Hi everyone, today my topic is *A Style-Based Generator Architecture for Generative Adversarial Networks.*

This paper has been accepted by CVPR 2019. Three authors all come from NVIDIA.

This is my content.

**Introduction**

First, let’s review about what is GAN, which stands for generative adversarial networks. GAN is based on a game theory. It trains two networks – one called generator and another called discriminators. The generator aims to generate images from a random noise and the discriminator aims to distinguish real images from generated images by producing a scalar. The larger the scalar is, the more likely the image is real. So during training, two networks do backpropgation and update based on each other and get stronger. The generator can generate more realistic images that the discriminator fails to distinguish. The discriminator also get stronger to classify the images.

Another GAN framework we should know is ProGAN, which stands for Progressive grow of GAN. If we set both networks to be normal CNNs, it is ok to generate images of small sizes like [32, 32]. But when it comes to high resolutions like [1024, 1024], the traditional CNN are NOT enough. Thus ProGAN provide a way to handle it. We first use the latent vector to generator a small image in size of [4, 4], and then use the [4, 4] images to generate a larger one like [8, 8] and [16, 16] and gradually increase the resolution up to 1024. Also, when it comes to discriminator, it reduce the resolution from 1024 all the way down to [4, 4] and produce a scalar. This architecture helps to generate a high resolution images and our StyleGAN is based on this ProGAN.

**Motivation**

As we can see, all generators will be fed with a latent vector. And it is proved that the latent vector is not only a random noise, it has some high level sematic interpretations. For example, in this picture, we do interpolation between two latent vectors. So as the input of generator smoothly changes, the output image, which is a bed, also changes from a no-window to having-window. So some value of the latent vector may control whether the image will contain a window or not. This is what we called high level sematic interpretations. So obviously we will come up with a question that apart from generate realistic high-resolution images, can we actually control the feature and style of the output image ?

The answer is that it will be quite challenging. The problem we will face is feature entanglement. Here, suppose we have a generator that produce fake human faces. If we want to change the hair color, then we use some method to locate that this value in the latent vector control the color hair. So we slightly change this value. But as a result, we may not only have the hair color change, but also other features like we may change the gender or the color of skin !

Also, this situation is that maybe we should modify several values to control one feature. Not to mention that we may get some unexpected change on other features. So this is feature entanglement, which means that several features are entangled in the same dimensions in the latent vector. And our StyleGAN is proposed to handle this problem,

StyleGAN will not only generate high resolution and realistic images, but also do feature disentanglement and thus provide superior control and understanding of the features of generated images

So now let’s get deeper inside into the architecture of StyleGAN

**Architecture Details**

First, as we can see in the title of the paper, a style-based **generator** architecture. So it does no modifications to the discriminator network. The discriminator is exactly the same as it is in ProGAN.

Mapping Network

So in generator, StyleGAN first put forward a Mapping Network. The mapping network will encode the latent vector into an intermediate vector and then input the intermediate vector into the generator instead of the latent vector. As an interpretation, the author said that the latent vector must follow the probability density of the training data thus causing feature entanglement. But the intermediate vector do not have to obey this rule so it can reduce the correlation between features.

In my opinion, this interpretation is not so convincing but this method really works.

AdaIN

Now, suppose we have a generator that generate human faces. If we want to paint a face with features like this. What would we do if we are going to paint by our own hand? We will first look at one of the features, say hey this guy has yellow skin, so we draw yellow skin and then look back to features, oh this guy has black hair, so we draw black hair. Actually we will draw a picture feature by feature, instead of looking at these hundreds of features only once and draw the picture immediately.

In other words, the source of features and information should be repeatedly injected into the generator. However, in traditional GANs, once we feed the latent vector into the generator as an input, it is never used again.

The StyleGAN fixes this problem by doing exactly what we expected — it allows the intermediate vector produced by mapping network to be injected into the generator every layer. The generator can keep referring back to the “style guide” in the same way that we paint our own images.

Technically speaking, how can we inject the feature information into the generator every layer? The answer is Adaptive Instance Normalization.

Here, f(w) represents a learned affine transformation, corresponding to the A in the picture. xi is an instance that we are applying AdaIN to, and y is a set of two scalars (ys,yb) that control the “style“ of the generated image.

This might seem very familiar to Batch Normalization. One key difference is that the mean and variance are computed per-channel and per-sample here, rather than for an entire mini batch.

By doing this, the generator can refer to the feature guide whenever it wants to, thus generating a more realistic image.

Constant input of the generator-

As we will input the feature guide w into every layer of the generator as sort of additional information. What about the original input of the generator? StyleGAN set it to be a constant that can be learned during the training.

This modification also has some intuitive interpretations. Because every kind of pictures will have the same skeleton. For example, all human faces have a round face, two eyes, one nose, one mouth. These are constant skeleton and the feature guide only modify the style of them. So the constant input represents the skeleton of the image dataset.

Stochastic variation

Variation is quite important in image generation. By adding variations, we can get several slightly different images based on the same input values. Like, we will have the same person with deferent hair style or wrinkles.

In the traditional GAN, we will apply variations like this. Injecting the variation term into the latent vector only once and then put them into the generator. But as we can see, different features needs different variations, so using the Adaptive Instance Normalization, in StyleGAN, we also inject variation information every layer. Here B is a learned per-channel scale.

By adding stochastic variation every layer, we can thus generate more realistic images.

Style Mixing

As we can see, the feature vector is independently injected into the generator and is independently handled by different modules. So if we inject different feature vectors into different position, we may mix the features in different vectors and generate a style mixed human face.

For example, we inject three feature vectors, and the generator will respectively pick three features and combine them to one. This is a useful way to control the output style. And we will see the real experiment result in the next part.

**Evaluation and result**

Evaluate image qualities: FID

First, we will evaluate the qualities of the generated images, which is the common goal for generative models. FID is a commonly used metric to evaluate the output image from the generative models. The smaller the FID is, the better the model is.

All the models are trained on two datasets. One called CelebA-HQ, another called FFHQ. As we can see, compared to the baseline ProGAN, the modified models are much more better. The state of art model is E and F.

Evaluate feature disentanglement

The paper comes up with two brand new methods to study the feature disentanglement.

One is perceptual path length. Suppose we are doing interpolation between latent vectors. If there are severe feature entanglement, the intermediate images will change dramatically. Because drastic changes mean that multiple features have changed together and that they might be entangled. Following this intuition, we subdivide a latent vector interpolation path into linear segments, and we can define the total perceptual difference from the segmented path.

Here, z1 z2 are two latent vector and f is the mapping network that transform z1z2 into intermediate vectors. And lerp here represents linear interpolation. Function g is a slide segment image during the interpolation. And d is short for distance which measures the difference between two images.

In the result table, we can see that by adding mapping network, we will significantly reduce the path length compared to the traditional generator. But when we applying style mixing, the path length again increases. This is because the style mixing will contribute to the feature entanglement.

Another criterion is called linear separiablity.