Manipulating the Latent Space of a Style Generative Adversarial Model

Andrew Grebenisan

1. INTRODUCTION

Research in generative-adversarial networks (GANs) has had some of the most significant strides in AI literature in the last few years due to advances in loss function stability, skip connection transformations, and recent ability to steer these models towards desired outputs. The most significant addition to the GAN toolbox was the creation of StyleGAN. Typically, images can be thought of as the composition of content and style, and StyleGANs allow for the addition of style to the content of image. In the remainder of this report, I will briefly discuss the StyleGAN architecture, how to embed an image into the StyleGAN space, and how to translate the embedded image in the latent space to apply semantic operations. Also, I wanted to point out that this is not supervised learning, since we are not learning a function that maps an input to an output. Rather, this is more similar to a search problem since we are looking for the an input that gives us the desired output in our space. Furthermore, when we are applying feature transformation to our desired image, we are doing so in an unsupervised manor.

2. STYLE-GAN ARCHITECTURE

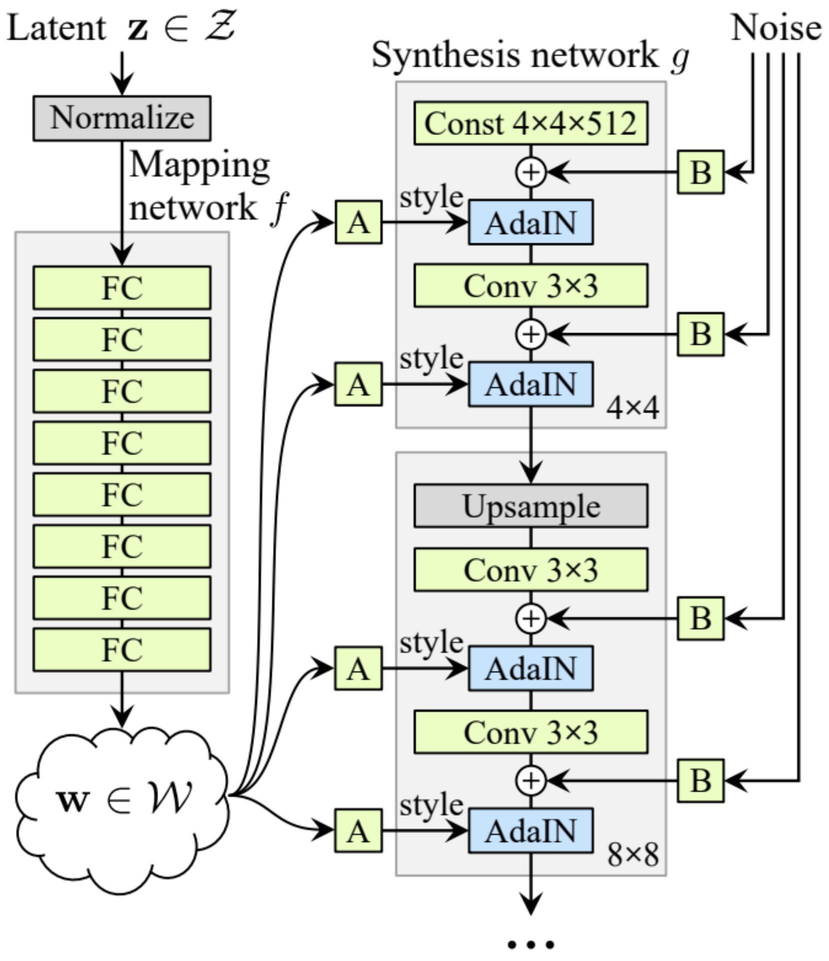


Figure 1: Architecture of the StyleGAN generator portion

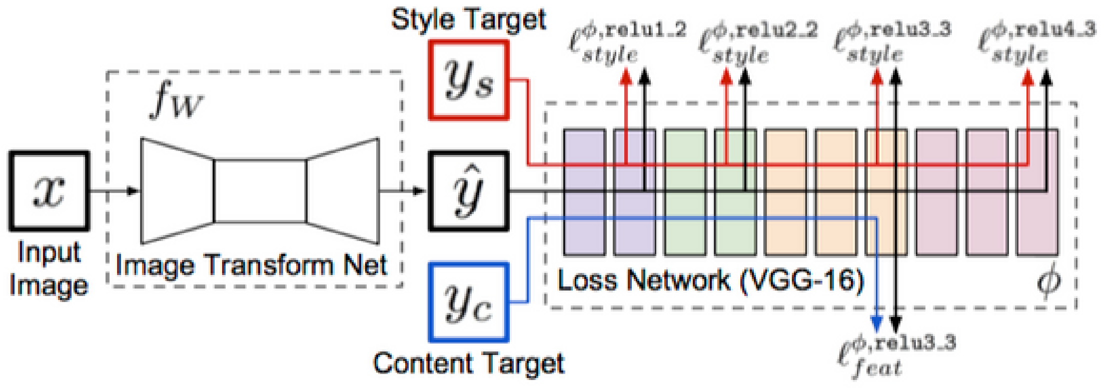
You can see in Figure 1 that only the generator architecture is visible. The reason for this is because the adversary is not important for this report, since I will only be using a pre-trained generator, and I will not be changing the weights. The input to the network is a vector $z \in \mathbb{R}^d$, where d represents the input dimensions of the generator (in this case, 512 dimensions). Each value in this vector is a random sample from a normal distribution with variance 1 and mean 0. The mapping network f maps the network to a latent space which allows for learned features to be as linearly separable as possible. The synthesis network g is the component of the architecture that actually generates an image given the latent vector w. The synthesis network then takes in noise as well as the w vector to produce an image. The reason why w is fed into the network numerous times is so that the synthesis network does not forget the style that the w vector represents. You can also see that the synthesis network is divided into blocks of convolution, AdaIN, and then followed by upsampling. The upsampling was inspired by previous research done by the same team in ProGAN (Progressive generative adversarial network). Instead of trying to generate a high resolution image all at once, each block is trained individually to first learn how to generate low-resolution images, and then the quality of the image improves with each subsequent block.

3. EMBEDDING AN IMAGE TO THE STYLEGAN SPACE

Once we have a pre-trained generator, it is possible to find the latent vector w that produces our desired image (from now on, I will refer to images that we want to produce as query images). Below is the latent code optimization algorithm

```
Input: An image $I \in \mathbb{R}^{n \times m \times 3}$ to embed; a pre-trained generator $G$
Output: The embedded latent code $w^*$ and the embedded image $G(w^*)$
Initialize latent code $w^* = w$;
while not converged do
    $L \leftarrow L_{perceptual}(G(w^*), I) + \|G(w^*) - I\|_2$
    $w^* \leftarrow w^* - \eta F(\nabla_{w^*} L)$
end while
```

The second term of the loss function represents the L2 pixel-by-pixel loss between the estimated image and the query image I. The perceptual loss is actually defined by a VGG network. Beyond image classification, VGG-16 has been shown to be a very promising feature extractor, and is widely used in transfer learning. To create an embedding using VGG-16, take a look at the figure below



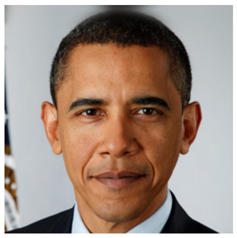
Style Target
y_s
$\ell_{style}^{\phi,relu1\_2}$
$\ell_{style}^{\phi,relu2\_2}$
$\ell_{style}^{\phi,relu3\_3}$
$\ell_{style}^{\phi,relu4\_3}$
f_W
x
Input Image
Image Transform Net
$\hat{y}$
y_c
Content Target
Loss Network (VGG-16)
ϕ
$\ell_{feat}^{\phi,relu3\_3}$

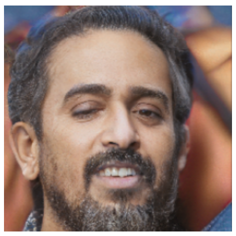
Figure 2: Loss network architecture

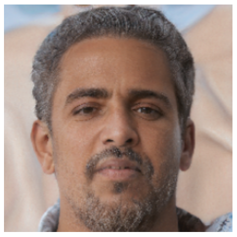
Each block in the diagram represents a series of convolutions and ReLU activations. The colour changes as soon as max pooling is used (example, max-pooling is used between the purple and green layers, between the green and cream layers, and so on). Since I am just using latent code optimization, I am only interesting in using the content loss, which are the activated filters out of the seventh block of the network (examine the blue arrow coming from y_c. Then, the perceptual loss is the MSE of the embedding of the query image and the estimated image.

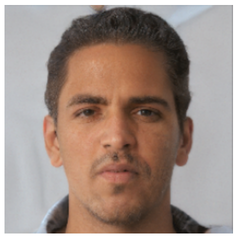
4. METHODS

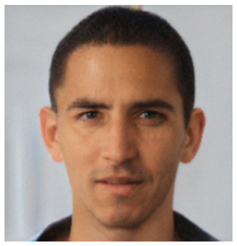
Typically, one would want to latent-code optimize a bunch of query images, however with the limited amount of time, I only chose two images: one with myself, and a picture of Obama. Additionally, Resnet is usually used to find a good initialization for the latent code w to speed up training, but I decided to remove this step from the pipeline for simplicity. I resized each of the images to be 1024×1024 pixels, and slightly changed the algorithm above for stability issues. I randomly inputed a vector z into the mapping network, and produced a w vector. I then did MSE on the pixel-by-pixel loss of the estimated image and the query image. This optimization technique quickly got stuck in a local minimum, but I further optimized the output using the perceptual loss, as described in the algorithm above. When using VGG, I had to


(a) Original image of Obama


(b) Output from random vector


(c)


(d) After 2000 epochs of optimization


(e) Final estimation using just MSE

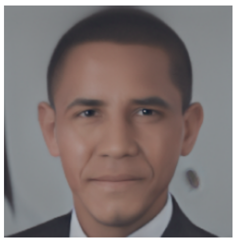
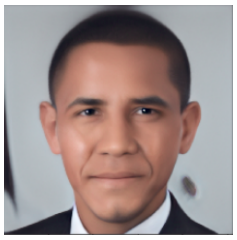
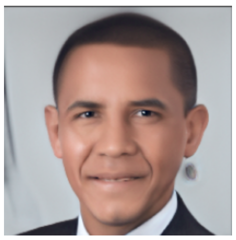
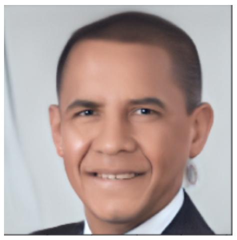
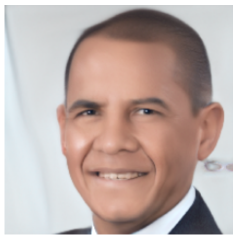

(f) After optimization using perceptual loss

Figure 3: Estimation of Obama


(a)


(b)


(c)


(d)

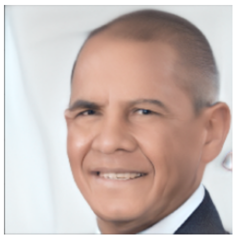

(e)

Figure 4: Applying the old feature vector the derived image of Obama

normalize the images fed into this network using mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225] across each channel respectively, as is typical with image normalization for ImageNet models. I used a learning rate of 0.001 and the Adam Optimizer. Once I obtained approximations for my query images in the StyleGAN space, I then used pre-trained latent codes obtained online for old age to translate my style vectors in the latent space.

5. RESULTS

In Figure 3 are the results of trying to optimize for Obama through time using just the MSE between the query image and output image, followed by using perceptual loss in (f). Overall, training took 5000 epochs on an NVIDIA GeFORCE 2070 GPU. Since I have a dataset of latent features vectors, I chose to take the feature vector for old age, map it through the mapping part of the generator, and apply it to my generated image of Obama. In Figure 4 are a few images showing Obama getting older through time.