**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**-oOo-**

# BÁO CÁO

## MÔN: Lập trình Python cho Máy học - CS116.M11

**Đề tài: Tìm hiểu Generative Adversarial Networks (GANs)**


**Giảng viên hướng dẫn:** TS. Nguyễn Vinh Tiệp

| **Sinh viên thực hiện**: | **Mã số sinh viên** |
|---|---|
| Trần Thanh Nguyên | 19520192 |
| Nguyễn Chí Cường | 19521299 |


**TP.HCM – 12/2021**

**NHẬN XÉT CỦA GIẢNG VIÊN**

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

# MỤC LỤC

## Contents

**TÓM TẮT**

Các model trước đây cũng dùng đến CNN đều làm được những điều rất tốt như phân biệt ngay lập tức được hàng trăm, hàng ngàn bước ảnh chó và mèo hoặc thẩm chí nhiều thứ khác. Nhưng điểm chung ở những bài toán trên là phải có gán nhãn (label), việc gán nhãn này làm tốn rất nhiều thời gian và công sức. Thế nên, Generative Adversarial Networks đã được sinh ra với kỳ vọng để tạo ra những model có độ chính xác cao mà không cần nhiều đến sự hoạt động của con người trong việc huấn luyện. (Unsupervised Learning). Trong bài báo cáo này sẽ tìm hiểu rõ về GANs là cái gì, GANs hoạt động như thế nào, tại sao lại dùng đến GANs.

# 1. CHƯƠNG 1: GIỚI THIỆU TỔNG QUAN

## 1.1. Giới thiệu quá trình hình thành

- Ý tưởng đặt hai thuật toán chống lại nhau bắt nguồn từ Arthur Samuel, một nhà nghiên cứu nổi tiếng trong lĩnh vực khoa học máy tính, người được ghi nhận là người đã phổ biến thuật ngữ "machine learning". Khi ở IBM, ông đã nghĩ ra một trò chơi cờ caro - the Samuel Checkers-playing Program - là một trong những trò chơi đầu tiên tự học thành công, một phần bằng cách ước tính cơ hội chiến thắng của mỗi bên ở một vị trí nhất định.
- Nhưng nếu Samuel là ông của GAN, Ian Goodfellow, cựu nhà khoa học nghiên cứu Google Brain và giám đốc máy học tại Nhóm các dự án đặc biệt của Apple, có thể là cha của nó. Trong một bài báo nghiên cứu năm 2014 có tiêu đề đơn giản là " generative adversarial net ", Goodfellow và các đồng nghiệp mô tả việc triển khai hoạt động đầu tiên của một mô hình chung dựa trên các mạng đối địch.
- Goodfellow thường tuyên bố rằng anh ta lấy cảm hứng từ "noise-contrastive estimation", một cách học phân phối dữ liệu bằng cách so sánh nó với phân phối nhiễu xác định (tức là, một hàm toán học đại diện cho dữ liệu bị hỏng hoặc bị bóp méo). Noise-contrastive estimation sử dụng các hàm tổn thất giống như GAN - nói cách khác, cùng một phép đo hiệu suất liên quan đến khả năng dự đoán kết quả mong đợi của mô hình.

## 1.2. Mục tiêu đề tài

- Tìm hiểu về generative adversarial networks(Gans)
- Thực hành, thử nghiệm demo các vấn đề gặp phải khi ứng dụng

### 1.3. Thông tin nhóm

- Danh sách thành viên:

**Bảng 1: Danh sách các thành viên trong nhóm**

| STT | MSSV | Họ và tên |
|---|---|---|
| 1 | 19520192 | Trần Thanh Nguyên |
| 2 | 19521299 | Nguyễn Chí Cường |

### 1.4. Bảng phân công công việc

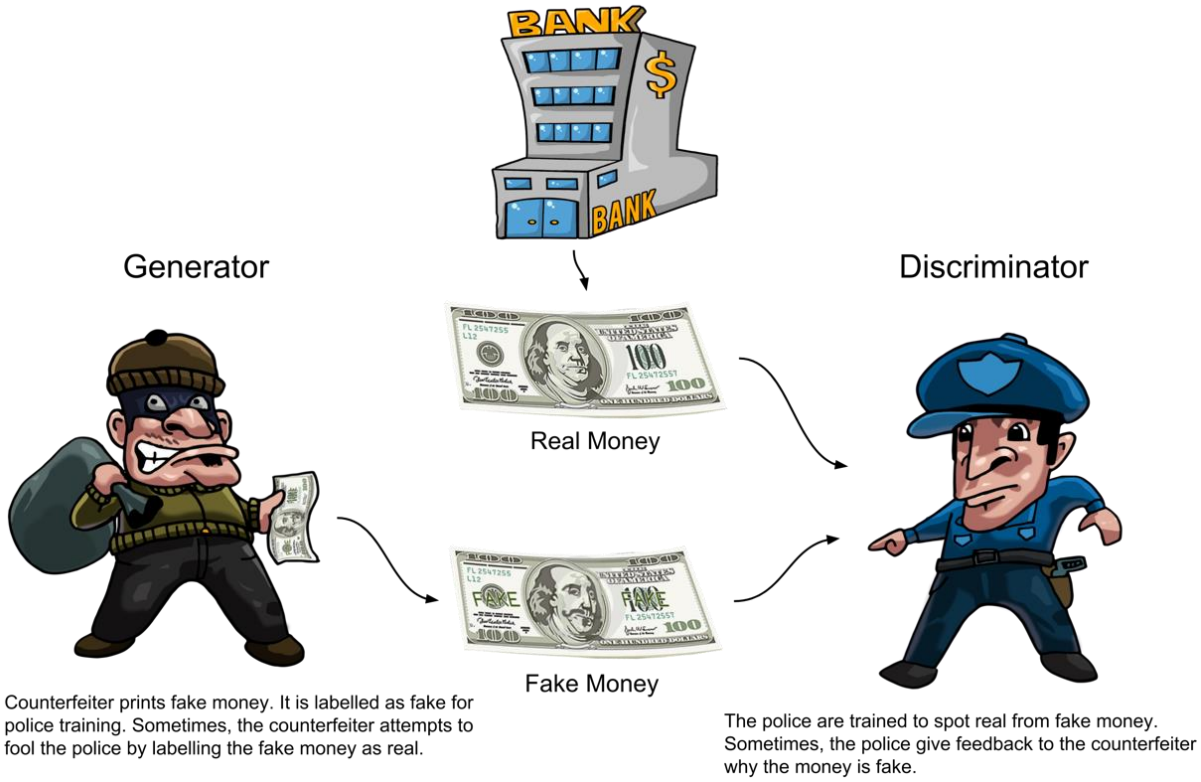
**Bảng 2 Bảng phân công công việc**

| Công việc | Người thực hiện |
|---|---|
| Tìm hiểu khái quát đề tài | Nguyên |
| Làm file báo cáo | Cả nhóm |
| Lên lịch thực hiện | Cường |
| Tìm hiểu ứng dụng của Gans | Cường |
| Phân tích tìm hiểu lý thuyết toán | Nguyên |
| Chỉnh sửa báo cáo | Cả nhóm |

## 2. CHƯƠNG 2: LÝ THUYẾT

### 2.1. Generative Adversarial Networks (GANs) là gì?

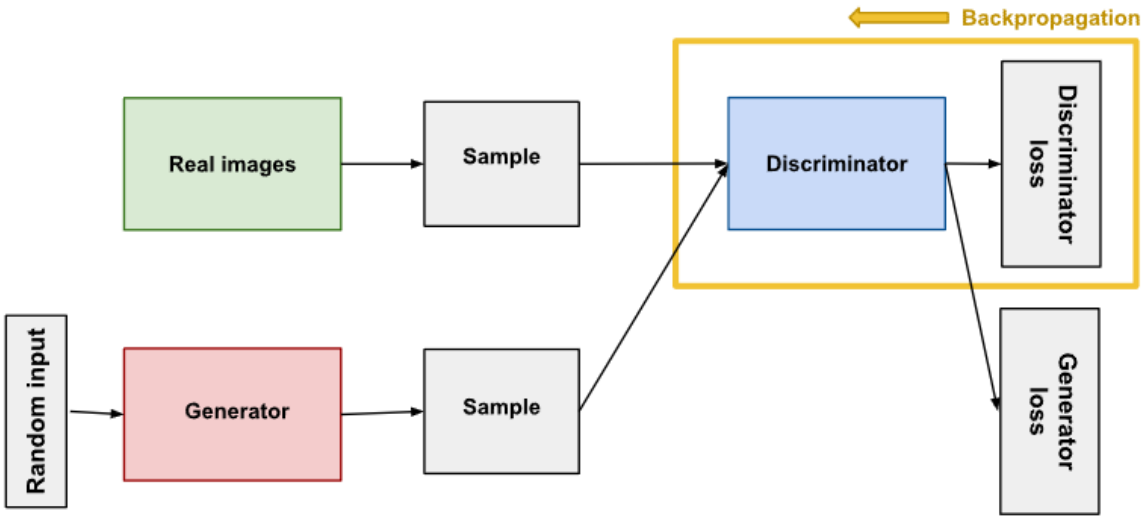
Generative Adversarial Networks Hướng tới việc sinh ra dữ liệu mới sau khi học như là tạo ra một khuôn mặt người, chữ viết, bản nhạc, hoặc những thứ khác như thế. Thế Generative Adversarial Networks là cái gì? Generative Adversarial Networks goi tắt là (GANs) - Mạng Chống đối Tạo sinh - hình thành trên ý tưởng về sự cạnh tranh của hai mạng neural network: - Discriminative network (mạng phân biệt) - Generative network (mạng sinh) Quan hệ giữa Discriminative network và Generative network?




Discriminative được ví như là một cảnh sát, còn Generative như là một tội phạm làm tiền giả. Tội phạm làm tiền giả thì sẽ học cách làm tiền giả sao cho cảnh sát không thể phân biệt được đó là tiền giả. Cảnh sát thì sẽ học cách phân biệt được đâu là tiền giả còn đâu là tiền thật, và báo cho tội phạm làm tiền giả biết là tiền giả của nó còn non lắm cần phải cải thiện thêm. Làm việc như trên như thế sẽ làm cho tội phạm làm tiền giả giống tiền thật hơn và cảnh sát cũng sẽ phân biệt được tiền giả tốt hơn.

## 2.2. Discriminative network

Discriminative đơn giản là một mô hình phân lớp (classifier) giống với bài toán Logistic Regression. Mô hình này sẽ học phân biệt đâu là data thực (real data) và đâu là data được tạo bởi Generator (fake data).




Discriminator được huấn luyện từ dữ liệu đến từ hai nguồn:

- Real data: Dữ liệu thực tế được lập trình viên đưa vào. Discriminator sẽ xem đây là nhãn positive trong suốt quá trình huấn luyên (training).

- Fake data: Dữ liệu được tạo bởi Generator. Discriminator sẽ xem đây là nhãn negative trong suốt quá trình huấn luyện (training).

## 2.3. Generative network

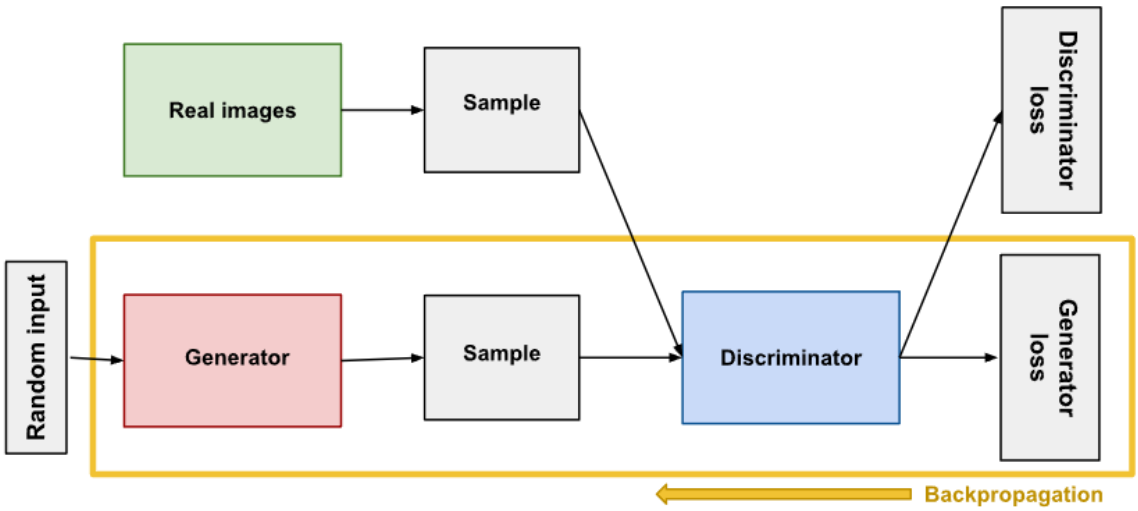
Generator sẽ tạo dữ liệu từ phản hồi của Discriminator. Và từ đó học làm sao để cho Discriminator không phân biệt được dữ liệu của nó tạo ra là giả. Khi huấn luyện Generator yêu cầu tích hợp chặt chẽ với Discriminator. Phần đào tạo Generator:

- Random input (random noise).

- Generator từ random input tạo ra dữ liệu.

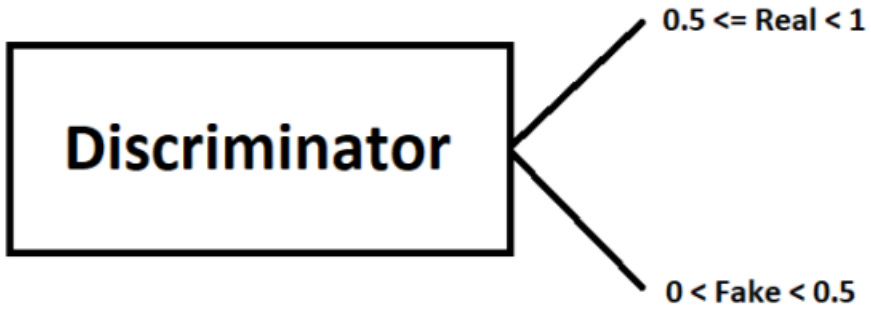
- Discriminator sẽ phân loại dữ liệu được tạo.

- Generator Loss




### 2.4. Loss Function

Cũng như thế GANs có hai loss function: một là cho Generator huấn luyện còn một là cho Discriminator huấn luyện. Làm rõ về hai Loss Function. Ta ký hiệu: z là đầu vào noise random, x là dữ liệu từ datasets. Ký hiệu mạng Generator là G, Discriminator là D và từ đó ta thì G(z) là dữ liệu sinh ra từ mạng Generator và D(x) là giá trị dự đoán của mạng Discriminator từ dữ liệu từ bộ datasets. Vậy D(G(z)) là giá trị dự đoán của mạng Discriminator về dữ liệu sinh ra từ Generator. Discriminator phân biệt thật giả giống như bài toán phân lớp Logistic Regression, vì thế nên Loss Function nó cũng dùng binary cross-entropy loss của bài sigmoid (binary cross-entropy dùng cho các bài toán binary classification).




Thế nên giá trị output của Discriminator nằm trong khoảng (0, 1). Discriminator sẽ train với dữ liệu trong datasets tiến tới gần bằng 1 nhất, với dữ liệu từ Generator thì sẽ gần bằng 0 nhất. Tức là ta muốn làm Loss Function của D(x) cao nhất có thể (maximize) và D(G(z)) thấp nhất có thể (minimize). Ta có minimize của

D(G(z)) tương đương với maximize (1 – D(G(z))), Loss function của Discriminator sẽ là:

$$\max_D D = E_{x\sim P_{data}}[\log D(x)] + E_{z\sim P_{z(z)}}[\log(1 - D(G(z)))]$$

Trong đó $E_{x\sim P_{data}}[\log D(x)]$ là thể hiện được khả năng nhận diện được data thật tốt hơn. Còn $E_{x\sim P_{z(z)}}[log\ 1 - D(G(z)))]$ cho thấy khả năng nhận diện data từ Generator tốt hơn.

Còn về Generator thì chỉ có một E kì vọng để thể hiện khả năng lừa được Discrimmator tốt hơn.

$$\min_G G = E_{z\sim P_{z(z)}}[\log(1 - D(G(z)))]$$

Và ta gộp lại như sau:

$$\min_G \max_D D, g = E_{x\sim P_{data}}[\log D(x)] + E_{z\sim P_{z(z)}}[\log(1 - D(G(z)))]$$

## 2.5. Một vài kiến trúc GAN

### 2.5.1. CycleGAN

CycleGAN là một kiến trúc GAN rất phổ biến, chủ yếu được sử dụng để học chuyển đổi giữa các hình ảnh có kiểu dáng khác nhau.

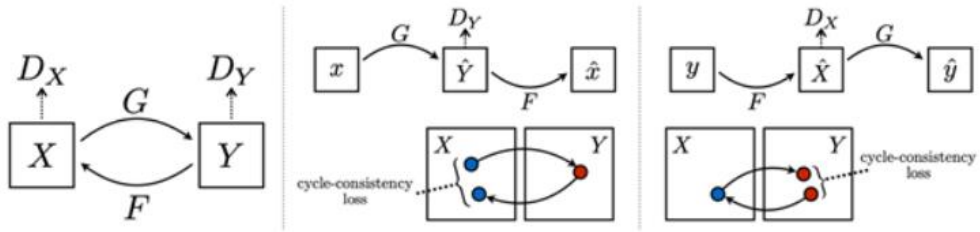
Ví dụ: chuyển đổi hình ảnh mùa đông và mùa hè, chuyển đổi hình ảnh giữa báo đen và báo đốm, chuyển đổi hình ảnh ngựa thường và ngựa vằn,…

FaceApp là một ứng dụng khá nổi tiếng xử dụng CycleGAN, nơi khuôn mặt chúng ta sẽ được biến đổi theo nhóm tuổi, độ tuổi khác nhau.


CycleGAN là một mở rộng của kiến trúc GAN cổ điển bao gồm 2 Generator và 2 Discriminator. Generator đầu tiên gọi là G, nhận đầu vào là ảnh từ domain X và convert nó sang domain Y . Generator còn lại gọi là Y, có nhiệm vụ convert ảnh từ domain Y sang X. Mỗi mạng Generator có 1 Discriminator tương ứng với nó

DYD_YDY: phân biệt ảnh lấy từ domain Y và ảnh được translate G(x).

DXD_XDX: phân biệt ảnh lấy từ domain X và ảnh được translate F(y).



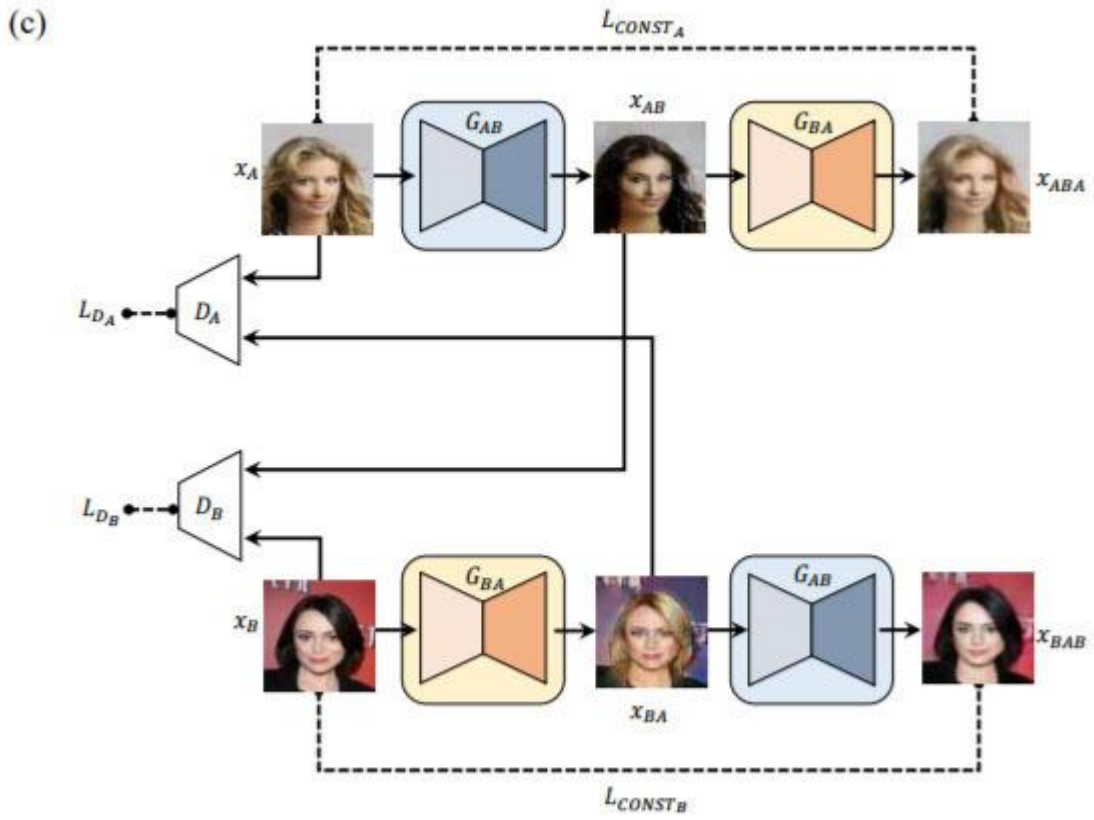

Trong quá trình huấn luyện, generator G cố gắng tối thiểu hóa hàm adversarial loss bằng cách translate ra ảnh G(x) (với x là ảnh lấy từ domain X) sao cho giống với ảnh từ domain Y nhất, ngược lại Discriminator DYD_YDY cố gắng cực đại hàm adversarial loss bằng cách phân biệt ảnh G(x) và ảnh thật y từ domain

### 2.5.2. DiscoGAN

DiscoGAN mới trở nên phổ biến trong thời gian gần đây nhờ vào khả năng học các mối liên hệ giữa các tên miền với dữ liệu không giám sát.

Nếu con người chúng ta nhìn vào hình trên dễ dàng phát hiện mối liên hệ giữa 2 miền là về bản chất của màu sắc. Tuy nhiên để máy có thể tìm ra mối liên hệ giữa 2 miền chưa được ghép nối là vô cùng khó khăn.

DiscoGAN và CycleGAN khá giống nhau về mặt thiết kế mạng. Một bên xử dụng phép chuyển đổi từ miền X sang miền Y, còn một bên thì học theo một ánh xạ ngược lại. Cả hai đều sử dụng reconstruction loss như một phép đo mức độ tái tạo sau hai lần chuyển đổi giữa các miền.




### 2.5.3. IsGAN

Trong thời gian gần đây, Generative Adversarial Networks đã chứng minh hiệu suất ấn tượng cho các tác vụ không có giám sát.

Trong GAN thông thường, Discrimination sử dụng hàm mất cross-entropy, đôi khi dẫn đến các vấn đề về gradient biến mất. Thay vào đó, lsGAN đề xuất sử dụng hàm mất bình phương nhỏ nhất cho dấu phân biệt. Công thức này cung cấp chất lượng hình ảnh cao hơn do GAN tạo ra.

Sau đây là công thức tối ưu hóa min-max trong đó Discrimination là một bộ phân loại nhị phân và đang sử dụng mất mát cross-entropy sigmoid trong quá trình tối ưu hóa.

$$\min_G \max_D V_{GAN}(D,G) = E_{x\sim P_{data}}[\log D(x)] + E_{z\sim P_{z(z)}}[\log(1 - D(G(z)))]$$

Như đã đề cập trước đó, công thức này thường gây ra các vấn đề về vanishing gradient đối với data point nằm ở phía bên phải của ranh giới quyết định nhưng ở xa vùng dày đặc. Công thức Least Square giải quyết vấn đề này và cung cấp khả năng học tập ổn định hơn về mô hình và tạo ra hình ảnh tốt hơn.

Sau đây là công thức tối ưu hóa được định dạng lại cho lsGAN trong đó:

- a là nhãn cho fake data
- b là nhãn của real data
- c biểu thị giá trị mà Generator muốn Discriminator tin cho là một fake data.

$$\min_D V_{LSGAN}(D) = \frac{1}{2} E_{x \sim P_{data}}(x)[(D(x) - b)^2] + \frac{1}{2} E_{z \sim P_{z(z)}}\left[\left(D\left(G(z)\right) - a\right)^2\right]$$

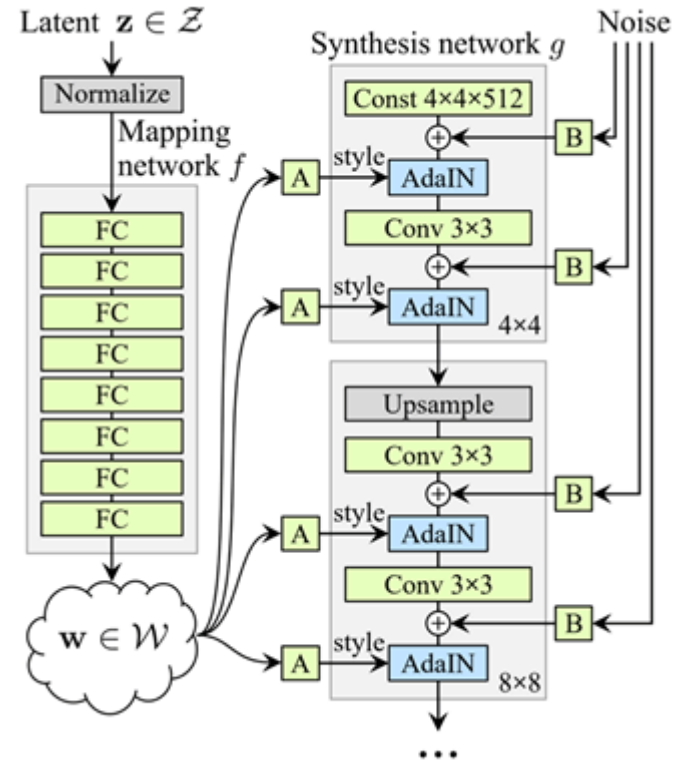
$$\min_G V_{LSGAN}(G) = \frac{1}{2} E_{z \sim P_Z}(z)[(D(G(z)) - c)^2]$$

Công thức trên tối ưu hóa 2 hàm loss. Một dùng để giảm thiểu với Discriminator, một là để giảm, tối ưu hóa cho Generator.

### 2.5.4. StyleGan

StyleGAN là một phần kiến trúc mở rộng đang ngày càng phát triển của GAN, là một cách tiếp cận để đào tạo các mô hình máy phát điện có khả năng tổng hợp các hình ảnh chất lượng cao rất lớn thông qua việc mở rộng gia tăng cả discriminator và generator từ hình ảnh nhỏ đến hình ảnh lớn trong quá trình đào tạo.

Generator StyleGAN không còn lấy một điểm từ "latent space" làm đầu vào; thay vào đó, có hai nguồn ngẫu nhiên mới được sử dụng để tạo ra hình ảnh tổng hợp: mạng ánh xạ độc lập và các lớp nhiễu.



Hình trên mô tả kiến trúc cơ bản của styleGAN, vactor “latent space” z được chuyển qua phép biến đổi ánh xạ bao gồm 8 lớp được kết nối đầy đủ trong khi mạng tổng hợp bao gồm 18 lớp, trong đó mỗi lớp tạo ra hình ảnh từ 4 x 4 đến 1024 x 1024. Lớp đầu ra xuất ra hình ảnh RGB thông qua một lớp tích chập riêng. Kiến trúc này có 26,2 triệu tham số và vì số lượng tham số có thể huấn luyện rất cao, mô hình này đòi hỏi một số lượng lớn các hình ảnh huấn luyện để xây dựng một mô hình thành công.

Có 6 điểm quan trọng trong kiến trúc của styleGan:

- Baseline Progressive GAN:

Progressive GAN là một phương pháp đào tạo trong đó kiến trúc GAN được phát triển từ từ theo kiểu ổn định trong quá trình đào tạo. Lúc đầu, một mô hình cơ sở với generator và discriminator được đào tạo với các hình ảnh nhỏ và khi quá trình đào tạo ổn định các khối lớp mới được thêm vào có khả năng thu được hình ảnh lớn hơn và mô hình được đào tạo lại. Phương pháp này được lặp lại cho đến khi đạt được kích thước hình ảnh mong muốn.

- Tuning:

Một số phương pháp được sử dụng để cải thiện kết quả, và một trong số đó là bilinear sampling(lấy mẫu song tuyến). Ở đây, sau mỗi lớp lấy mẫu lên và trước mỗi lớp lấy mẫu xuống, các activations được lọc bằng bộ lọc nhị thức bậc hai.

Kỹ thuật này thay thế cách lấy mẫu thông thường được sử dụng trong Progressive GAN .

- Add mapping and styles:

Tiếp theo, thay vì cấp “latent space” trực tiếp cho mạng tổng hợp, nó được chuyển qua 8 lớp được kết nối đầy đủ và đầu ra được gọi là Style vector. Style vector sau đó được kết hợp vào từng khối của mạng tổng hợp, sau các lớp convolution. Điều này được thực hiện thông qua một phương pháp được gọi là chuẩn hóa phiên bản thích ứng hoặc AdaIN.

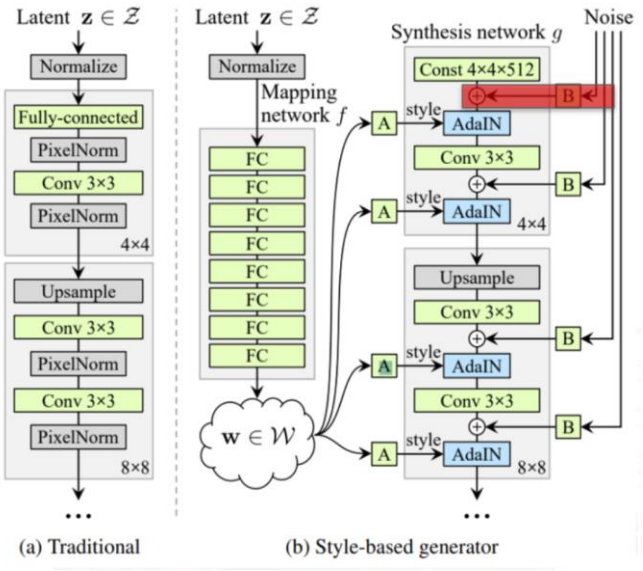
$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

- Remove traditional input:

Thay vì sử dụng "latent space" thông thường làm đầu vào, một giá trị cố định được sử dụng làm đầu vào cho mạng tổng hợp. Đầu vào này được học trong quá trình đào tạo.

- Add noise inputs:

Tiếp theo, các hình ảnh đơn kênh bao gồm nhiễu Gaussian không tương quan, hay được gọi đơn giản là nhiễu trên giấy, được đưa vào từng lớp của mạng tổng hợp. Hình ảnh nhiễu được truyền tới tất cả các feature map bằng cách sử dụng các hệ số tỷ lệ đã học cho mỗi đối tượng và sau đó được thêm vào đầu ra của convolution tương ứng.

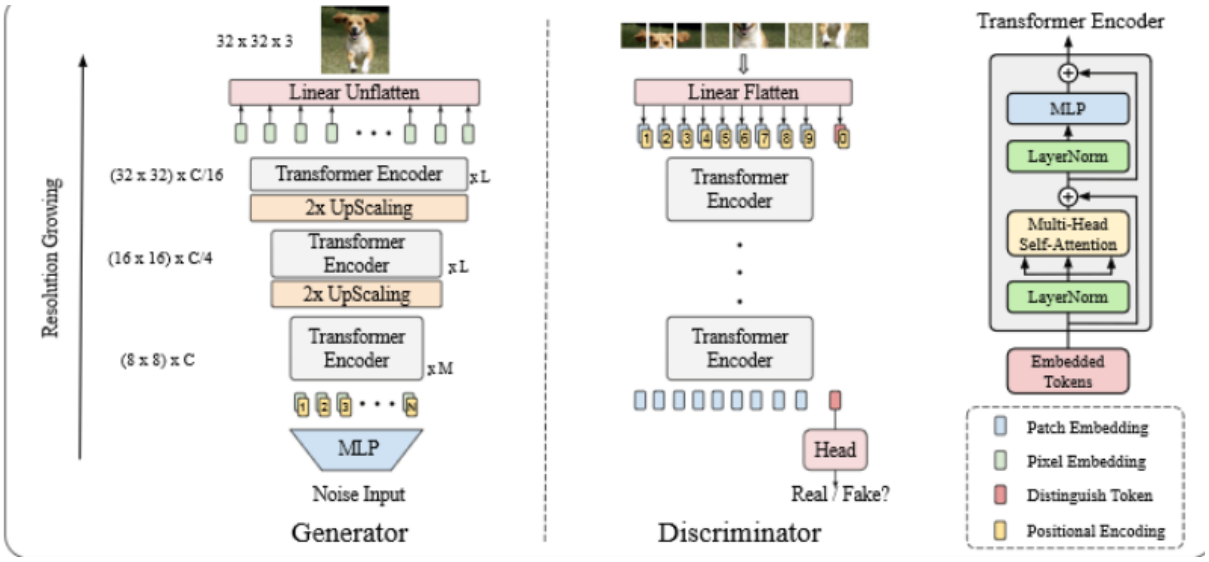



- Addition Mixing regularization:

Cuối cùng, kỹ thuật Mixing regularization ngăn không cho mạng giả định rằng các kiểu liền kề có tương quan với nhau. Nó liên quan đến việc đào tạo mạng với hai nút "latent space" ngẫu nhiên cho một tỷ lệ hình ảnh được tạo nhất định, thay vì một "latent space". Trong quá trình tạo ra những hình ảnh này, các "latent nodes" được chuyển đổi tại một điểm được chọn ngẫu nhiên trong mạng tổng hợp.

### 2.5.5. TransGAN

Cho đến nay, transformers hoặc attentions đã được sử dụng trong Generative Adversarial Networks (GAN), nhưng chúng luôn có một thành phần tích chập trong kiến trúc của chúng.




Kiến trúc của TransGAN tương đối đơn giản. Một TransGAN bao gồm một bộ tạo G và một bộ phân biệt D. Các discriminator và generator trong một GAN thường được tạo thành từ các convolution. TransGAN thay thế các chập này bằng stranformer.

- The generator:

Mô hình bộ tạo tạo ra hình ảnh 32 x 32 chiều với ba kênh màu (đỏ, lục và lam).

Ở giai đoạn đầu tiên của mô hình stranformoer, các đầu vào nhiễu ngẫu nhiên được đưa vào perceptron nhiều lớp (MLP). Quá trình này tạo ra trình tự ban đầu. Các chuỗi ban đầu này có kích thước 8 x 8 x C, trong đó C là số kênh. Bộ mã hóa vị trí cũng được sử dụng trong bộ tạo.

Chúng đưa thông tin về vị trí trình tự tương đối hoặc tuyệt đối của các mảng hình ảnh. Ví dụ, hình ảnh nào đến trước và hình ảnh nào đến sau cùng. Các bộ mã hóa vị trí này có cùng kích thước với các trình tự đầu vào để chúng có thể được thêm vào với nhau. Các đầu ra này sau đó được đưa vào các lớp M của mô-đun Bộ mã hóa biến áp.

Bước tiếp theo liên quan đến việc nâng cấp độ phân giải ở các giai đoạn khác nhau. Quá trình này liên quan đến việc tăng độ phân giải của hình ảnh trong khi giảm số lượng kênh trong hình ảnh.

Quá trình này giúp đạt được mật độ pixel cao hơn (tăng độ phân giải) trong hình ảnh. Nó cũng tiết kiệm không gian bộ nhớ bằng cách giảm số lượng kênh và ngăn chặn các vụ nổ máy tính. Quá trình nâng cấp mang lại đầu ra kích thước (16 x 16) x C / 4 và cuối cùng đạt được kích thước mục tiêu là (32 x 32) x C / 16.

- The discriminator:

Mô hình phân biệt nhận đầu vào của nó từ generator model. Nó chia hình ảnh 32 x 32 x 3 này thành các bản vá hình ảnh được đưa vào bộ mã transformer. Transformer kiểm tra mỗi bản vá hình ảnh dưới dạng nhúng mã thông báo, như trường hợp trong NLP.

Vì một transformer thiếu convolution, điều quan trọng là phải thêm mã hóa vị trí vào các bản vá hình ảnh. Nó phải được thêm vào như một transformer và không có ý tưởng về vị trí tuần tự của các bản vá hình ảnh bị chia nhỏ.

Ở giai đoạn cuối của mô hình Discriminator , bạn có mã thông báo [cls] để phân loại đầu ra là thật hay giả. Mã thông báo CLS được sử dụng cho các nhiệm vụ phân loại trong NLP.

# 3. CHƯƠNG 3: THỰC TIỄN

## 3.1. Anime Face.

**Thực hiện với bài toán tạo ra mặt Anime. Và sử dụng model DCGAN**

Kết quả trong quá trình Train với:

Learning rate: 0.0002

Beta hyperparameter for Adam: 0.5

| | | | | |
|---|---|---|---|---|
| [0/5][0/497] | Loss_D: 1.5753 | Loss_G: 25.0250 | D(x): 0.4624 | D(G(z)): 0.4404 / 0.0000 |
| [0/5][50/497] | Loss_D: 1.4232 | Loss_G: 4.7655 | D(x): 0.6376 | D(G(z)): 0.3454 / 0.0244 |

| | | | | |
|---|---|---|---|---|
| [0/5][100/497] | Loss_D: 0.8033 | Loss_G: 2.0211 | D(x): 0.6158 | D(G(z)): 0.1938 / 0.1499 |
| [0/5][150/497] | Loss_D: 1.0326 | Loss_G: 6.5730 | D(x): 0.8167 | D(G(z)): 0.4857 / 0.0027 |
| [0/5][200/497] | Loss_D: 1.2800 | Loss_G: 3.1691 | D(x): 0.6626 | D(G(z)): 0.3427 / 0.0766 |
| [0/5][250/497] | Loss_D: 1.1344 | Loss_G: 2.8768 | D(x): 0.6358 | D(G(z)): 0.3692 / 0.0820 |
| [0/5][300/497] | Loss_D: 0.6702 | Loss_G: 2.5953 | D(x): 0.6604 | D(G(z)): 0.1574 / 0.0819 |
| [0/5][350/497] | Loss_D: 0.8969 | Loss_G: 3.9533 | D(x): 0.7591 | D(G(z)): 0.4140 / 0.0249 |
| [0/5][400/497] | Loss_D: 1.9696 | Loss_G: 4.2511 | D(x): 0.5418 | D(G(z)): 0.6312 / 0.0229 |
| [0/5][450/497] | Loss_D: 1.2910 | Loss_G: 1.7278 | D(x): 0.3969 | D(G(z)): 0.1032 / 0.2098 |
| [1/5][0/497] | Loss_D: 1.0281 | Loss_G: 0.5791 | D(x): 0.5682 | D(G(z)): 0.2716 / 0.5940 |
| [1/5][50/497] | Loss_D: 1.4846 | Loss_G: 2.1247 | D(x): 0.3621 | D(G(z)): 0.1656 / 0.1647 |
| [1/5][100/497] | Loss_D: 1.6035 | Loss_G: 2.8369 | D(x): 0.3804 | D(G(z)): 0.3170 / 0.0717 |
| [1/5][150/497] | Loss_D: 1.0855 | Loss_G: 4.1846 | D(x): 0.7155 | D(G(z)): 0.4798 / 0.0198 |
| [1/5][200/497] | Loss_D: 1.3951 | Loss_G: 3.5897 | D(x): 0.6049 | D(G(z)): 0.4386 / 0.0441 |
| [1/5][250/497] | Loss_D: 1.3289 | Loss_G: 4.6923 | D(x): 0.8964 | D(G(z)): 0.6777 / 0.0112 |
| [1/5][300/497] | Loss_D: 2.3190 | Loss_G: 3.4859 | D(x): 0.1605 | D(G(z)): 0.0120 / 0.0416 |
| [1/5][350/497] | Loss_D: 1.3040 | Loss_G: 4.1718 | D(x): 0.9192 | D(G(z)): 0.6421 / 0.0239 |

| | | | | |
|---|---|---|---|---|
| [1/5][400/497] | Loss_D: 1.4531 | Loss_G: 3.6899 | D(x): 0.7629 | D(G(z)): 0.6384 / 0.0340 |
| [1/5][450/497] | Loss_D: 1.1356 | Loss_G: 2.8923 | D(x): 0.4075 | D(G(z)): 0.0474 / 0.0918 |
| [2/5][0/497] | Loss_D: 1.0066 | Loss_G: 2.5750 | D(x): 0.5354 | D(G(z)): 0.2183 / 0.0921 |
| [2/5][50/497] | Loss_D: 1.2051 | Loss_G: 2.1214 | D(x): 0.5502 | D(G(z)): 0.3225 / 0.1493 |
| [2/5][100/497] | Loss_D: 0.8443 | Loss_G: 3.1657 | D(x): 0.5158 | D(G(z)): 0.0597 / 0.0598 |
| [2/5][150/497] | Loss_D: 0.9074 | Loss_G: 3.8094 | D(x): 0.7572 | D(G(z)): 0.4126 / 0.0298 |
| [2/5][200/497] | Loss_D: 0.8091 | Loss_G: 3.1398 | D(x): 0.7849 | D(G(z)): 0.3950 / 0.0561 |
| [2/5][250/497] | Loss_D: 0.6809 | Loss_G: 3.3801 | D(x): 0.8723 | D(G(z)): 0.3768 / 0.0539 |
| [2/5][300/497] | Loss_D: 0.8239 | Loss_G: 4.5494 | D(x): 0.8013 | D(G(z)): 0.3915 / 0.0148 |
| [2/5][350/497] | Loss_D: 2.5738 | Loss_G: 2.6979 | D(x): 0.1317 | D(G(z)): 0.0287 / 0.1296 |
| [2/5][400/497] | Loss_D: 0.9248 | Loss_G: 3.4160 | D(x): 0.6866 | D(G(z)): 0.3603 / 0.0389 |
| [2/5][450/497] | Loss_D: 1.4760 | Loss_G: 2.4974 | D(x): 0.3288 | D(G(z)): 0.0335 / 0.1092 |
| [3/5][0/497] | Loss_D: 1.1421 | Loss_G: 6.1331 | D(x): 0.7192 | D(G(z)): 0.4745 / 0.0037 |
| [3/5][50/497] | Loss_D: 1.8564 | Loss_G: 5.2781 | D(x): 0.7606 | D(G(z)): 0.7357 / 0.0083 |
| [3/5][100/497] | Loss_D: 0.6968 | Loss_G: 3.2886 | D(x): 0.8134 | D(G(z)): 0.3531 / 0.0432 |
| [3/5][150/497] | Loss_D: 1.0601 | Loss_G: 2.2329 | D(x): 0.4418 | D(G(z)): 0.0636 / 0.1334 |

| | | | | |
|---|---|---|---|---|
| [3/5][200/497] | Loss_D: 0.5356 | Loss_G: 3.6102 | D(x): 0.7532 | D(G(z)): 0.1677 / 0.0401 |
| [3/5][250/497] | Loss_D: 1.1558 | Loss_G: 6.0974 | D(x): 0.8478 | D(G(z)): 0.5487 / 0.0043 |
| [3/5][300/497] | Loss_D: 1.1682 | Loss_G: 1.5157 | D(x): 0.3916 | D(G(z)): 0.0389 / 0.2576 |
| [3/5][350/497] | Loss_D: 1.8165 | Loss_G: 5.5344 | D(x): 0.8856 | D(G(z)): 0.7342 / 0.0070 |
| [3/5][400/497] | Loss_D: 0.8756 | Loss_G: 6.6434 | D(x): 0.8971 | D(G(z)): 0.4890 / 0.0023 |
| [3/5][450/497] | Loss_D: 1.0906 | Loss_G: 6.4293 | D(x): 0.9264 | D(G(z)): 0.5794 / 0.0027 |
| [4/5][0/497] | Loss_D: 0.9871 | Loss_G: 4.8448 | D(x): 0.7151 | D(G(z)): 0.4019 / 0.0110 |
| [4/5][50/497] | Loss_D: 0.8543 | Loss_G: 4.7641 | D(x): 0.8866 | D(G(z)): 0.4666 / 0.0121 |
| [4/5][100/497] | Loss_D: 0.9830 | Loss_G: 6.5441 | D(x): 0.8136 | D(G(z)): 0.4708 / 0.0022 |
| [4/5][150/497] | Loss_D: 0.4622 | Loss_G: 4.1300 | D(x): 0.7975 | D(G(z)): 0.1527 / 0.0225 |
| [4/5][200/497] | Loss_D: 1.0703 | Loss_G: 5.0785 | D(x): 0.8232 | D(G(z)): 0.4892 / 0.0128 |
| [4/5][250/497] | Loss_D: 1.2295 | Loss_G: 5.7448 | D(x): 0.8487 | D(G(z)): 0.5701 / 0.0052 |
| [4/5][300/497] | Loss_D: 0.9906 | Loss_G: 3.0875 | D(x): 0.6907 | D(G(z)): 0.3666 / 0.0696 |
| [4/5][350/497] | Loss_D: 0.6184 | Loss_G: 3.6089 | D(x): 0.8085 | D(G(z)): 0.2859 / 0.0398 |
| [4/5][400/497] | Loss_D: 1.0039 | Loss_G: 2.7811 | D(x): 0.5461 | D(G(z)): 0.1926 / 0.1077 |
| [4/5][450/497] | Loss_D: 0.7287 | Loss_G: 5.8128 | D(x): 0.8438 | D(G(z)): 0.3635 / 0.0044 |

Trên kia có tất cả là 5 vòng lặp trong khi train (5 epochs):

Hình ảnh kết quả trong từng vòng lặp:

- Vòng lặp thứ nhất (0/5): Hình ảnh còn khá mờ chưa rõ rang được. Kết quả khi kết thúc vòng lặp này:

| Loss_D: 1.2910 Loss_G: 1.7278 D(x): 0.3969 D(G(z)): 0.1032 / 0.2098 |
|---|

Con số D(x) con thấp so với mặt bằng chung trong suốt quá trình Train.

- Vòng lặp thứ hai (1/5): kết quả sau khi kết trúc vòng lặp.

Nhiều đã train nhiều hơn nên kết quả theo đánh giá nhóm em là tốt hơn lần epoch 1.

| Loss_D: 1.1356 Loss_G: 2.8923 D(x): 0.4075 D(G(z)): 0.0474 / 0.0918 |
| --- |

- Vòng lặp thứ ba (2/5): Có màu sau hơn epoch trước.

| Loss_D: 1.4760 | Loss_G: 2.4974 | D(x): 0.3288 | D(G(z)): 0.0335 / 0.1092 |
|---|---|---|---|

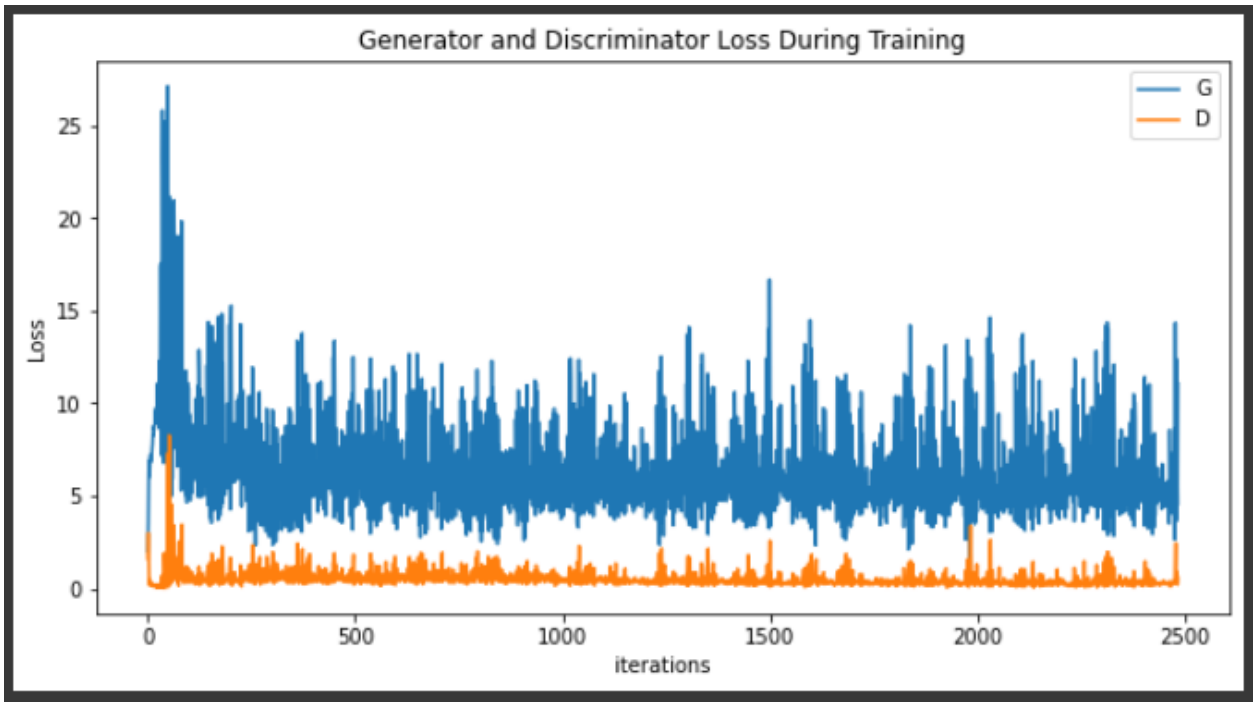
- Vòng lặp thứ 4 (3/5): Không khác lắm so với ở trên.

| Loss_D: 1.0906 Loss_G: 6.4293 D(x): 0.9264 D(G(z)): 0.5794 / 0.0027 |
|---|

- Vòng lặp cuối cùng (4/5): Khả quan hơn nhiều.

| Loss_D: 0.7287 Loss_G: 5.8128 D(x): 0.8438 D(G(z)): 0.3635 / 0.0044 |
|---|

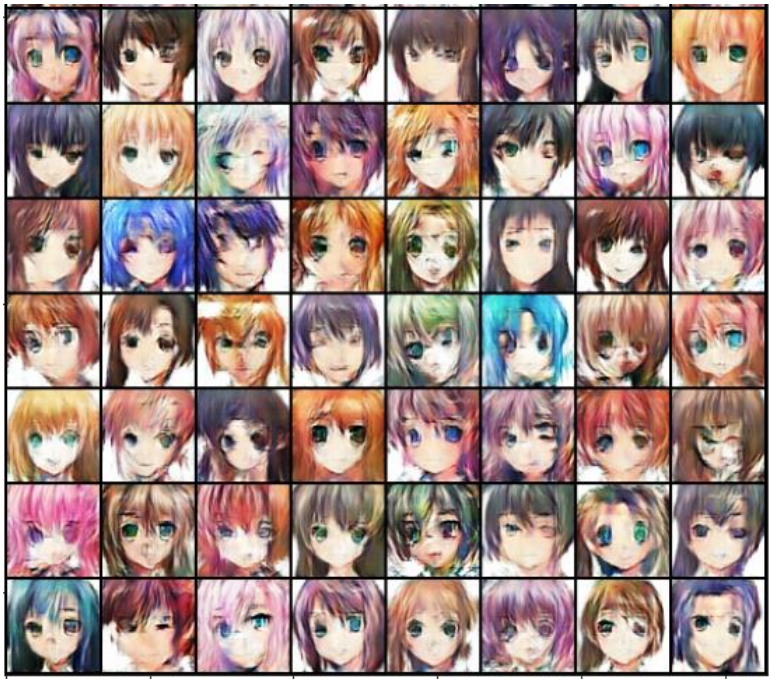
Biểu đồ giá trị Loss của Generator và Discrimnator trong suốt quá trình Train:



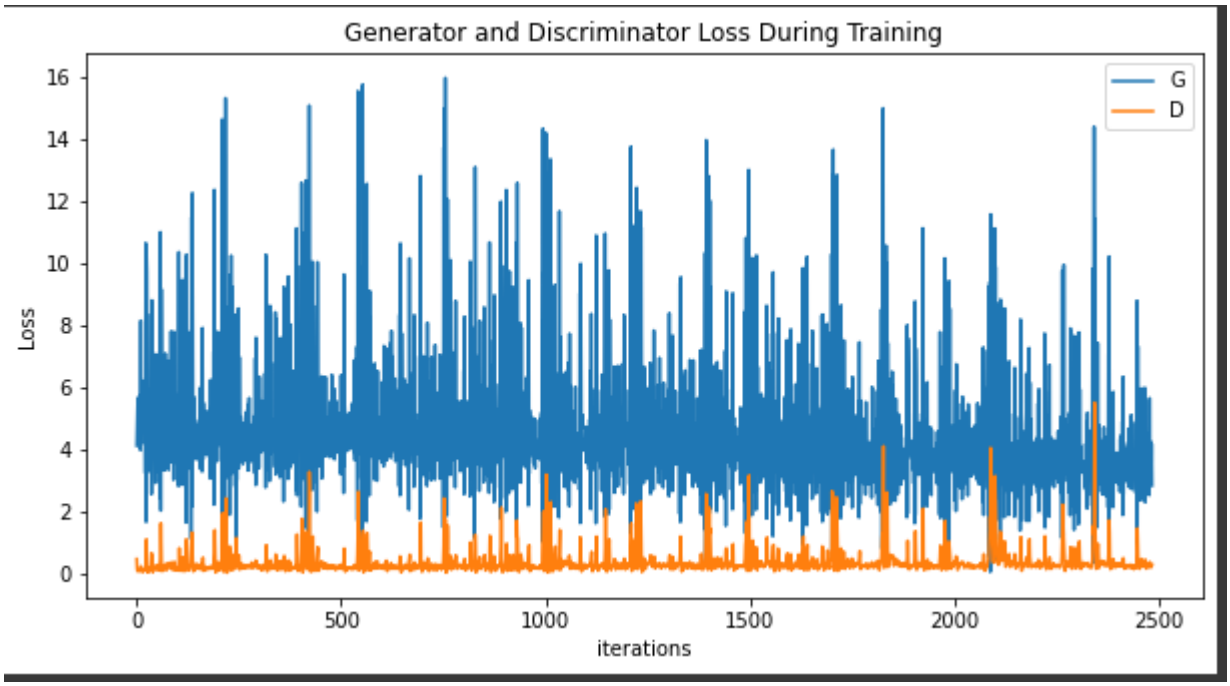

Quyết đinh train thêm 5 epochs tiếp với việc giảm Learning rate xuống gấp đôi và giảm beta còn 0.3. Liệu có thể tối ưu hóa hơn không?

Learning rate: 0.0001

Beta hyperparameter for Adam: 0.3



Biểu đồ Loss trong suốt quá trình Train:




Kết quả khả quan rất nhiều. Có thể là đã train nhiều hơn do chưa train hết ảnh trong bộ dữ diệu.

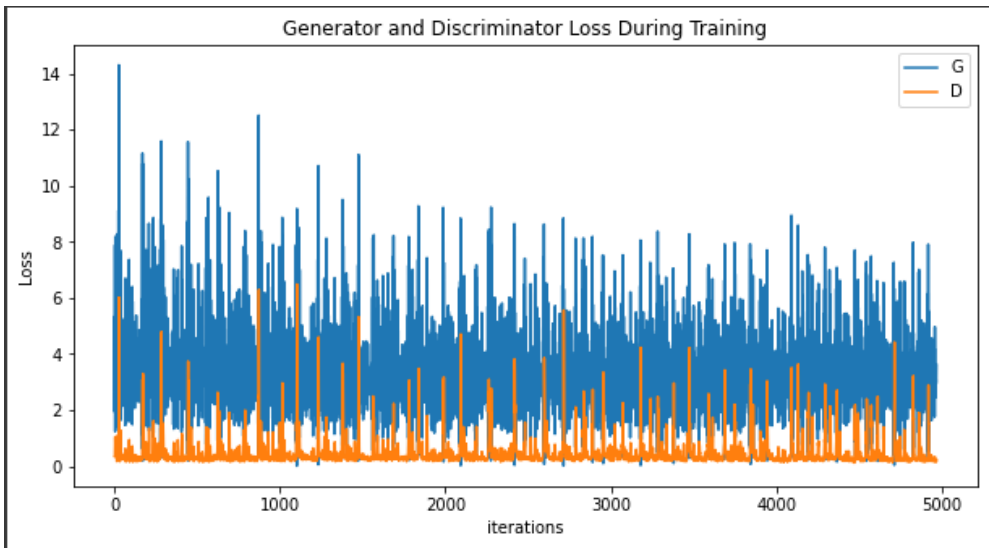
Hình ảnh tuy khả quan hơn lần trước nhưng vẫn ko tốt khi so với ảnh gốc trong bộ dữ liệu.

- Chúng em quyết định train thêm 10 epochs nữa lần này thì sẽ cho learning rate là 0.0001 giảm 10 lần và beta1 đưa về con 0.5 như ban đầu.

Learning rate: 0.0001

Beta hyperparameter for Adam: 0.5

Biểu đồ Loss trong suốt quá trình Train:




Hình ảnh trong có vẻ mượt hơn so với trước:


- Train thêm 10 epochs xem có cải thiện thêm không? Giảm learning rate đi lần để Loss có thể đạt được con số thấp nhất.

Learning rate: 0.00001

Beta hyperparameter for Adam: 0.5


So sánh với lần train trên và ảnh thực, kết quả không khá hơn nhiều.

Kết quả lần cuối cùng train:

Loss_D: 0.1113 Loss_G: 4.0270 D(x): 0.9619 D(G(z)): 0.0659 / 0.0258

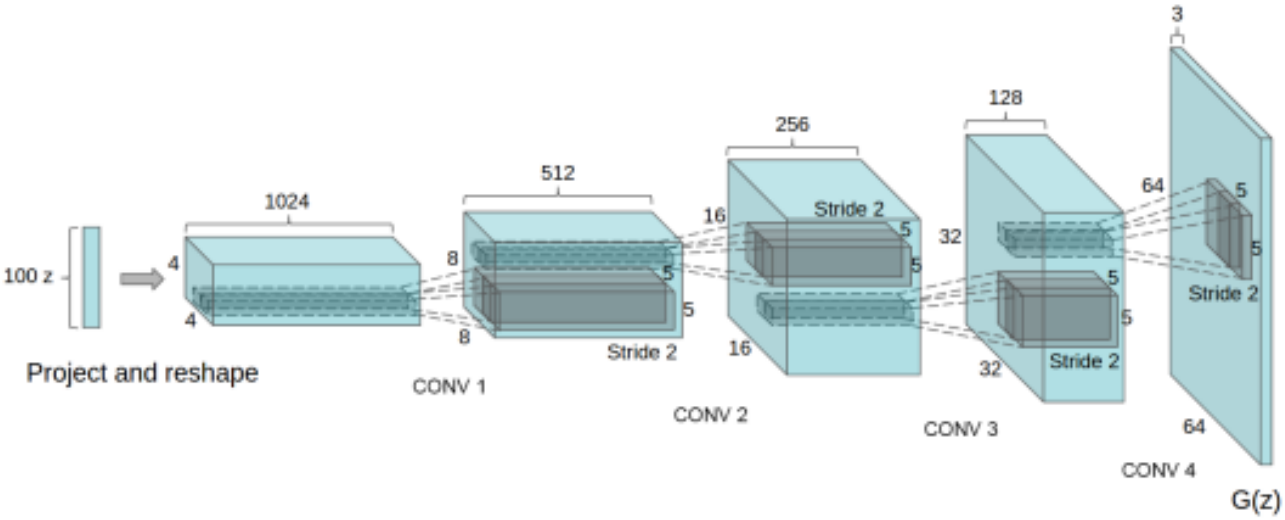
**Dừng lại:** Thấy được kết quả đã được cải thiện nhưng so với ảnh thật thì không được tốt. Sau nhiều lần thay đổi learning rate và train thêm, theo dự đoán tụi em dù có train thêm kết quả có thể sẽ không tốt hơn được nhiều.

**Nguyên nhân:** Vì ảnh Anime có màu sắc quá đa dạng như tóc, mắt, có cả những ảnh có cả kính.

### DCGAN là gì? (Deep Convolutional Generative Adversarial Network)

DCGAN là một phần mở rộng của GAN. Nó sử dụng các Convolution và các Convolution-Transpose trong các model Generator và Discriminator.

- Discriminator được tạo từ Convolution layer, Batch norm layer và LeakyReLU activation xếp theo thứ tự.
    - Input: Là bức ảnh có kích thức (row, col, channels) thường sẽ chọn là (64, 64, 3) để xử lý với ảnh màu hệ RGB.
    - Output: Là một giá trị xác xuất.
- Generator bao gồm: convolutional-transpose layers, batch norm layers, và ReLU activation.

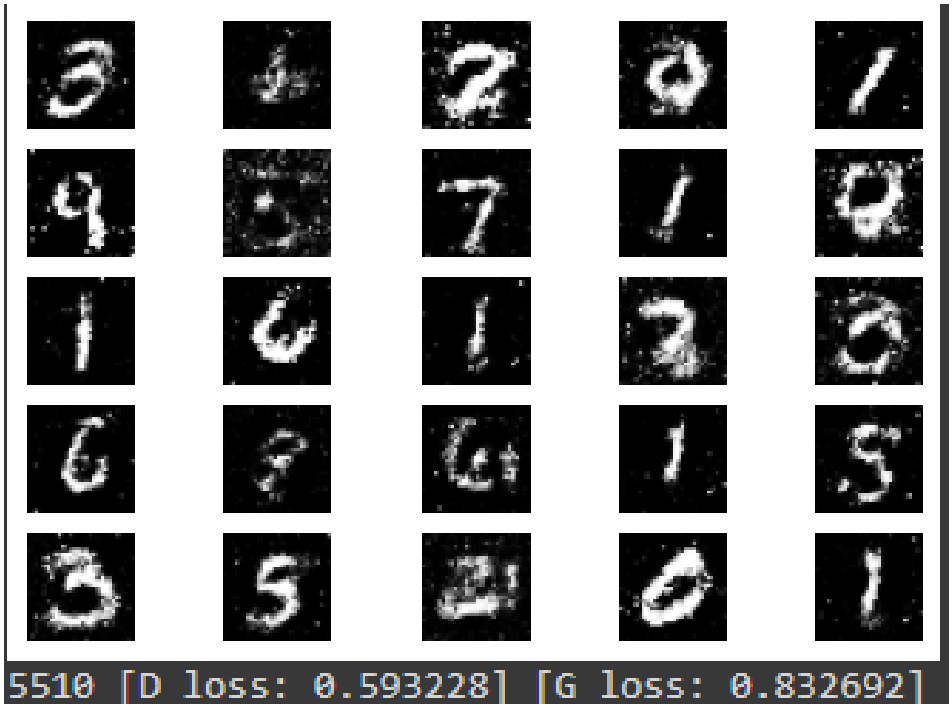



- Input: Là một vector noise z.
- Output: Là một fake image có kích thước (row, col, channels) thường sẽ chọn là (64, 64, 3) để xử lý với ảnh màu hệ RGB.

## 3.2. Hand Written.

**Thực nghiệm, tạo hình ảnh các chữ số viết tay gộp toàn bộ label và train.**

Build model từ các lớp convolution neural network.



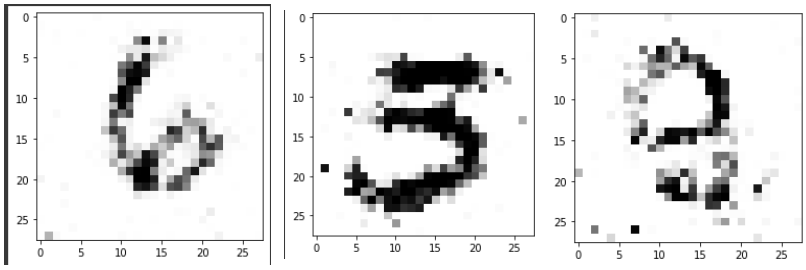

Kết quả cuối cùng sau khi train. So với ảnh gốc thì đánh giá 7/10 một kết quả tốt nhưng không đánh giá cao. Với:

- D loss: 0.593228.
- G loss: 0.832692.

Các thông số train:

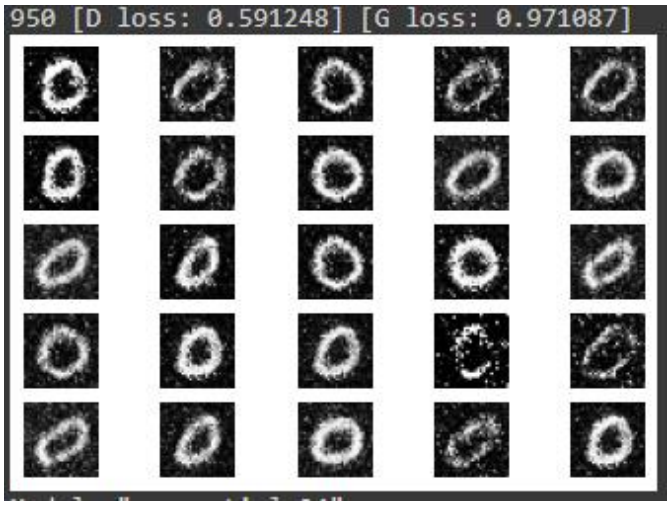
- Optimizer: Adam(learning rate = 0.0002, beta1 = 0.5
- Epochs: 5000
- Hàm kích hoạt:
    - Generator: tanh
    - Discriminator: sigmoid

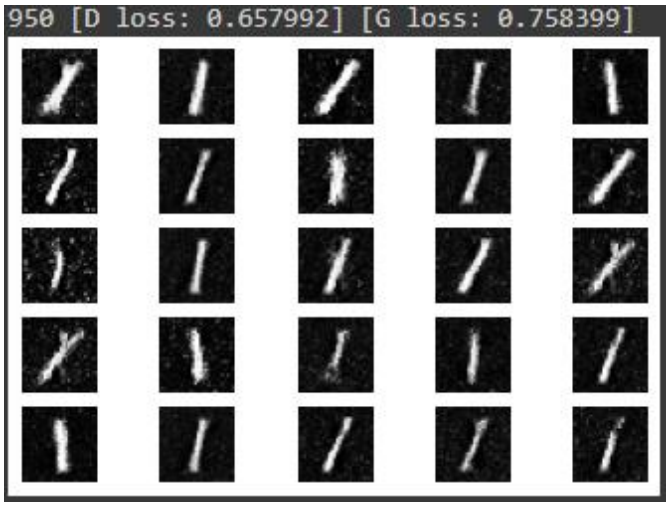
**Kết quả hình ảnh các con số được model tạo ra.**



**Chia các label ra riêng lẽ và mỗi label sẽ có 1 model riêng.**

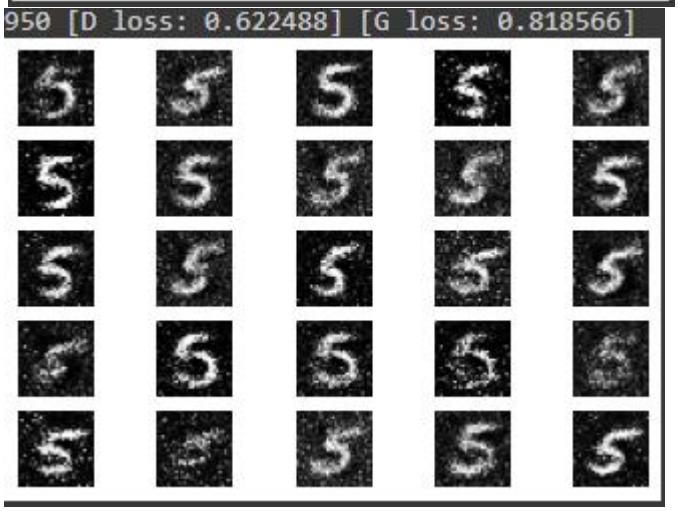
Kết quả mỗi khi train của mỗi model.

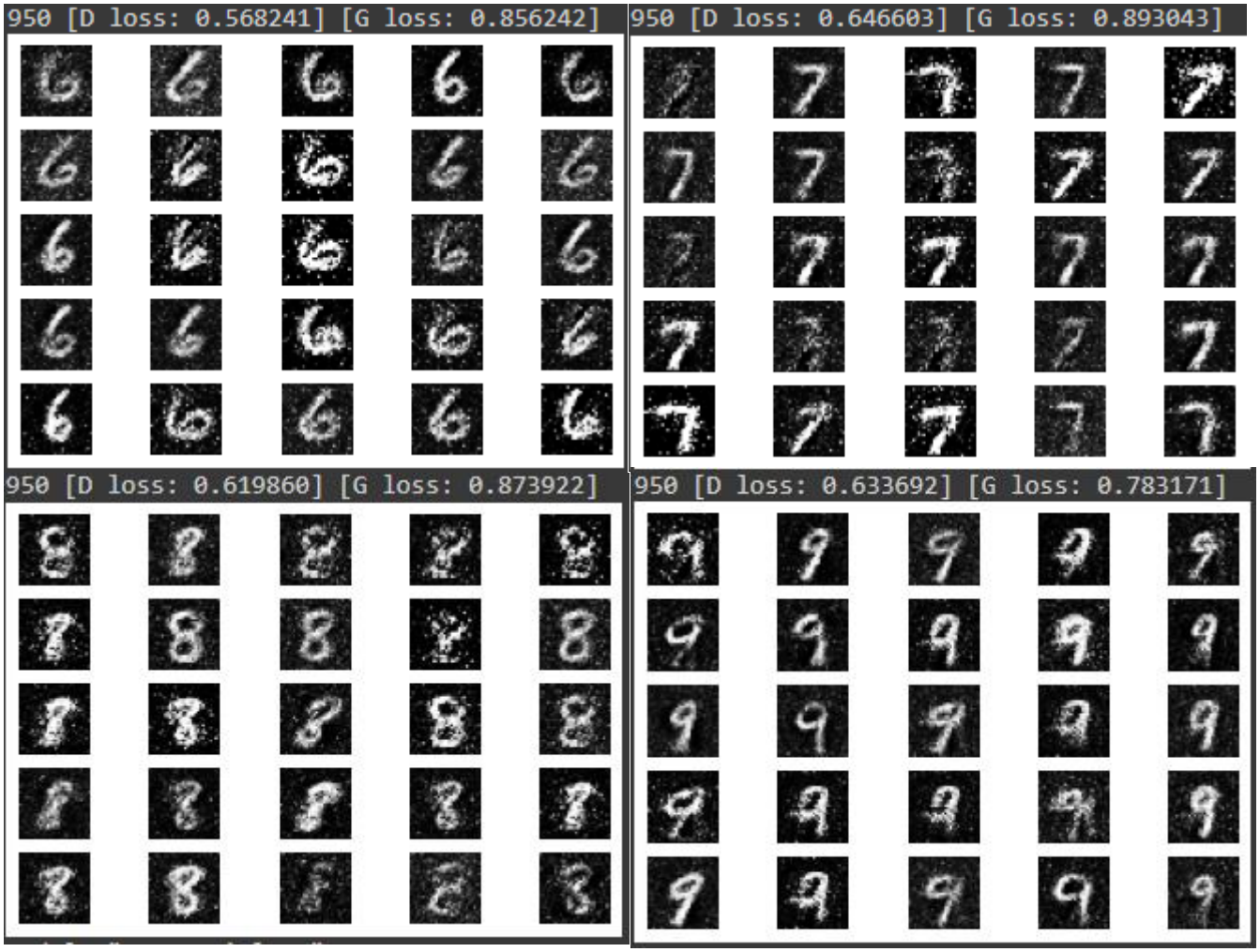











Thông tin train:

- Epochs mỗi lần train: 1000 epochs
- Learning Rate: 0.0002
- Beta1: 0.5

Đánh giá: So với ảnh thực khá tốt.

# 4. CHƯƠNG 5: KẾT LUẬN

## 4.1. Kết luận

Tiềm hiểu rõ về Generative Adversarial Networks(GANs) được cấu thành từ 2 mạng Generator và Discriminator.

GANs giải quyết vấn đề về các hoạt động huấn luyện mà không cần can thiệp nhiều đến từ còn người trong việc huấn luyện cụ thể rõ như gán nhãn cho dữ liệu.

Hiểu vai trò của từng mạng Generator và Discriminator trong GANs. Hai mạng này được xem giống như là đối trọng của nhau, và cũng như cùng nhau phát triển. Discriminator báo Generator dữ liệu làm giả của nó không được tốt, Discriminator thì

sẽ học được khả năng phân biệt dữ liệu thật và giả nhờ dữ liệu được cung cấp từ Generator.

Hiểu được Ưu và nhược điểm của GANs.

Tìm hiểu sơ hàm Loss Function của model GANs

Biết được một vài kiến trúc GANs phổ biến

### 4.1.1. Ưu điểm

GAN tạo ra dữ liệu giống với dữ liệu gốc. Nếu cung cấp cho GAN một hình ảnh thì nó sẽ tạo ra một phiên bản mới của hình ảnh trông giống với hình ảnh gốc. Tương tự, nó có thể tạo ra các phiên bản khác nhau của văn bản, video, âm thanh.

GAN đi sâu vào chi tiết dữ liệu và có thể dễ dàng diễn giải thành các phiên bản khác nhau, vì vậy nó rất hữu ích trong việc thực hiện công việc học máy.

Bằng cách sử dụng GAN và máy học, có thể dễ dàng nhận ra cây cối, đường phố, người đi xe đạp, người và ô tô đang đỗ và cũng có thể tính toán khoảng cách giữa các đối tượng khác nhau.

### 4.1.2. Nhược điểm

Khó đào tạo hơn: Cần cung cấp liên tục các loại dữ liệu khác nhau để kiểm tra xem nó có hoạt động chính xác hay không.

Việc tạo kết quả từ văn bản hoặc giọng nói rất phức tạp

## TÀI LIỆU THAM KHẢO

Sách tham khảo:

1.GANs in Action: Deep Learning with Generative Adversarial Networks

Tài liệu tham khảo:

2. https://nttuan8.com/

3. https://www.researchgate.net/publication/334484229_OnkoGan_Bangla_Handwritten_Digit_Generation_with_Deep_Convolutional_Generative_Adversarial_Networks

4. https://machinelearningmastery.com

5. DCGAN Tutorial — PyTorch Tutorials 1.10.1+cu102 documentation

6. 1511.06434.pdf (arxiv.org)
7. python_for_microscopists/125_126_GAN_training_mnist.py at master · bnsreenu/python_for_microscopists (github.com)