Academic year 2023

Module Code: CS4740

Module Name: Deep learning

Module Leader: Dr James Borg

Coursework Title: Technical Blog

Student ID: 220283889 / 190114871 / 220262172

Introduction

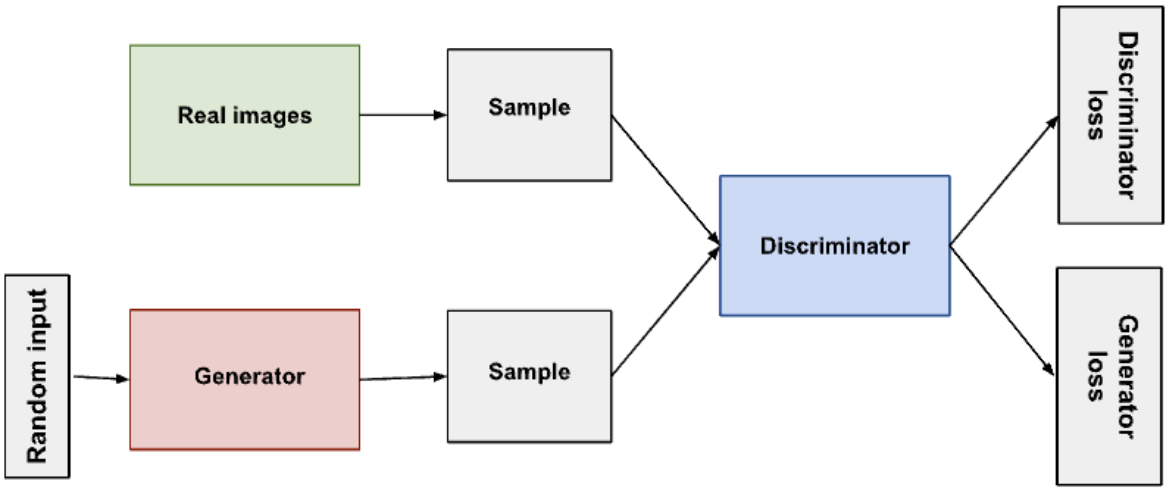
In the past several years, deep learning has made amazing strides in the field of computer vision, and generative models have played a crucial part in this success. One of the numerous types of generative models, known as Generative Adversarial Networks (GANs), has emerged as a prominent area of investigation. GANs are able to generate realistic images. However, the generator network in GANs continues to remain as a black box, it takes inputs and generates images which can be completely random. This lack of control in GANs is a major downside that the researchers at NVIDIA tackled in creating the StyleGAN model.

In this blog, we will discuss the specific technical features of the StyleGAN architecture, describe the method to implement StyleGAN, evaluate StyleGANs benefits and drawbacks, provide interesting implementations of StyleGAN and provide a comparison of StyleGAN with other similar methods. By the end of this blog, you should have a comprehensive understanding of StyleGAN and its potential impacts of it in the field of computer vision.

Background

In 2014, Ian Goodfellow [3] was the first to propose the Generative Adversarial Nets model, he had the revolutionary idea to make two neural networks compete with each other. One neural network, called the generator network, tries to generate realistic data, and the other network, the discriminator network, tries to discriminate between real data and data generated by the generator network. The generator network uses the discriminator as a loss function and updates its parameters to generate data that starts "fooling" the discriminator, thus, the data starts to look more realistic. On the other hand, the discriminator network

updates its parameters to make itself better at picking out fake data from real data. This is done until the generator is able to create data which looks real enough and forces the discriminator to guess randomly.

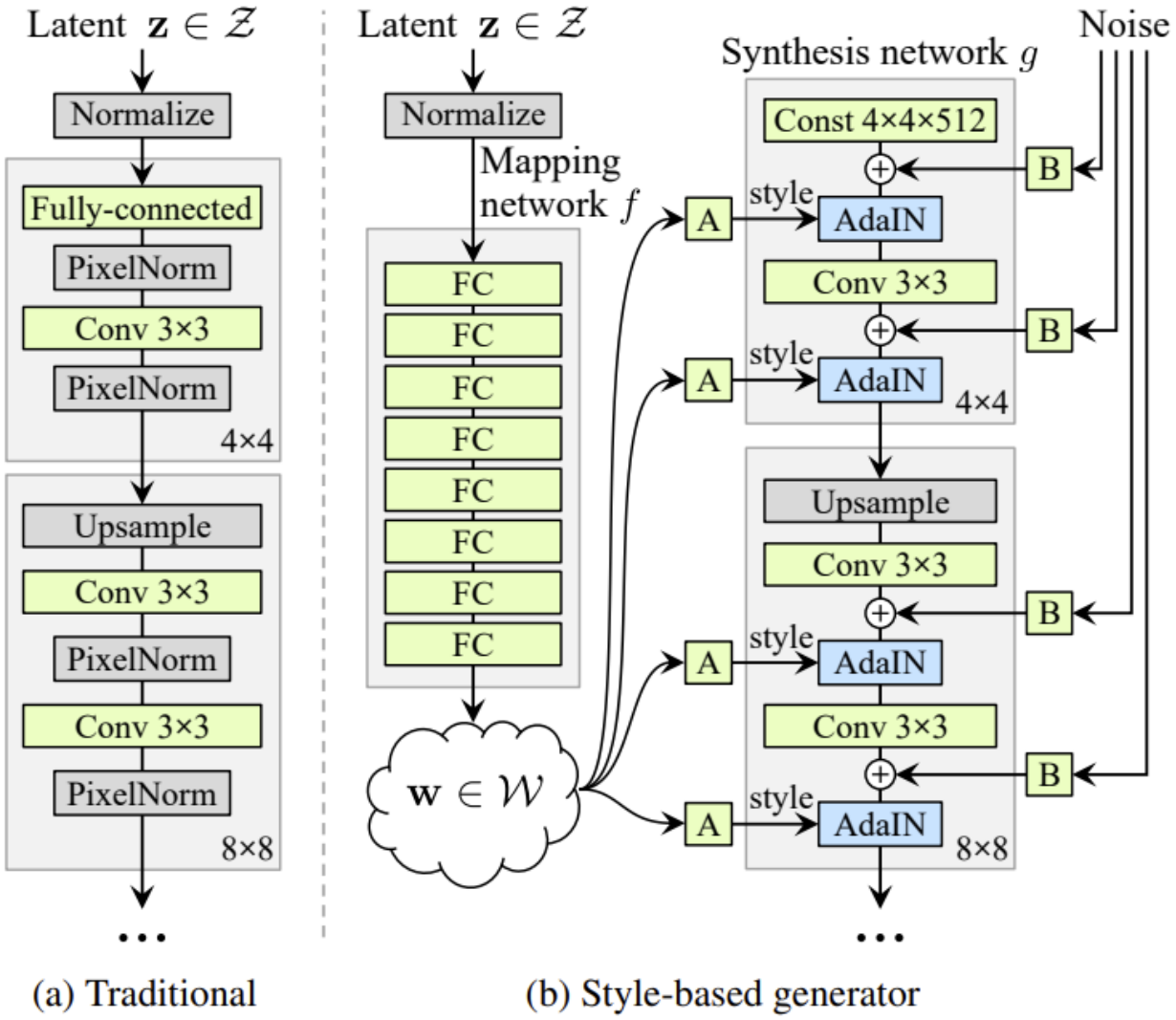


[4]

Throughout the years the quality and resolution of the GAN model outputs continued to improve, through innovations such as progressive growing, minibach discrimination and normalisation in the generator and discriminator to name a few.[2] Yet, "the generators continue to operate as black boxes", we don't have full visibility into the decision-making process that the generator network is using to generate new data. Furthermore, "the properties of the latent space are also poorly understood" and comparing the performance of different generators is also not possible.[1]

StyleGAN

This issue was addressed by Karras et al. in their 2018 paper [1] where they proposed StyleGAN, an extension to the GAN architecture to give control over the disentangled style properties of generated images. The team made major changes to the generator of the progressive growing GAN architecture (which involved incrementally increasing the size of the model during training) allowing users to tune hyperparameters to control individual characteristics of data. Below you can see both the traditional progressive growing GAN and the style-based generator network "StyleGAN":



[1]

In the traditional network (left) latent vectors directly pass into the block after normalization, however, in StyleGAN the latent vector passes through the "mapping network ", which is an 8-layered MLP, this network acts as a non-linear mapping function and maps the latent vectors to w. This is done in the hopes to produce a more disentangle latent space W. But what is the benefit of this?

Well, a more disentangled latent space refers to a space where each dimension corresponds to one specific independent feature of the input data. This means that if we change a single dimension in the latent space, this would only affect one aspect of the output. For example, in a disentangled latent space, different dimensions would correspond to different features such as hair color, eye color etc... and changing the dimension of hair color would only affect the hair color of the generated image, rather than having all the features change together.

This the w latent vectors are passed to the "A" block (called learned affine transformations), which linearly transform the vector to produce styles:

$$\mathbf{y} = (\mathbf{y}_s, \mathbf{y}_b)$$

The style y has component ys which is the scale and yb which is the bias. Since, the NVIDIA team which

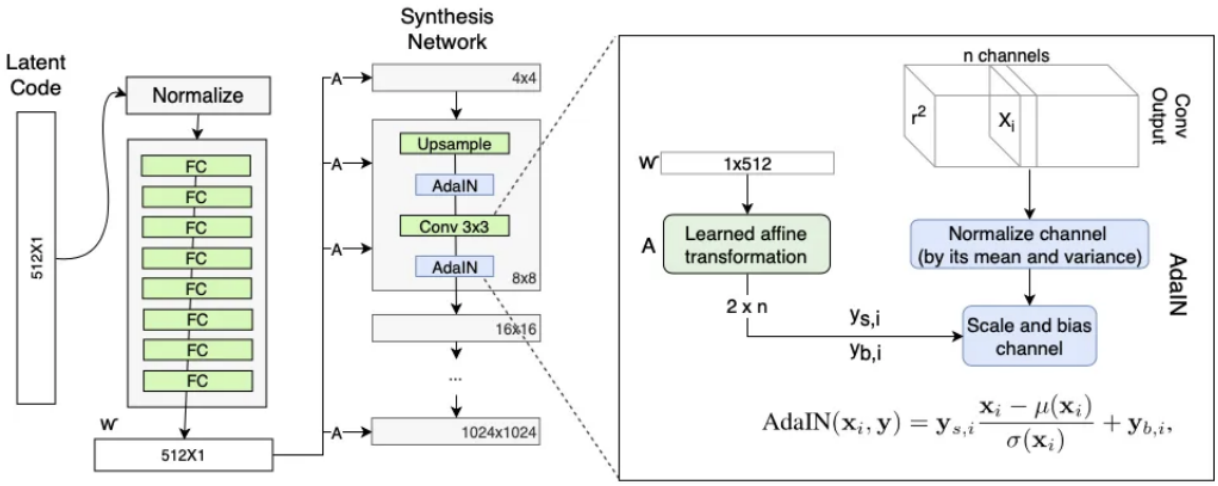
created StyleGAN have made a github repo which is publicly available at [6], here is the implementation of the "A" block in StyleGAN:

In [ ]:
```
def style_mod(x, dlatent, **kwargs):
    with tf.variable_scope('StyleMod'):
        style = apply_bias(dense(dlatent, fmaps=x.shape[1]*2, gain=1, **kwargs))
        style = tf.reshape(style, [-1, 2, x.shape[1]] + [1] * (len(x.shape) - 2))
        return x * (style[:,0] + 1) + style[:,1]
```

These styles are injected into AdaIN (as shown in the above graph, stands for Adaptive Instance Normalization) which is able to control these styles. The AdaIN operation is defined as:

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

As shown by the equation above, the AdaIN operation first normalizes each channel of the input image xi, and then an element-wise multiplication is performed with ys which is the scale component of style. Lastly, the bias yb is added. This whole process is done to transfer the encoded information w into the generated image. Here is a graph which visualizes the AdaIN module:



The generator's Adaptive Instance Normalization (AdaIN)

[5]

The following code is the AdaIN imlementation in StyleGAN:

In [ ]:
```
if use_styles:
            x = style_mod(x, dlatents_in[:, layer_idx], use_wscale=use_wscale)
        return x
```

Another interesting change made in StyleGAN is a constant value as the initial image of the generator. In the traditional GAN shown in the graph above, a random initial image is created first, however, this is omitted in StyleGAN as the image features are controlled by the style w and the AdaIN module.

Lastly, in the StyleGAN framework, Gaussian noise is fed into a block "B" which "applies learned per-channel scaling factors to the noise input" [1]. This simply means that block B decides how much Gaussian noise is

introduced into the network. And this noise is added after each convolution. Below is the implementation of this "B" block:

In [ ]:
```
def apply_noise(x, noise_var=None, randomize_noise=True):
    assert len(x.shape) == 4 # NCHW
    with tf.variable_scope('Noise'):
        if noise_var is None or randomize_noise:
            noise = tf.random_normal([tf.shape(x)[0], 1, x.shape[2], x.shape[3]], dtype=
        else:
            noise = tf.cast(noise_var, x.dtype)
        weight = tf.get_variable('weight', shape=[x.shape[1].value], initializer=tf.init
        return x + noise * tf.reshape(tf.cast(weight, x.dtype), [1, -1, 1, 1])
```

Method

Since StyleGAN is very complex and requires a vast amount of GPU power (at least 2 GPUs) and time (several days in some cases)to train, we will use a pre-trained network, which is also provided by the NVIDIA team in their GitHub, to show the output of StyleGAN:

In [ ]:
```
!git clone https://github.com/NVlabs/stylegan.git
```

In [ ]:
```
import os
import pickle
import numpy as np
import PIL.Image
import dnnlib
import dnnlib.tflib as tflib
import config

# Initialize TensorFlow.
tflib.init_tf()

# Load pre-trained network.
url = 'https://drive.google.com/uc?id=1MEGjdvVpUsu1jB4zrXZN7Y4kBBOzizDQ' # karras2019sty
with dnnlib.util.open_url(url, cache_dir=config.cache_dir) as f:
    _G, _D, Gs = pickle.load(f)
 # _G = Instantaneous snapshot of the generator. Mainly useful for resuming a previous t
 # _D = Instantaneous snapshot of the discriminator. Mainly useful for resuming a previo
 # Gs = Long-term average of the generator. Yields higher-quality results than the insta

# Print network details.
Gs.print_layers()

# Pick latent vector.
rnd = np.random.RandomState(5)
latents = rnd.randn(1, Gs.input_shape[1])

# Generate image.
fmt = dict(func=tflib.convert_images_to_uint8, nchw_to_nhwc=True)
images = Gs.run(latents, None, truncation_psi=0.7, randomize_noise=True, output_transfor

# Save image.
os.makedirs(config.result_dir, exist_ok=True)
png_filename = os.path.join(config.result_dir, 'example.png')
PIL.Image.fromarray(images[0], 'RGB').save(png_filename)
```

The above code is from the StyleGAN public github's pretrained_example.py [6]. This example shows how a pre-trained StyleGAN can be used. You will need to get a copy of the source code and the pre-trained network, which is provided by the url. The generated image be different depending on the random state,

changing this random state will result in a different image generated. This random state is used in the latent vector in the line latents = rnd.randn(1, Gs.input_shape[1]). To generate the image, we run the generator in images = Gs.run(latents, None, truncation_psi=0.7, randomize_noise=True, output_transform=fmt), and the latent vector is used as one of the parameters. Truncation_psi is another very important parameter as this is what influences the change in the generated image, it is used to determine how much change we want in the generated image.

Below is the implementation of the generator which uses both G_mapping and G_synthesis networks as shown in the graphs above:

In [ ]:
```
def G_style(
    latents_in,                                 # First input: Latent vectors (Z) [m
    labels_in,                                  # Second input: Conditioning labels
    truncation_psi          = 0.7,              # Style strength multiplier for the
    truncation_cutoff       = 8,                # Number of layers for which to appl
    truncation_psi_val      = None,             # Value for truncation_psi to use du
    truncation_cutoff_val   = None,             # Value for truncation_cutoff to use
    dlatent_avg_beta        = 0.995,            # Decay for tracking the moving aver
    style_mixing_prob       = 0.9,              # Probability of mixing styles durin
    is_training             = False,            # Network is under training? Enables
    is_validation           = False,            # Network is under validation? Choos
    is_template_graph       = False,            # True = template graph constructed
    components              = dnnlib.EasyDict(),    # Container for sub-networks. Retain
    **kwargs):                                  # Arguments for sub-networks (G_mapp

    # Validate arguments.
    assert not is_training or not is_validation
    assert isinstance(components, dnnlib.EasyDict)
    if is_validation:
        truncation_psi = truncation_psi_val
        truncation_cutoff = truncation_cutoff_val
    if is_training or (truncation_psi is not None and not tflib.is_tf_expression(truncat
        truncation_psi = None
    if is_training or (truncation_cutoff is not None and not tflib.is_tf_expression(trun
        truncation_cutoff = None
    if not is_training or (dlatent_avg_beta is not None and not tflib.is_tf_expression(d
        dlatent_avg_beta = None
    if not is_training or (style_mixing_prob is not None and not tflib.is_tf_expression(
        style_mixing_prob = None

    # Setup components.
    if 'synthesis' not in components:
        components.synthesis = tflib.Network('G_synthesis', func_name=G_synthesis, **kwa
    num_layers = components.synthesis.input_shape[1]
    dlatent_size = components.synthesis.input_shape[2]
    if 'mapping' not in components:
        components.mapping = tflib.Network('G_mapping', func_name=G_mapping, dlatent_bro

    # Setup variables.
    lod_in = tf.get_variable('lod', initializer=np.float32(0), trainable=False)
    dlatent_avg = tf.get_variable('dlatent_avg', shape=[dlatent_size], initializer=tf.in

    # Evaluate mapping network.
    dlatents = components.mapping.get_output_for(latents_in, labels_in, **kwargs)

    # Update moving average of W.
    if dlatent_avg_beta is not None:
        with tf.variable_scope('DlatentAvg'):
            batch_avg = tf.reduce_mean(dlatents[:, 0], axis=0)
            update_op = tf.assign(dlatent_avg, tflib.lerp(batch_avg, dlatent_avg, dlaten
            with tf.control_dependencies([update_op]):
                dlatents = tf.identity(dlatents)
```

```
    # Perform style mixing regularization.
    if style_mixing_prob is not None:
        with tf.name_scope('StyleMix'):
            latents2 = tf.random_normal(tf.shape(latents_in))
            dlatents2 = components.mapping.get_output_for(latents2, labels_in, **kwargs)
            layer_idx = np.arange(num_layers)[np.newaxis, :, np.newaxis]
            cur_layers = num_layers - tf.cast(lod_in, tf.int32) * 2
            mixing_cutoff = tf.cond(
                tf.random_uniform([], 0.0, 1.0) < style_mixing_prob,
                lambda: tf.random_uniform([], 1, cur_layers, dtype=tf.int32),
                lambda: cur_layers)
            dlatents = tf.where(tf.broadcast_to(layer_idx < mixing_cutoff, tf.shape(dlat

    # Apply truncation trick.
    if truncation_psi is not None and truncation_cutoff is not None:
        with tf.variable_scope('Truncation'):
            layer_idx = np.arange(num_layers)[np.newaxis, :, np.newaxis]
            ones = np.ones(layer_idx.shape, dtype=np.float32)
            coefs = tf.where(layer_idx < truncation_cutoff, truncation_psi * ones, ones)
            dlatents = tflib.lerp(dlatent_avg, dlatents, coefs)

    # Evaluate synthesis network.
    with tf.control_dependencies([tf.assign(components.synthesis.find_var('lod'), lod_in
        images_out = components.synthesis.get_output_for(dlatents, force_clean_graph=is_
    return tf.identity(images_out, name='images_out')
```

Evaluation

StyleGAN excels in creating realistic images, below is an example of the capabilities StyleGAN:

[1]

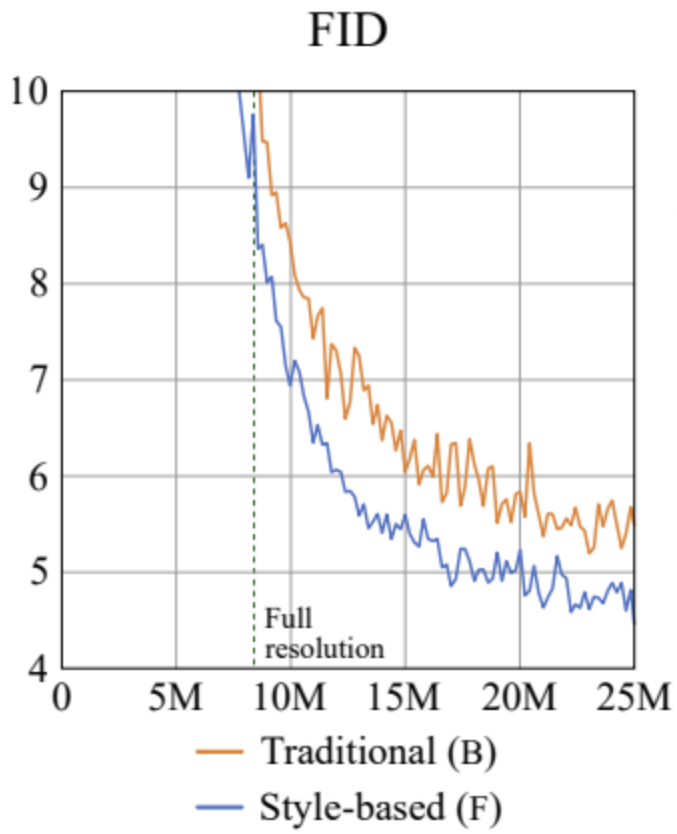
The above image shows the capabilities of StyleGAN, in this example StyleGAN the latent factors which generate image "Source B" are used to derive the coarse style, and the finer spatial resolutions are gathered from "Source A". This means that the generated image will have the "high-level" styles such as poses, face shape and accessories from "Source B" and have the finer facial features of "Source A".

Furthermore, StyleGAN has very impressive Fréchet Inception Distance (FID) scores; these scores are metrics to evaluate GANs, the FID is used to compute the distance between the feature vectors of the original image and the fake image. The lower the FID score is the better the quality of the image generated, as it is more similar to the original image. Below are the FID scores of StyleGAN when trained on the CelebA-HQ and FFHQ datasets when the respective implementations were added:

	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**

[1]

Comparing the StyleGAN and traditional progressive GANs FID scores we get the following graph, showing an effective reduction:



[1]

One major downside of StyleGAN is the large training time of the model, the paper stated that their training time was approximately one week on an NVIDIA DGX-1 with 8 Tesla V100 GPUs [1], to train the model using the CelebA-HQ and FFHQ datasets. Furthermore, the table below shows the expected training times for the default configuration of StyleGAN using Tesla V100 GPUs:

GPUs	1024×1024	512×512	256×256
1	41 days 4 hours	24 days 21 hours	14 days 22 hours
2	21 days 22 hours	13 days 7 hours	9 days 5 hours
4	11 days 8 hours	7 days 0 hours	4 days 21 hours
8	6 days 14 hours	4 days 10 hours	3 days 8 hours

[6]

This means training a model using large datasets like the FFHQ dataset used in the paper is not feasible for people using "standard" computers or laptops, therefore, most will rely on the pre-trained networks provided by the team, which may not be what a user needs.

In addition, another disadvantage of the StyleGAN model is that all the generated images have "water droplet-like artefacts" which are noticeable from 64x64 resolution images onwards [7]:

[7]

This was first discovered and demonstrated in the paper by the same NVIDIA team which made StyleGAN [7]. These artefacts can not be fixed in StyleGAN as they stem from the AdaIN operation, in the process of normalizing each channel of the input image information on the magnitudes of the features relative to each other is lost. Furthermore, another artefact was found that involved certain features like teeth or eyes being stuck in place when transformations were made to the image:

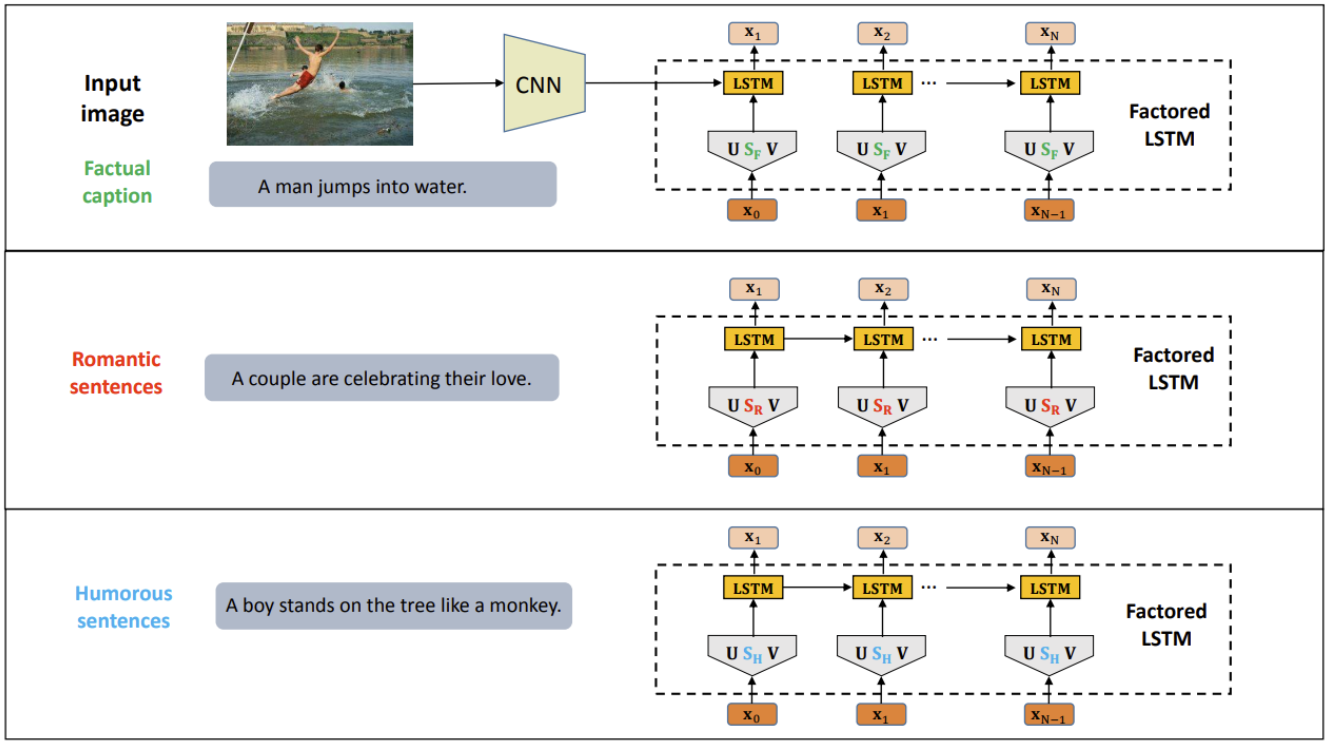
[7]

This artefact is also unavoidable when using StyleGAN as the problem stems from the progressively grown generator; for each resolution created the image is shortly serves as the output resolution and this inteferes with the shift invariance.

Interesting applications of StyleGAN

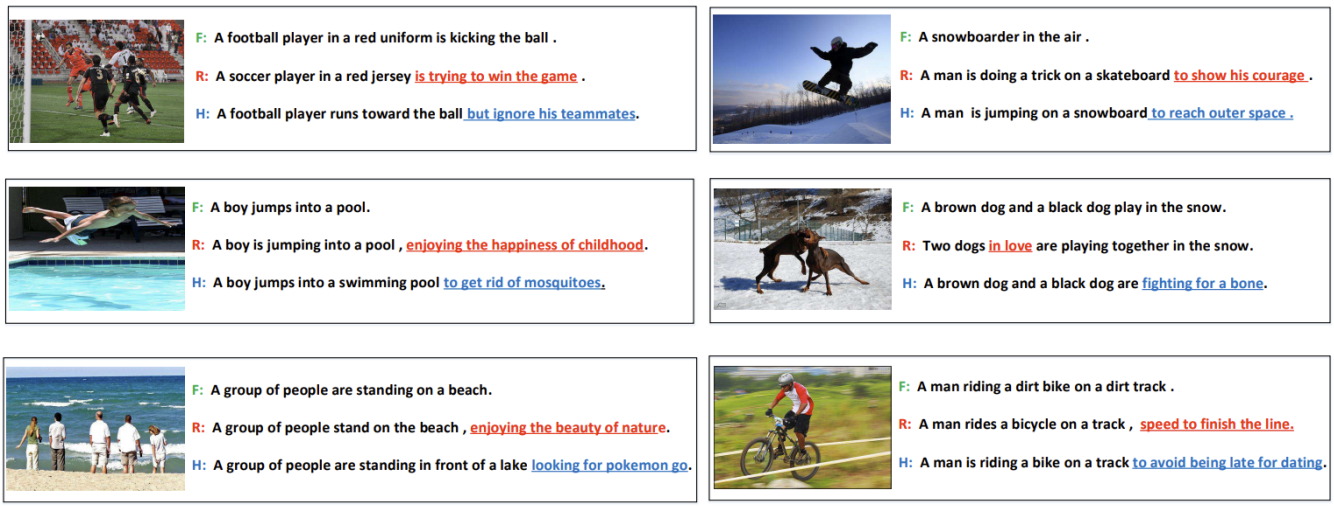
StyleGAN can be used in interesting applications such as image-to-image translations, for example, translating a satellite map picture into Google Maps equivalent images; other applications can be generating assets for games or even in music, the StyleGAN just needs to be trained on the appropriate dataset. Another very interesting application is extracting line drawing from 3D models, proposed by Wang et al. [11]. In this paper, the authors proposed using StyleGAN and StyleNET for the purpose of line synthesis, but what is StyleNET?

StyleNet is a new way to make comments for pictures and videos with different styles that are both interesting and accurate. StyleNet uses a new model part called factored LSTM, which naturally pulls out the style factors from the monolingual text corpus. The image below shows the framework:



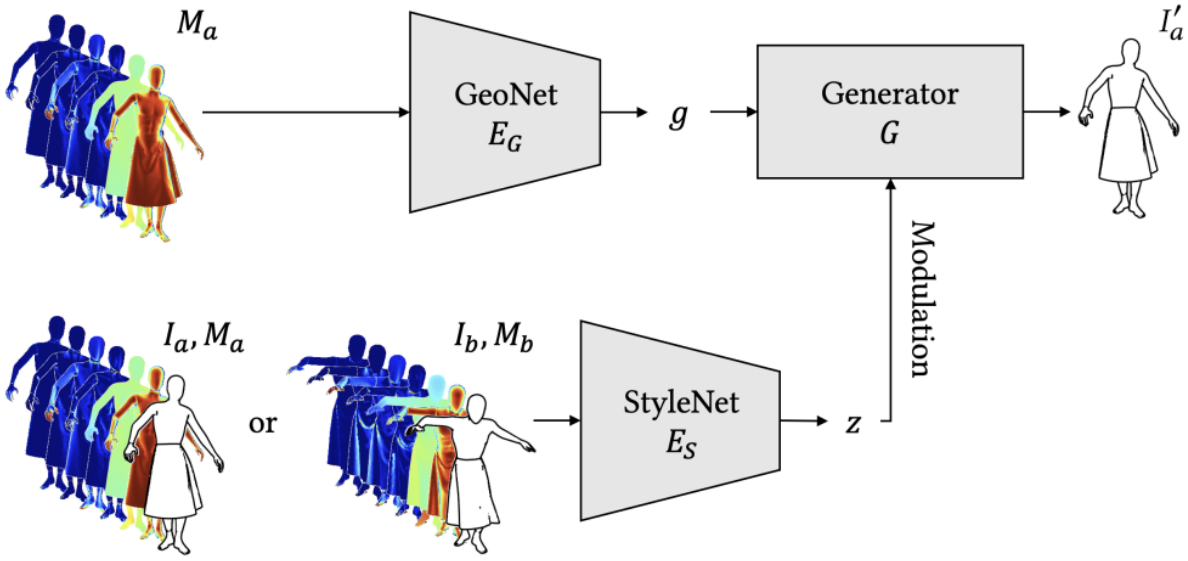
[10]

For the encoder part, the CNN that has already been trained takes an input picture and pulls out the feature vector. The feature vector is linearly changed so that it has the same number of dimensions as the LSTM network's input. Source and target messages are already set up for the decoder part. The encoder part is almost the same in the test phase as it was in the training phase. The only change is that the batch norm layer does not use mini-batch statistics. Instead, it uses moving average and variance. This can be done quickly with encoder.eval.(). When it comes to the decoder, the teaching phase and the test phase are very different. During the test step, the image description can't be seen by the LSTM decoder. To fix this, the LSTM encoder sends the word that was just made back to the next input. A for-loop can be used to do this. the image below shows an example of captions generated from images using StyleNET:



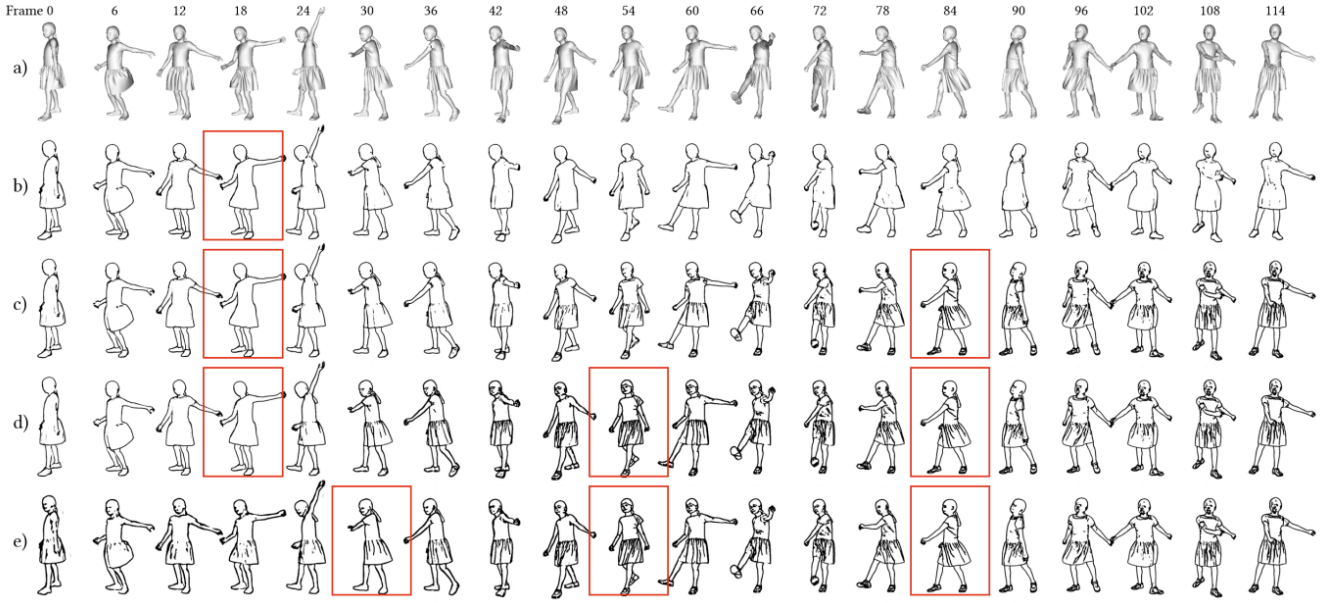
[10]

Now that we know how StyleNET works, lets see how it was implemented with StyleGAN for line synthesis of a 3D model:



[11]

As shown in the diagram above, the authors developed an autoencoder-like framework to learn the latent style from an input line drawing. This was done by en-coding the geometry of the input into a 2D feature map using GeoNet, StyleNET was used to encode the 2D line drawing into a 1D style code and this was all fed into the StyleGAN generator [11]. The following diagram shows the system in action:



[11]

Interesting implementations of StyleGAN such as these show the promising future of StyleGAN.

Comparing other similar methods

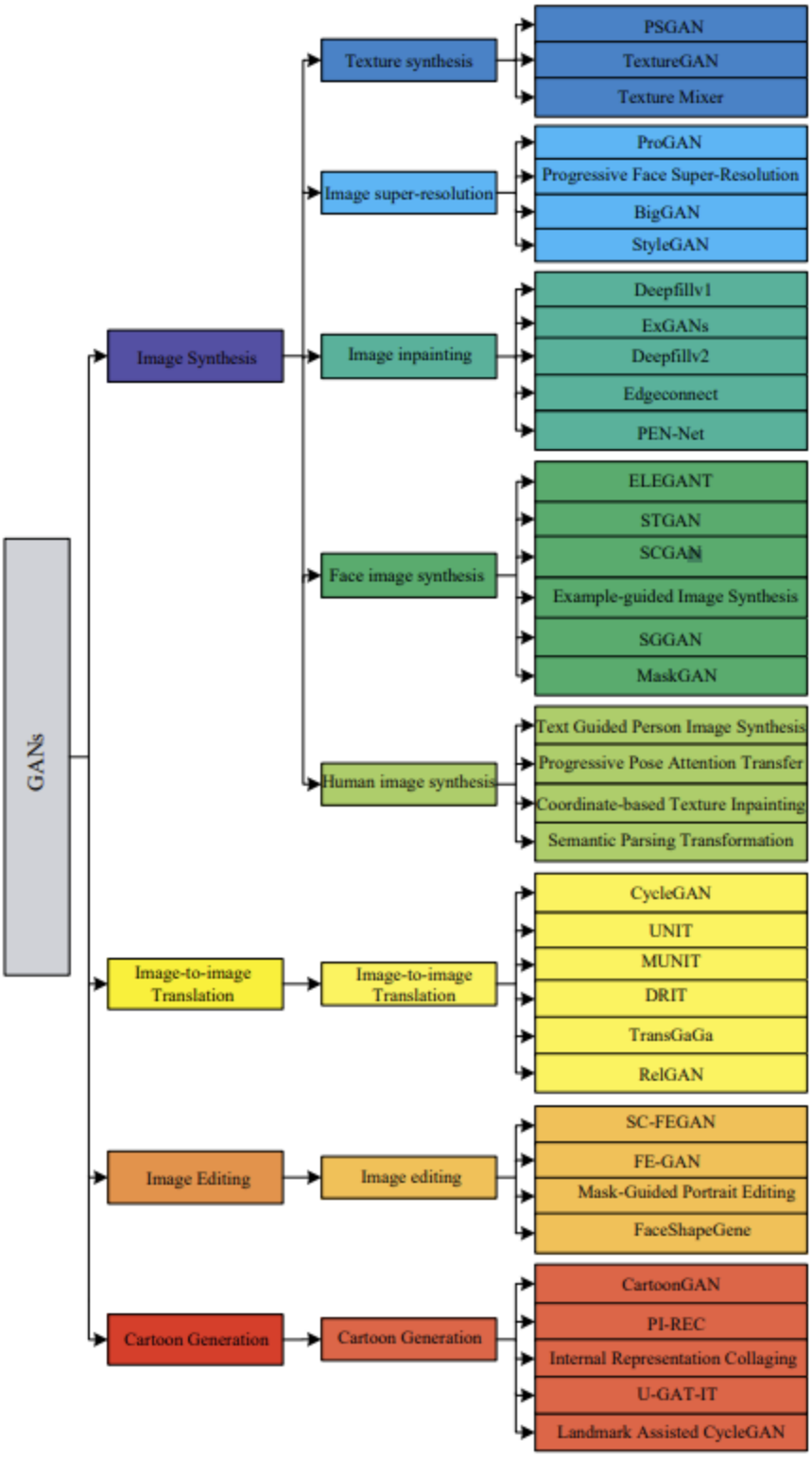
StyleGAN was first introduced in 2018 since then many improvements have been made to it, namely the releases of StyleGAN2 and StyleGAN3. Out of these two new releases, the most notable improvement was in

StyleGAN2, as StyleGAN3 was only an improvement over StyleGAN2 for video purposes (making transitions more natural), only StylaGAN2 will be used as a comparison. StyleGAN2 improved upon the initial StyleGAN implementation by removing the unwanted artefacts generated, and several other changes were made, the changes and their respective FID scores are below:

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	–	–	–	–

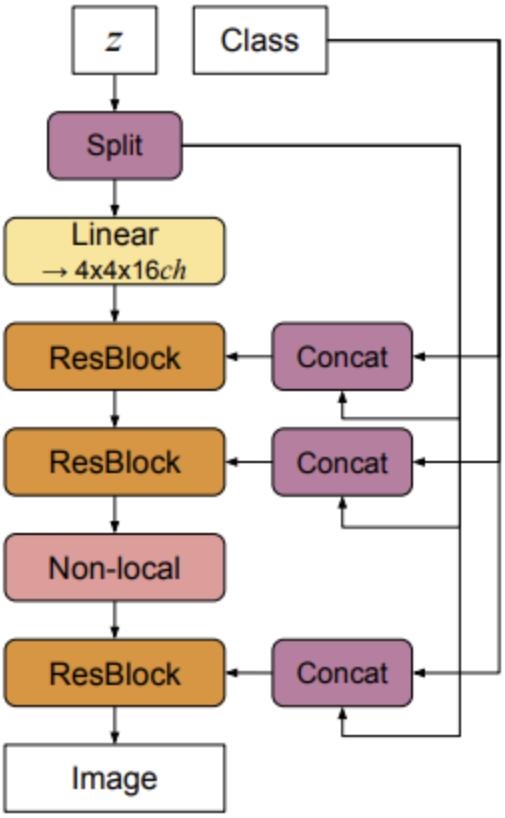
[7]

These changes also lowered the training time of StyleGAN2, in comparison the original StyleGAN trained at 37 images per second on an Nvidia DGX-1 with 8 Tesla V100 GPUs, whereas the StyleGAN2 managed to train at 61 images per second, a full 40% improvement [7]. The main field StyleGANs are used in image synthesis, and many GANs models are available, this graph shows a good representation of this:



[8]

A comparison will be made with BigGAN as it is in the same field as StyleGAN. BigGAN is focused on scaling up the traditional GAN model, it introduces more model parameters, larger batch size and some architectural changes. The BigGAN was trained on JFT-300M, which is one of the biggest datasets it contains over 300 million images from over 15,000 classes, compared to the 1.2 million images and 1,000 classes of ImageNet. The main architectural change made is the following:



[9]

Here a hierarchical latent space is applied, where the latent vector z is split into chunks of equal sizes, and each chunk is concatenated to the shared class embedding. Furthermore, the BigGAN model employs a "truncation trick" which involves setting a threshold on the distribution of the input vector, meaning that if the values fall outside a range they are resampled to fall inside the range. This trades off quality with higher diversity, BigGAN is able to control this. Overall, BigGAN made big improvements to previous models, and when trained on ImageNet its FID score was a low 7.4 [9]. Comparing BigGAN to StyleGAN, StyleGAN has better image quality as it is able to generate images of 1024x1024 pixels in size, compared to 512x512 of BigGAN. Furthermore, there is no style control available in BigGAN, you are not able to control features such as hair color, on the other hand, BigGAN has the advantage in how diverse the output images can be compared to StyleGAN. Overall, both BigGAN and StyleGAN are powerful generative models and both have their strengths and weaknesses, one may be better for certain scenarios, which is the case for most of these generative models.

Conclusion

StyleGAN improved on the "black-box" nature of the generator, which is what it was trying to tackle, however, the model still retains this "black-box" nature. It is still challenging to understand how specific changes in the latent space will affect the generated images. Furthermore, it is still too computationally intensive, however, it is still a very exciting and interesting development, and we can see it being implemented in many fields in the future; as the strategy employed by StyleGAN can also be used for audio and music, it is not limited to image synthesis; however, such implementations have yet to be made due to how recent StyleGAN is, seeing these implementations created and tested will show the true staying power of this model.

Overall, StyleGAN is a huge contribution to the field of deep learning, and this blog has shown its impressive capabilities in image synthesis. StylaGAN has the potential to have a vast impact on a variety of different fields. This method has the potential to completely transform the ways in which we generate and work with digital material if additional study and development are carried out on it.

References

[1] Karras, T., Laine, S. and Aila, T. (2018) “A style-based generator architecture for generative Adversarial Networks,” 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) [Preprint]. Available at: https://arxiv.org/pdf/1812.04948.pdf.

[2] Karras, T., Laine, S., Aila, T., Lehtinen, J. (2017) “Progressive Growing of GANs for Improved Quality, Stability, and Variation". Available at: https://arxiv.org/pdf/1710.10196.pdf.

[3] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2020. Generative adversarial networks. Communications of the ACM, 63(11), pp.139-144. Available at: https://arxiv.org/pdf/1406.2661.pdf.

[4] Overview of gan structure | machine learning | google developers (no date) Google. Google. Available at: https://developers.google.com/machine-learning/gan/gan_structure.

[5] Horev, R. (2019) Explained: A style-based generator architecture for gans - generating and tuning realistic..., Medium. Towards Data Science. Available at: https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431.

[6] NVlabs (no date) NVlabs/stylegan: StyleGAN - Official Tensorflow Implementation, GitHub. Available at: https://github.com/NVlabs/stylegan.

[7] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J. and Aila, T., 2020. Analyzing and improving the image quality of stylegan. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 8110-8119). Available at: https://arxiv.org/pdf/1912.04958.pdf.

[8] Wang, L., Chen, W., Yang, W., Bi, F. and Yu, F.R., 2020. A state-of-the-art review on image synthesis with generative adversarial networks. IEEE Access, 8, pp.63514-63537. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9043519

[9] Brock, A., Donahue, J. and Simonyan, K., 2018. Large scale GAN training for high fidelity natural image synthesis. arXiv preprint arXiv:1809.11096. Available at: https://arxiv.org/pdf/1809.11096.pdf

[10] Gan, C., Gan, Z., He, X., Gao, J. and Deng, L., 2017. Stylenet: Generating attractive visual captions with styles. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3137-3146). Available at: https://openaccess.thecvf.com/content_cvpr_2017/papers/Gan_StyleNet_Generating_Attractive_CVPR_2017_pape

[11] Graphics, P., Yang, Y., Parakkat, A.D., Deng, B. and Noh, S.T., 2022. Learning a Style Space for Interactive Line Drawing Synthesis from Animated 3D Models. Available at: https://diglib.eg.org/bitstream/handle/10.2312/pg20221237/001-006.pdf?sequence=1&isAllowed=y