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

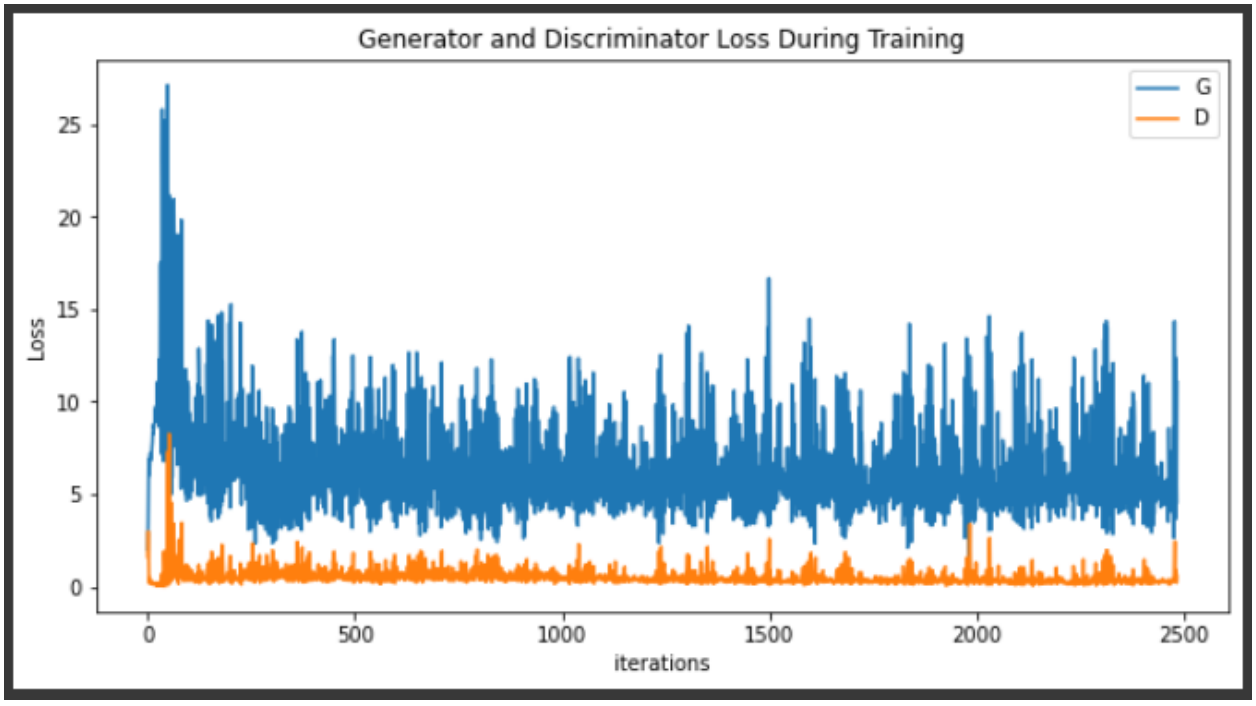
- Vòng lặp thứ 4 (3/5): Không khác lắm so với ở trên.

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

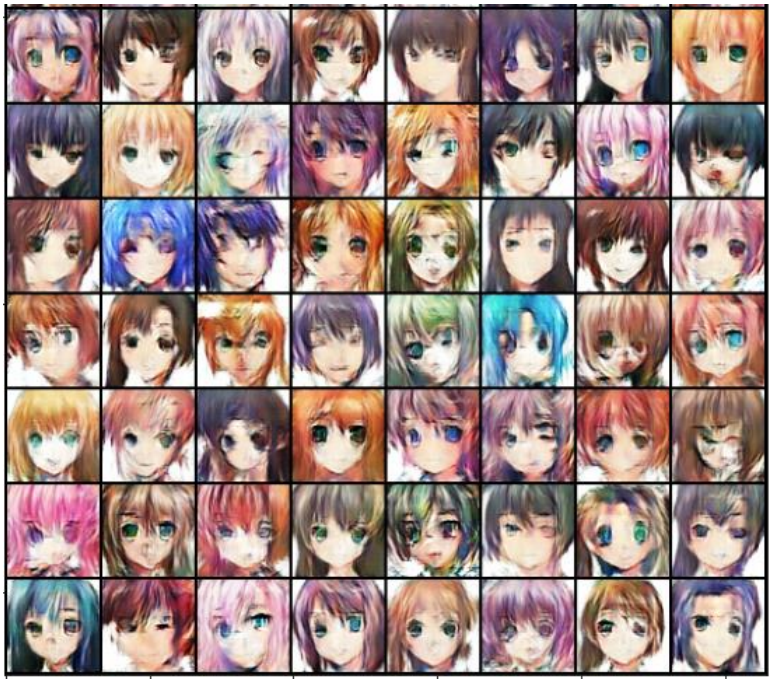
Biểu đồ giá trị Loss của Generator và Discrimnator trong suốt quá trình Train:



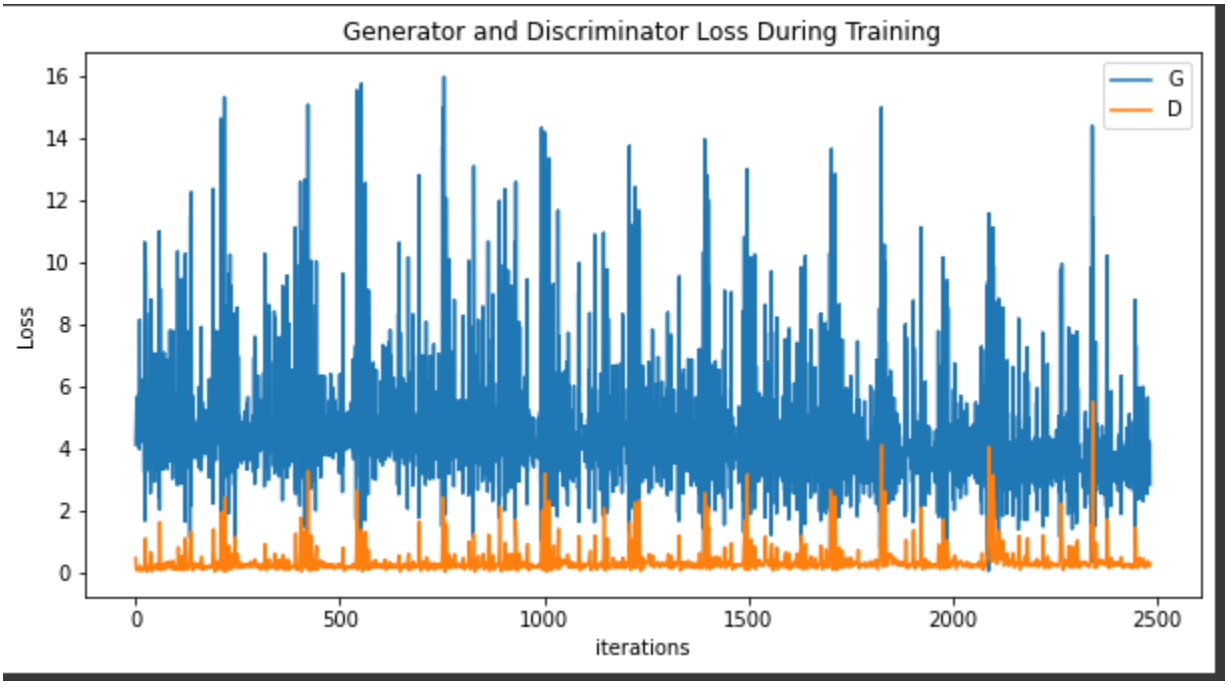
Quyết đinh train thêm 5 epochs tiếp với việc giảm Learning rate xuống gấp đôi và giảm beta còn 0.3. Liệu có thể tối ưu hóa hơn không?

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.

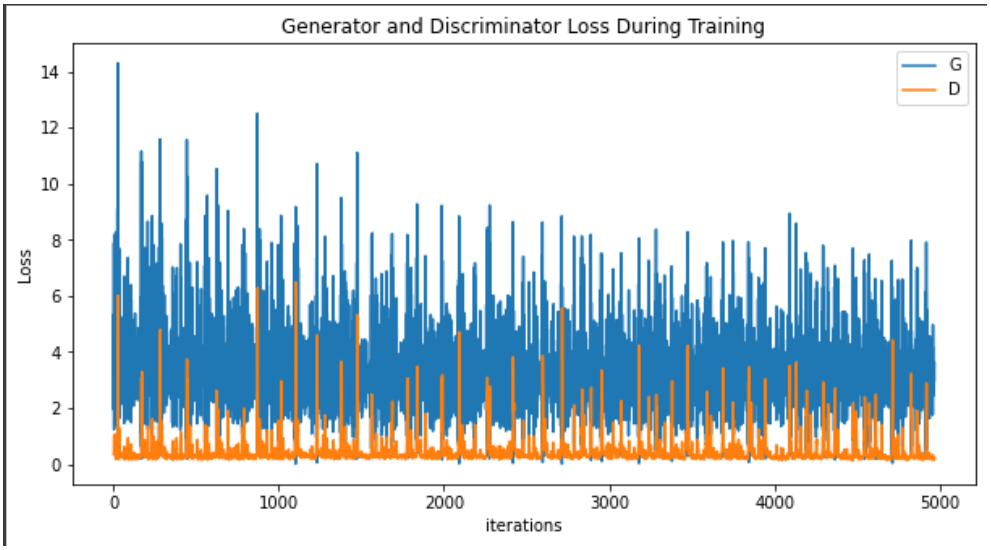
Hình ảnh tuy khả quan hơn lần trước nhưng vẫn ko tốt khi so với ảnh gốc trong bộ dữ liệu.

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

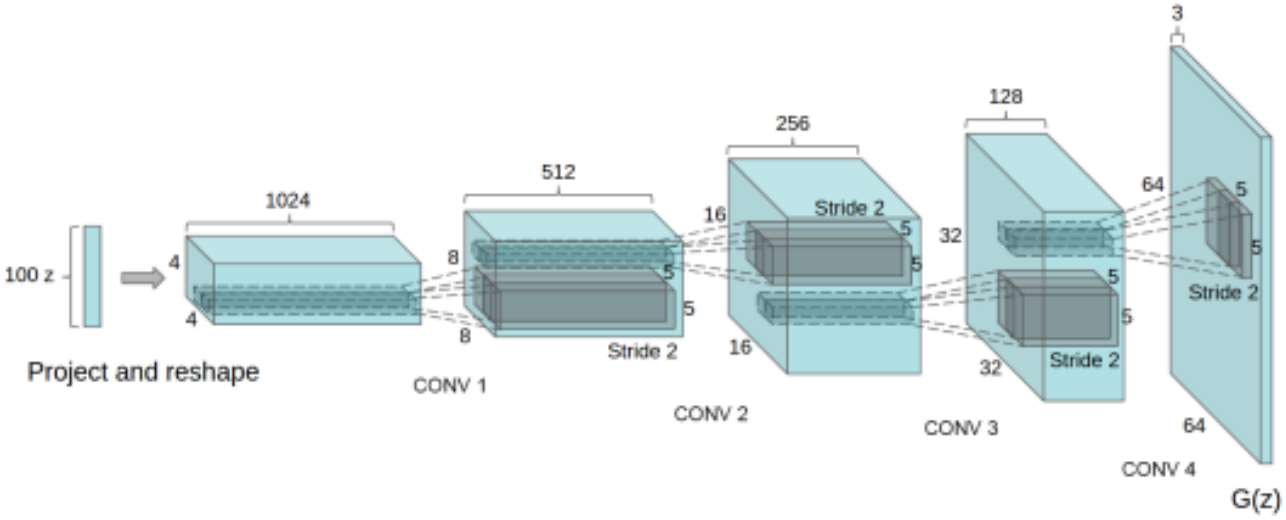
Dừng lại: Thấy được kết quả đã được cải thiện nhưng so với ảnh thật thì không được tốt. Sau nhiều lần thay đổi learning rate và train thêm, theo dự đoán tụi em dù có train thêm kết quả có thể sẽ không tốt hơn được nhiều.

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.

Tham khảo.

- DCGAN Tutorial — PyTorch Tutorials 1.10.1+cu102 documentation
- 1511.06434.pdf (arxiv.org)