Improved Multi-Domain Image-to-Image Translation GAN

Jeongik Cho

Abstract

StarGAN has shown excellent performance in image-to-image translation using adversarial, reconstruction, and classification losses in multi-domain image-to-image translation. The Style-Based Generator Architecture boosts generator performance through the Embedder and AdaIn modules. I propose here an attribute loss, which is like having multiple GANs, which is enhanced by combining StarGAN's conditional GAN loss (adversarial loss and classification loss) to improve learning speed. And suggest the new generator architecture, whose name is bi-directional progressive growing Style-Based U-Net generator, to improve learning speed.

1. Introduction

StarGAN[1] uses reconstruction loss of cycleGAN[2] and adversarial loss and classification loss, which are losses of conditional GAN[3].

$$L_D = -L_{adv} + \lambda_{cls} L_{cls}^{r}$$

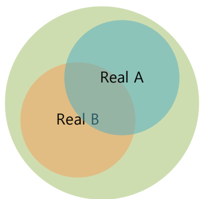
$$L_{cls}^{r} = E_{x,att\sim P_r(x,att)}[-\log\left(D_{cls}(\mathrm{att}|\mathrm{x})\right)]$$

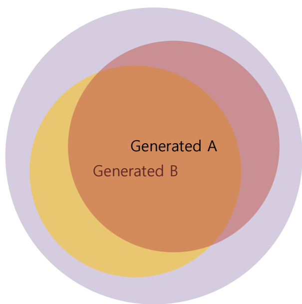
$$L_G = L_{adv} + \lambda_{cls} L_{cls}^{g}$$

$$L_{cls}^{g} = E_{x',att'\sim P_g(x',att')}\left[-\log\left(D_{cls}(att'|x')\right)\right]$$

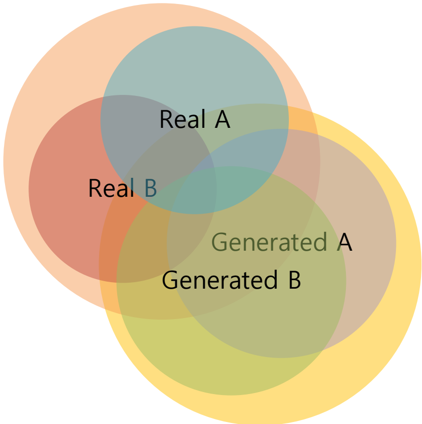
These are the losses of conditional GAN. In $x, att \sim P_r(x, att)$, x means real data, and att is the binary vector that expresses the attribute of real data. In $x', att' \sim P_g(x', att')$, x' means generated data and att' is the binary vector which is put in a generator to make x' .

In the star GAN, adversarial loss trains model well because there are well known the metrics such as LSGAN[4] or WGAN-GP[5] that can measure the distance between real data distribution and generated data distribution even if they are far from each other. However, classification loss of conditional GAN which is using cross-entropy is hard to be learned if the generated conditional data distribution is far from real conditional data distribution because cross-entropy trains the model to reduce KL-divergence only.



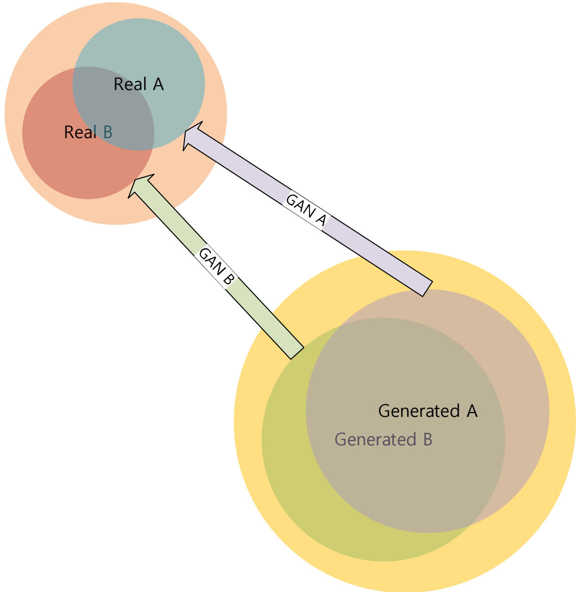


In the above figure, the circle containing Real A and Real B is the distribution of the real data, and the circle containing Generated A and Generated B is the distribution of the generated data. Real A is real data with attribute A and Generated A is data generated by the generator with condition A. In the early stage of learning, the classification loss does not have any meaning because the distance between the actual data distribution and the generated data distribution is far, so learning is conducted only as an adversarial loss.



As the learning progresses to some extent, the actual data distribution and the generated data distribution are somewhat similar, and classification loss starts to have meaning when each conditional data distributions overlap(Real A-Generated A, Real B-Generated B).

To solve the problem that classification loss does not have meaning at the beginning of learning, I made several GANs that learn only one attribute instead of conditional GAN losses. Each Generator only generates data with each attribute. Each Discriminator determines that it is true only for the real data with each attribute and that it is a fake for the data that the generator generates for each attribute.



Attribute loss is the sum of each GANs loss. Each GANs have their adversarial loss. So if you use LSGAN loss or WGAN-GP loss that can train models even if generated data distribution and real data distribution, the model can be trained well at the beginning of learning. Also, since each discriminator shares all layers except the output layer, and each generator shares all layers except the input layer, the learning time does not increase significantly.

I also used embedder and generator that is a simplified architecture of Few-Shot Adversarial Learning of Realistic Neural Talking Head Models[6] and U-Net architecture of Pix2Pix[7] and AdaIn module and embedder of Style-based generator[8]. I replaced all batch normalization layer to AdaIn module. To improve the learning speed, the generator grows in both input and output directions, not just in one direction like the style-based generator. Also, I used the activation functions of DCGAN[9].

2. Improved Star GAN

First, it is assumed that attribute information is matched with real data.

2.1 Loss

Overall Loss is as follows.

$$L_D = L_{att}^D$$

$$L_G = L_{att}^G + \gamma_{cnt} L_{cnt}$$

Attribute Loss

attribute loss is as follows.

$$L_{att}^D = \sum_c^{att} L_c^D$$

$$L_{att}^G = \sum_c^{att} L_c^G$$

$$L_c^D = E_{x,c\sim P_r(x,c)}[(D_c(x)-1)^2] + E_{x'\sim P_{G_c}(x',1)}[D_c(x')^2]$$

$$L_c^G = E_{x\sim P_r(x)}[(D_c(G_c(x,1))-1)^2]$$

c means one specific attribute among several attributes. L_c^D and L_c^G are the losses of one discriminator and one generator that discriminate against a particular attribute c. L_{att}^D is the sum of the attribute losses of all discriminators and L_{att}^G is the sum of the attribute losses of all generators.

G_c is a generator that converts an image x to have an attribute c when the image x and 1 are received as inputs. G_c tries to trick D_c only if 1 is entered with x, and does not care if 0 is entered(not learn).

D_c determines only about attribute c. D_c discriminates real only for real data with attribute c and doesn't care about real data without attribute c and determines fake when received the fake image from G_c that receives 1.

This is an example of using the least square loss as an adversarial loss, but you can use other losses such as Wasserstein-GP.

L_{att}^D is the sum of each discriminator. Each discriminator shares all layers with other discriminators except the output layer. Considering this aggregated discriminators as one discriminator, the loss can be changed like below.

$$L_{att}^D = E_{x,att\sim P_r(x,att)}\left[(D(x)-1)^2 \cdot att\right] + E_{x',att'\sim P_g(x',att')}[D(x')^2 \cdot att']$$

In $x, att \sim P_r(x, att)$, x is the real image, and att is attribute binary vector. '$\cdot$' means inner product.

In $x', att' \sim P_g(x', att')$, x' is generated image and att' is a binary attribute vector input generator to make x'.

Since each generator also shares all layers except the input layer, L_{att}^G can be written as the following by treating the grouped generators as one.

$$L_{att}^G = E_{x\sim P_r(x)}[\left(D(G(x,att'))-1\right)^2 \cdot att']$$

att' is a binary vector representing the attribute you want to change in the real image x. Use random binary vectors for training.

Incidentally, $G_c(x,0)$ does not convert x to x'

that doesn't have attribute c but simply disables G_c. Therefore, if you want to remove attribute c from image x, you need to add the attribute 'not c' while training.

Content Loss

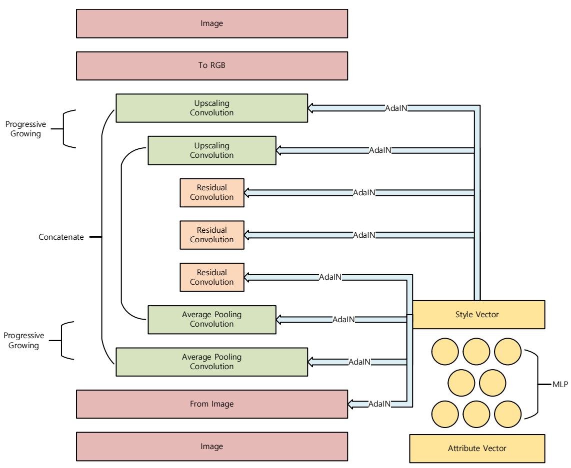
The content loss uses the loss of cycleGAN[2]. Use l1 loss if the architecture is not grown enough so the resolution of the image is too low to fit on the network, but if the architecture grows enough to fit on the network, use l1 loss of the output that is put on the network. Because I aimed for face attributes change, I used sliced pre-trained FaceNet[11]. This idea comes from Few-Shot Adversarial Learning of Realistic Neural Talking Head Models[6].

$$L_{cnt} = E_{x\sim P_{r,c}(x)}[||(G(G(x, c'), c)) - (x)||_1]$$

$$L_{cnt} = E_{x\sim P_{r,c}(x)}[||Net(G(G(x, c'), c)) - Net(x)||_1]$$

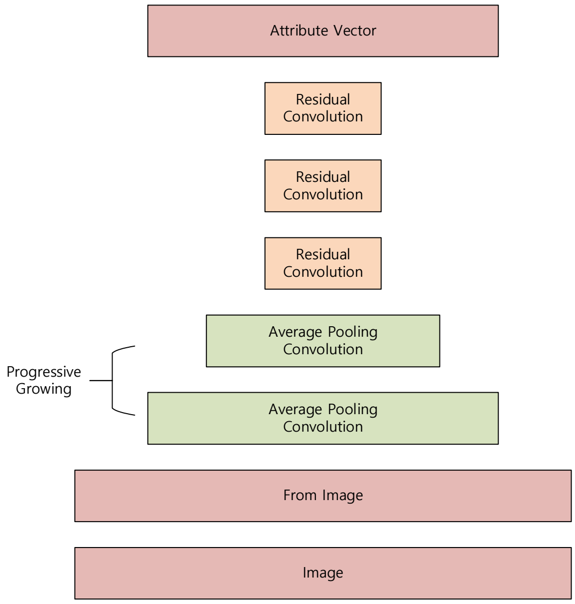
3.2 Architecture

Generator



I also used embedder and generator that is a simplified architecture of Few-Shot Adversarial Learning of Realistic Neural Talking Head Models[6] and U-Net architecture of Pix2Pix[7] and AdaIn module and embedder of Style-based generator[8]. To improve the learning speed, the generator grows in both input and output directions, not just in one direction like the style-based generator. Also, I used the activation functions of DCGAN.

Discriminator



Discriminator has attribute outputs that each output discriminates whether real image with each attribute or generated image with each attribute.

3. Experiments

3.1 Loss compare

I compared star GAN loss and attribute loss with reconstruction loss. I used the same architecture without the output layer. Star GAN uses cross-entropy, so activation function of Star GAN loss attribute output is sigmoid while attribute loss with reconstruction loss uses leaky Relu output. I used Adam optimizer with learning rate 0.00001 and beta1 0.5 and beta2 0.999. In Star GAN, Reconstruction loss weight is 10, and classification and adversarial loss weight are 1 that is recommending weights by the author of StarGAN. In attribute loss, reconstruction loss weight is 30 and attribute loss weight is 1. Dataset is Celeb A[10] and both models trained 24% of Celeb A dataset only once. I used resized image that resolution is 72 by 88, and the trained domain is 8 ('smiling', 'not smiling', 'black hair', 'not black hair', 'male', 'not male', 'young', 'not young'). It takes almost an hour on RTX2080ti.

Results

The left images are images generated by starGAN loss and the middle images are generated by attribute, reconstruction losses, and the right images are the original image. Input attribute vector was 'not smiling', 'not black hair', 'male', 'young'. The images were randomly picked.

In this case, the input attribute vector was 'smiling', 'not black hair', 'not male', 'young'

In this case, the input attribute vector was 'not smiling', 'black hair', 'not male', 'not young'

3.2 Progressive growing compare

I compared the bi-directional progressive growing model and non-progressive growing model. Both models trained approximately 1900sec on rtx2080ti (1901sec for the bi-directional progressive growing model, 1942sec for non-progressive growing model). The bi-directional progressive growing model learned 2% of celeb A dataset in resolution 18 by 22 with 150 sec, 4% in resolution 36 by 44 with 462sec, 8% in resolution 72 by 88 with 1289sec. The non-progressive model learned 12% of celeb A dataset in resolution 72 by 88 with 1942sec.

Results

Left pictures are results of the non-progressive growing model, middle pictures are a bi-directional progressive growing model, and the right pictures are original pictures.

References

[1] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, Jaegul Choo https://arxiv.org/abs/1711.09020

[2] Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros https://arxiv.org/abs/1703.10593

[3] Mehdi Mirza, Simon Osindero https://arxiv.org/abs/1411.1784

[4] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, Stephen Paul Smolley https://arxiv.org/abs/1611.04076

[5] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville https://arxiv.org/abs/1704.00028

[6] Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, Victor Lempitsky https://arxiv.org/abs/1905.08233

[7] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros https://arxiv.org/abs/1611.07004

[8] Tero Karras, Samuli Laine, Timo Aila https://arxiv.org/abs/1812.04948

[9] Alec Radford, Luke Metz, Soumith Chintala https://arxiv.org/abs/1511.06434

[10] Ziwei Liu Ping Luo Xiaogang Wang Xiaoou Tang http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[11] Florian Schroff, Dmitry Kalenichenko, James Philbin https://arxiv.org/abs/1503.03832