Amanda Ahn

Final Report:
Image Filter using Generative Adversarial Networks

Introduction

Background

Digital image processing is very prevalent in society today and only continues to grow as society spends more and more time online. The creation and application of image filters play a crucial role in enhancing and manipulating images for various purposes. Filters have wide-ranging applications in several industries. For example, social media companies, such as instagram, snapchat, and tiktok utilize image filters for the purpose of entertainment and fulfilling users' wants to look a desired way or alter their appearance in an amusing manner. Video/photo editing softwares, such as VSCO and Adobe, give the opportunity to enhance and transform images. UI/UX Design is another rising industry that uses applications and platforms (Figma, Adobe, etc.) that weigh great importance on image processing. Even telecommunication involves image processing as people blur out their backgrounds on zoom or touch up their skin complexion. These are the main industries involved, but this does not exclude companies involving technology and security. Facial recognition is widely used to unlock our phones or identify criminals. Overall, research in image processing benefits several industries and discovers ways to enhance and manipulate digital images effectively, while also opening new avenues for creativity and analysis.

Goal/Problem

The goal of this project is to explore and develop an image processing filter that manipulates the individual components of an image, while enhancing the quality, aesthetics, and utility. Training and implementing filter techniques using GANs, while addressing specific image processing requirements, will give way to more opportunities to discover how to improve digital image transformations. Generating a one-shot fast learned image filter that alters image attributes to transform to a certain style, while simultaneously minimizing undesirable distortions, creates a valuable manipulation that can be used by all people and industries.

Audience

Our selected audience would be industries in entertainment, media, photography, design, and telecommunication. Designing new image filters using GANs and fine-tuning methods will assist in improving companies' image processing softwares and application of filters. Additionally, this project is also geared towards society and individuals who use these image filters, whether it be for professional or personal, creative use.

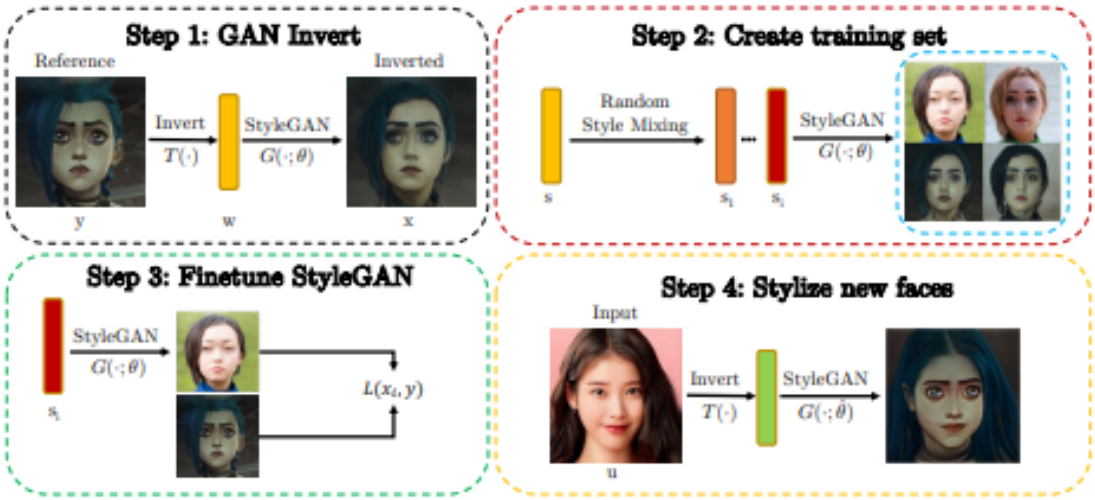
Data

Code was borrowed from JoJoGAN and e4e github repositories. JoJoGAN contains the generator, discriminator, and predictor networks that are used in training desired style using

style reference images. E4e is an encoder for styleGAN image manipulation that is also implemented in JoJoGAN, but is used as a baseline for the simple styleGAN, before fine tuning. The Input image of celebrity Nina Dobrev and style reference images of watercolor paintings were both retrieved from Google.

Overview

Using PyTorch as the neural network framework, this code uses one-shot learning techniques and GAN inversions that fine-tune a pre-trained StyleGAN with only one example of a particular image style. Essentially, the process learns a style mapper from only one reference style image, which is beneficial because collecting data sets per style would be too time consuming. StyleGANs learn image to image translation fairly fast with a relatively small dataset, but obtaining a small dataset per different style would be difficult; therefore, utilizing this code's method results in faster and more quality results.



Networks

Generator

The generator class is responsible for creating images based on a given latent vector and style information. First, there is the initialization which has the 'init' constructor that takes in the resolution of generated images, dimension of style vector, number of multi-layer perceptron layers, scaling factor for number of channels in model, a blur kernel list for convolution, and the learning rate. The class starts by creating a style multi-layer perceptron network that has several layers that transform the style vector to get all the different features and information from the image. Initially, the first layers of this generator class are the constant input layer, convolutional layer, and toRGB layer, which all have low-resolution images. Then, it iterates through a loop over different resolution levels, while upsampling the image, adding a pair of convolutional and RBG layers for each resolution that are stored in ModuleList. Throughout this looping, the noise is injected to make the images more diverse and reduce overfitting. During the process of

looping and upsampling, the image goes through a series of layers that are defined outside the generator class. These layers are PixelNorm, EqualLinear, ConstantInput, NoiseInjection, Styleconv, ModulatedConv2d, Upsample, Downsample, Blur, ToRGB. PixelNorm basically stabilizes the training process and gives more control over generated images. Equallinear essentially does the same by balancing the weights of linear layers to be scaled based on the number of inputs to the layer. ConstantInput is a tensor input that remains constant in the generator network. StyleConv layer includes the modulatedconv2d layer and combines convolution with style modulation, which allows the generator to create images with a specific style. This layer also takes in upsample, used to increase resolution of image as it goes through the loop, downsample, decreasing the resolution, and blur, smooths out the transitions. ToRGB layer is the final layer that transforms all these generated images at different resolutions to RGB images.

Discriminator

The discriminator network serves to distinguish between the real images and the new, generated images. It takes in image inputs and produces feature maps that evaluate the authenticity of the input images. First, the init constructor takes in size, the resolution of the input images, the multiplier of the number of channels in the model, and the blur kernel for convolution. The ConvLayer class creates the first single convolutional layer, which takes in 3 input RGB channels and output channels. Then the log_size is calculated to determine the number of layers the discriminator will have based on the looped downsampling in the network. Then, it enters a loop that creates a series of residual blocks. This loop goes through the highest to lowest resolution images, and each residual block applies this convolution process and stores the resulting layers in a ModuleList. Lastly, the final forward method contains a tuple of indices of all the layers where the outputs will be collected. The code iterates through those layers, applies the residual blocks, collects the intermediate outputs at each layer, and returns them as a list of feature maps. These feature maps will be used as training classifiers and calculating feature matching losses.

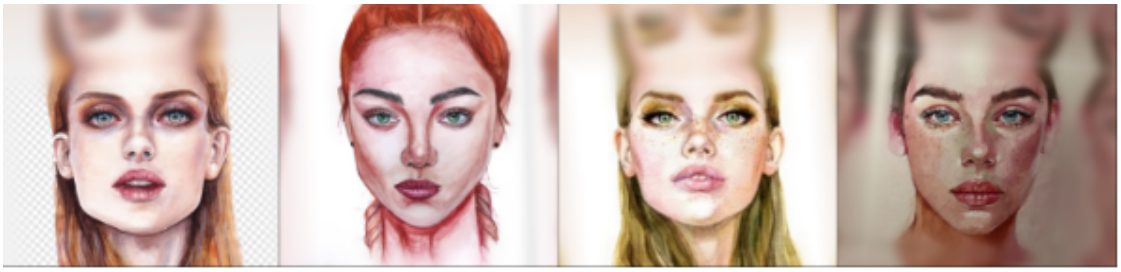
Predictor

The predictor class performs a style transfer onto an input image, using either pre-trained or generator based styles using the generator we created previously. JoJoGAN included pre-trained styles of art, arcane_multi, sketch_multi, arcane_jinx, arcane_caitlyn, jojo_yasuho, jojo, disney that users could apply onto their input image. For the purpose of my project, I trained my own style using four style reference images of watercolor paintings of faces. The predictor takes in the input face, pretrained or not parameter, the number of fine-tuning iterating steps, and the strength of the style. When run, it returns the input image transformed with the selected style.

Training

By inputting my own four style reference images of watercolor paintings, we can use GAN inversion from the e4e projection function to start the training process. It first returns a latent vector from the input style images that is saved. After running through the e4e projection, it finds the style code that exists and loads the latent vector from this disk. The style code refers to a vector or set of values that capture specific stylistic features that represent the style of the image the best. From this deduced style code, it converts the style image to a tensor and

performs transformation to create the new paired dataset of these watercolor painting styles. It is notable that the training process only took about 1 min.



Fine-tuning

After training the StyleGAN using GAN inversion, we are able to fine-tune the StyleGAN. First, the discriminator pre-trained above is loaded in to calculate the perceptual loss. The optimizer set for updating the generator's parameters is Adam because this fine-tuning process constantly updates the generator's weights. To start the process, the original latent vector is mixed with the mean style vector to create a modified latent vector. This modified latent vector is given to the generator to create an image. Next, the perceptual loss is calculated as the L1 loss between the discriminator features of the style image and of the newly created image. As it loops through, it replaces layers that the discriminator had recognized as fake to find the best optimizations. This process also includes the option to modify values for alpha, preserve_color, and the number of fine-tuning iterations. For alpha, I chose a value of 0.5 because the greater the value of alpha is, the more it resembles the style reference photo. Therefore, having an alpha value of 1 would greatly change the original input image with heavy stylization. I set preserve_color to True because again, we want to closely emulate the original photo. Lastly, I set the number of iterations to 400 because the style referenced I used (watercolor painting) is very similar to real images, so it's necessary to have a larger number of fine-tuning steps to pay more attention to the details of the style, like the brush strokes. The runtime for fine-tuning took around 12 minutes, which is most likely due to the large number of iterations I had.

Conclusion

Results

My results were very pleasing. The generated image looks similar to the original and depicts the watercolor brush strokes very clearly. The colors and facial positioning of the two images are also constant. In addition, I tested some random faces from the database onto my reference style to see how they did and these were also very accurate. However, there could be some further fine tuning steps to implement because as shown in the fourth random sample, the outline of their face is very blurry. It is unclear why only that photo transformed that way, but based on my research it might have to do with the background lighting being too light colored and blurred.

Future Work

Therefore, for further work, I could strive to fine-tune the styleGAN to prevent this obscure blur from occurring. I would love to keep working on this project to improve the accuracy and generate other styles that can be used for individuals and companies. Additionally, my project was geared towards a more technical audience because I was focused on learning the

implementation and structural components of Generative Adversarial Networks. However, continuing forward, I want to create a website or application that implements my project and makes it possible for non-technical individuals to use for their benefits.

My sample

Random samples

References/Citation

Code borrowed from JoJoGAN and e4e github repositories.

```
@article{chong2021jojogan,
 title={JoJoGAN: One Shot Face Stylization},
 author={Chong, Min Jin and Forsyth, David},
 journal={arXiv preprint arXiv:2112.11641},
 year={2021}
}
```