Notes: “Analyzing and Improving the Image Quality of StyleGAN”, AKA StyleGAN2

Sun

May 25, 2020

1 Notes:

1.1 Summary:

1. The authors note that in almost, if not every single, image generated by StyleGAN, there are artifacts.
2. They propose a fix for this, and make the generated images better.
3. They flex a cool metric.

1.2 Low level:

No high-level overview due to the fact I believe this paper mostly contains low-level optimization and modifications to get scores better and resolve some minor issues.

The authors note, then blame AdaIN for, the existance of “water-droplet”-esque artificats (see figure (1)). The justification they provides was that, since AdaIN normalizes each feature map completely independent of the other (aside from implicit scaling and biasing from the learned transformation), there are large amounts of spatial information destroyed. The generator doesn’t want this; at its’ core, it still is a style-transfer network, and spatial information (for images) is extremely important. As such, the generator instead creates powerful local spikes of activations, which, when naively normalized by AdaIN, stay the strongest activations in the layer.

NOTE: With figure 2, you might notice the modulation and normalization of AdaIN appear to be seperated by a conv+noise block. I checked the code, and couldn’t find this anywhere. I believe this is actually a seperate type of normalization (batchnorm maybe?) that they did not mention in the paper.

1.2.1 New normalization:

Regardless, look at Figre 2c to get an overview of StyleGAN2. We now only modulated the standard deviation, not the mean, in addition to structuring the network slightly differently.

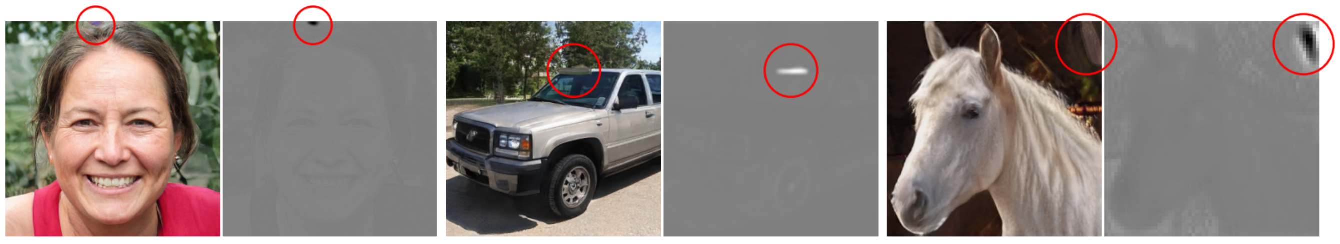

Figure 1. Instance normalization causes water droplet -like artifacts in StyleGAN images. These are not always obvious in the generated images, but if we look at the activations inside the generator network, the problem is always there, in all feature maps starting from the 64x64 resolution. It is a systemic problem that plagues all StyleGAN images.

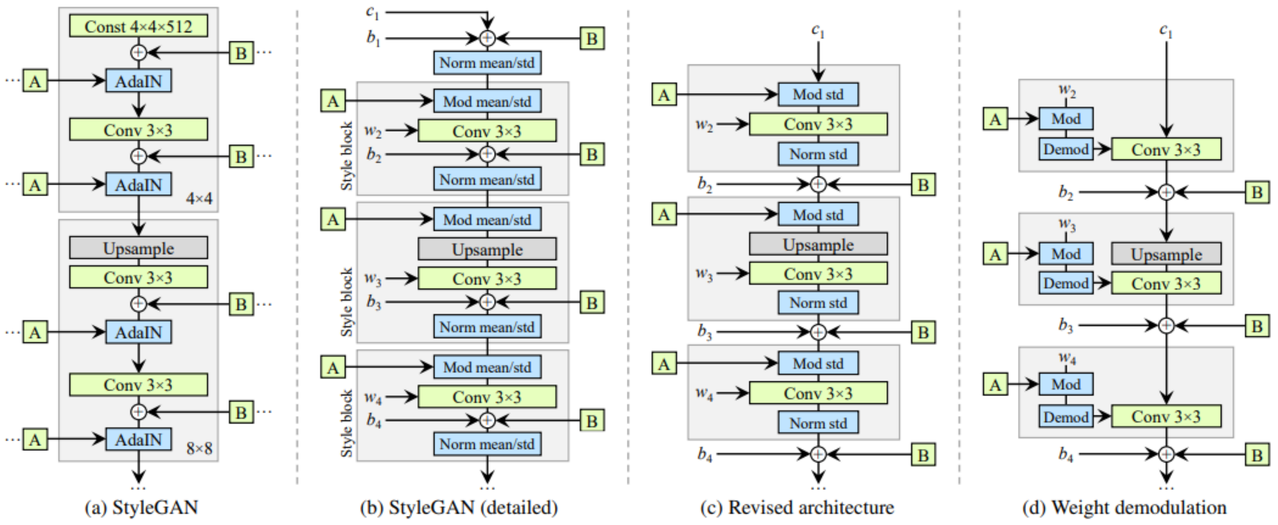

Figure 2. We redesign the architecture of the StyleGAN synthesis network. (a) The original StyleGAN, where $\boxed{A}$ denotes a learned affine transform from $\mathcal{W}$ that produces a style and $\boxed{B}$ is a noise broadcast operation. (b) The same diagram with full detail. Here we have broken the AdaIN to explicit normalization followed by modulation, both operating on the mean and standard deviation per feature map. We have also annotated the learned weights (w), biases (b), and constant input (c), and redrawn the gray boxes so that one style is active per box. The activation function (leaky ReLU) is always applied right after adding the bias. (c) We make several changes to the original architecture that are justified in the main text. We remove some redundant operations at the beginning, move the addition of b and $\boxed{B}$ to be outside active area of a style, and adjust only the standard deviation per feature map. (d) The revised architecture enables us to replace instance normalization with a "demodulation" operation, which we apply to the weights associated with each convolution layer.

Consider the effects of modulation followed by a convolution: The modulation itself only scales each feature map by a constant, which, instead, could be internalized by the weights of the layer itself:

$$w^{'}_{ijk} = s_i \cdot w_{ijk} \tag{1}$$

Where i denotes the input feature map, j denotes the output feature maps, k denotes spatial features of the feature map, and s_i is a scalar.

The purpose of instance normalization (seen in figure 2c) is to remove the effect of s from the statistics of the output feature maps. If we instead assume that input activations are random variables with unit std, the output activations have std of:

$$\sigma_j = \sqrt{\sum_{i,k} {w^{'}_{ijk}}^2} \tag{2}$$

which corresponds to the outputs being scaled by the L_2 norm of the weights.

We can once again instead internalize this:

$$w^{''}_{ijk} = w^{'}_{ijk} \Big/ \sqrt{\sum_{i,k} {w^{'}_{ijk}}^2 + \epsilon} \tag{3}$$

where ϵ is to avoid numerical instability.

We now have the entire style block within a single convolution layer: simply learn an affine transformation to project a given style, y, onto the correct dimension of s, then input that into eq. (1) and (3).

This change removes the droplet artifacts seen in (1), in addition to keeping FID score (a GAN performance metric) the same.

1.2.2 More regularization:

The authors note that a fixed-step size in $\mathcal{W} \sim f(\mathbf{z})$ should result in a non-zero, fixed-magnitude change in the end resulting image ($g(\mathbf{w} \in \mathcal{W})$); empirically, we can measure

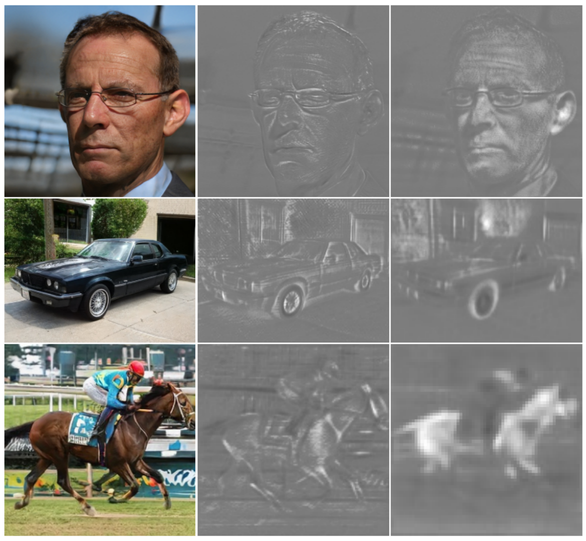
Figure 3: Replacing normalization (right) with demodulation (left) removes artifacts from images and activations.

this by stepping in random directions in the image space, then observing the $\mathbf{w}$ gradients.

But what does this actually mean? We want small steps in input to lead to small steps in output; the input being a given style, $\mathcal{W}$, the output being an image, $\mathcal{Y}$. Put more abstractly, for the function $f(x)$, we want a $f(x+\epsilon) - f(x) \approx a$. In matrices, consider the problem $f(x) := Ax$. The magnitude of change for a given small step can be represented by $||A(\mathbf{x}+\epsilon) - A(\mathbf{x})||_2$, therefore, for all steps ϵ of roughly the same size, we want $\left[||A(\mathbf{x}+\epsilon) - A(\mathbf{x})||_2 - a\right]^2$ to be roughly constant. However, we don't actually need this $\mathbf{x}$, as the transformation is linear, and it instead can be written as:

$$\left[||A(\epsilon))||_2 - a\right]^2 \tag{3}$$

The authors formulate the regularization mathematically as:

$$\mathbb{E}_{\mathbf{w},\mathbf{y}\sim\mathcal{N}(0,1)}\left(||\mathbf{J}_{\mathbf{W}}^{\mathrm{T}}\mathbf{y}||_2 - a\right)^2 \tag{4}$$

with $\mathbf{J_W} = \partial g(\mathbf{w})/\partial \mathbf{w}$, where $g(\cdot)$ is the generator. Intuitively, the random "image" $\mathbf{y}$ corresponds to a random step of roughly constant magnitude (the ϵ from (3)). In the case where $g(\cdot)$ is a linear transformation, eq. (4) is reduced to eq. (3). We obviously are not dealing with linear functions, however, we can approximate the function linearly using the Jacobian well, assuming small steps (small $\mathbf{y}$). We use a random style, $\mathbf{w} \in \mathcal{W}$ as the point to linearize the Jacobian, where after, it is reduced to eq. (3). The authors optimize this using standard backpropogation, putting it in the form $\mathbf{J_W} = \nabla_{\mathbf{W}}(g(\mathbf{w}) \cdot \mathbf{y})$.

This regularization term "smooths" the latent space by incentivizing the model to produce small changes in the image for small changes in style. It also empirically produces easier to optimize and better behaved models.

0.0.1 Lazy regularization:

The authors note that the regularization terms can be calculated every now and then, rather then every optimization step, without degradation of performance.

0.0.2 Replacement for progressive growing:

Previously, StyleGAN used progressive growing (a method of slowly increasing the size of the network, for stability in training at high resolutions), however, there are cons to

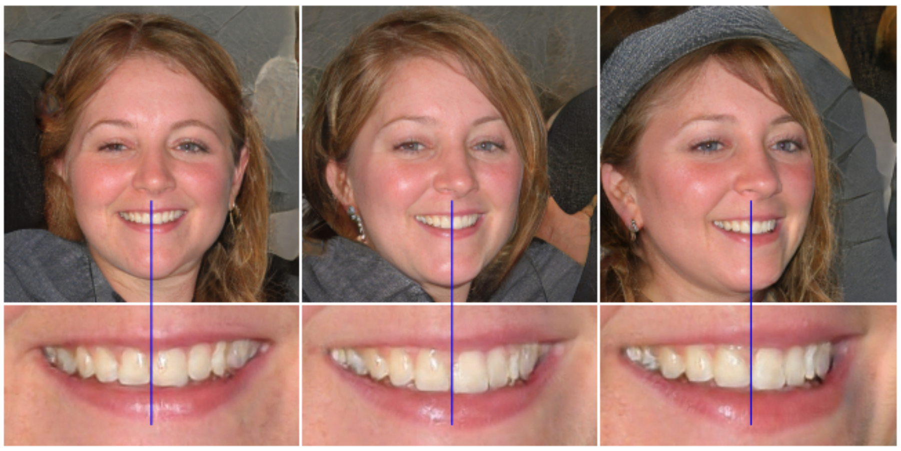

Figure 4: Progressige growing leads to “phase” artifacts; common features stay in a single state until it is absolutely necessary for them to change

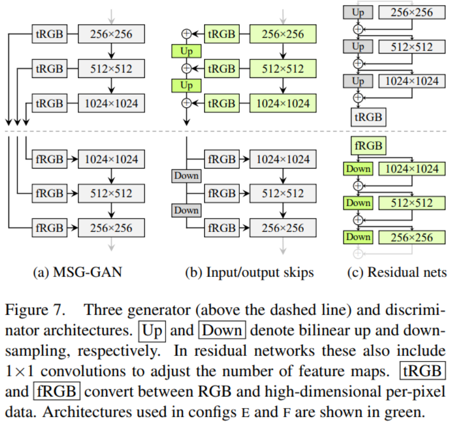

Figure 7. Three generator (above the dashed line) and discriminator architectures. Up and Down denote bilinear up and downsampling, respectively. In residual networks these also include 1×1 convolutions to adjust the number of feature maps. tRGB and fRGB convert between RGB and high-dimensional per-pixel data. Architectures used in configs E and F are shown in green.

Figure 5: Three generator/discriminator architectures. Up/down denote bilinear up/downsampling. tRGB and fRGB convert between RGB and high dimensional data. This (tRGB/fRGB) wasn’t explained anywhere else but in this graphic, and as far as I can tell, wasn’t explained in any citation either.

this; namely, “phase” artifacts seen in figure 4.

Instead of using progressive growing, StyleGAN uses a mix of a sparsely-connected-skip-connection generator, with a traditional residual discriminator; this is explained primarily in figure (5). Intuitively, by letting the generator use these sparse connections, the generator is incentized to focus on lower resolution images first, then the high resolution ones, by nature of the upsampling.

0.1 Conclusion:

This paper was primarily hacks, however, a few of them were cool. The new architecture that replaced progressive growing was smart, as was the justification for moving away from AdaIN and the latent space smoothing.

The results from StyleGAN2 are much better than StyleGAN, and, as far as I know, still is SOTA for natural high-resolution images.