, both the loss of the generator and the discriminator remain stable, with the generator having a slight increase.

Fig. 8. CelebA generating results with the enhanced GAN (1/3).

Fig. 9. CelebA generating results with the enhanced GAN (2/3).

Fig. 10. CelebA generating results with the enhanced GAN (3/3).

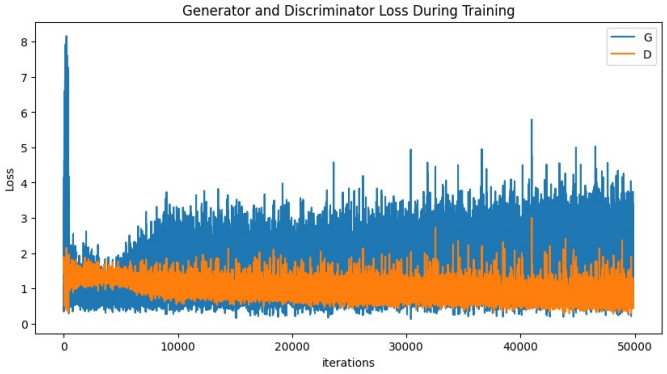

Fig. 11. Training loss on CelebA dataset.

The results throughout iterations of the enhanced GAN on Frechet Inception Distance (FID) and Kernel Inception Distance (KID) metrics are displayed in table IV and V. Note that the whole dataset of CelebA that we trained on contains approximately 6332 batches, which means 6332 iterations per epoch. For MNIST, one dataset contains 1875 batches. As we can see, the FID and KID scores decreases gradually, showing the performance of the GAN model improving by iterations.

TABLE IV
RESULTS THROUGHOUT ITERATIONS OF THE ENHANCED GAN ON FRECHET INCEPTION DISTANCE AND KERNEL INCEPTION DISTANCE METRIC ON CELEBA DATASET.

Iteration	FID	KID
1	324.0590	0.5369, 0.0066
2000	20.8359	0.4013, 0.0095
4000	5.8091	0.3234, 0.0133
6000	4.2050	0.2258, 0.0102
8000	2.1601	0.1688, 0.0072
10000	1.6035	0.1293, 0.0065
12000	1.8569	0.1214, 0.0072
14000	2.0928	0.1089, 0.0061
16000	2.2177	0.0809, 0.0059
18000	1.6322	0.0842, 0.0048
20000	1.2611	0.0648, 0.0041

TABLE V
RESULTS THROUGHOUT ITERATIONS OF THE ENHANCED GAN ON FRECHET INCEPTION DISTANCE AND KERNEL INCEPTION DISTANCE METRIC ON MNIST DATASET.

Iteration	FID	KID
1	24.3796	0.6944, 0.0091
2000	2.0513	0.1909, 0.0068
4000	0.1448	0.0520, 0.0039
6000	0.0479	0.0355, 0.0049
8000	0.0290	0.0177, 0.0033

Comparing the FID and KID scores of original and enhanced GAN, applying Convolutional Neural Network shows a remarkable improvement in the overall result. Despite the disparity in measurement methodologies between the two GAN versions — one recorded on each epoch and the other on each iteration — the CNN layers demonstrate a significant improvement in overall performance. CNN helps model convergence better than the Multi-Layer Perceptron (MLP) approaches, thus proving our successful efforts in advancing GAN capabilities.

VII. ABLATION STUDY

An ablation study investigates how well the AI models perform by pruning some layers to gain insights into the contribution of the layers to the overall system. Comparing our two main GAN versions, we don't really understand how batch normalization layers can help in the training process. In fact, the graph which is conducted from the original GAN is smoother than the graph of the enhanced GAN. As a consequence, our main objective in the ablation study is to investigate the role of batch normalization layers in enhanced GAN. Since we had struggled in building the original GAN architecture, cutting one or several layers in the original GAN is too risky and would bring garbage-generated samples only.

A. Experiment setup

To conduct the experiment, we use 4 pairs of discriminator and generator models: normal-normal, ablation-normal, normal-ablation, ablation-ablation. "Normal" means the normal version of enhanced GAN, while "ablation" means the version whose batch normalization layers had been removed. After that, we set up the same configuration on each pair, train each of them in 10 epochs, then record the losses in every step. It took one hour to train each pair on Google Colab with NVIDIA T4 as the hardware accelerator.

B. Result

1) Loss of models: Interestingly, in the charts of both models losses, batch normalization layers seem to make the training process more fluctuate. Moreover, pruning the batch normalization layers in both models can deliver a much smoother line chart. Normally, a stable record of loss values implies a success in training process that model gradually studies the features of input data within a reasonable learning rate. We continued to evaluate the performance between versions.

2) Performance of models: We used the discussed metric and evaluate the results from the last epoch of each model, then recorded in the table VI. Despite the stable training processes, it seems that "ablation-normal" and "ablation-ablation" models were collapsed with abnormal high scores. On the other hand, pruning in generator model and keeping the discriminator model gave a slightly higher performance.

TABLE VI
DISCRIMINATOR AND GENERATOR TYPES VS. FID AND KID SCORES

Discriminator Type	Generator Type	FID	KID
Normal	Normal	0.5702	(0.3948, 0.0426)
Ablation	Normal	2.3231	(1.7774, 0.1175)
Normal	Ablation	0.5570	(0.3888, 0.0512)
Ablation	Ablation	6.7573	(8.4936, 0.4497)

3) Conclusion on ablation study: Despite our limited understanding of the underlying causes of this phenomenon, we derive the following insights:

- GAN is generally unstable and need much more tuning effort than other type of generative models.
- Batch normalization layers can play a vital role in training GAN and should be implemented to enhance the result.
- A careful analysis and practical experiences play important roles in achieving success in GAN training processes. Real-time visualization is recommended to figure out any problems when training immediately.
- Our conventional understanding of the contributions of layers in typical deep learning problems may not directly apply to the image generation tasks (such as stable loss values doesn't mean a converge model.

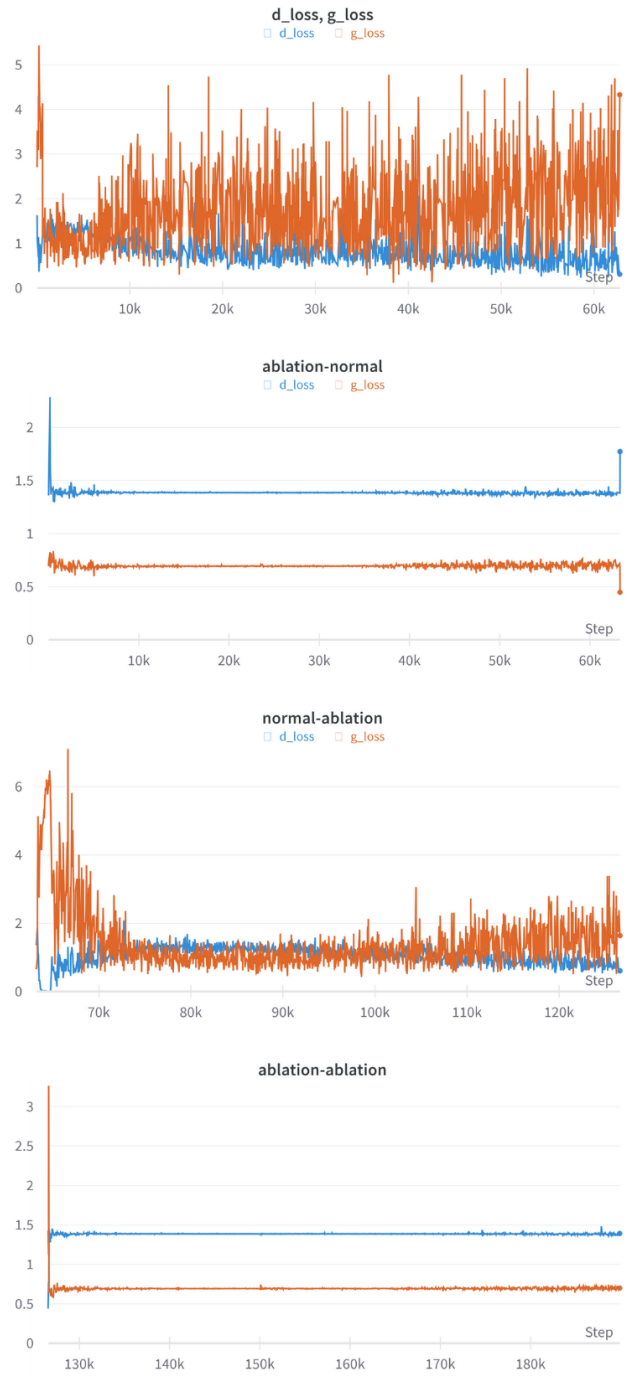

Fig. 12. Graphs from our experiments. From top to bottom: normal-normal, ablation-normal, normal-ablation, ablation-ablation.

VIII. Member Contributions

Both team members made equal and substantial contributions throughout every phase of the project, encompassing research, implementation, enhancement, and experimental evaluation. In our repository, we didn't use any available templates on the Internet, while our deep learning models were carefully built with enormous effort to test their efficient. Also, despite some suspicious comment lines in our python files, we wrote it by ourselves to conform the best practices and Pylint's checks. We believe this skill is beneficial and needed for our future.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[2] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems*, vol. 2, 1989.

[3] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," December 2015.

[4] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," 2018.

[5] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton, "Demystifying mmd gans," 2021.