ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BÁO CÁO BÀI TẬP LỚN

MÔN HỌC: MACHINE LEARNING

GIẢNG VIÊN: TẠ VIỆT CƯỜNG

CHỦ ĐỀ: MÔ HÌNH GAN

CHU THANH TÙNG	**MSSV: 19021386**
BÙI ĐĂNG NAM BÌNH	**MSSV: 19021225**
NGUYỄN QUỐC KHÁNH	**MSSV: 19021313**

I. Định nghĩa mô hình GAN

Mô hình mạng sinh đối lập tổng quan mạng gồm thành phần sinh và thành phần phân biệt, quá trình huấn luyện giống như zero-sum game trong lý thuyết trò chơi thành phần sinh sẽ cố gắng sinh ra dữ liệu sao cho giống với dữ liệu mẫu còn thành phần phân biệt sẽ cố gắng xác định đâu là dữ liệu sinh đâu là dữ liệu mẫu. Quá trình huấn luyện có kết quả lý tưởng khi thành phần sinh được huấn luyện rất tốt mà chỉ có thành nắm được 50% đâu dữ liệu sinh có là thật hay không

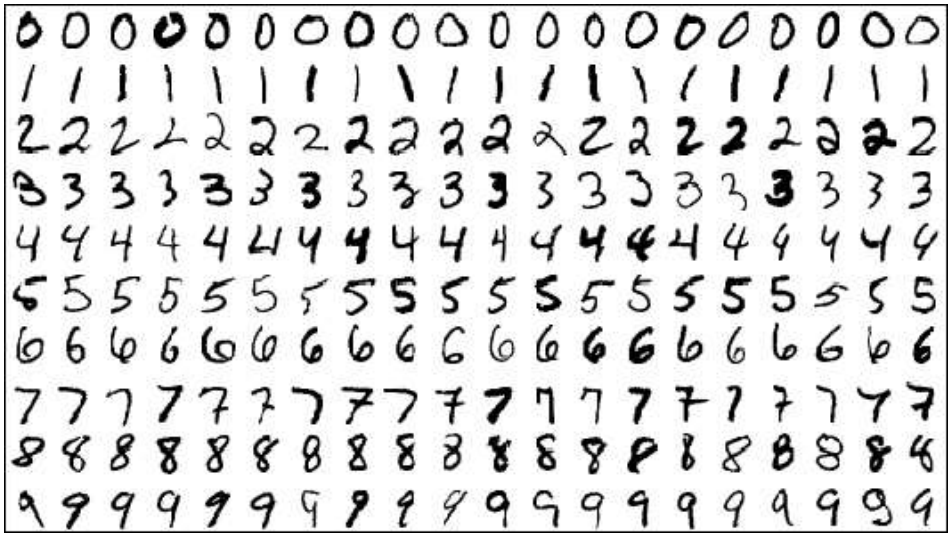
II. Ý tưởng

Ở đây chúng ta chỉ làm rõ cách mô hình GAN hoạt động trên bộ dữ liệu chữ số MNIST, và kết quả là các chữ số viết tay do GAN sinh ra

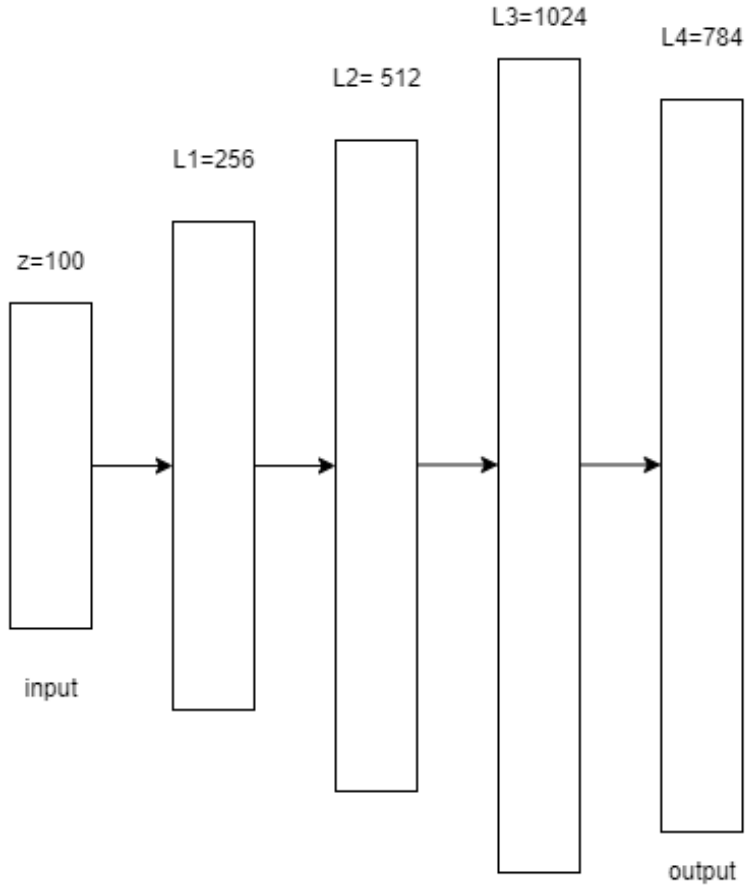
Các thành phần cơ bản

GAN gồm 2 thành phần chính

- *Generator*: Thành phần sinh dữ liệu
    - Generator có input là noise (random vector) là output là chữ số viết tay.
    - Vì các chữ số khi viết ra sẽ khác nhau. Ví dụ như bộ dữ liệu MNIST minh họa dưới đây, ta có thể thấy các chữ số 0 ở hàng đầu không giống nhau, tuy nhiên chúng vẫn là số 0. Thế nên với input là noise (random vector) khi ta thay đổi noise ngẫu nhiên thì Generator vẫn có thể sinh ra 1 biến dạng khác của chữ số 0 viết tay. Noise cho Generator thường được sinh ra từ normal distribution hoặc uniform distribution.



    - Input của Generator là noise vector 100 chiều.
      Sau đấy mô hình neural network được áp dụng với số node trong hidden layer lần lượt là 256, 512, 1024.
    -



GENERATOR

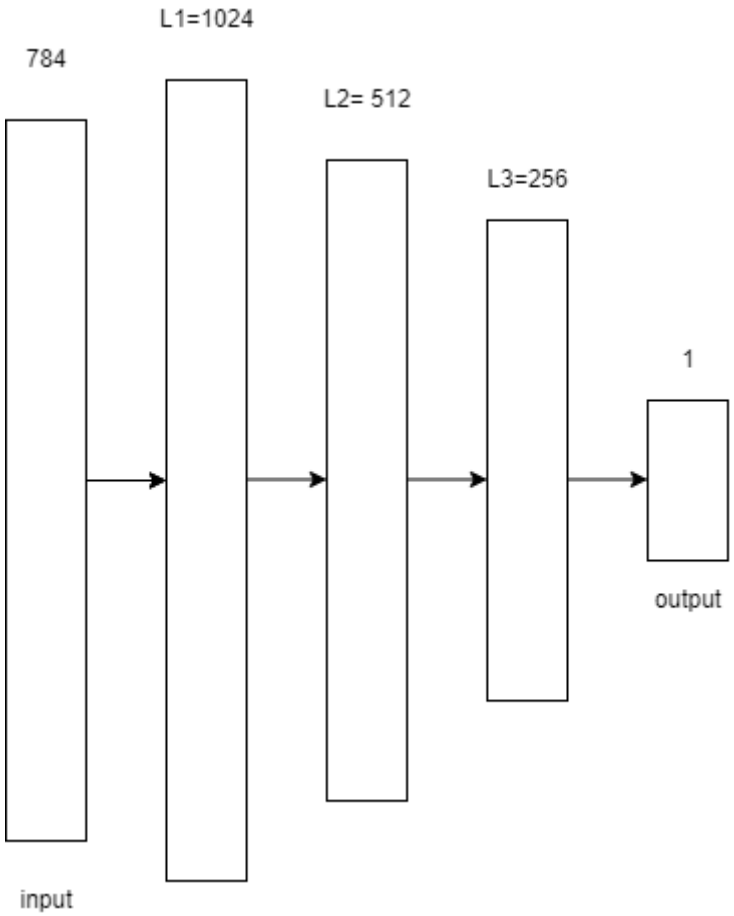
- Output đầu ra là ảnh giống với dữ liệu MNIST, ảnh xám kích thước 28*28 (784 pixel).

```
# Mô hình Generator
g = Sequential()
g.add(Dense(256, input_dim=z_dim, activation=LeakyReLU(alpha=0.2)))
g.add(Dense(512, activation=LeakyReLU(alpha=0.2)))
g.add(Dense(1024, activation=LeakyReLU(alpha=0.2)))
# Vì dữ liệu ảnh MNIST đã chuẩn hóa về [0, 1] nên hàm G khi sinh ảnh ra
#cần sinh ra ảnh có pixel value trong khoảng [0, 1] => hàm sigmoid được chọn
g.add(Dense(784, activation='sigmoid'))
g.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])
```

- Output là vector kích thước 784*1 sẽ được reshape về 28*28 đúng định dạng của dữ liệu MNIST.
- Vì dữ liệu ảnh MNIST đã chuẩn hóa về [0, 1] nên hàm G khi sinh ảnh ra cần sinh ra ảnh có pixel value trong khoảng [0, 1] => hàm sigmoid được chọn

- ***Discriminator:*** thành phần phân biệt đâu là dữ liệu được lấy mẫu với dữ liệu được sinh ra bởi Generator.

- Discriminator có input là ảnh biểu diễn bằng 784 chiều, output là ảnh thật hay ảnh giả (output là 0 hoặc 1).
- Input của Discriminator là ảnh kích thước 784 chiều.
- Sau đấy mô hình neural network được áp dụng với số node trong hidden layer lần lượt là 1024, 512, 256. Mô hình đối xứng lại với Generator.
- Output là 1 node thể hiện xác suất ảnh input là ảnh thật, hàm sigmoid được sử dụng.
- Giá trị output của model qua hàm sigmoid nên sẽ trong (0, 1) nên Discriminator sẽ được train để input ảnh ở dataset thì output gần 1, còn input là ảnh sinh ra từ Generator thì output gần 0.



DISCRIMINATOR

```
# Mô hình Discriminator
d = Sequential()
d.add(Dense(1024, input_dim=784, activation=LeakyReLU(alpha=0.2)))
d.add(Dropout(0.3))
d.add(Dense(512, activation=LeakyReLU(alpha=0.2)))
d.add(Dropout(0.3))
d.add(Dense(256, activation=LeakyReLU(alpha=0.2)))
d.add(Dropout(0.3))
# Hàm sigmoid cho bài toán binary classification
d.add(Dense(1, activation='sigmoid'))
d.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])
```

- Loss function:
  - Kí hiệu z là noise đầu vào của generator, x là dữ liệu thật từ bộ dataset.
  - Kí hiệu mạng Generator là G, mạng Discriminator là D. G(z) là ảnh được sinh ta từ Generator. D(x) là giá trị dự đoán của Discriminator xem ảnh x là thật hay không, D(G(z)) là giá trị dự đoán xem ảnh sinh ra từ Generator là ảnh thật hay không.
  - Ta sẽ thiết kế 2 hàm loss, một cho G và một cho D
  - Loss function muốn maximize D(x) và minimize D(G(z)). Ta có minimize D(G(z)) tương đương với maximize (1 – D(G(z))). Do đó loss function của Discriminator là:

$$\max_D V(D) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{recognize real images better}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{recognize generated images better}}$$

Loss Discriminator. Nguồn: https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf
(E là kì vọng, là lấy trung bình của tất cả dữ liệu, hay maximize D(x) với x là dữ liệu trong traning set.)

  - Generator sẽ học để đánh lừa Discriminator rằng số nó sinh ra là số thật, hay D(G(z)) -> 1. Hay loss function muốn maximize D(G(z)), tương đương với minimize (1 – D(G(z)))

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

Optimize G that can fool the discriminator the most.

Loss Generator. Nguồn: https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

- Do đó ta có thể viết gộp lại loss của mô hình GAN:

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

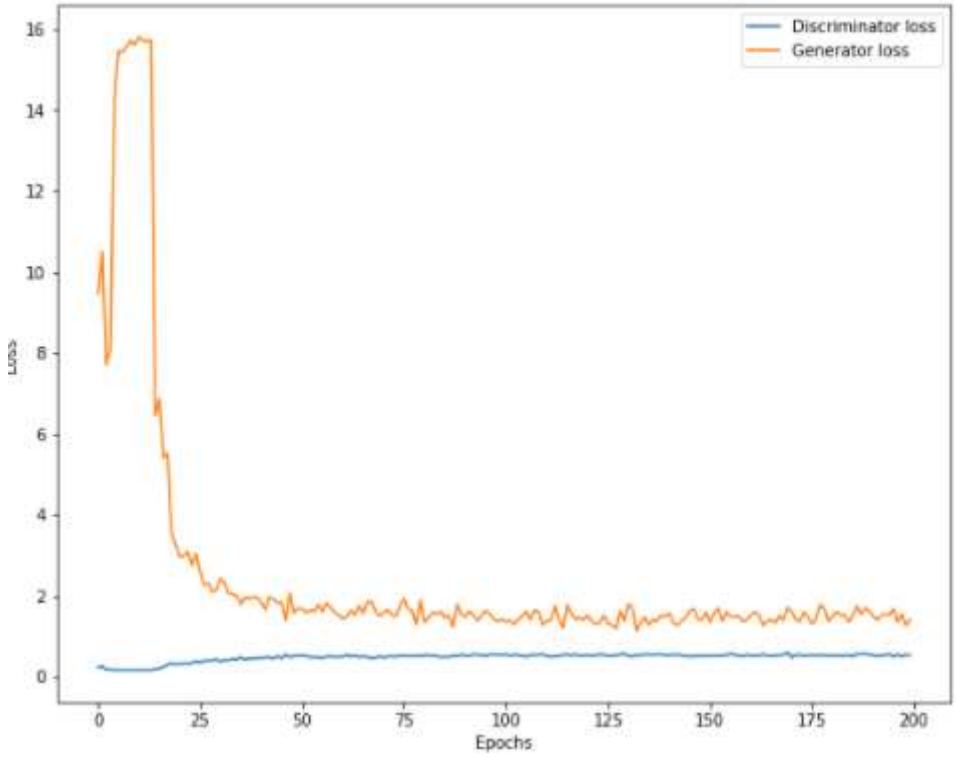
Loss GAN. Nguồn: https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

Từ hàm loss của GAN có thể thấy là việc train Generator và Discriminator đối nghịch nhau, trong khi D cố gắng maximize loss thì G cố gắng minimize loss. Quá trình train GAN kết thúc khi model GAN đạt đến trạng thái cân bằng của 2 models, gọi là Nash equilibrium.

III. Kết quả GAN

- Sau khi chạy với epoch tăng dần, từ 1 epoch đến 200 (bước nhảy 20) epoch và dataset là bộ số viết tay của MNIST ta có kết quả sau

- Có biểu đồ loss của 2 hàm loss Generator loss và Discrimination loss


Discriminator loss
Generator loss
Loss
Epochs
0
2
4
6
8
10
12
14
16
0
25
50
75
100
125
150
175
200

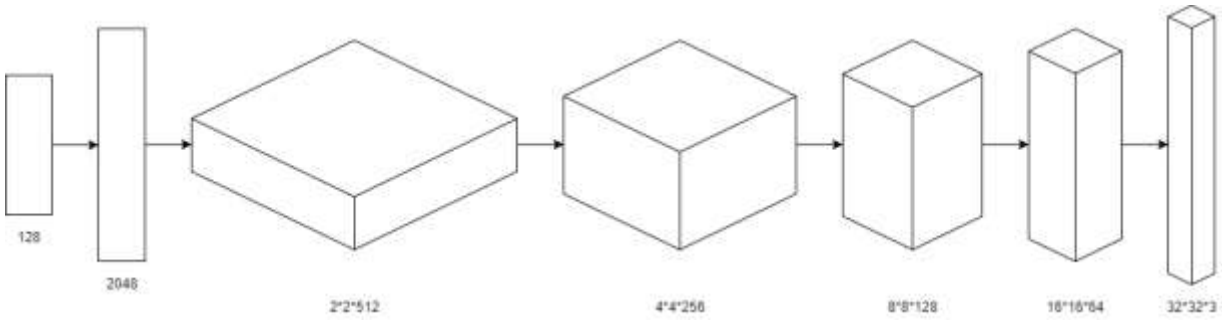
IV. DCGAN

Deep Convolutional Generative Adversarial Network sẽ được sử dụng cho các dữ liệu là ảnh thay vì mạng GAN thông thường.

Về cơ bản, DCGAN cũng sẽ có 2 thành phần giống mô hình GAN cơ bản, gồm 2 thành phần là Generator và Discriminator, tuy nhiên chúng sẽ được xây dựng bằng mô hình CNN với 2 layers chính là convolutional layer và transposed convolutional layer.

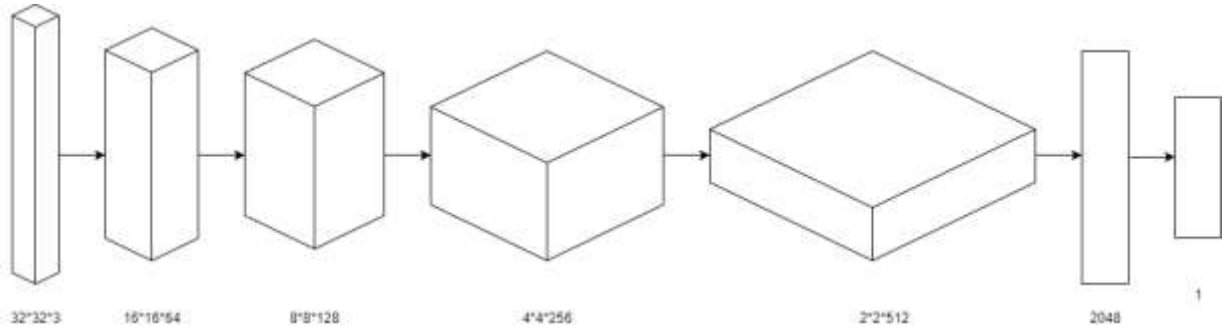
- ***Generator:***
  Các layer trong mạng

  - Dense (fully-connected) layer: 128*1 -> 2048*1
  - Flatten chuyển từ vector về dạng tensor 3d, 2048*1 -> 2*2*512
  - Transposed convolution stride=2, kernel=256, 2*2*512 -> 4*4*256
  - Transposed convolution stride=2, kernel=128, 4*4*256 -> 8*8*128
  - Transposed convolution stride=2, kernel=64, 8*8*128 -> 16*16*64
  - Transposed convolution stride=2, kernel=3, 16*16*64 -> 32*32*3



Mô hình generator của DCGAN

- ***Discriminator:***
  Mô hình discriminator đối xứng lại với mô hình generator. Ảnh input được đi qua convolution với stride = 2 để giảm kích thước ảnh từ 32*32 -> 16*16 -> 8*8 -> 4*4 -> 2*2. Khi giảm kích thước thì depth tăng dần. Cuối cùng thì tensor shape 2*2*512 được reshape về vector 2048 và dùng 1 lớp fully connected chuyển từ 2048d về 1d.

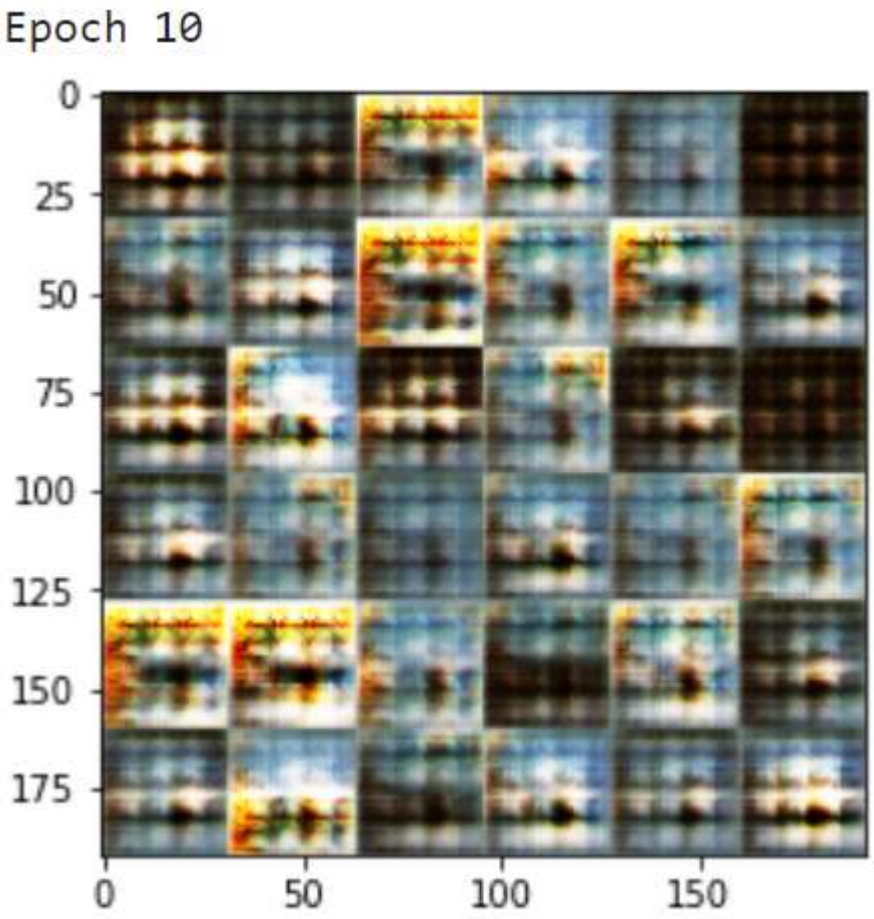


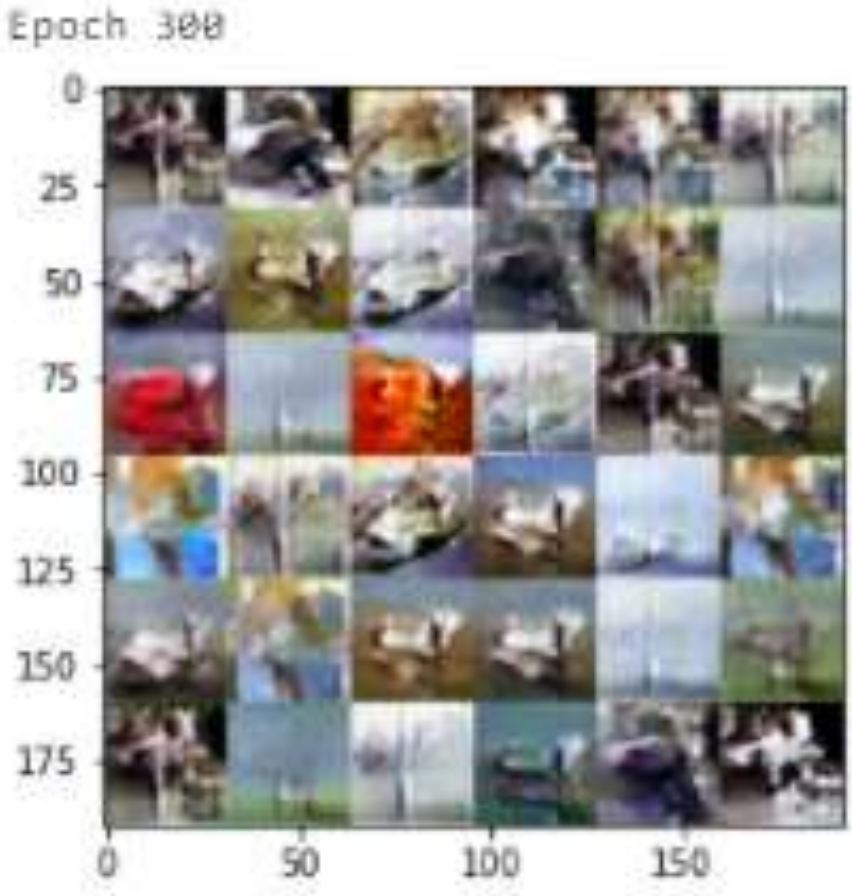
Mô hình discriminator của DCGAN

- ***Loss function:***
  Loss trong DCGAN giống với mạng GAN thông thường.

V. Kết quả DCGAN

Sau khi chạy với epoch tăng dần, từ 10 epoch đến 300 epoch (bước nhảy 10) và dataset là CIFAR-10 (CIFAR-10 dataset bao gồm 60000 ảnh màu kích thước 32×32 thuộc 10 thể loại khác nhau. Mỗi thể loại có 6000 ảnh).
ta có kết quả sau:





Phân chia công việc:

- Tất cả các thành viên đều sẽ phải đọc và dịch các bài báo, phần code liên quan.
- Các thành viên sẽ cùng thống nhất các phần thông tin liên quan và viết report
- Tìm hiểu về DCGAN: Chu Thanh Tùng, Nguyễn Quốc Khánh.
- Tìm hiểu về Generator, Discriminator: Bùi Đăng Nam Bình
- Tìm hiểu về Loss: Nguyễn Quốc Khánh
- Code DCGAN: Chu Thanh Tùng
- Code GAN: Nguyễn Quốc Khánh
- Viết báo cáo: tất cả các thành viên
- Trình bày: Nguyễn Quốc Khánh