Semantic Editing using Style GAN

Dantu Venkata Sai Kamal[1]

Bharat Institute of Engineering and Technology, Hyderabad ; Telangana, India
Email: dantusaikamal@gmail.com

Abstract

Generative Adversarial Networks, commonly referred to as GANs are a type of neural network architecture for generative modeling which have grown popular in recent times. Generative Adversarial Networks were initially proposed in 2014 and have been widely used for specific use cases.

GANs are used for generative modeling which involves generate new examples from an existing distribution of samples. For example, new faces can be generated from existing dataset of faces.

The StyleGAN is a continuation of the progressive, developing GAN that is a proposition for training generator models to synthesize enormous high-quality photographs via the incremental development of both discriminator and generator models from minute to extensive pictures. [2]

We are building a StyleGAN encoder that is capable of generating different hair styles for any given face. Such encoders can be used for various applications. For example, checking out a new hairstyle before getting an actual haircut.

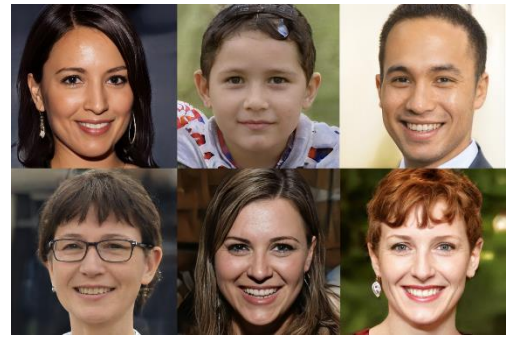
Keywords: Generative adversarial network, StyleGAN, Deep Learning

1. Introduction

GANs are an unsupervised model for generating new elements from a set of similar elements. For instance, to produce original face pictures given a collection of face images or create new tunes out of preexisting melodies.

GANs have found applications for image, text, and sound generation, being at the core of technologies such as AI music, deep fakes, and content-aware image editing. Besides pure generation, GANs have also been applied to transforming images from one domain to another and as a means for style transfer. To add one more application, they fit as an intelligent data augmentation technique for semi-supervised learning [3]

A typical example of a GAN application is to produce artificial face pictures by learning from a dataset of notable faces.



Artificially generated faces using StyleGAN

While GAN images became further vivid over time, one of their main hurdles is regulating their output, such as replacing explicit features such as pose, face shape, and hairstyle in an illustration of a face.

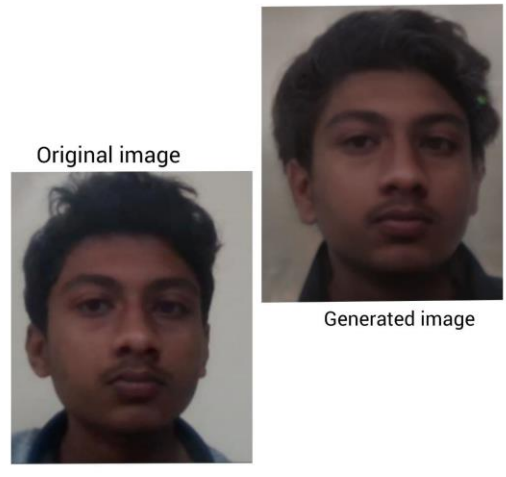
The Style Generative Adversarial Network, or StyleGAN for short, is an addition to the GAN architecture that introduces significant modifications to the generator model. StyleGAN produces the simulated image sequentially, originating from a simple resolution and enlarging to a huge resolution (1024×1024).

By transforming the input of each level individually, it examines the visual features that are manifested in that level, from standard features (pose, face shape) to minute details (hair color), without altering other levels.

The resulting model is proficient in producing impressively photorealistic high-quality photos of faces and grants control over the characteristic of the created image at different specification levels by changing the style vectors and noise. [4]

2. Motivation and contribution

The main objective behind this project is to synthesize different hairstyles for faces using StyleGAN. Researchers demonstrate that the generator's radical redesign or compromise produced image quality but considerably enhances it. So, there does not exist a trade-off between picture quality and interpolation skills.



In this way, we can synthesize enormous high-quality photographs of faces with various hairstyles via the incremental development of both discriminator and generator models from minute to extensive pictures.

.

We can edit one attribute of the human face by finding the hyperplane boundary in the latent space, which can generate amazing yet not perfect results. So far, we can set one attribute as a conditional attribute along with the primary attribute, as discussed in InterfaceGAN.

Additionally, when using one attribute to edit our face, some other attributes may also be changed because of their correlations. We believe using a better classifier can control more than two conditions at the same time and make the boundary more explicit. [5]

The features can be divided into three types:

- Coarse - resolution of up to 82 - affects pose, general hair style, face shape, etc
- Middle - resolution of 162 to 322 - affects finer facial features, hair style, eyes open/closed, etc.
- Fine - resolution of 642 to 10242 - affects color scheme (eye, hair and skin) and micro features.

The Mapping Network's goal is to encode the input vector into an intermediate vector whose different elements control different visual features.

3. Implementation:

Set up your Google Collab environment and select a runtime environment to run the model.
a. To run the model, click *Runtime* from the top and select *Run all cells*.
b. Once the model begins to run, scroll to *Get images* section, and click on the visual input you see on screen to capture an image.
c. Check the contents of the image in the next section and double click to edit the captured image.
d. Wait for the resultant image to be generated.

1. Install the required dependencies:

The first cell contains the following code:

```
!pip install 'h5py<3.0.0'   # h5py is used to store & manipulate Numeric data
!pip install --upgrade tqdm     # tqdm is used for creating progress bars
```

The above dependencies must be manually installed in the collab environment since the default versions of h5py and tqdm are not supported by StyleGAN.

Similarly, TensorFlow version ***1.14.x / 1.15.x*** must strictly be used in the environment.

2. Clone Git repo and create directories for importing images

Clone the repository to use the sample dataset and also the latent representations, latent space markings and the adaptive loss functions.

```
!git clone https://github.com/Azmarie/stylegan-encoder.git
mkdir aligned_images raw_images
```

3. Get images

To capture image from live video input, we are using multiple modules in Python.

Firstly, IPython.display is used to execute HTML code. We need it in order to further execute JavaScript code inside HTML script tags. The live video that is streamed inside the browser is developed using JavaScript methods.

The *MediaDevices.getUserMedia()* method prompts the user for permission to use a media input which produces a MediaStream with tracks containing the requested types of media. The JS listens for a click on the button, then calls navigator.mediaDevices.getUserMedia() asking for the video.

Next, to display the image, pillow library is using an image class within it. The image module inside pillow package contains some important inbuilt functions like, load images or create new images, etc.

Once the camera starts capturing, we are using *var canvas = document.createElement('canvas')*
To create a canvas of the size we require from the live video capture. This canvas is stored in the variable data which is then used in take_photo method to create a snapshot of the canvas.

HTMLCanvasElement.toDataURL() method returns a data URI containing a representation of the image in the format specified by the type parameter. Here, our desired type is jpeg

We are also assigning the timestamp to every image that is capture using the following block:

```
timestampStr = datetime.now().strftime("%d-%b-%Y (%H:%M:%S.%f)")
filename = 'raw_images/photo_%s.jpeg' %timestampStr
```

Example of an image captured*: photo_17-Dec-2021 (19:10:21.422345).jpeg*

The take_photo method takes quality and resolution as parameters and returns a snapshot of the canvas. And the snapshot is displayed in the browser.

4. Check the contents of our image folder before we start:

Before we proceed further, let us check whether or not the image captured earlier is saved to the ‘raw_images’ directory we created in the beginning. If the captured image is present, we can go ahead.

5. Auto-Align faces

The next step is to crop the image and eliminate unnecessary portion of the image. In the canvas, we are going to crop the image and align the face into center of the image. This will result into an aligned image of size 1024*1024.

- Look for faces in the images
- Crop out the faces from the images
- Align the faces (center the nose and make the eyes horizontal)
- Rescale the resulting images and save them in "aligned_images" folder

This is done by executing align_images.py file which uses Keras, argparse and multiprocessing to detect the landmarks and features in the face and align the image accordingly. The LandmarksDetector class is pre-trained trained on similar dataset, and is used to extract features in the image. Then, the aligned image is printed. We say two models are aligned if they share the same architecture, and one of them (the child) is obtained from the other (the parent) via fine-tuning to another domain, a common practice in transfer learning.

6. Encoding faces into StyleGAN latent space:

a. Download a pretrained ResNet encoder

```
!gdown https://drive.google.com/uc?id=1aT59NFy9-bNyXjDuZOTMl0qX0jmZc6Zb
!mkdir data
!mv finetuned_resnet.h5 data
```

The pretrained StyleGAN network from NVIDIA trained on faces and a pretrained VGG-16 network, trained on ImageNet will be downloaded. After guessing the initial latent codes using the pretrained ResNet, it will run gradient descent to optimize the latent faces

b. Generate encoded images using TensorFlow

```
!python encode_images.py --optimizer=lbfgs --face_mask=False --iterations=50 --use_lpips_loss=0 --use_discriminator_loss=0 --output_video=True aligned_images/ generated_images/ latent_representations/
```

Since the execution takes a heavy toll on GPU of the system, the fast version and slow versions can be used accordingly to the requirements.

The slow version takes additional parameters including decay rate, decay steps, early stopping threshold, average best loss, and has 8 times more number of iterations resulting in a high accuracy encoded image of the aligned image.

```
!python encode_images.py --optimizer=adam --lr=0.02 --decay_rate=0.95 --decay_steps=6 --         use_l1_penalty=0.3 --face_mask=True --iterations=400 --early_stopping=True --    early_stopping_threshold=0.05 --average_best_loss=0.5 --use_lpips_loss=0 -- use_discriminator_loss=0 --output_video=True aligned_images/ generated_images/  latent_representations/
```

7. Generating output from Encoded image:

Dnnlib is a standalone library used by Nvidia, hence there is no public documentation available. But this lib was made for parsing and easy configuration, and also participates in creating and managing TensorFlow sessions

The StyleGAN model is loaded into TensorFlow session and is executed using tflib.init_tf().

Lastly, added noise is included in the network to generate more stochastic details in the images. This noise is just a single-channel picture consisting of uncorrelated Gaussian noise. Before each AdaIN operation, noise is supplied at a specific convolutional layer. Additionally, there is a scaling part for the noise, which is decided per feature.

A separate sample of noise for each block is evaluated based on the scaling factors of that layer.

```
synthesis_kwargs = dict(output_transform=dict(func=tflib.convert_images_to_uint8, nchw_to_nhwc=True), minibatch_size=1)
```

6 .

Kwargs, Or keyword arguments let our synthesis function take an arbitrary number of keyword arguments. The tflib is a low-level library used to convert images to an 8-bit unsigned integer while the output transform argument is a helper argument.

The remaining keyword arguments are optional and can be used to further modify the operation. The output is a batch of images, whose format is dictated by the output_transform argument

Then, the *generator.run* method is used to generate the synthesized images and are plotted using matplotlib library.

8. Final steps

We can compare the original image with encoded image and the generated images. Also, store the same images to the disk or mount GDrive and store the images to a folder on Google Drive.

4. Applications:

1. This model can be used to generate examples for Image Datasets. This includes generation of fake faces to train Machine Learning models and neural networks in order to eliminate bias, overfitting and other common problems.
2. Checking out a new hairstyle before getting an actual haircut.
3. This GAN can be used in generating Cartoon characters and Anime characters. A huge number of people are using various applications and filters to generate synthesized images of their faces in cartoon style. It can be achieved seamlessly through this model.
4. Generating realistic photographs by training the ResNet neural network with a sample dataset of very high quality and realistic images.

5. Conclusion:

Generative adversarial networks are effective at generating high-quality and large-resolution synthetic images.

The generator model takes as input a point from latent space and generates an image. This model is trained by a second model, called the discriminator, that learns to differentiate real images from the training dataset from fake images generated by the generator model. As such, the two models compete in an adversarial game and find a balance or equilibrium during the training process.

The architecture of StyleGAN model GAN model that introduces control over the style of generated images at different levels of detail. [6]

As a result, Impressive results can be achieved with the StyleGAN architecture when used to generate synthetic hairstyles and human faces.

References

1. https://dantusaikamal.github.io/
2. https://www.analyticsvidhya.com/blog/2021/05/stylegan-explained-in-less-than-five-minutes/
3. https://towardsdatascience.com/gan-papers-to-read-in-2020-2c708af5c0a4
4. https://github.com/Dantusaikamal/Telecom-users-churn-analysis
5. https://www.sfu.ca/~ysa195/projects/CMPT743Project/
6. https://machinelearningmastery.com/introduction-to-style-generative-adversarial-network-stylegan/