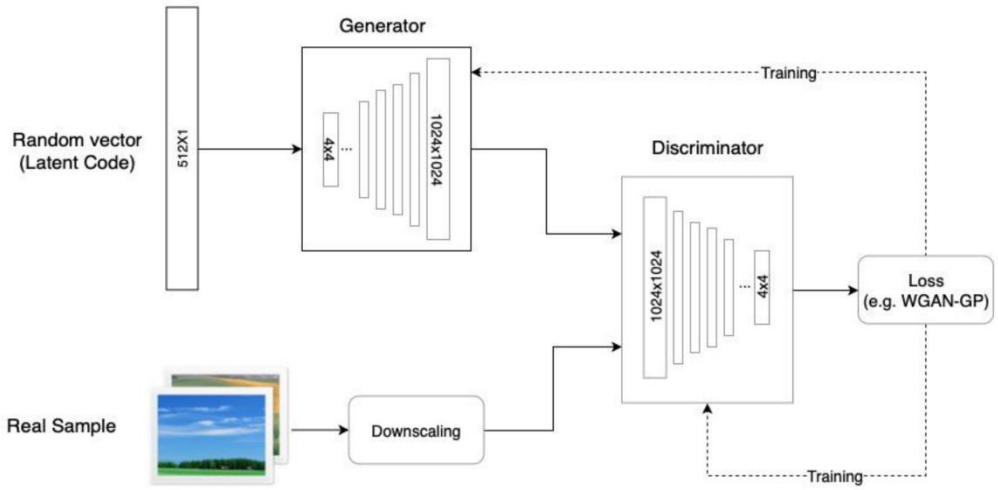
Style GAN Study Notes

1. What is Style GAN?

StyleGAN is a novel generative adversarial network (GAN) introduced by Nvidia researchers in December 2018, and made source available in February 2019. StyleGAN depends on Nvidia's CUDA software, GPUs and TensorFlow. [1] It's able to generate images that are of high reality and resolution. The second version of StyleGAN, called StyleGAN2, is published on 5 February 2020. It removes some of the characteristic artifacts and improves the image quality. [1]

2. The structure of Style GAN

2.1. ProGAN



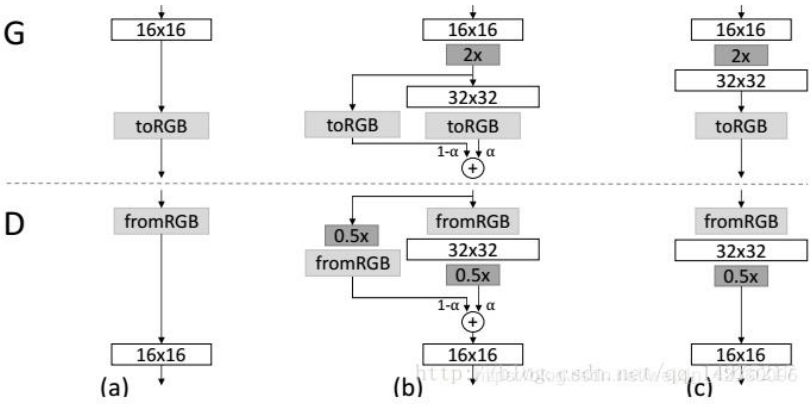
Style GAN could be viewed as an improvement based on ProGAN (Progressive Growing GAN). ProGAN model aims at training and generating the images at low resolution first, and then progressively add details and features to get images at higher and higher resolution. In this way, training the model would be more stable and saves much more time, compared with those traditional models.

Generator	Act.	Output shape	Params
Latent vector	–	512 × 1 × 1	–
Conv 4 × 4	LReLU	512 × 4 × 4	4.2M
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Upsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Upsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Upsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Upsample	–	512 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	1.2M
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Upsample	–	256 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	295k
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Upsample	–	128 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	74k
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Upsample	–	64 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	18k
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Upsample	–	32 × 1024 × 1024	–
Conv 3 × 3	LReLU	16 × 1024 × 1024	4.6k
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 1 × 1	linear	3 × 1024 × 1024	51
Total trainable parameters			**23.1M**

Discriminator	Act.	Output shape	Params
Input image	–	3 × 1024 × 1024	–
Conv 1 × 1	LReLU	16 × 1024 × 1024	64
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 3 × 3	LReLU	32 × 1024 × 1024	4.6k
Downsample	–	32 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Conv 3 × 3	LReLU	64 × 512 × 512	18k
Downsample	–	64 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Conv 3 × 3	LReLU	128 × 256 × 256	74k
Downsample	–	128 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Conv 3 × 3	LReLU	256 × 128 × 128	295k
Downsample	–	256 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Conv 3 × 3	LReLU	512 × 64 × 64	1.2M
Downsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Downsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Downsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Downsample	–	512 × 4 × 4	–
Minibatch stddev	–	513 × 4 × 4	–
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Conv 4 × 4	LReLU	512 × 1 × 1	4.2M
Fully-connected	linear	1 × 1 × 1	513
Total trainable parameters			**23.1M**

Table 2: Generator and discriminator that we use with CELEBA-HQ to generate 1024×1024 images.

Above is the structure of ProGAN. After we trained the model at each of the resolution level, we would add a layer that doubly upsamples the generator layer and downsapmles the discriminator layers to a half. To smoothly fade in the new layer, we would need additional parameter α for the increasement of the image's dimention.



Here 2x and 0.5x represents doubling and halving the resolution of the image via nearest neighbor filtering and average pooling respectively. α will grow linearly from 0 to 1, so that the old layer will progresively fade away and the new layer will then fade in. For example:

X = X16pixel * (1 −α) + X32pixel * α

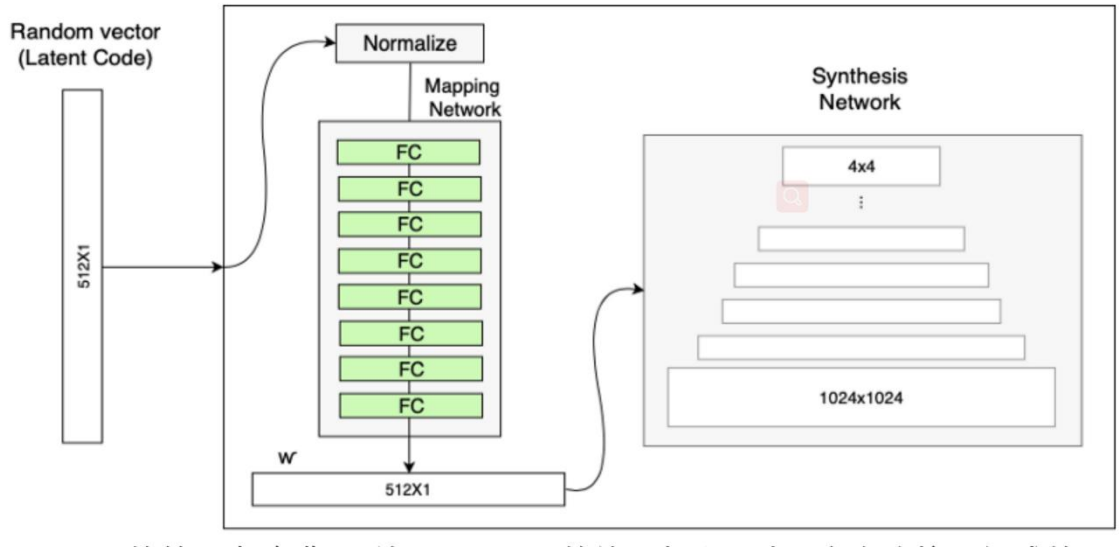
enables the resolution of the image grows from 16 * 16 to 32 * 32.

Although ProGAN effectively improve the stability and efficiency of traditional GAN model, there are still problems for such a model. We are unable to control what additional features the model learn in each of the new layer, as ProGAN progressively generate images between each layer and that the features are related to each other. To solve this shortcome, StyleGAN might work well. Based on the resolution, StyleGAN classifies the features at three levels:

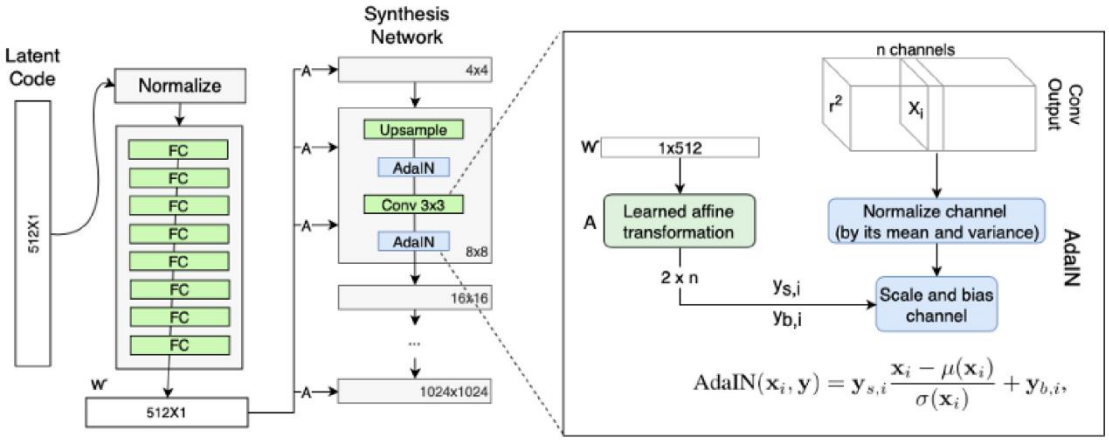
coarse spatial resolutions (4^2 – 8^2) brings high-level aspects such as pose, general hair style, face shape, eyeglasse, all colors (eyes, hair, lighting) and finer facial features;[2]
middle resolutions (16^2 – 32^2) inherit smaller scale facial features, hair style, eyes, while the pose, general face shape, and eyeglasses from A are preserved;[2]
the fine styles (64^2 – 1024^2) brings mainly the color scheme and microstructure.[2]

2.2. Mapping Network

To better control each of the feature of the image, we need to disentangle the latent factors of variation. With only latent factors as input, it's hard to disentangle the features as the latent factors must follow a certain distribution, which increases the correlation among certain features. Thus, StyleGAN contains a mapping network composed of 8 layers of MLP. Through neural network, the generator could get vectors that don't have to follow a certain distribution, AKA the intermediate latent space W, which disentangles the correlation among each feature.



2.3. AdaIN



After we got the intermediate latent space W, we'll need to pass it to AdaIN as the controller in the generator. Since we need to progressively train the images from 4*4 to 1024*1024, there are 9 stages in all. In each of the stage we need two AdaINs to impact on the output images, following each of the two convolution layers. Learned affine transformations then specialize w to styles y = (ys, yb) that control adaptive instance normalization (AdaIN) operations after each convolution layer of the synthesis network g:

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i}\frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$ [2]

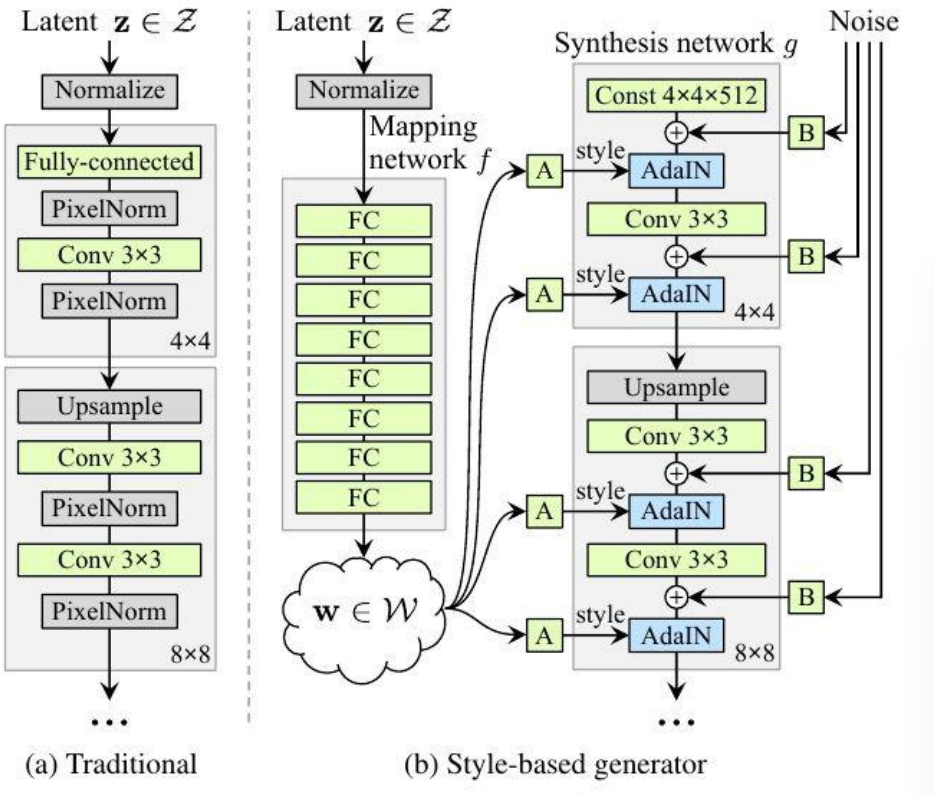
where each feature map xi is normalized separately, and then scaled and biased using the corresponding scalar components from style y. Thus the dimensionality of y is twice the number of feature maps on that layer. In this way, we are able to control the certain features through style y.

2.4. Removal of latent vector input

Because that now we rely on W and AdaIN to control the features of the image, we could omit the input layer altogether and starting from a learned constant instead. In this way, we could better avoid generating strange outputs due to the distribution of Z, and strengthen the disentanglement as well.

2.5. Stochastic variation

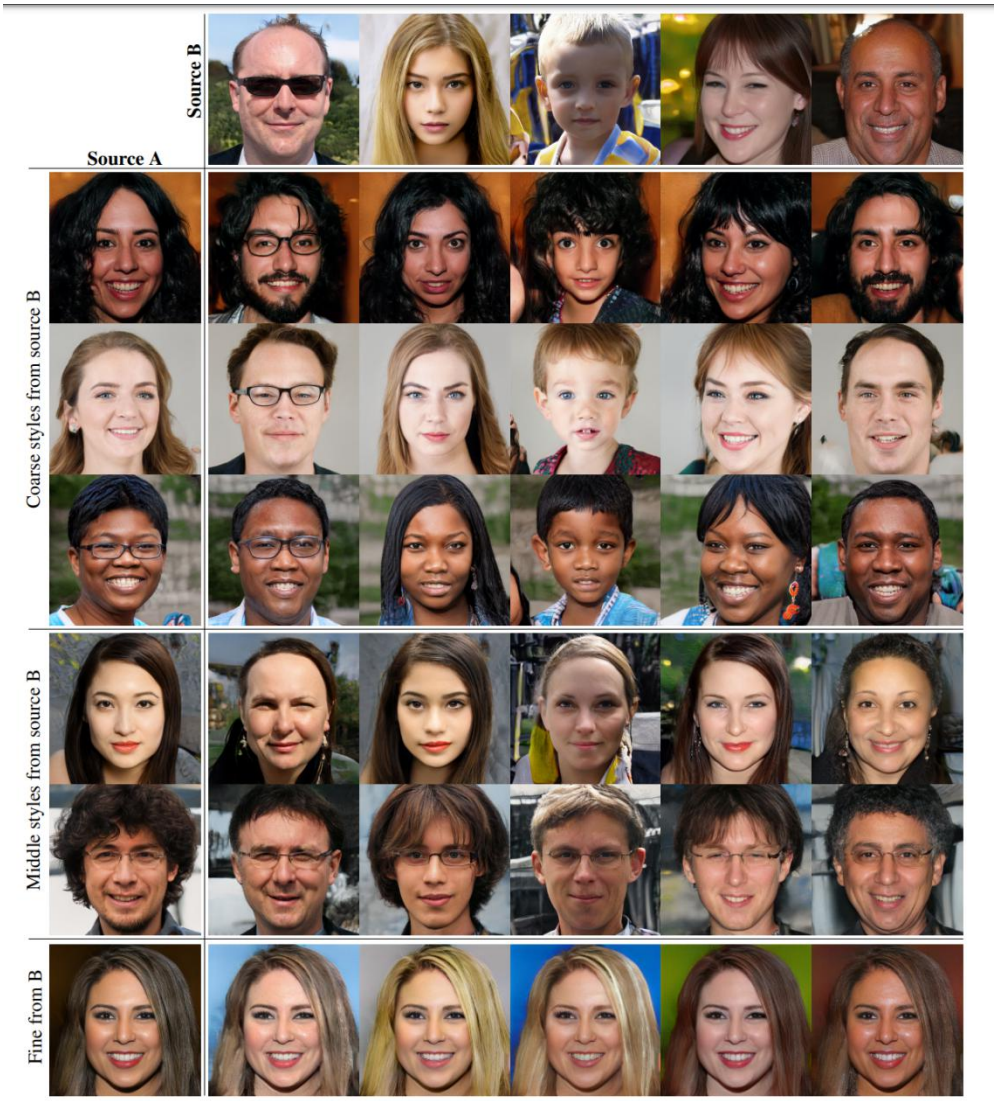
There are many aspects in human portraits that can be regarded as stochastic, such as the exact placement of hairs, stubble, freckles, or skin pores. Any of these can be randomized without affecting our perception of the image as long as they follow the correct distribution. Thus apart from the input layer, the networks need to introduce some of the spatially-varying pseudorandom numbers from early activations.



Thus we could see “B” applies learned per-channel scaling factors to the noise input.

2.6. Style Mixing

As AdaIN overwrites the former layers through normalization, the features of each layer is disentangled. To further weaken the correlation of the features among each of the resolution level, we employ mixing regularization, where a given percentage of images are generated using two random latent codes instead of one during training. When generating such an image, we simply switch from one latent code to another — an operation we refer to as style mixing — at a randomly selected point in the synthesis network. To be specific, we run two latent codes z1, z2 through the mapping network, and have the corresponding w1, w2 control the styles so that w1 applies before the crossover point and w2 after it.[2] This regularization technique prevents the network from assuming that adjacent styles are correlated.



Through this way, the generator combines the features that each of the two latent codes learned and mix them into another style, as shown above. The first column of the images are generated from source A latent code and the first row of the images are generated from the source B latent code. Different resolution of the features we copy from source B would be mixed accordingly to source A.

2.7. Truncation trick in W

In terms of the training data, there always exists the areas of low density which are poorly represented and thus likely to be difficult for the generator to learn.[2] Thus, to solve this kind of issues, we could choose to draw latent vectors from a truncated or otherwise shrunk sampling space which might tend to improve average image quality, although some amount of variation is lost.[2] To conduct the truncation, we scaled the deviation of a given w from the center as $w' = \bar{w} + \psi(w - \bar{w})$, where $\psi < 1$.[2] Here if we are generating the faces, $\bar{w}$ represents the average of the faces in the sample. For the StylrGAN model, truncation works well even if we don't change the loss function.

2.8. Hyperparameters and training details

StyleGAN inherits most of the training details from ProGAN, such as discriminator architecture, resolutiondependent minibatch sizes, Adam hyperparameters, and exponential moving average of the generator. There are also several modifications so in that case the model could improve the overall result quality. For example, the model replaces the nearest-neighbor up/downsampling in both networks with bilinear sampling. Also for the progressive growing the model starts from 8*8 instead of 4*4. For the FFHQ dataset, the model switches from WGAN-GP to the non-saturating loss with R1 regularization using γ = 10.[2]

3. References

[1] Wikipedia contributors. "StyleGAN." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 11 Aug. 2020. Web. 21 Aug. 2020.

[2] Tero Karras and Samuli Laine and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks, 2018. arXiv: 1812.04948v3 [cs.NE]

[3] a312863063. “Note_StyleGAN.” http://www.seeprettyface.com/research_notes.html, 21 Nov. 2019. Web. 9 Sep. 2020