CS838 Homework Assignment 3

Akshata Bhat
akshatabhat@cs.wisc.edu

Rohit Kumar Sharma
rsharma@cs.wisc.edu

5 December 2019

Part I - Paper Summary

Why is this paper interesting?

Synthesizing artificial images that are realistic and indistinguishable from real-world images is an interesting problem that GANs solve. StyleGAN [KLA18] developed by a research group at NVIDIA generates high quality images and also gives control over some characteristics such as facial expressions, hair style, etc. StyleGAN is also very popular[1] in the timeline of GAN models. Additionally, availability of open-source code and the pre-trained models made us select this paper for the study.

Goals of StyleGAN

- Generate impressive photorealistic high-quality photos of faces.
- Get some control over the style of the generated images at different levels of detail by varying the style vectors and noise.

Contributions of StyleGAN

- Introduction of a standalone ***mapping network*** to control style.
- Using ***Adaptive Instance Normalization (AdaIN)*** [HB17] to integrate style vectors.
- Addition of ***mixing regularization*** to intermix different style vectors.
- ***Perceptual path length*** and ***linear separability*** to quantify disentanglement.
- A new high quality and diverse human faces dataset FFHQ by parsing images from Flickr.

Key Ideas of StyleGAN

The generator takes a learned constant input and adjusts the style of image at each convolution based on the intermediate latent code, resulting in better control over image features at different scales. The per block incorporation of style vector and noise allows each block to localise the interpretation of style and stochastic variation to a given level of detail.

StyleGAN model architecture

Given a latent code $\mathbf{z}$ in the latent space Z, a non-linear mapping network (MLP) produces $w \in W$. Next, the learned affine tranformations specialise w to styles $y = (y_s, y_b)$ which are inputs to AdaIN. AdaIN is applied after each convolution layer of synthesis network. Stochastic variation (Gaussian noise) is generated by introducing noise inputs to each layer of synthesis network. The noise input is broadcasted to all feature maps using learned feature scaling factors and added to corresponding convolution outputs.

[1] As of today, the official GitHub repository https://github.com/NVlabs/stylegan has 8362 stargazers and 1.7k forks.

Experimental setup and results

The training takes one week on Nvidia DGX-1 with 8 Tesla V100 GPUs. For evaluation, Frechect Inception Distance (FID) is used. Ablation studies to study the impact of bilinear upsampling, mapping network, AdaIN operator, addition of noise inputs, mixing regularization is done. The results indicate that the FID values of the images generated are very low demonstrating the high quality of the images.

StyleGAN is also evaluated using the new disentanglement metrics: perceptual path length and linear separability. The results indicate that the intermediate latent-space points generated by StyleGAN can be easily separated.

Part II - Experiment with Code

Implementation Details

The StyleGAN is implemented on top of Progressive GAN in TensorFlow. The model uses same hyperparameters as Progressive GAN. The techniques such as batch normalization, attention, dropout, etc., are not employed. Two loss functions: WGAN-GP and non-saturating loss with R1 regularizer are used in the training. The constant input in the synthesis network is initialised to one. The biases and noise scaling factors are initialised to zero. FFHQ dataset is primarily used for training, it consists of 700,000 high quality images at 1024^2 resolution. Training time of 25M images is used. At a high level, the implementation has the following structure

- Configurations: Configure result, data and cache directory paths.
- Pre-processing dataset: The training and evaluation scripts need the datasets to be stored as multi-resolution TFRecords. `dataset_tool.py` converts datasets into the required format.
- Training: Dataset and training configurations are set in `train.py`. Disabling individual features (pixel normalization, leaky ReLU, smoothing, minibatch repeats), setting numerical precision (`fp32` or `fp16`), number of GPUs and debugging options can be configured in this file. The results are written to a newly created directory `results`.
- Evaluation: `run_metrics.py` contains the quality and disentanglement metrics (FID, Perceptual Path Length and Linear Separability).
- Pre-trained Networks: Images can be generated by loading the corresponding pre-trained models as given in `pretrained_example.py`. Style mixing, truncation and effect of noise input can be explored using `generate_figures.py`
- The `training` directory contains the model networks. `networks_stylegan.py` contains modules for synthesis, mapping, base generator and discriminator part of the network. `training_loop.py` contains the code that builds tensorflow graph, loops through batches, saves snapshots, etc. This directory also contains separate files for computing loss and loading dataset.
- The `dnnlib` directory contains code for optimization, network abstraction, utility libraries, etc.
- The `metrics` directory contains the implementation for the metrics mentioned above.

Tricks in Implementation

- *Truncation trick* is used in the intermediate latent space W. The center of mass is computed, in case of FFHQ, this point represents an average face. The deviation of a point in intermediate latent space is scaled from the center.
- Increasing the depth of mapping network tends to make the training unstable with high learning rates. Hence, StyleGAN reduces the learning rate by an order of two for the mapping network.
- Run-time speed is increased by using half-precision floating point arithmetic.

Figure 1: Faces generated using pre-trained StyleGAN

Experimenting with the pre-trained model

We used the pre-trained StyleGAN model trained on FFHQ, Cars and Bedrooms datasets and generated the images shown in figures 1, 2 and 3 respectively.

As training the model from scratch on a different dataset takes a lot of time and resources (around 1 week with 8 Nvidia Tesla V100 GPUs), and as we are limited by the resources and time to train such a model, we did not train the model from scratch. Instead, we used the pre-trained model to perform some experiments such as style-mixing.

To perform the experiments described in this section, we had to do some code modifications:

- Modification to the `pretrained_example.py` file to randomly sample seeds to generate latent representations that help in generating images using StyleGAN's pre-trained models for various data sets such as FFHQ, Cars and Bedrooms.
- Using perceptual loss trick to generate latent representation of custom images (our own images and some celebrities) as performed in StyleGAN Encoder repository[2].
- Modification to the `generate_figures.py` script to use the custom latent representations instead of randomly sampling and applying style-mixing on them to generate stylized images as shown in fig 4.

Discussion of results The generated images from the pre-trained models shown in figures 1, 2 and 3 look realistic. However there are a few artifacts introduced in some of the images. These are especially clear in some bedroom images and in some car images. The reason for such artifacts are variations in camera angles, repeated textures, zoom levels and background clutter as compared to the faces dataset.

The style-mixing results shown in figure 4 still resemble realistic images in some ways but they are not as good as the results shown in the paper. This is because of the difference in the approach used to generate them: in StyleGAN, random latent representations are sampled to create style-vectors using the mapping network. But, in our approach, we generated these latent representations from our own custom

[2]https://github.com/Puzer/stylegan-encoder

Figure 2: Cars generated using pre-trained StyleGAN

Figure 3: Bedrooms generated using pre-trained StyleGAN

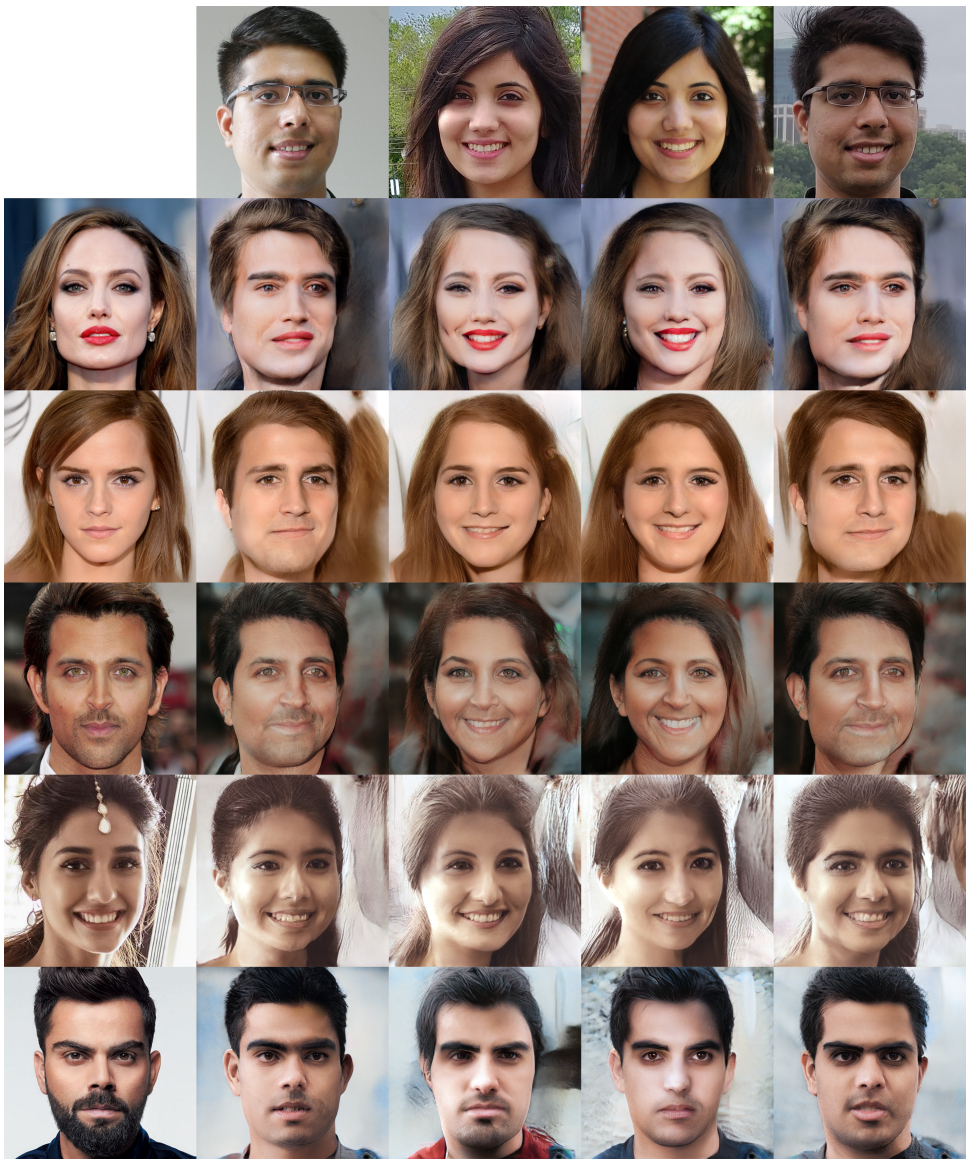

Figure 4: Style-mixing using our custom images and celebrities

images using the perceptual loss trick as described above. This is why the images do not look as realistic as presented in the paper.

Team Members and Contribution

In general, both the team members contributed equally by collaborating in all aspects. On a high level, the individual contributions can be broken down as follows:

Name	Contributions
Rohit Kumar Sharma	• Reading the StyleGAN paper • Understanding the StyleGAN code layout • Generating sample images using the pre-trained StyleGAN models • Studying about perceptual loss trick to generate latent representations from custom images • Using our custom images to perform style-mixing • Writeup (Part 1 - Why this paper, Goals, Contributions, Experimental setup; Part 2 - Custom Experiments and Discussion, Conclusion)
Akshata Bhat	• Reading the StyleGAN paper • Understanding the StyleGAN code layout • Generating sample images using the pre-trained StyleGAN models • Studying about perceptual loss trick to generate latent representations from custom images • Using our custom images to perform style-mixing • Writeup (Part 1 - Key Ideas, StyleGAN architecture; Part 2 - Implementation details, Code organization, Tricks in implementation)

Conclusion

StyleGAN is a Generative Adversarial Networks model that succeeds in producing photo-realistic high resolution images as described in this report due to some novel techniques employed. Not only does StyleGAN produce realistic images, but also gives the user some control over various aspects of the generated images at various levels of detail.

References

[HB17] Xun Huang and Serge Belongie. "Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (Oct. 2017). DOI: 10.1109/iccv.2017.167. URL: http://dx.doi.org/10.1109/ICCV.2017.167.

[KLA18] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2018. arXiv: 1812.04948 [cs.NE].