CSE 676: Deep Learning Project1

Tingting wang

Abstract: This report will include the conception of Deep Convolution GAN and Self Attention GAN, also will include the performance comparison of both. There are frame of this report:

- Deep Convolution Generative Adversarial Networks
  - Architecture
  - Training
  - Performance
- Self Attention Generative Adversarial Networks
  - Conditional GAN
  - Wasserstein Loss
  - Attention Layer
  - Spectral Normalization
- Comparison
- Reference
  - Code
  - Paper

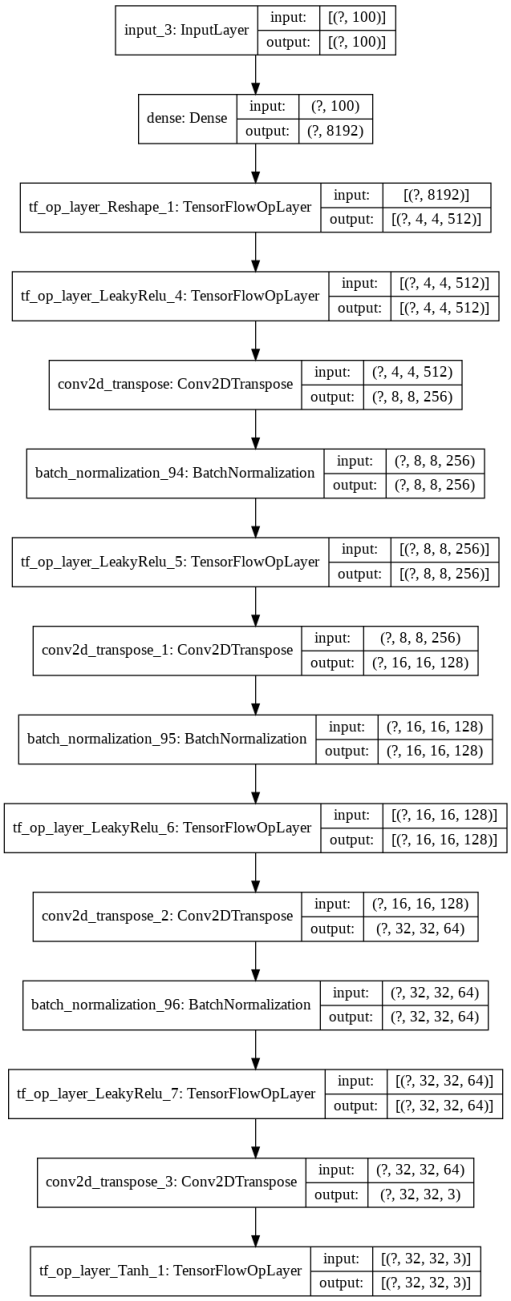
1. DCGAN

1.1. Generator

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 100)]	0
dense_2 (Dense)	(None, 8192)	827392
tf_op_layer_Reshape_2 (Tenso	[(None, 4, 4, 512)]	0
tf_op_layer_LeakyRelu_16 (Te	[(None, 4, 4, 512)]	0
conv2d_transpose_4 (Conv2DTr	(None, 8, 8, 256)	2097152
batch_normalization_97 (Batc	(None, 8, 8, 256)	1024
tf_op_layer_LeakyRelu_17 (Te	[(None, 8, 8, 256)]	0
conv2d_transpose_5 (Conv2DTr	(None, 16, 16, 128)	524288
batch_normalization_98 (Batc	(None, 16, 16, 128)	512
tf_op_layer_LeakyRelu_18 (Te	[(None, 16, 16, 128)]	0
conv2d_transpose_6 (Conv2DTr	(None, 32, 32, 64)	131072
batch_normalization_99 (Batc	(None, 32, 32, 64)	256
tf_op_layer_LeakyRelu_19 (Te	[(None, 32, 32, 64)]	0
conv2d_transpose_7 (Conv2DTr	(None, 32, 32, 3)	1728
tf_op_layer_Tanh_2 (TensorFl	[(None, 32, 32, 3)]	0

Total params: 3,583,424
Trainable params: 3,582,528
Non-trainable params: 896

(a)



(b)

Figure 1: Framework of Generator

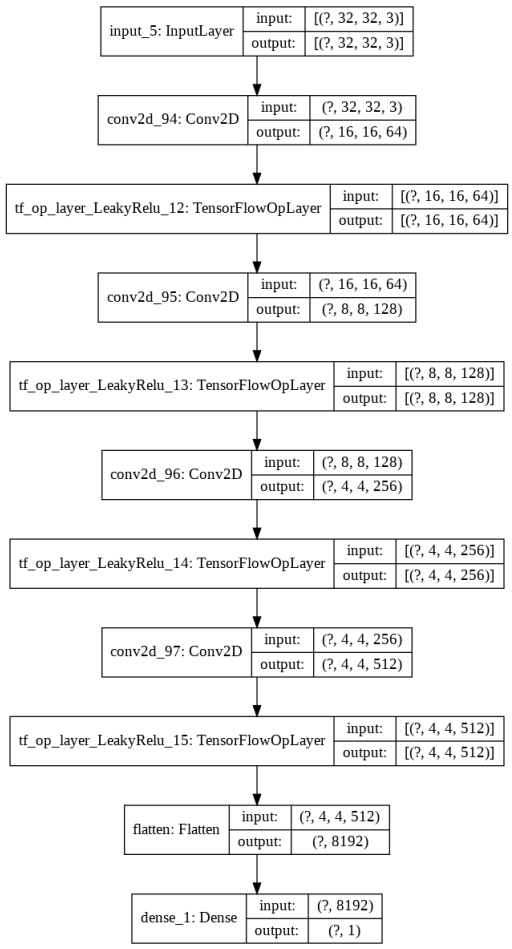
In the Generator, I apply dense, convolution transpose, batch normalization and leacky rulu like Figure 1.

1.2. Discriminator

Layer (type)	Output Shape	Param #
input_8 (InputLayer)	[(None, 32, 32, 3)]	0
conv2d_98 (Conv2D)	(None, 16, 16, 64)	3136
tf_op_layer_LeakyRelu_24 (Te	[(None, 16, 16, 64)]	0
conv2d_99 (Conv2D)	(None, 8, 8, 128)	131200
tf_op_layer_LeakyRelu_25 (Te	[(None, 8, 8, 128)]	0
conv2d_100 (Conv2D)	(None, 4, 4, 256)	524544
tf_op_layer_LeakyRelu_26 (Te	[(None, 4, 4, 256)]	0
conv2d_101 (Conv2D)	(None, 4, 4, 512)	1180160
tf_op_layer_LeakyRelu_27 (Te	[(None, 4, 4, 512)]	0
flatten_1 (Flatten)	(None, 8192)	0
dense_3 (Dense)	(None, 1)	8193

Total params: 1,847,233
Trainable params: 1,847,233
Non-trainable params: 0

(a)

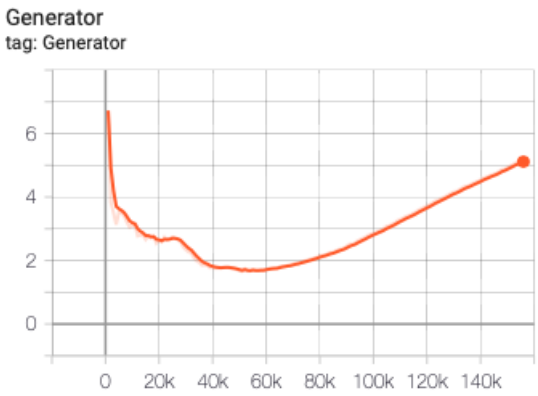


(b)

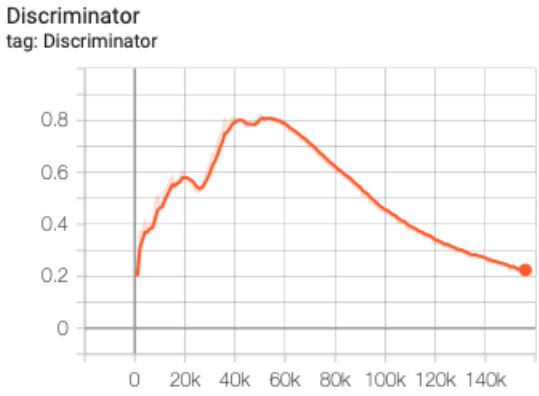
Figure 2: Framework of Discriminator

In the Discriminator of DCGAN, I apply dense, convolution, fatten and leacky relu like Figure 2

1.3. Training Process



(a) generator loss



(b) discriminator loss

Figure 3: training loss

As we can see it from Figure 3, during the process of training, the discriminator loss first increases, and then decreases, but the generator loss first decreases, and then increases. The turning point is about 60k.

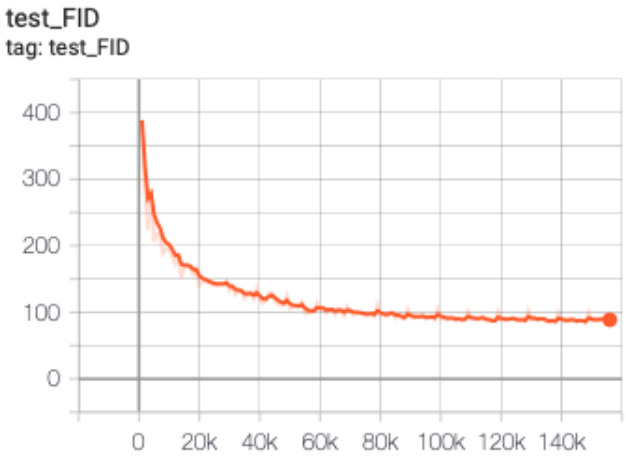
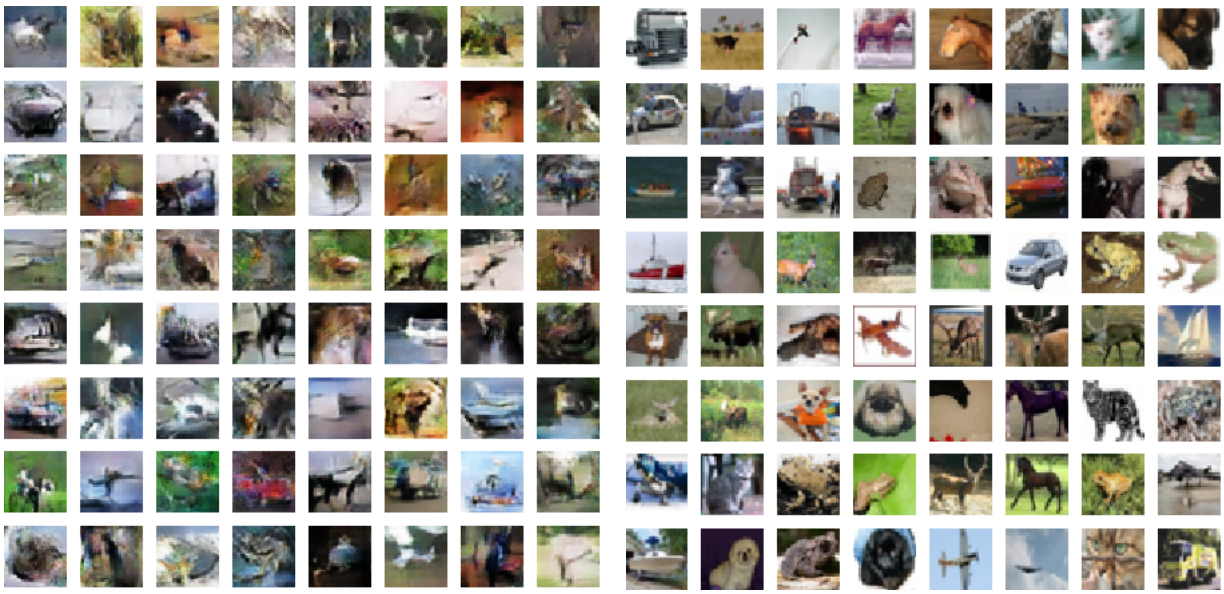


Figure 4: FID

As we can see it from Figure 4, during the process of training, before this turning point, the test fid goes down quickly, but after that, it still goes down, however in a super slow way, so I just pick the newest model as best model.

1.4. Performance



(a) Generated img (b) Ground truth

Figure 5: Generated vs Ground-truth

As we can see it from Figure 4, the best model is the newest model, so I restore the checkpoint, and use random noise to generate some image, like Figure 5, the quality of generated img is kind of ok when compared to the Ground truth.

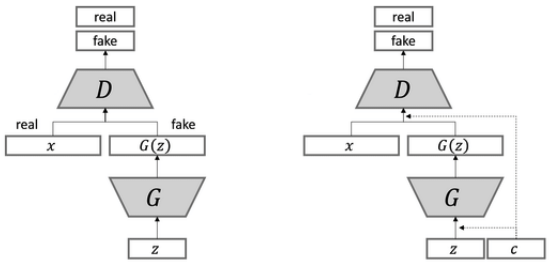
And finally, the mean FID 90, which is calculated on test dataset(10k) and it's calculated with every 1024 test data and 1024 generated images, and calculate the mean of them, you can recover the result from my DCGAN.ipynb, it looks pretty high, but the FID between the test dataset and train dataset(every 1024) is about 50.

2. SAGAN

As we all known, one limitation of DCGAN is that it can't model long term dependencies for generating image, due to convolution operator has a local receptive field, like it doesn't know how many eyes one image already have, then it may produce a image of human with more than 2 eyes. However, unlike DCGAN, SAGAN, which is also called Self-Attention Generative Adversarial Networks, can take care of modeling long-range dependencies and also the computational efficiency Networks.

2.1. Conditional GAN

The first difference between our implemented SAGAN and DCGAN is condition, like Figure 6



(a) DCGAN (b) SAGAN

Figure 6: Architecture of DCGAN and SAGAN

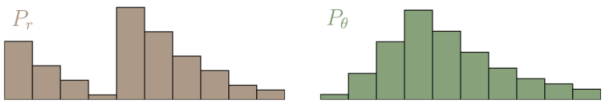
In Figure 6, G means Generator, and D means Discriminator, and x means image, and z means noise, and c means condition/label.

First talking about the generator, in DCGAN, we don't combine the condition(which is the label of train image), which means we only use noise(100-dimention random normal vector) to generate image; But, in SAGAN, we use the combination of label and noise((110-dimention vector, 10/110 represents the one-hot format label) to generate the image, which is also called conditional SAGAN.

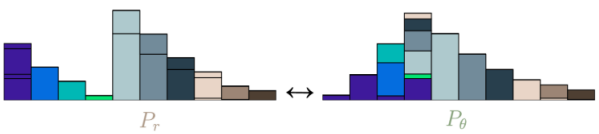
Second talking about discriminator, in DCGAN, the input of discriminator is only the ground truth image(without label) or the generated image(without label); But, in SAGAN, both the image(ground truth or generated) and the paired label are put into discriminator.

2.2. Wasserstein Loss

The second difference between our implemented SAGAN and DCGAN is whether use Wasserstein loss or not. Since the real images and generated images are low-dimension manifold in high-dimension space, the overlapping of they is very very small, which can be ignored. And it will lead to JS divergence is always same constant(log 2), which sometimes results in mode collapse. In order to prevent this issue, we use Wasserstein Loss, which can provide a meaningful and smooth representation of 2 distribution distance, even without data distribution overlaping.



(a) Probability distribution



(b) Optimal transportation

Figure 7: Wasserstein distance

The Wasserstein distance, Figure 7(Source:https://vincentherrmann.github.io/blog/wasserstein/), which is also called Earth Movers Distance, is the minimum cost of transporting mass in converting one data distribution to another one. There are 3 ways to achieve Wasserstein loss: weight clipping, hinge loss and gradient penalty. Among weight clipping, hinge loss and gradient penalty, gradient penalty is the most robust and stable, so I choose to use gradient penalty to finish it, so the final discriminator loss looks like Figure 8.

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right]}_{\text{Our gradient penalty}}.$$

Figure 8: Gradient Penalty

2.3. Attention Layer

Most GAN model use convolutional layers to generate image, but convolutional operations only processes the data in a local-neighborhood way, then it's very computationally inefficient for modeling long-range dependencies in image. In SAGAN, there is self attention layer which can enable both generator ans discriminator to efficiently model long-range relationship in image, like Figure 9.

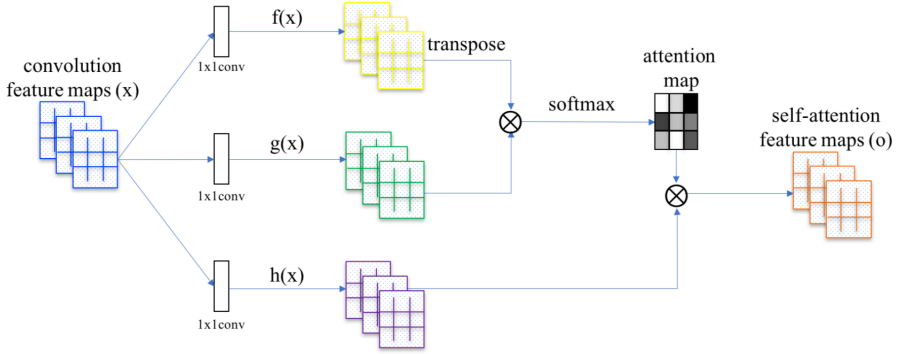


Figure 9: Self Attention

The convolutional feature maps(x), coming from the previous hidden layers $x \in \mathbb{R}^{C\times N}$ (C is the number of channels and N is the number of feature location of previous layer), are first transformed into 2 feature spaces: f, g in order to calculate the attention, where $f(x) = W_f x$, $g(x) = W_g x$, and $\beta_{j,i}$ means the extent to which the model attends to the i_{th} location when synthesizing the j_{th} region.

$$\beta_{j,i} = \frac{\exp(S_{ij})}{\sum_{n=1}^{N}\exp(S_{ij})}, S_{ij} = f(x_i)^{\mathrm{T}} g(x_i) \tag{1}$$

And the output of this layer is $o = (o_1, o_2, o_3, ..., o_j, ..., o_N) \in \mathbb{R}^{C\times N}$, where,

$$o_j = v(\sum_{n=1}^{N}\beta_{j,i} h(x_i)), h(x_i) = W_h x_i, v(x_i) = W_v x_i. \tag{2}$$

In the above equation, $W_g \in \mathbb{R}^{\overline{C}\times C}$, $W_f \in \mathbb{R}^{\overline{C}\times C}$, $W_h \in \mathbb{R}^{\overline{C}\times C}$ and $W_v \in \mathbb{R}^{C\times\overline{C}}$ are 1×1 convolutions, and $\overline{C}$ are set to be C/8. After that, keep multiply the output of attention layer by a scale parameter and add the input feature map, like below, where γ is a learnable scalar and it is initialized as 0.

$$y_i = \gamma o_i + x_i, \tag{3}$$

In conclusion, I apply the self-attention layer in both Generator and Discriminator, then all of them could learn the long-range dependencies very efficiently, which will lead to a better quality of generated images.

2.4. Spectral Normalization

Most of this chapter(Lipschitz Continuity, The multidimension and Spectral normaliza) comes from https://christiancosgro
normalization-explained.html

2.4.1. Lipschitz Continuity

Suppose we have a GAN discriminator $D : I \to \mathbb{R}$, where I is the space of images (e.g., $\mathbb{R}^{32\times32}$). Because both the domain and codomain of this function have inner products, we have a natural metric (distance function) in both spaces: the L2 distance. If our discriminator is K-Lipschitz continuous, then for all x and y in I,

$$||D(x) - D(y)|| \leq K||x - y|| \tag{4}$$

where $|\cdot|$ is the L2 norm. Here, if K is a minimum, then it is called the Lipschitz constant of the discriminator.

Let's talk about 1-Lipschitz. Its clear that sin is 1-Lipschitz continuous since the maximum value of its derivative is less than 1.And lipschitz continuity will bounds the gradients in our model.

2.4.2. The multidimensional case

Suppose we have a linear function $A : \mathbb{R}^n \to \mathbb{R}^m$. We can compute the spectral norm of this function, which is defined as the largest singular value of the matrix A, i.e., the square root of the largest eigenvalue of A^TA. Since A is linear, we can set our point of reference y to zero. In other words, if A is K-Lipschitz at zero, then it is K-Lipschitz everywhere. This require follows:

$$||Ax|| \leq K||x|| \tag{5}$$

$$\langle Ax, Ax\rangle \leq K^2\langle x, x\rangle, \forall x \in I \tag{6}$$

$$\langle (A^TA - K^2)x, x\rangle \leq 0, \forall x \in I. \tag{7}$$

$$\langle (A^TA - K^2)x, x\rangle = \langle (A^TA - K^2)\sum_i x_iv_i, \sum_j x_jv_j\rangle = \sum_i\sum_j x_ix_j\langle (A^TA - K^2)v_i, v_j\rangle = \sum_i(\lambda_i - K^2)x_i^2 \leq 0 \tag{8}$$

Since A^TA is positive semidefinite, all the λ_is must be nonnegative. To guarantee the above sum to be nonnegative, each term must be nonnegative, so

$$K^2 - \lambda_i \geq 0 for all i = 1 \dots n. \tag{9}$$

Therefore, the Lipschitz constant of a linear function is its largest singular value, or its spectral norm. Analogously to the 1D case, it is easy to observe that the Lipschitz constant of a general differentiable function $f : \mathbb{R}^n \to \mathbb{R}^m$ is the maximum spectral norm (maximum singular value) of its gradient over its domain.

$$||f||_{Lip} = \sup_x \sigma(\nabla f(x)) \tag{10}$$

Now, lets introduce a function $g : \mathbb{R}^m \to \mathbb{R}^l$. The underlying premise of the spectral normalization paper is that if we can find, or at least bound, the Lipschitz constant of the composition of f and g, then we can obtain a bound on the Lipschitz constant of an arbitrary multilayer discriminator, which is just a composition of linear maps and componentwise nonlinearities,

$$\nabla(g \circ f)(x) = \nabla g(f(x))\nabla f(x). \tag{11}$$

To find the spectral norm of a composition of functions, express it in terms of the spectral norm of the matrix product of its gradients. We can write the spectral norm (maximum singular value) in another convenient form:

$$\sigma(\nabla f(x)) = \sup_{||v||\leq 1} ||[\nabla f(x)]v|| \tag{12}$$

$$\sigma(\nabla(g \circ f)(x)) = \sup_{||v||\leq 1} ||[\nabla g(f(x))][\nabla f(x)]v|| \tag{13}$$

Since the supremum is convex, we can bound the result in (2) as follows:

$$\sup_{||v||\leq 1} ||[\nabla g(f(x))][\nabla f(x)]v|| \leq \sup_{||u||\leq 1} ||[\nabla g(f(x))]u|| \sup_{||v||\leq 1} ||[\nabla f(x)]v||. \tag{14}$$

Or,

$$||g \circ f||_{Lip} \leq ||g||_{Lip}||f||_{Lip}. \tag{15}$$

The solution goes like this: if we can fix each of the factors in the right-hand side of this inequality to 1, then we can ensure that the model is at most 1-Lipschitz.

2.4.3. Spectral normalization

Spectral normalization simply replaces every weight W with $W/\sigma(W)$. But how do we efficiently compute $\sigma(W)$, the largest singular value of W? The answer is a cheap and effective technique called power iteration.

Suppose we have a linear map $W : \mathbb{R}^n \to \mathbb{R}^m$. Suppose we have a random vector in the domain of our matrix, $v \in \mathbb{R}^n$, and a vector in the codomain, $u \in \mathbb{R}^m$.

Lets first consider v. We can form the square matrix $W^TW : \mathbb{R}^n \to \mathbb{R}^n$. Power iteration involves computing the recurrence relation

$$v_{t+1} = \frac{W^TWv_t}{||W^TWv_t||} \tag{16}$$

$$v_t = \frac{(W^TW)^tv}{||(W^TW)^tv||} \tag{17}$$

By the spectral theorem, we can write v in an orthonormal basis of eigenvectors of W^TW. Lets denote $\lambda_1 \dots \lambda_n$ as the descending eigenvalues of W^TW and $e_1 \dots e_n$ the corresponding eigenvectors.

$$v_t = \frac{(W^TW)^t \sum_i v_i e_i}{||(W^TW)^t \sum_i v_i e_i||} = \frac{\sum_i v_i \lambda_i^t e_i}{||\sum_i v_i \lambda_i^t e_i||} = \frac{v_1 \lambda_1^t \sum_i \frac{v_i}{v_1} \left(\frac{\lambda_i}{\lambda_1}\right)^t e_i}{||v_1 \lambda_1^t \sum_i \frac{v_i}{v_1} \left(\frac{\lambda_i}{\lambda_1}\right)^t e_i||}. \tag{18}$$

Now note that since λ_1 is the largest eigenvalue of W^TW, upon power iteration $\lim_{t\to\infty} \frac{\lambda_i}{\lambda_1} = 0$ for $igt;1$. So, after many iterations of the above procedure, v_t converges to e_1. We call the intermediate computation $\frac{Wv_t}{||Wv_t||} = u_t$. The power iteration procedure becomes:

$$u_{t+1} = Wv_tv_{t+1} = W^Tu_{t+1}. \tag{19}$$

Since the singular values of W^T and W are the same, it must be that the spectral norm is $\sigma(W) = \sqrt{\lambda_1} = ||Wv||$. Since $||u||$ is of unit length, we can conveniently compute the spectral norm as follows:

$$\sigma(W) = ||Wv|| = u^TWv. \tag{20}$$

Now, the algorithm of spectral normalization should appear simple. For every weight in our network, we randomly initialize vectors u and v. Because the weights change slowly, we only need to perform a single power iteration on the current version of these vectors for each step of learning.

Algorithm 1 SGD with spectral normalization

- Initialize $\tilde{\boldsymbol{u}}_l \in \mathcal{R}^{d_l}$ for $l = 1, \dots, L$ with a random vector (sampled from isotropic distribution).
- For each update and each layer l:
  1. Apply power iteration method to a unnormalized weight W^l:
  $$\tilde{\boldsymbol{v}}_l \leftarrow (W^l)^{\mathrm{T}}\tilde{\boldsymbol{u}}_l / \|(W^l)^{\mathrm{T}}\tilde{\boldsymbol{u}}_l\|_2$$
  $$\tilde{\boldsymbol{u}}_l \leftarrow W^l\tilde{\boldsymbol{v}}_l / \|W^l\tilde{\boldsymbol{v}}_l\|_2$$
  2. Calculate $\bar{W}_{\mathrm{SN}}$ with the spectral norm:
  $$\bar{W}^l_{\mathrm{SN}}(W^l) = W^l/\sigma(W^l), \text{ where } \sigma(W^l) = \tilde{\boldsymbol{u}}_l^{\mathrm{T}}W^l\tilde{\boldsymbol{v}}_l$$
  3. Update W^l with SGD on mini-batch dataset $\mathcal{D}_M$ with a learning rate α:
  $$W^l \leftarrow W^l - \alpha\nabla_{W^l}\ell(\bar{W}^l_{\mathrm{SN}}(W^l), \mathcal{D}_M)$$

Figure 10: Spectral Normalization

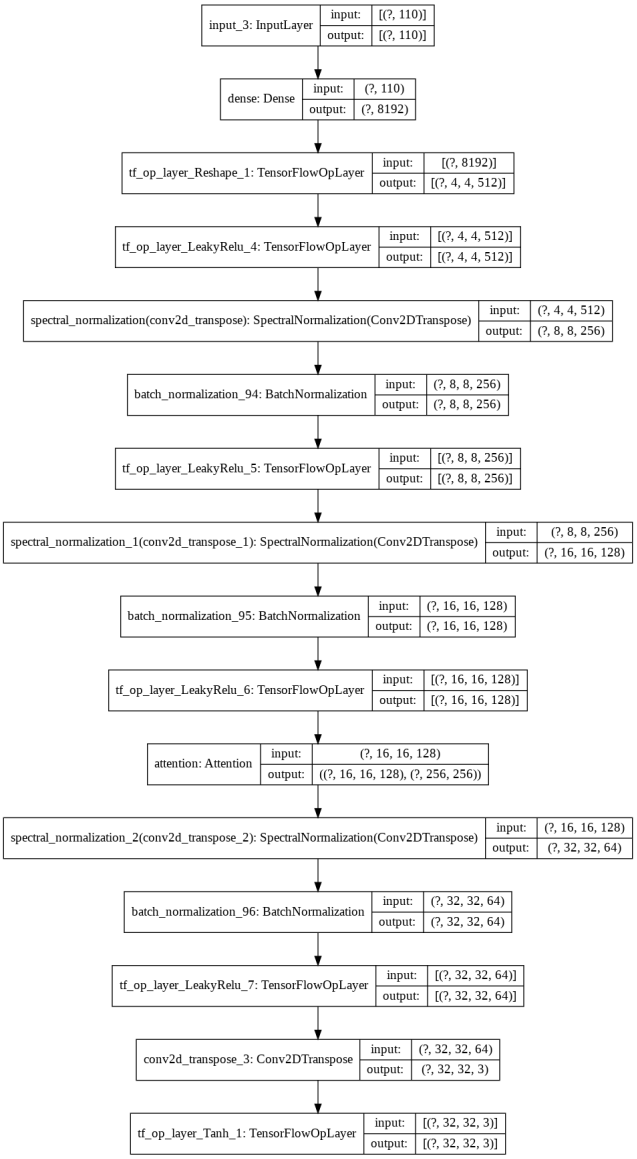
3. Architecture

3.1. Generator

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 110)]	0
dense_1 (Dense)	(None, 8192)	909312
tf_op_layer_Reshape (TensorF	[(None, 4, 4, 512)]	0
tf_op_layer_LeakyRelu (Tenso	[(None, 4, 4, 512)]	0
spectral_normalization_3 (Sp	(None, 8, 8, 256)	2101760
batch_normalization_97 (Batc	(None, 8, 8, 256)	1024
tf_op_layer_LeakyRelu_1 (Ten	[(None, 8, 8, 256)]	0
spectral_normalization_4 (Sp	(None, 16, 16, 128)	526592
batch_normalization_98 (Batc	(None, 16, 16, 128)	512
tf_op_layer_LeakyRelu_2 (Ten	[(None, 16, 16, 128)]	0
attention_1 (Attention)	((None, 16, 16, 128), (No	37153
spectral_normalization_5 (Sp	(None, 32, 32, 64)	132224
batch_normalization_99 (Batc	(None, 32, 32, 64)	256
tf_op_layer_LeakyRelu_3 (Ten	[(None, 32, 32, 64)]	0
conv2d_transpose_7 (Conv2DTr	(None, 32, 32, 3)	1728
tf_op_layer_Tanh (TensorFlow	[(None, 32, 32, 3)]	0

Total params: 3,710,561
Trainable params: 3,701,601
Non-trainable params: 8,960

(a)



(b)

Figure 11: Framework of Generator

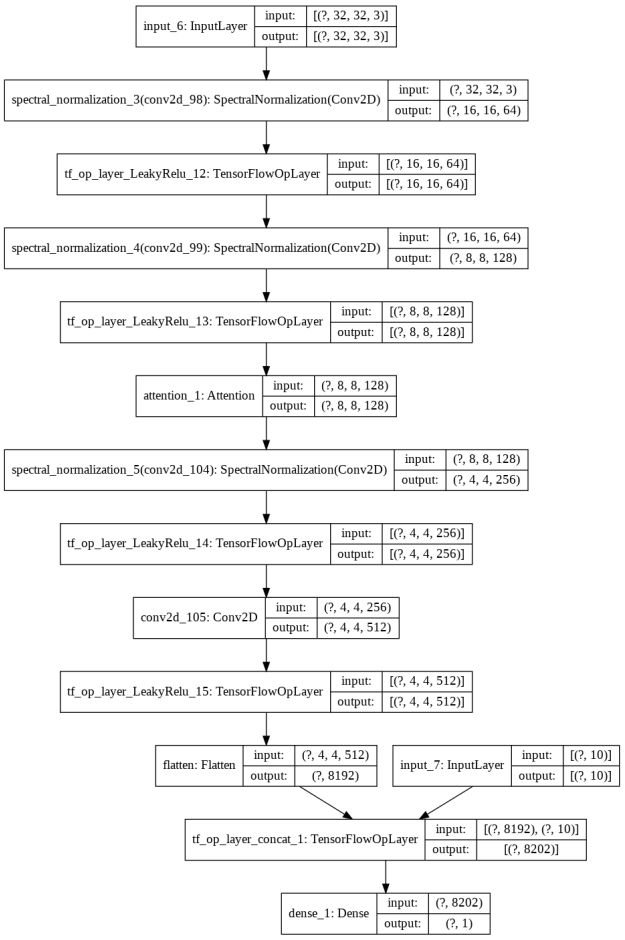
In the Generator, I apply dense, convolution transpose, batch normalization, attention, spectral normalization, leacky rulu and so on like Figure 11.

3.2. Discriminator

Layer (type)	Output Shape	Param #	Connected to
input_4 (InputLayer)	[(None, 32, 32, 3)]	0	
spectral_normalization_6 (Spect	(None, 16, 16, 64)	3248	input_4[0][0]
tf_op_layer_LeakyRelu_4 (Tensor	[(None, 16, 16, 64)]	0	spectral_normalization_6[0][0]
spectral_normalization_7 (Spect	(None, 8, 8, 128)	132352	tf_op_layer_LeakyRelu_4[0][0]
tf_op_layer_LeakyRelu_5 (Tensor	[(None, 8, 8, 128)]	0	spectral_normalization_7[0][0]
attention_2 (Attention)	(None, 8, 8, 128)	37153	tf_op_layer_LeakyRelu_5[0][0]
spectral_normalization_8 (Spect	(None, 4, 4, 256)	526848	attention_2[0][0]
tf_op_layer_LeakyRelu_6 (Tensor	[(None, 4, 4, 256)]	0	spectral_normalization_8[0][0]
conv2d_109 (Conv2D)	(None, 4, 4, 512)	1180160	tf_op_layer_LeakyRelu_6[0][0]
tf_op_layer_LeakyRelu_7 (Tensor	[(None, 4, 4, 512)]	0	conv2d_109[0][0]
flatten (Flatten)	(None, 8192)	0	tf_op_layer_LeakyRelu_7[0][0]
input_5 (InputLayer)	[(None, 10)]	0	
tf_op_layer_concat (TensorFlowO	[(None, 8202)]	0	flatten[0][0] input_5[0][0]
dense_2 (Dense)	(None, 1)	8203	tf_op_layer_concat[0][0]

Total params: 1,887,964
Trainable params: 1,884,396
Non-trainable params: 3,568

(a)

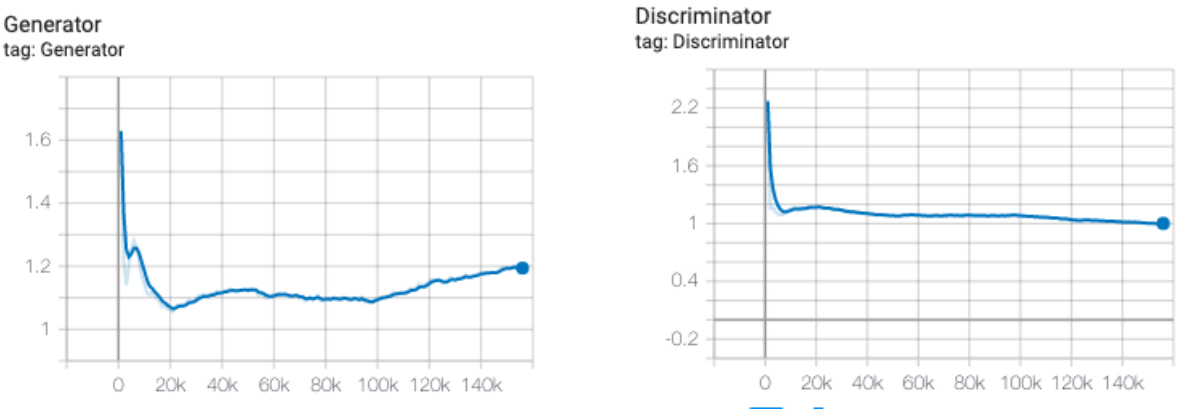


(b)

Figure 12: Framework of Discriminator

In the Discriminator, I apply dense, convolution, attention, spectral normalization, leacky rulu and so on like Figure 12.

4. Training



(a) generator loss

(b) discriminator loss

Figure 13: training loss

As we can see it from Figure 13, during the process of training, the discriminator loss keeps decreasing, but the generator loss first decreases, and then increases a little.

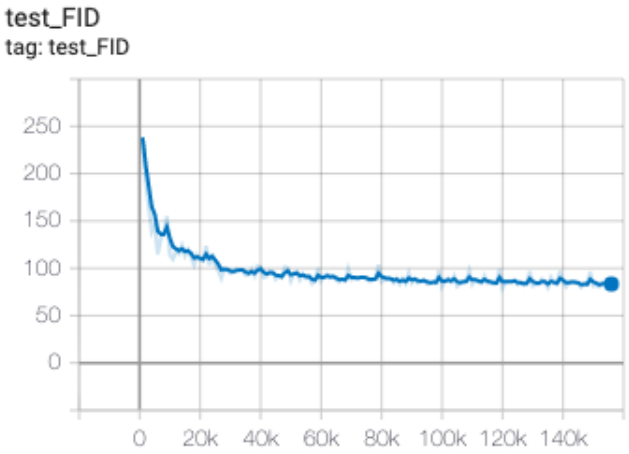
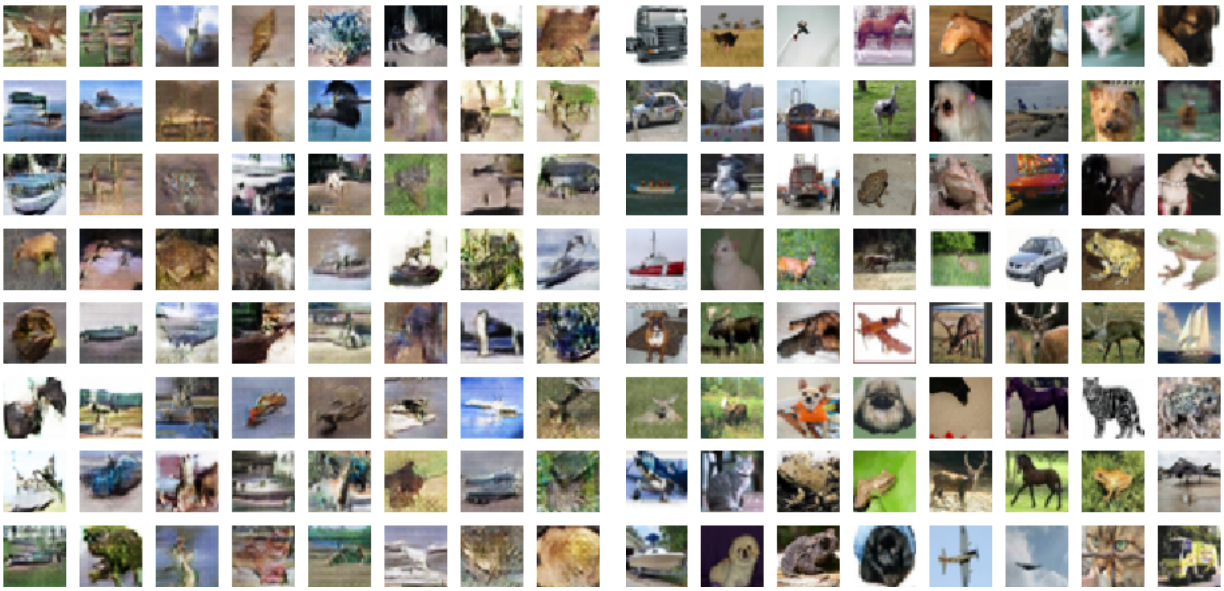


Figure 14: FID

As we can see it from Figure 14, during the process of training, the fid firstly decrease vary fast, but after some point(20k), it still decreases but at a very very slow rate.

5. Performance



(a) Generated img (b) Ground truth

Figure 15: Generated vs Ground-truth

As we can see it from Figure 14,the fid still decrease, but I don't have anytime, so we just choose the latest model as the best model(about 140k). So I restore the checkpoint, and use random vector (noise + label)to generate some image, like Figure 16, the quality of generated img is kind of ok when compared to the Ground truth.

And finally, the mean FID 88.95, which is calculated on test dataset(10k) and it's calculated with every 1024 test data and 1024 generated images, and calculate the mean of them, you can recover the result from my SAGAN.ipynb, it looks pretty high, but the FID between the test dataset and train dataset(every 1024) is about 50.

6. Visualization of Attention Maps

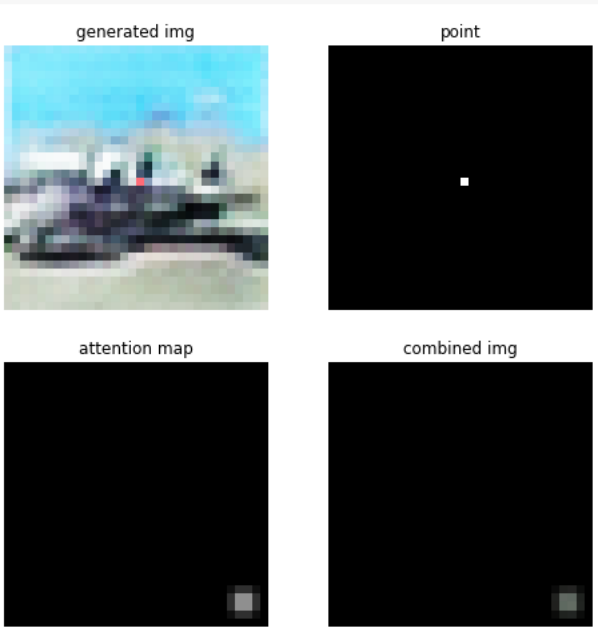


Figure 16: One point

Figure 16, we choose the middle point(16, 16), and check that point's attention map.

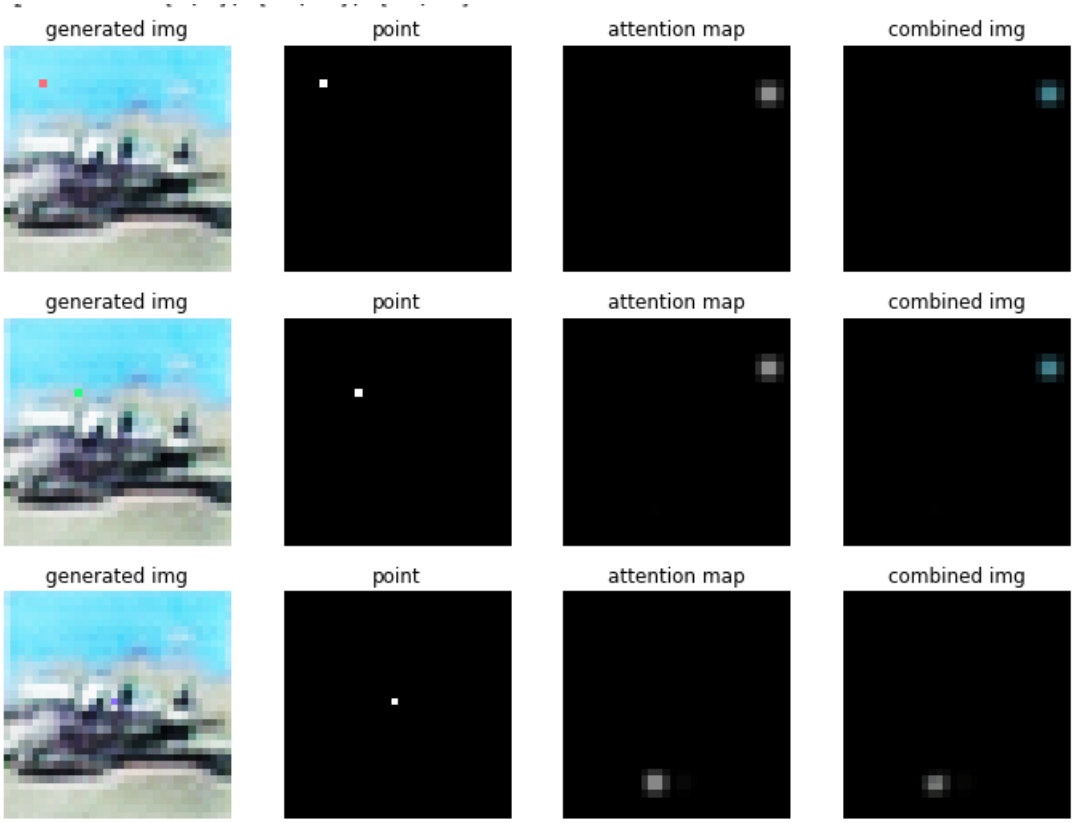
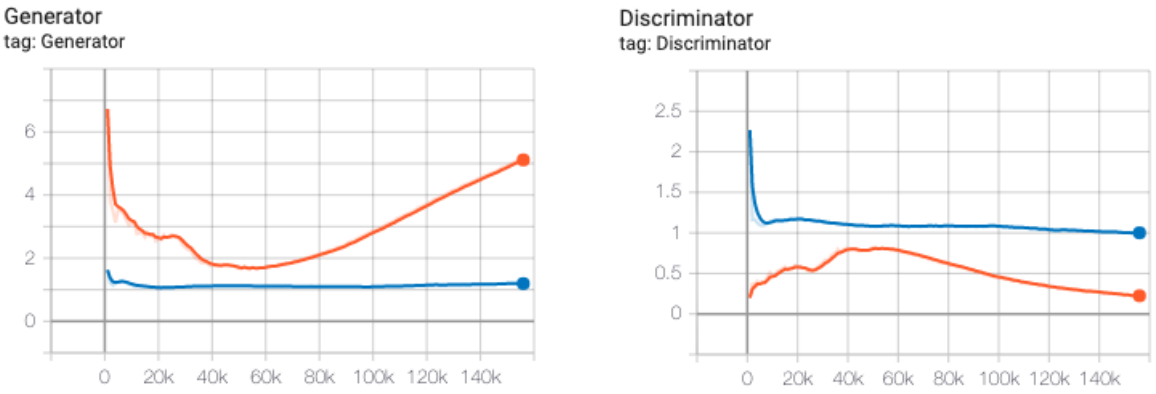


Figure 17: Three points

Figure 17, we choose 3 points((5, 5),(10,10),(15,15)) and check the attention map of that points.

7. Comparison

7.1. Training Loss



(a) generator loss

(b) discriminator loss

Figure 18: training loss

As we can see it from Figure 18, the orange line means DCGAN, and the blue line means SAGAN. during the process of training, SAGAN is more stable and robust than DCGAN in terms of generator loss and discriminator loss, since we apply Wasserstein Loss, Self-Attention, Spectral Normalizat and TTUR on SAGAN, not DCGAN.

7.2. Training FID

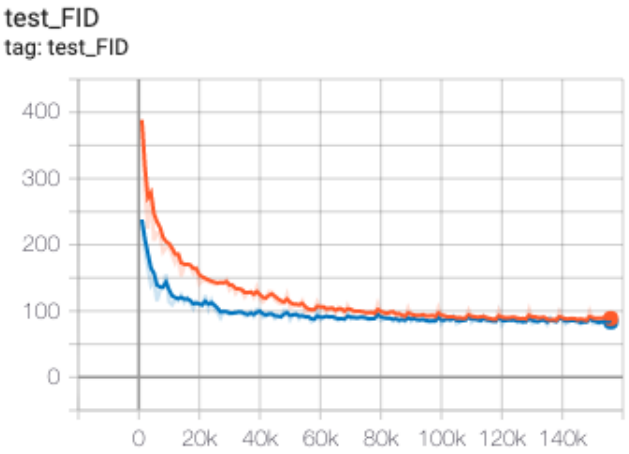
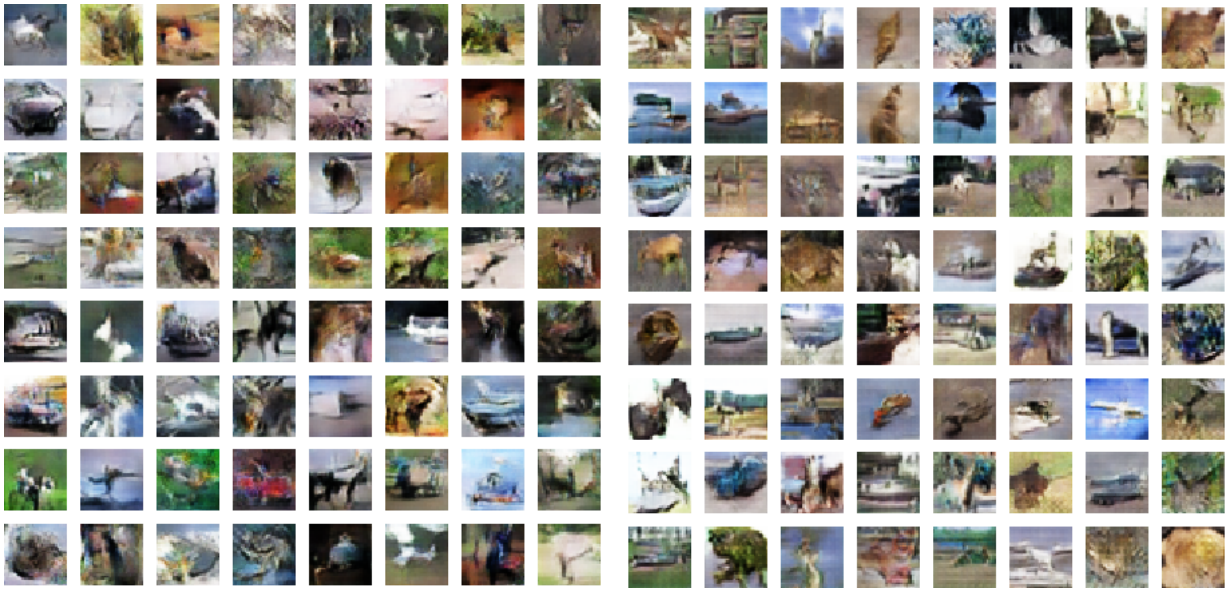


Figure 19: Training FID

As we can see it from Figure 19,the orange line means DCGAN, and the blue line means SAGAN. SAGAN goes down faster than DCGAN in the beginning, so it reach the best FID at about 25k, but DCGAN reach the best FID at about 80k. Both of them have similar best FID(about 90), I think the most reason for that is the training dataset's size is (32, 32, 3), which is very small.

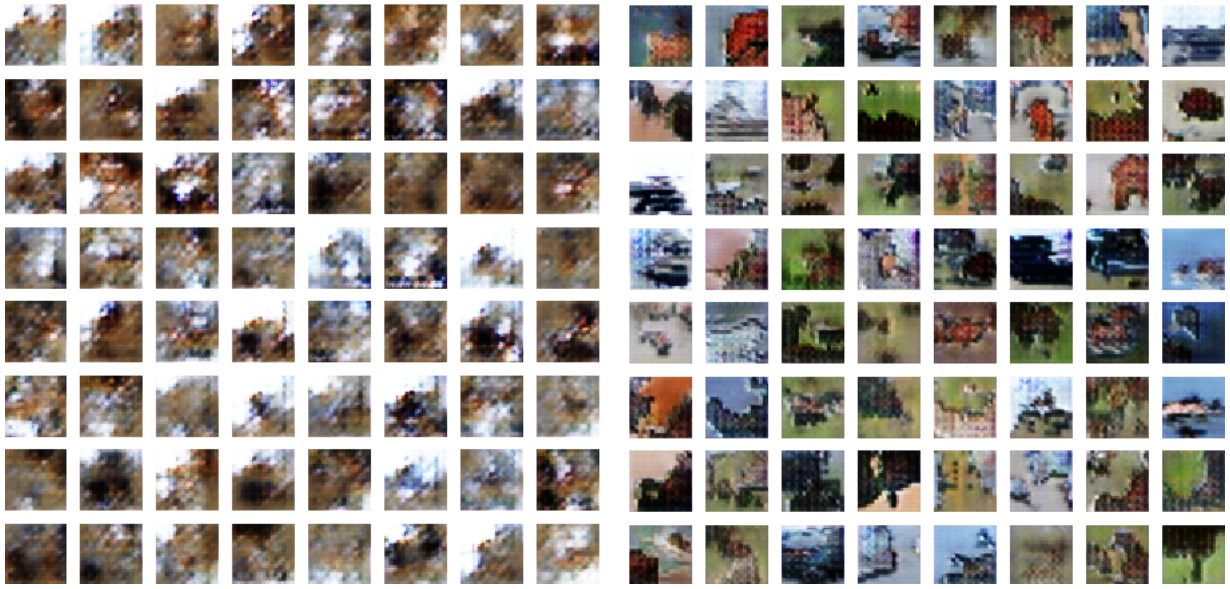
7.3. Generated image

As we can see it from Figure 19 and 20, the quality of final images generated by DCGAN and SAGAN are very similar. Belows are some images generated during training process, and we can see that SAGAN get a better performance than DCGAN during the early epoch.



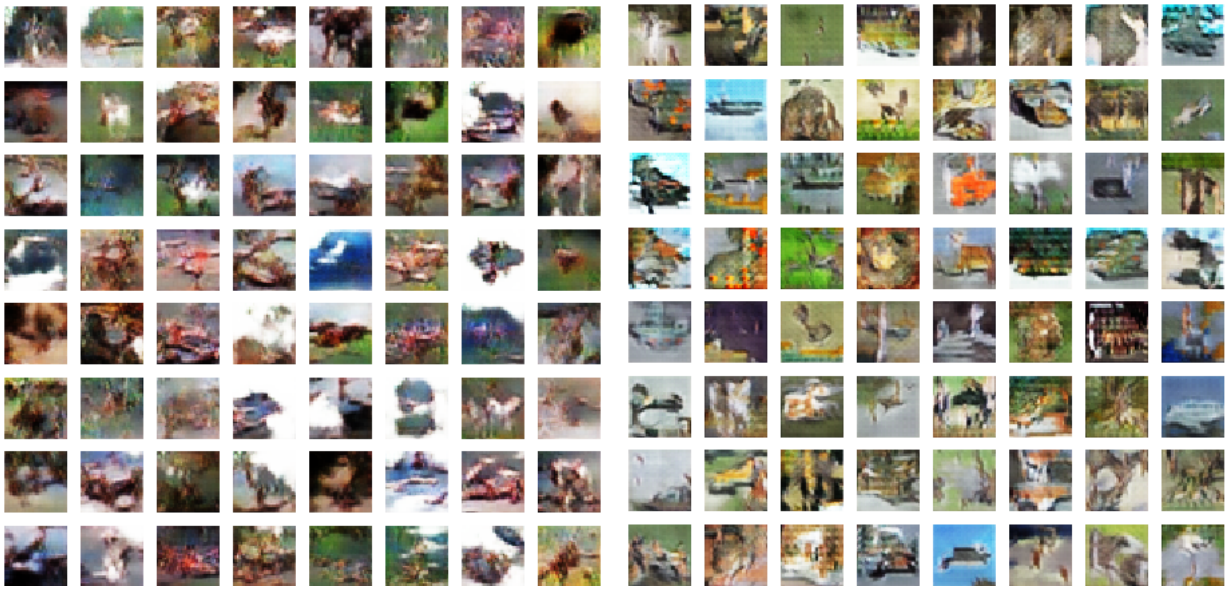
(a) DCGAN (b) SAGAN

Figure 20: Generated image



(a) DCGAN (b) SAGAN

Figure 21: Epoch 10



(a) DCGAN (b) SAGAN

Figure 22: Epoch 50

(a) DCGAN (b) SAGAN

Figure 23: Epoch 100

8. Reference

8.1. Code Reference

- GIF:https://www.tensorflow.org/tutorials/generative/dcgan
- FID:https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch
- Spectral Normalization:https://github.com/thisisiron/spectral$_n ormalization - tf2/blob/master/sn.py$

8.2. Paper Reference

- Denton, Emily, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep Generative Image Models Using a Laplacian Pyramid of Adversarial Networks. ArXiv:1506.05751 [Cs], June 18, 2015. http://arxiv.org/abs/1506.05751.

  Mirza, Mehdi, and Simon Osindero. Conditional Generative Adversarial Nets. ArXiv:1411.1784 [Cs, Stat], November 6, 2014. http://arxiv.org/abs/1411.1784.

- Hoogi, Assaf, Brian Wilcox, Yachee Gupta, and Daniel L. Rubin. Self-Attention Capsule Networks for Image Classification. ArXiv:1904.12483 [Cs], April 29, 2019. http://arxiv.org/abs/1904.12483.

- Arjovsky, Martin, Soumith Chintala, and Lon Bottou. Wasserstein GAN. ArXiv:1701.07875 [Cs, Stat], January 26, 2017. http://arxiv.org/abs/1701.07875.