CS 597 REPORT 2

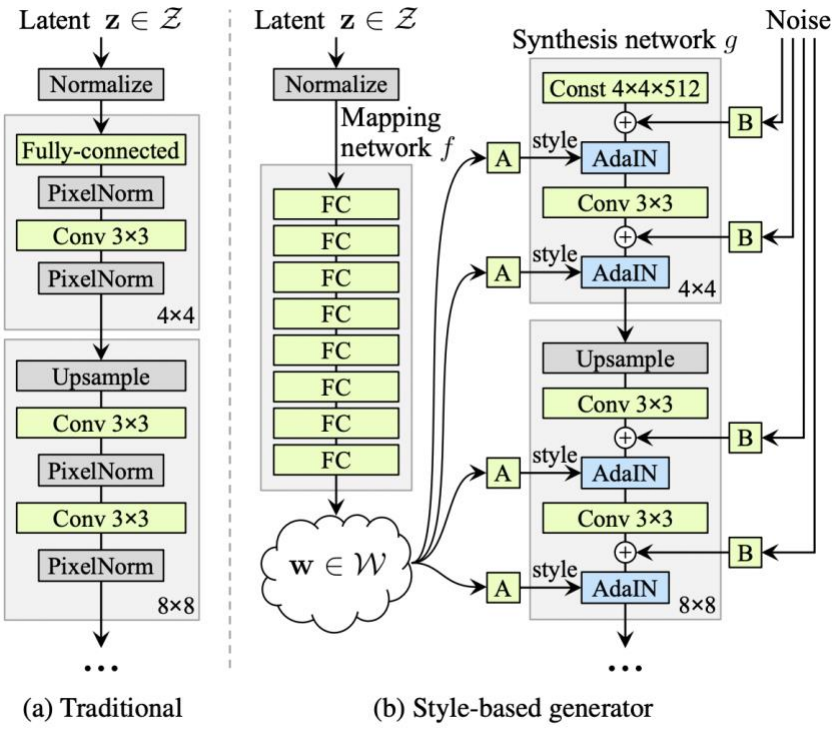
Yash Pradeep Gupte
MS COMPUTER SCIENCE
CWID: A20472798

I] A Style-Based Generator Architecture for Generative Adversarial Networks

1) Introduction and Novel Contribution

- Proposed an alternative generator architecture in GANs, which can precisely control the synthesis of image that is clearly separate the pose, identity, face freckles, hair, color, etc.
- This novel generator architecture can control the style of the generated image.
- Dataset – Flickr Face-HQ (FFHQ) – 70k images of 1024x1024 resolution consisting of diverse faces.
- The problem with GANs is that we do not have control over the style of the generated image, thus this architecture was developed using some novel techniques which are discussed later in this report.

2) Architecture



(a) Traditional (b) Style-based generator

(A) Mapping Network

- Take input as the latent vector z and maps it to another vector w.
- Affine transformation 'A' and Adaptive Instance Normalization (AdaIN)
- Noise vectors along with transformation 'B'
- In this architecture they have passed four noise vectors 'B', at different stages of the generator network
- We know batch normalization computes the mean and standard deviation of x and also has parameters gamma and beta (scale and translation factors)

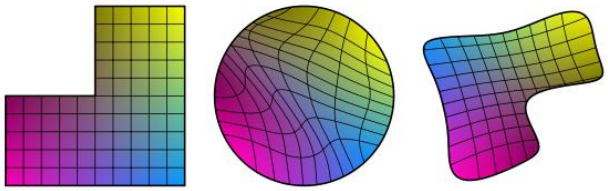
$$BN = \gamma \frac{x - \mu(x)}{\sigma(x)} + \beta$$

- Instance Normalization : breakdown or batch normalization of specific instance of a batch , treating each sample in the batch separately

$$IN(x_i) = \gamma \frac{x_i - \mu(x_i)}{\sigma(x_i)} + \beta$$

- Now, Adaptive Instance Normalization (AdaIN) is further specialization of Insance normalization where we have input xi and also have input yi. Here xi, is the features from previous layer convolutions and yi is composed of ys (scale factor) and yb (translation factor)
- Style y comes from weight vectors(w) and convolutional features(x) come from previous layers

$$AdaIN(x_i, y) = y_{s,i} \frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}$$



(a) Distribution of features in training set (b) Mapping from $\mathcal{Z}$ to features (c) Mapping from $\mathcal{W}$ to features

Figure 6. Illustrative example with two factors of variation (image features, e.g., masculinity and hair length). (a) An example training set where some combination (e.g., long haired males) is missing. (b) This forces the mapping from $\mathcal{Z}$ to image features to become curved so that the forbidden combination disappears in $\mathcal{Z}$ to prevent the sampling of invalid combinations. (c) The learned mapping from $\mathcal{Z}$ to $\mathcal{W}$ is able to "undo" much of the warping.

(B) Evolution of Generator with respect to architecture

	Method	CelebA-HQ	FFHQ
A	Baseline Progressive GAN [30]	7.79	8.04
B	+ Tuning (incl. bilinear up/down)	6.11	5.25
C	+ Add mapping and styles	5.34	4.85
D	+ Remove traditional input	5.07	4.88
E	+ Add noise inputs	**5.06**	4.42
F	+ Mixing regularization	5.17	**4.40**

- In StyleGAN the architecture has evolved from a baseline progressive GAN with a normal generator and discriminator architecture.

-Method B from above image was implemented using simple bilinear up/down sampling and longer training

- In method B, they added AdaIN and mapping network

- Method D, learned 4x4x512 constant tensor and removed traditional input noise. This actually improved the performance further

- Method E was the final architecture with noise inputs that is 'B'(Affine transformation)

- Method F was mixing regularization with a set of z and w vectors

(C) Mixing Regularization

- Instead of passing one latent input 'z' and getting one vector 'w' as output, they pass atleast two 'z1' and 'z2' and get two vectors 'w1' and 'w2'

- Then they pass 'w1' and 'w2' to randomly chosen input styles 'A'.

- In the next iteration, we choose completely random pints / styles and again pass 'w1' and 'w2'

- Use this mixing with certain percentage of training data only

- The following are results of mixing regularization

	Mixing regularization	Number of latents during testing 1	2	3	4
E	0%	4.42	8.22	12.88	17.41
	50%	4.41	6.10	8.71	11.61
F	90%	**4.40**	**5.11**	6.88	9.03
	100%	4.83	5.17	**6.63**	**8.40**

Table 2. FIDs in FFHQ for networks trained by enabling the mixing regularization for different percentage of training examples. Here we stress test the trained networks by randomizing 1 . . . 4 latents and the crossover points between them. Mixing regularization improves the tolerance to these adverse operations significantly. Labels E and F refer to the configurations in Table 1.

3) RESULTS

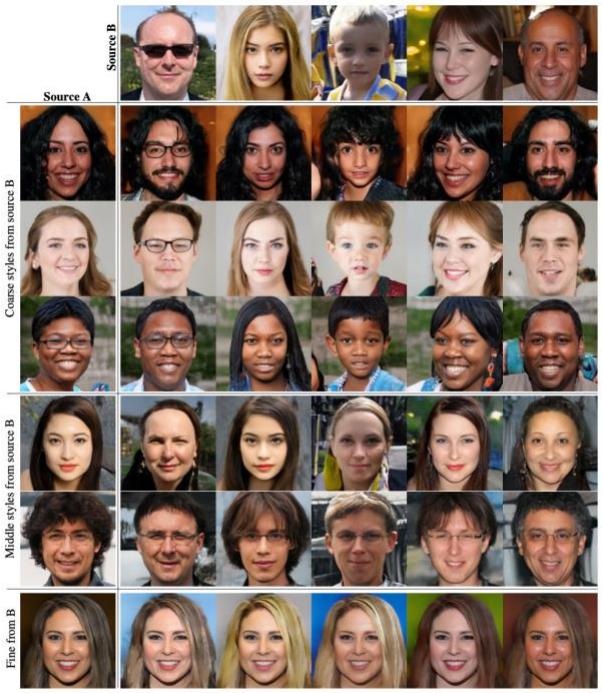


Figure 3. Two sets of images were generated from their respective latent codes (sources A and B); the rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A. Copying the styles corresponding to coarse spatial resolutions ($4^2 - 8^2$) brings high-level aspects such as pose, general hair style, face shape, and eyeglasses from source B, while all colors (eyes, hair, lighting) and finer facial features resemble A. If we instead copy the styles of middle resolutions ($16^2 - 32^2$) from B, we inherit smaller scale facial features, hair style, eyes open/closed from B, while the pose, general face shape, and eyeglasses from A are preserved. Finally, copying the fine styles ($64^2 - 1024^2$) from B brings mainly the color scheme and microstructure.

- They generate set of images called source A and source B.
- Now we have set of latent vectors corresponding to these images generated
- Combine these latent vectors at different scales and they see how combining these set of vectors results in appearance of the images.
- They noticed that when we mix coarse level vectors of source A and B, we notice that there are changes in pose and hairstyle of the images generated
- But if we mix the fine level details of latent vector A with that of B, we notice change in color scheme and microstructures which are very subtle.

4) Effect of varying noise vector 'B'

- Style GAN takes input four noise vectors on other side of the network.
- Noise vector has much more subtle effect in results of generation compared to GAN, where changing noise vector will generate entirely different image.
- In case of Style GAN, changes are very subtle

5)Effect of adding noise vector at different levels

- Add noise at different levels and not at all the levels.
- When they add noise at fine level, they get fine or subtle changes in the image generated.

- But when they add no noise at all to the network, then the generated image is very blurry, without any prominent features.

- Noise at coarse level was also compared and the results were as follows,

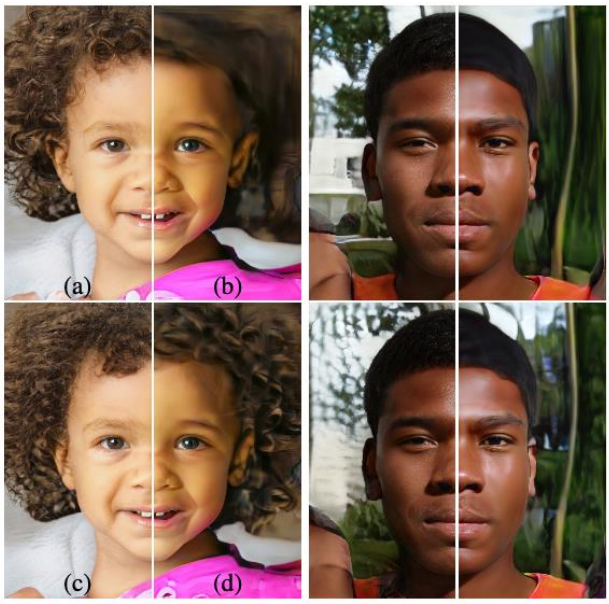


Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only (64^2 – 1024^2). (d) Noise in coarse layers only (4^2 – 32^2). We can see that the artificial omission of noise leads to featureless "painterly" look. Coarse noise causes large-scale curling of hair and appearance of larger background features, while the fine noise brings out the finer curls of hair, finer background detail, and skin pores.

II] Analyzing and Improving the Image Quality of StyleGAN

1) Introduction and Contribution

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.

- Instance normalization in style GAN v1 causes water droplets like effects in generated images.

- Removing these Normalization Artifacts in generated image can be achieved by restructuring the Adaptive Instance Normalization (AdaIN).

2) Architecture changes in Style GAN - (Style GAN version 2)

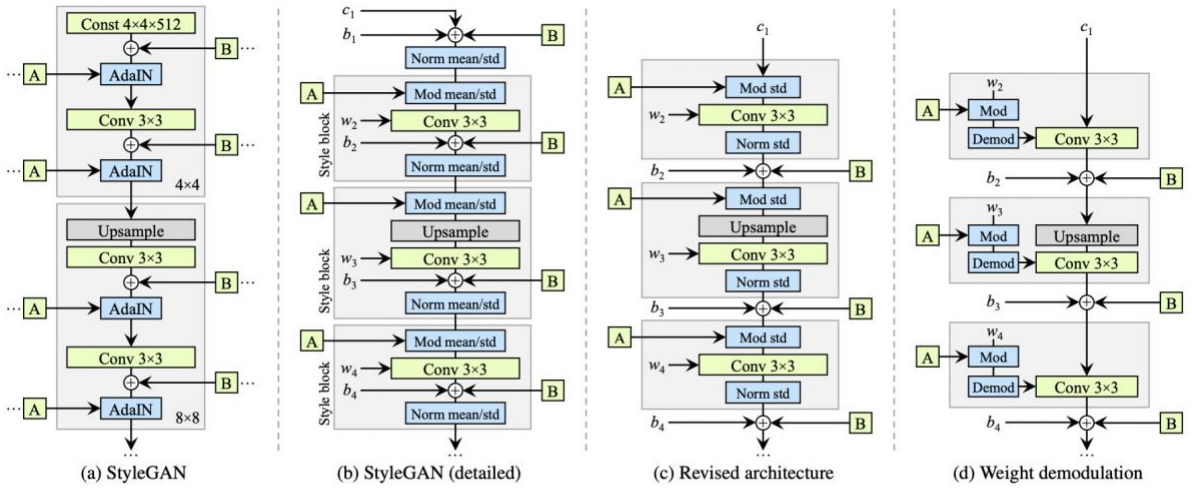


Figure 2. We redesign the architecture of the StyleGAN synthesis network. (a) The original StyleGAN, where A denotes a learned affine transform from $\mathcal{W}$ that produces a style and B is a noise broadcast operation. (b) The same diagram with full detail. Here we have broken the AdaIN to explicit 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 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.

(A) Weight Demodulation

- Separate out the addition of gaussian noise 'B' with AdaIN 'A', stating that these might have conflicting interest which by sort of block each other out. -> separate style blocks (grey color in above figure 2) in revised architecture.
- Next idea is to get rid of AdaIN and instead use Weight demodulation layers
- The idea is to scale the parameters by using s_i from AdaIN from w latent vector and then demodulate it to, assuming that features have unit variance, changing the weight parameter of 3x3 convolutional layer (in figure 2 (d)) instead of having intermediate modulation layer and normalization layer (figure 2(b)).

$$w'_{ijk} = s_i \cdot w_{ijk},$$

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

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

- Without demodulation, normalization artifacts or water droplets are introduced in generated images.

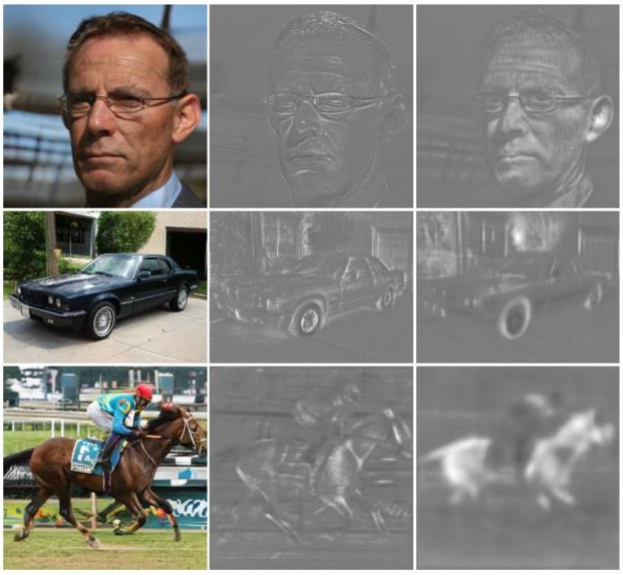


Figure 3. Replacing normalization with demodulation removes the characteristic artifacts from images and activations.

(B) Perceptual Path Length Regularization (PPL)

- Add PPL to the loss function of the generator
- The high-level idea of the PPL is to have changes in latent vector z, now results too dramatic of the changes in the generated image. -> So we slightly change this latent vector z, we want it to be smooth semantic changes in generated image rather than having completely different image.
- PPL image quality metric -> FID score or Precision / Recall. Low PPL score is better than high PPL score.

(a) Low PPL scores (b) High PPL scores

Figure 4. Connection between perceptual path length and image quality using baseline StyleGAN (config A) with LSUN CAT. (a) Random examples with low PPL ($\leq 10^{\text{th}}$ percentile). (b) Examples with high PPL ($\geq 90^{\text{th}}$ percentile). There is a clear correlation between PPL scores and semantic consistency of the images.

- Implementation is done by using high level Jacobian matrix $Jw = \partial g(w)/\partial w$. -> partial derivative of the output with respect to small changes in latent vector w.
- Use small changes to see how much the output changes

$$\mathbb{E}_{\mathbf{w},\mathbf{y}\sim\mathcal{N}(0,\mathbf{I})}\left(\left\|\mathbf{J}_{\mathbf{w}}^{T}\mathbf{y}\right\|_{2}-a\right)^{2}$$

- Where y is a random image, and a is exponential moving average
- Lazy Regularization idea – add this to loss function only every 16 steps

(C) Is Progressive Growing Necessary?

- In progressive growing network there are a lot of hyper parameters search – complicated for training and this causes a bottleneck for architecture.

Figure 6. Progressive growing leads to "phase" artifacts. In this example the teeth do not follow the pose but stay aligned to the camera, as indicated by the blue line.

- Recent Paper – Multiscale Gradient for GANs(MSG-GAN) – idea is to enforce intermediate feature maps in the generator – by taking it out and projecting it and sort of providing additional features to the discriminator

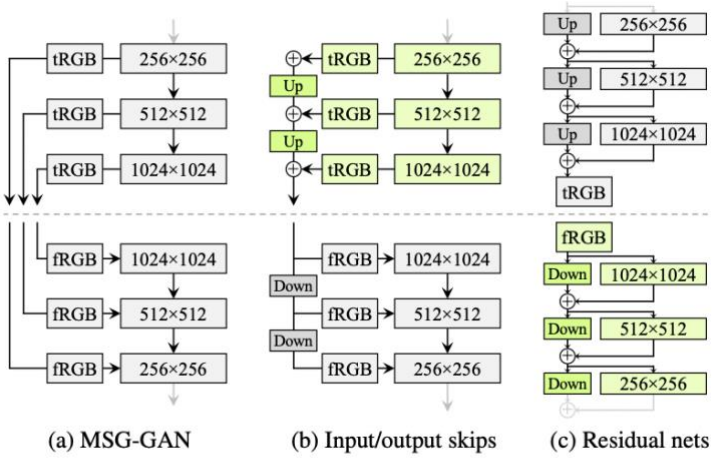


Figure 7. Three generator (above the dashed line) and discriminator architectures. Up and Down denote bilinear up and downsampling, respectively. In residual networks these also include 1×1 convolutions to adjust the number of feature maps. tRGB and fRGB convert between RGB and high-dimensional per-pixel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

- Modification from progressive growing
  - ResNet style architecture – isolate 256x256 feature maps and then perform upsampling, then do elementwise addition, at the end convert it into 3 channel RGB image.
  - Input/Output skips – In this method, first convert 256x256 feature map into RGB image by flattening it into 3 channels, then sum this up and perform upsampling and go to the next level.

(D) Ablation

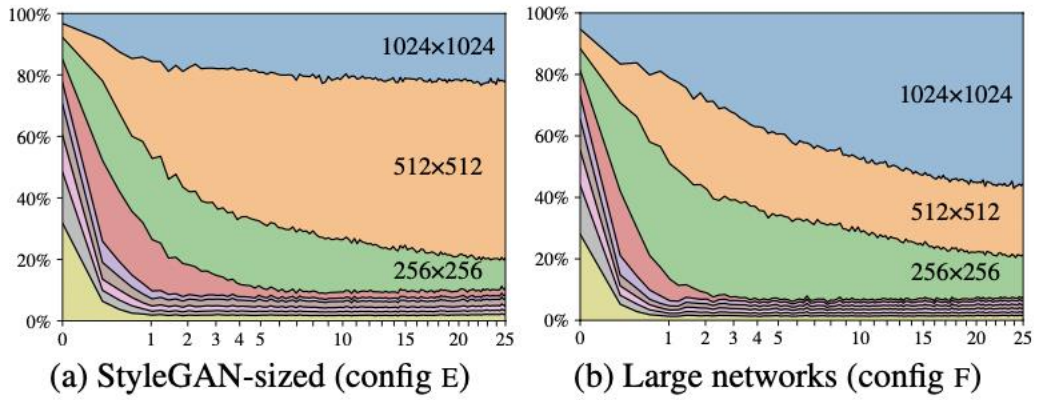
- Shows that , there is not much difference in skips and residual network , in the resulting image quality

FFHQ	D original		D input skips		D residual	
	FID	PPL	FID	PPL	FID	PPL
G original	4.32	265	4.18	235	3.58	269
G output skips	4.33	169	3.77	127	**3.31**	**125**
G residual	4.35	203	3.96	229	3.79	243

LSUN Car	D original		D input skips		D residual	
	FID	PPL	FID	PPL	FID	PPL
G original	3.75	905	3.23	758	3.25	802
G output skips	3.77	544	3.86	**316**	3.19	471
G residual	3.93	981	3.40	667	**2.66**	645

Table 2. Comparison of generator and discriminator architectures without progressive growing. The combination of generator with output skips and residual discriminator corresponds to configuration E in the main result table.

3) Stability of Multi Scale Training



- ResNet takes intermediate feature maps, does elementwise sum to produce final feature map

- We can see from above graph how much each feature map is contributing to final output across different training iterations

4) Projection of images back to latent space

- Idea is to find the latent vector w that produce the target image (DeepFake)

5) Results

Configuration	FFHQ, 1024×1024				LSUN Car, 512×384			
	FID ↓	Path length ↓	Precision ↑	Recall ↑	FID ↓	Path length ↓	Precision ↑	Recall ↑
A Baseline StyleGAN [24]	4.40	212.1	**0.721**	0.399	3.27	1484.5	**0.701**	0.435
B + Weight demodulation	4.39	175.4	0.702	0.425	3.04	862.4	0.685	0.488
C + Lazy regularization	4.38	158.0	0.719	0.427	2.83	981.6	0.688	0.493
D + Path length regularization	4.34	**122.5**	0.715	0.418	3.43	651.2	0.697	0.452
E + No growing, new G & D arch.	3.31	124.5	0.705	0.449	3.19	471.2	0.690	0.454
F + Large networks (StyleGAN2)	**2.84**	145.0	0.689	**0.492**	**2.32**	**415.5**	0.678	**0.514**
Config A with large networks	3.98	199.2	0.716	0.422	–	–	–	–

Table 1. Main results. For each training run, we selected the training snapshot with the lowest FID. We computed each metric 10 times with different random seeds and report their average. *Path length* corresponds to the PPL metric, computed based on path endpoints in $\mathcal{W}$ [24], without the central crop used by Karras et al. [24]. The FFHQ dataset contains 70k images, and the discriminator saw 25M images during training. For LSUN CAR the numbers were 893k and 57M. ↑ indicates that higher is better, and ↓ that lower is better.

-Training Speed Gains for 1024x1024 images

- StyleGAN v1 - 37 images per second
- V2 Config E - 61 (40% faster)
- V2 Config F – 31 (larger network)
- V2 Config F – 9 days on 8 tesla v100 GPUs for FFHQ and 13 days for LSUN CAR

Figure 11. Four hand-picked examples illustrating the image quality and diversity achievable using StylegGAN2 (config F).