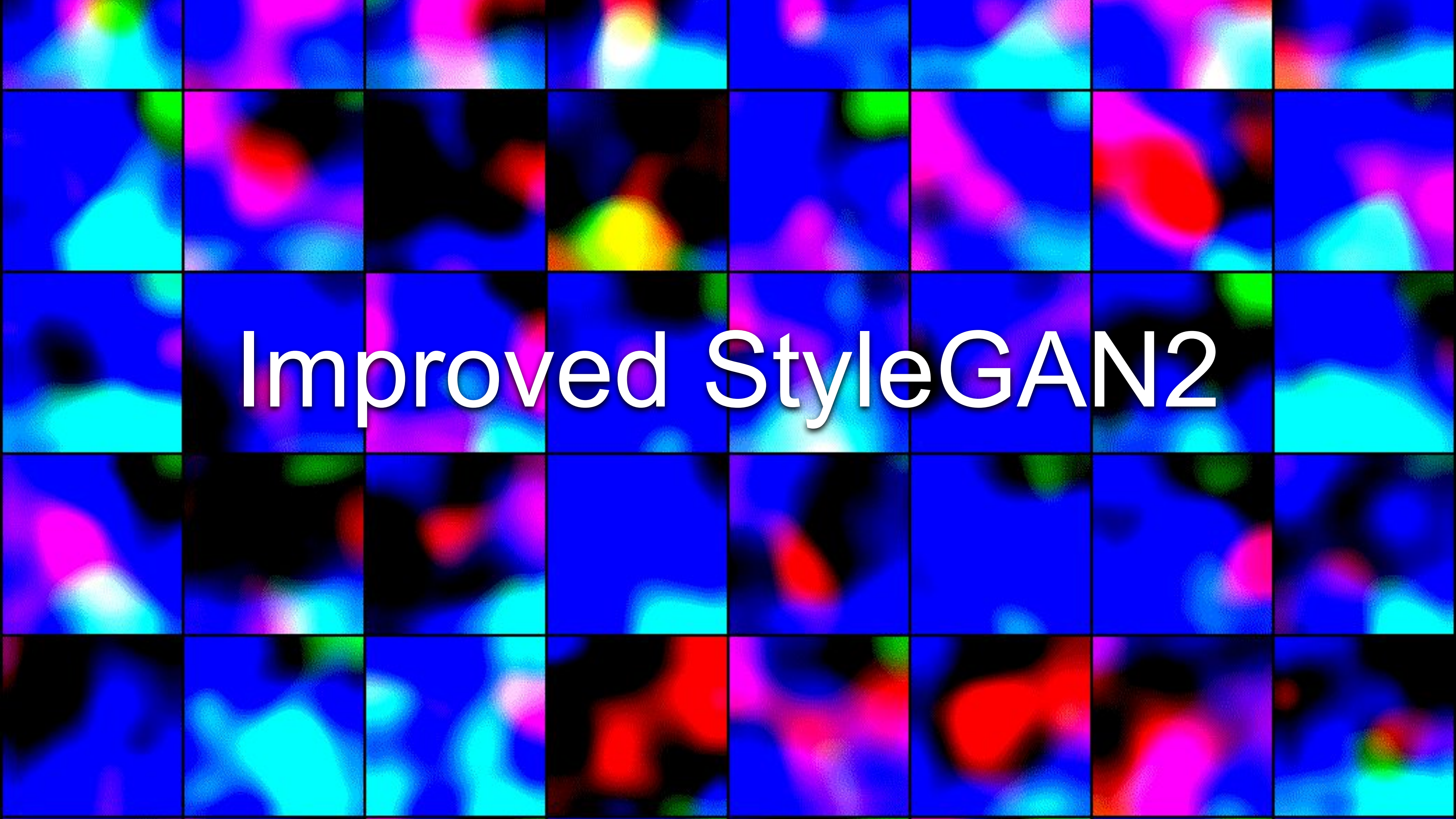
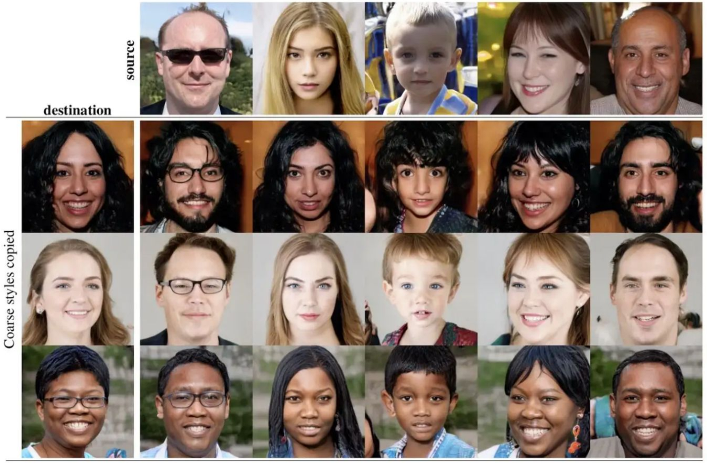
Improved StyleGAN2



What is StyleGAN2?

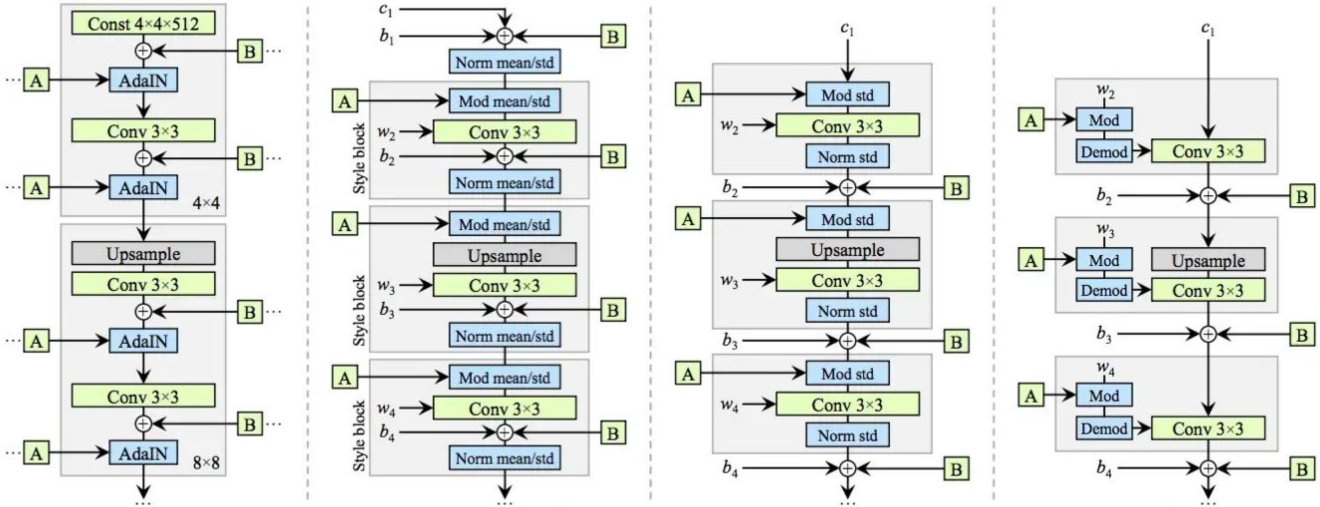
- A state-of-the-art Generative Adversarial Network
- Developed by researchers at NVIDIA
    - ProGAN -> StyleGAN -> StyleGAN2
        - Progressive growing (ProGAN)
        - Adaptive Instance Normalization: AdaIN (StyleGAN)
        - Weight demodulation (StyleGAN2)

- Diminishes artifacts/blobs in generations
    - Increasing resolution layers
    - Weight demodulation (normalization)
        - Improves speed



StyleGAN2 Improvements

- Restructures StyleGAN architecture
    - Replaces AdaIN, separates style blocks with noise maps, increases parallelization
        - Revised architecture ~40% faster, fewer artifacts



(a) StyleGAN (b) StyleGAN (detailed) (c) Revised architecture (d) Weight demodulation

Competitive Architectures

- Skip connections
  - MSG-GAN: Multi-Scale Gradients for Generative Adversarial Networks
    - Improves gradient passing between discriminator and generator

- Residual networks
  - Improved Training of Wasserstein GANs
    - Gradient penalty instead of weight clipping
    - Can train deep residual networks in GAN settings

- Hierarchical methods
  - StackGAN: Text to Photo-realistic Image Synthesis
    - Multiple GANs work together in a sketch-refinement process

StyleGAN2 Methodology

- Analyze StyleGAN limitations
  - Implement improvements
  - Compare architectures
- Datasets
  - FFHQ, 70k images, 1024x1024
  - LSUN Car, 893k images, 512x384
- Metrics
  - Frechet inception distance (FID)
    - Differences of distribution densities in feature space
  - Path length, precision, recall

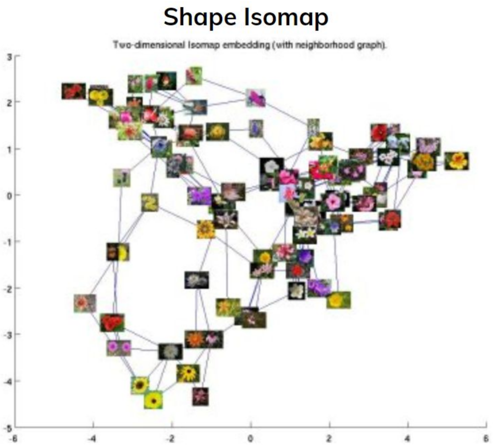
StyleGAN2 Results

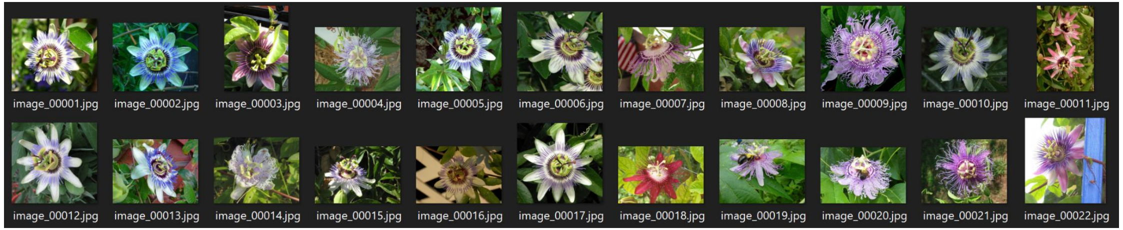
	Configuration	FFHQ, 1024×1024				LSUN Car, 512×384			
		FID ↓	Path length ↓	Precision ↑	Recall ↑	FID ↓	Path length ↓	Precision ↑	Recall ↑
A	Baseline StyleGAN [24]	4.40	212.1	**0.721**	0.399	3.27	1484.5	**0.701**	0.435
B	+ Weight demodulation	4.39	175.4	0.702	0.425	3.04	862.4	0.685	0.488
C	+ Lazy regularization	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	–	–	–	–

- Path Length
  - estimates quality of latent space interpolations
- StyleGAN2 improves on all metrics except precision

Simulating StyleGAN2

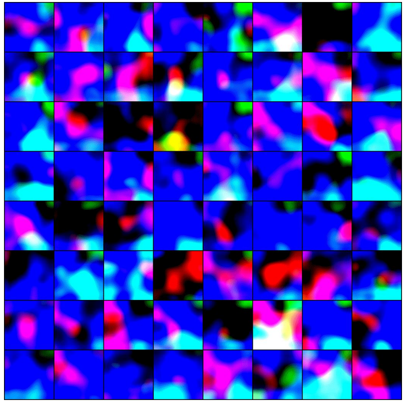
- Original StyleGAN2 implementation is in TensorFlow
  - https://github.com/NVlabs/stylegan2
- Third-party Pytorch implementation:
  - https://github.com/lucidrains/stylegan2-pytorch
- Dataset
  - oxford_flowers102
    - 102 different flowers, 1,000 images, 128x128
  - Smaller sample size due to less computing power





Results

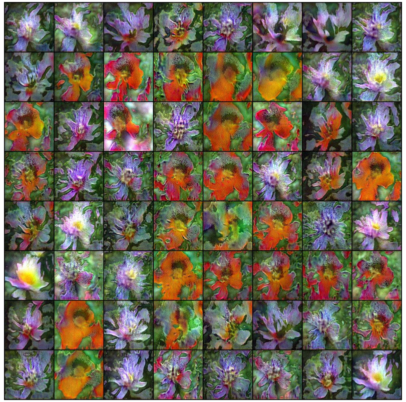
- Trained for 5,000 iterations
  - ~10 hours training time
  - Checkpoints every 1,000 iterations
  - Outputs show improving quality



1,000

3,000

4,000



StyleGAN2 More Details

- Has two main components:
  - Generator - Generates artificial images based on input images
  - Discriminator - classifies images
    - Classifies between “true” (real image) and “generated” (fake image)

- Both generator and discriminator use the Adam optimizer
- Generator has lr of 2e-4
- Discriminator has lr of 4e-4

Improving StyleGAN2

- StyleGAN2 implementations do not support 3D image data
- 3D generations important for:
  - Molecular and material design
  - Magnetic resonance imaging
  - CAD and modeling

- Requires
  - 3D convolutions, transposes, filters, kernels, strides
  - 3D upsampling, image channel adjustments

Improving StyleGAN2: Methodology

- Methodology is the exact same of StyleGAN2
  - With addition of 3D capabilities

- Same:
  - Base architecture structure
  - Style transfer capabilities

- Different:
  - Network parameters
  - Data channels
  - Image dimensionality

Code Adjustments

```python
class LinearAttention(nn.Module):
    def __init__(self, dim, dim_head=64, heads=8):
        super().__init__()
        ...
        self.to_q = nn.Conv2d(dim, inner_dim, 1, bias=False)
        self.to_kv = DepthWiseConv2d(dim, inner_dim * 2, 3, padding=1, bias=False)
        self.to_out = nn.Conv2d(inner_dim, dim, 1)

    def forward(self, fmap):
        ...

# one layer of self-attention and feedforward, for images

attn_and_ff = lambda chan: nn.Sequential(*[
    Residual(PreNorm(chan, LinearAttention(chan))),
    Residual(PreNorm(chan, nn.Sequential(nn.Conv2d(chan, chan * 2, 1), leaky_relu(),
    nn.Conv2d(chan * 2, chan, 1))))])
```

```python
class LinearAttention(nn.Module):
    def __init__(self, dim, dim_head=64, heads=8):
        super().__init__()
        ...
        self.to_q = nn.Conv3d(dim, inner_dim, kernel_size=(1, 1, 1), bias=False)
        self.to_kv = DepthWiseConv3d(dim, inner_dim * 2, 3, padding=1, bias=False)
        self.to_out = nn.Conv3d(inner_dim, dim, kernel_size=(1, 1, 1))

    def forward(self, fmap):
        ...

# one layer of self-attention and feedforward, for images

attn_and_ff = lambda chan: nn.Sequential(*[
    Residual(PreNorm(chan, LinearAttention(chan))),
    Residual(PreNorm(chan, nn.Sequential(nn.Conv3d(chan, chan * 2, 1), leaky_relu(),
    nn.Conv3d(chan * 2, chan, 1))))])
```

```python
class DepthWiseConv2d(nn.Module):
    def __init__(self, dim_in, dim_out, kernel_size, padding=0, stride=1, bias=True):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(dim_in, dim_in, kernel_size=kernel_size, padding=padding,
            groups=dim_in, stride=stride, bias=bias),
            nn.Conv2d(dim_in, dim_out, kernel_size=1, bias=bias))

    def forward(self, x):
        return self.net(x)
```

```python
class DepthWiseConv3d(nn.Module):
    def __init__(self, dim_in, dim_out, kernel_size, padding=1, stride=1, bias=True):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv3d(dim_in, dim_in, kernel_size=(3, 3, 3), padding=padding,
            groups=dim_in, stride=(1, 1, 1), bias=bias),
            nn.Conv3d(dim_in, dim_out, kernel_size=(1, 1, 1), bias=bias))

    def forward(self, x):
        return self.net(x)
```

Code Adjustments Cont.

- Handles ‘n’ data channels instead of only 3 RGB channels

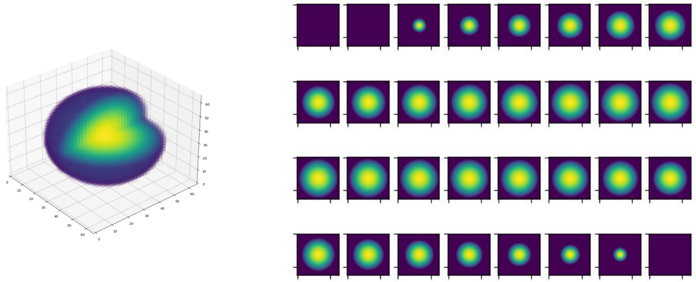
```
class RGBBlock(nn.Module):
    def __init__(self, latent_dim, input_channel, upsample, rgba=False):
        super().__init__()
        ...
        out_filters = 3 if not rgba else 4
        self.conv = Conv2DMod(input_channel, out_filters, 1, demod=False)

        self.upsample = nn.Sequential(
            nn.Upsample(scale_factor=2, mode='bilinear', align_corners=False),
            Blur()
        ) if upsample else None

    def forward(self, x, prev_rgb, istyle):
        b, c, h, w = x.shape
        ...
```

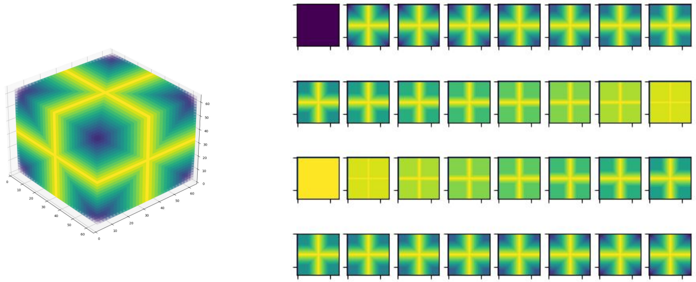
```
class DCBlock(nn.Module):
    def __init__(self, latent_dim, input_channel, upsample, rgba=False):
        super().__init__()
        ...
        out_filters = 2
        self.conv = Conv3DMod(input_channel, out_filters, 1, demod=False)

        self.upsample = nn.Sequential(
            nn.Upsample(scale_factor=2, mode='trilinear', align_corners=False),
            Blur()
        ) if upsample else None

    def forward(self, x, prev_rgb, istyle):
        b, c, de, h, w = x.shape
        ...
```

Experimental Setup

- Dataset
  - Synthetic 3D energy grids, 32x32x32
  - 2 channels:
    - 1st: Spherical
    - 2nd: Cubic
  - 7000 samples

- Trained for 2,000 iterations
  - ~8 hours
  - 3D data is slower
    - More information to process

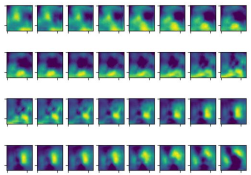




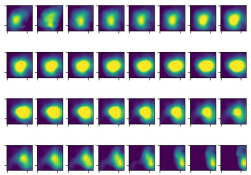
Experimental Results

- Decreasing generator loss
- Learns both energy distributions over time
- Some additional noise in results
  - Possible causes:
    - low training iterations
    - Leftover noise from weight initializations

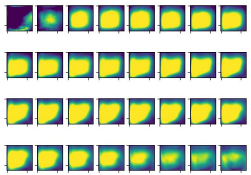
iter 1,000 sphere

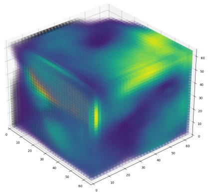


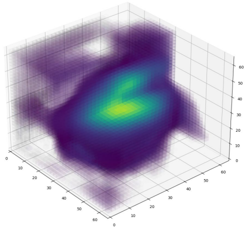
iter 2,000 sphere



iter 2,000 cube







```
G: 8.23 | D: 0.00 | GP: 0.12
G: 48.25 | D: 2.57 | GP: 2234.11
G: 3.11 | D: 0.57 | GP: 0.14
G: 19.89 | D: 0.02 | GP: 0.16
G: 5.76 | D: 2.23 | GP: 334.03
G: 4.01 | D: 1.60 | GP: 8.12
G: 2.96 | D: 0.28 | GP: 1.68
G: 2.04 | D: 0.81 | GP: 1.25
G: -0.94 | D: 1.02 | GP: 143.31
G: 13.75 | D: 0.09 | GP: 0.27
G: 5.13 | D: 1.06 | GP: 33.70
G: -1.75 | D: 7.01 | GP: 8.39
G: 1.75 | D: 0.47 | GP: 1.32
G: 3.61 | D: 0.03 | GP: 0.37
G: 1.76 | D: 3.31 | GP: 0.50
G: 2.46 | D: 0.35 | GP: 0.22
G: 6.83 | D: 0.05 | GP: 0.15
G: 2.42 | D: 0.20 | GP: 4.22
G: 0.89 | D: 1.00 | GP: 0.92
```

Simulation and Experimentation Discussion

- Able to reproduce original results
- 3D version of StyleGAN2 is available at:
  - https://github.com/jbarks1234/improved-stylegan2

- In the end, StyleGAN2 is a very useful and powerful GAN
  - Has uses in video games for auto generating cities or other models
  - Also used in auto generating voices

- We showed:
  - StyleGAN2's results can be reproduced on new datasets
  - StyleGAN2's architecture can be improved for 3D samples

References

- Prior Works: https://www.qblocks.cloud/creators/synthesize-high-resolution-images-with-stylegan2
- Flower Image Dataset: https://www.robots.ox.ac.uk/~vgg/data/flowers/17/index.html
- Original StyleGAN: https://github.com/lucidrains/stylegan2-pytorch/blob/master/stylegan2_pytorch/stylegan2_pytorch.py
- Our improved StyleGAN2: https://github.com/jbarks1234/improved-stylegan2


Thank You! Any questions?