A Style-Based Generator Architecture for Generative Adversarial Networks

Liana Isayan

Computer Vision, final presentation

A Style-Based Generator Architecture for Generative Adversarial Networks

Tero Karras
NVIDIA
tkarras@nvidia.com

Samuli Laine
NVIDIA
slaine@nvidia.com

Timo Aila
NVIDIA
taila@nvidia.com

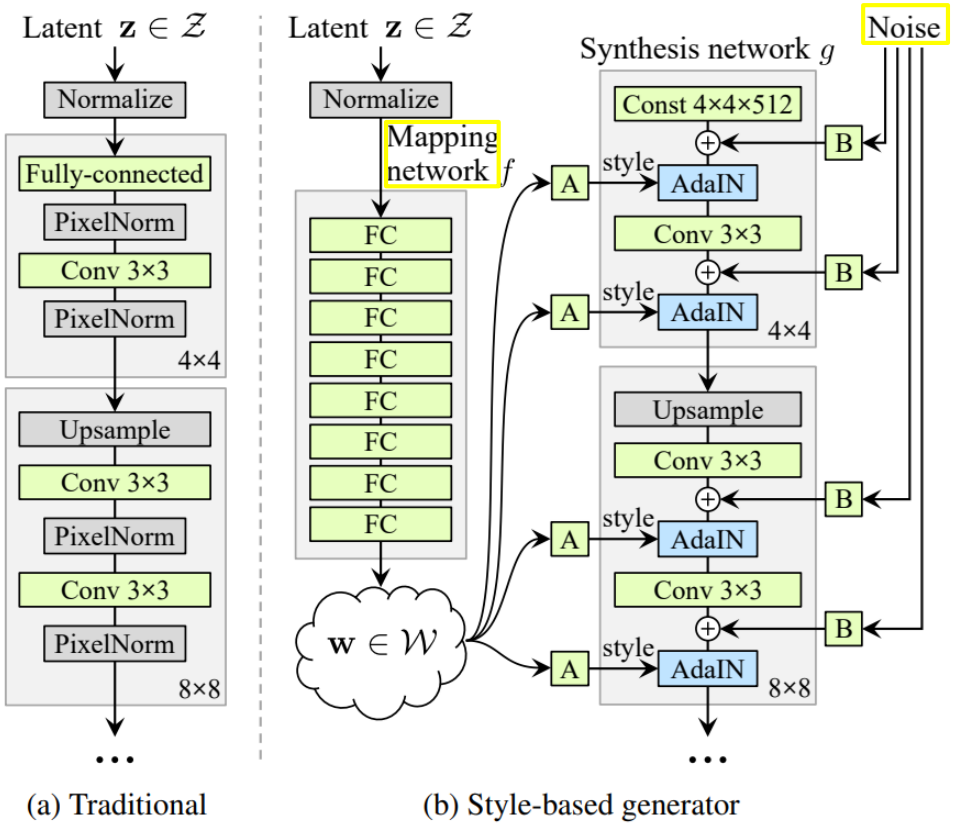
Abstract

We propose an alternative generator architecture for generative adversarial networks, borrowing from style transfer literature. The new architecture leads to an automatically learned, unsupervised separation of high-level attributes (e.g., pose and identity when trained on human faces) and stochastic variation in the generated images (e.g., freckles, hair), and it enables intuitive, scale-specific control of the synthesis. The new generator improves the state-of-the-art in terms of traditional distribution quality metrics, leads to demonstrably better interpolation properties, and also better disentangles the latent factors of variation. To quantify interpolation quality and disentanglement, we propose two new, automated methods that are applicable to any generator architecture. Finally, we introduce a new, highly varied and high-quality dataset of human faces.

(e.g., pose, identity) from stochastic variation (e.g., freckles, hair) in the generated images, and enables intuitive scale-specific mixing and interpolation operations. We do not modify the discriminator or the loss function in any way, and our work is thus orthogonal to the ongoing discussion about GAN loss functions, regularization, and hyperparameters [24, 45, 5, 40, 44, 36].

Our generator embeds the input latent code into an intermediate latent space, which has a profound effect on how the factors of variation are represented in the network. The input latent space must follow the probability density of the training data, and we argue that this leads to some degree of unavoidable entanglement. Our intermediate latent space is free from that restriction and is therefore allowed to be disentangled. As previous methods for estimating the degree of latent space disentanglement are not directly applicable in our case, we propose two new automated metrics —

Style-based generator



(a) Traditional (b) Style-based generator

Figure 1. While a traditional generator [30] feeds the latent code though the input layer only, we first map the input to an intermediate latent space $\mathcal{W}$, which then controls the generator through adaptive instance normalization (AdaIN) at each convolution layer. Gaussian noise is added after each convolution, before evaluating the nonlinearity. Here "A" stands for a learned affine transform, and "B" applies learned per-channel scaling factors to the noise input. The mapping network f consists of 8 layers and the synthesis network g consists of 18 layers — two for each resolution ($4^2 - 1024^2$). The output of the last layer is converted to RGB using a separate 1×1 convolution, similar to Karras et al. [30]. Our generator has a total of 26.2M trainable parameters, compared to 23.1M in the traditional generator.

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

where each feature map $\mathbf{x}_i$ is normalized separately, and then scaled and biased using the corresponding scalar components from style $\mathbf{y}$. Thus the dimensionality of $\mathbf{y}$ is twice the number of feature maps on that layer.

Properties of the style-based generator

- **Style mixing**
  Increases the diversity that the model sees during the training
- **Stochastic variation**
  Stochastic noise causes small variations to the output
- **Separation of global effects from stochasticity**
  The network learns to use the global and local channels appropriately, without explicit guidance

Disentanglement studies

- **Perceptual path length**

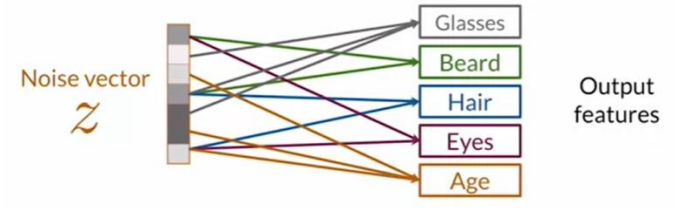
  Intermediate latent space is perceptually more linear than the input latent space

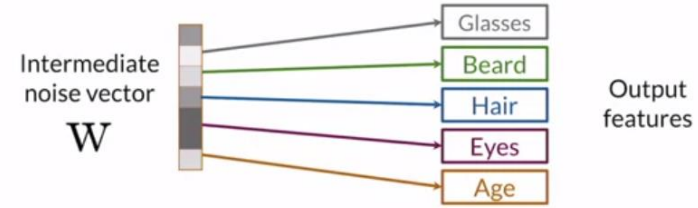
$$l_{\mathcal{Z}} = \mathbb{E}\left[\frac{1}{\epsilon^2} d\big(G(\text{slerp}(\mathbf{z}_1, \mathbf{z}_2; t)), G(\text{slerp}(\mathbf{z}_1, \mathbf{z}_2; t+\epsilon))\big)\right]$$

$$l_{\mathcal{W}} = \mathbb{E}\left[\frac{1}{\epsilon^2} d\big(g(\text{lerp}(f(\mathbf{z}_1), f(\mathbf{z}_2); t)), g(\text{lerp}(f(\mathbf{z}_1), f(\mathbf{z}_2); t+\epsilon))\big)\right]$$

- **Linear separability**

  Less entangled, controlling single output features





```
            loss_g = loss_fn_g(tf.ones_like(logits_fake), logits_fake)
        grads = tape.gradient(loss_g, generator.trainable_variables)
        optimizer_g.apply_gradients(zip(grads, generator.trainable_variables))

    # Print the progress
    print("Time for epoch {} is {} sec".format(epoch + 1, time.time() - start_time))

    # Save generated images
    if (epoch + 1) % sample_interval == 0:
        noise = tf.random.normal((16, latent_dim))
        generated_images = generator(noise, training=False)
        for i in range(generated_images.shape[0]):
            save_img('C:/Users/ASUS/Downloads/generated_' + str(i) + '.png', generated_images[i])
```

In [2]:
```
# Load the dataset
dataset = load_dataset()
dataset
```

Out[2]:
```
<PrefetchDataset element_spec=TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None)>
```

In [*]:
```
# train the model
train(dataset)
```

```
Time for epoch 1 is 9208.322114229202 sec
Time for epoch 2 is 9290.493214845657 sec
Time for epoch 3 is 9025.061330080032 sec
Time for epoch 4 is 8755.84211730957 sec
Time for epoch 5 is 8854.225080490112 sec
Time for epoch 6 is 8779.539382696152 sec
Time for epoch 7 is 8773.808583974838 sec
Time for epoch 8 is 8787.269788742065 sec
Time for epoch 9 is 8833.937662601471 sec
Time for epoch 10 is 40082.33596634865 sec
```

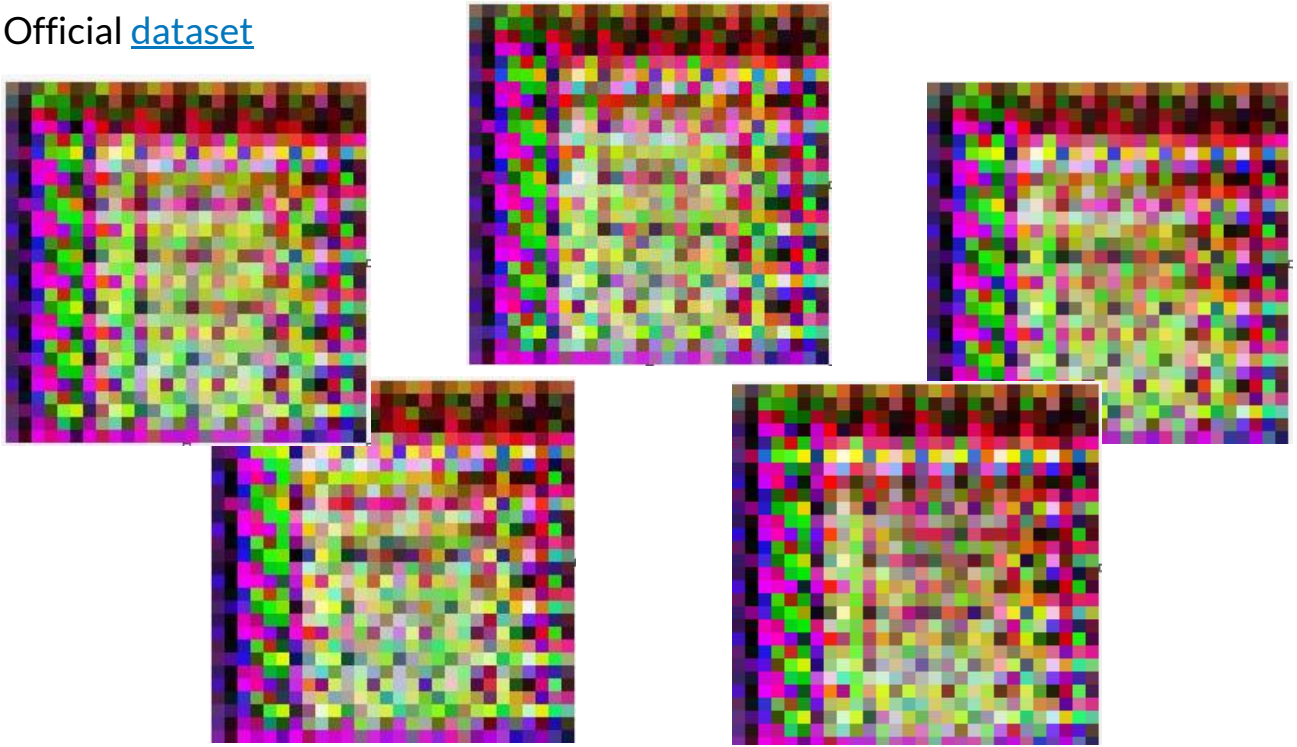
The tensorflow implementation

Link1 - original implementation:
https://github.com/NVlabs/stylegan

Link2 - my implementation:
https://github.com/LianaIs/styleGAN/blob/main/StyleGAN.ipynb

Results with the 100 input images

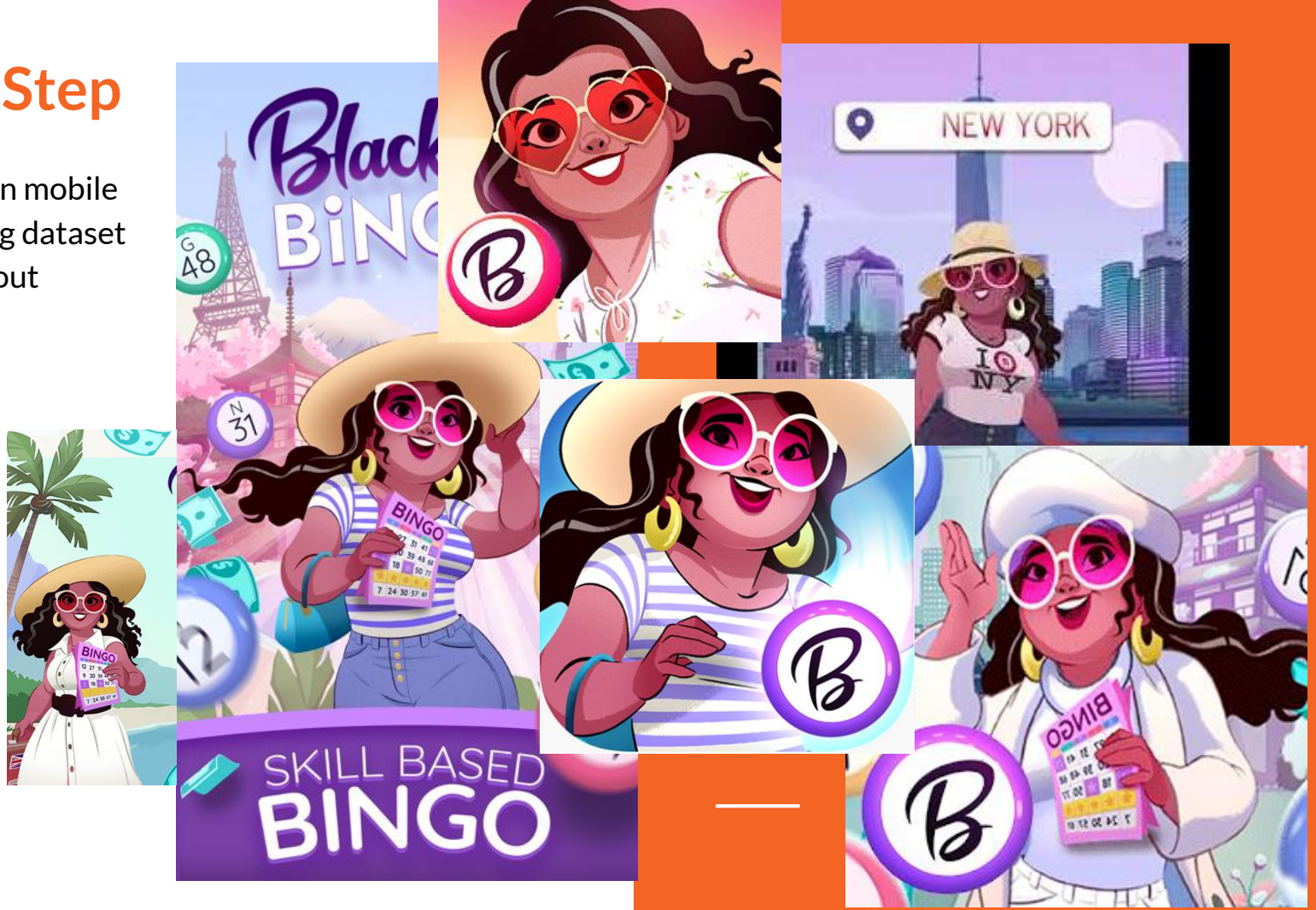
Official [dataset]



```
Time for epoch 1 is 26.325513124465942 sec
Time for epoch 2 is 27.825599193572998 sec
Time for epoch 3 is 28.306364059448242 sec
Time for epoch 4 is 28.61549186706543 sec
Time for epoch 5 is 29.49031710624695 sec
Time for epoch 6 is 29.780969381332397 sec
Time for epoch 7 is 30.82942795753479 sec
Time for epoch 8 is 29.02382254600525 sec
Time for epoch 9 is 29.379705667495728 sec
Time for epoch 10 is 29.101566076278687 sec
Time for epoch 11 is 29.461113691329956 sec
Time for epoch 12 is 29.42032551765442 sec
Time for epoch 13 is 30.247260093688965 sec
Time for epoch 14 is 30.669936180114746 sec
Time for epoch 15 is 31.875087022781372 sec
Time for epoch 16 is 30.04021453857422 sec
Time for epoch 17 is 28.960651636123657 sec
Time for epoch 18 is 29.361088514328003 sec
Time for epoch 19 is 29.603143453598022 sec
Time for epoch 20 is 29.148345947265625 sec
Time for epoch 21 is 29.843295335769653 sec
Time for epoch 22 is 30.28518009185791 sec
Time for epoch 23 is 31.34323525428772 sec
Time for epoch 24 is 31.765979528427124 sec
Time for epoch 25 is 29.12954592704773 sec
Time for epoch 26 is 28.953335285186768 sec
Time for epoch 27 is 29.6609365940094 sec
Time for epoch 28 is 29.68863034248352 sec
Time for epoch 29 is 30.006385326385498 sec
Time for epoch 30 is 29.901216983795166 sec
Time for epoch 31 is 31.28024911880493 sec
Time for epoch 32 is 31.558868408203125 sec
Time for epoch 33 is 32.7503399848938 sec
Time for epoch 34 is 29.10699701309204 sec
Time for epoch 35 is 29.635565757751465 sec
Time for epoch 36 is 29.656879663467407 sec
Time for epoch 37 is 30.14802575111389 sec
Time for epoch 38 is 30.213967323303223 sec
Time for epoch 39 is 30.58687663078308 sec
Time for epoch 40 is 30.91081714630127 sec
Time for epoch 41 is 32.26446747779846 sec
Time for epoch 42 is 31.703819274902344 sec
Time for epoch 43 is 29.441234588623047 sec
Time for epoch 44 is 29.61560869216919 sec
Time for epoch 45 is 30.27293086051941 sec
Time for epoch 46 is 30.01142430305481 sec
Time for epoch 47 is 31.20807695388794 sec
Time for epoch 48 is 32.08082914352417 sec
Time for epoch 49 is 32.35410404205322 sec
Time for epoch 50 is 29.321226835250854 sec
```

Next Step

Trying it on mobile advertising dataset (eg. Blackout Bingo)



Thank you!

Regards,

Liana Isayan

liana.isayan@yahoo.com