Addressing the Noise: Image Synthesis with Deep Image Priors and StyleGAN

Jason Uwaeze[1]

[1]**ju6@rice.edu**

https://colab.research.google.com/drive/1lQP7tOrUYhAaxwqbd1L1Y8SR91pom2cU?usp=sharing

1 INTRODUCTION

In this project, I implemented and demonstrated Deep Image Priors (Ulyanov et al. (2018)) for denoising and inpainting. I used StyleGAN2 [2] for controlled image generation. I also performed experiments on special uses cases.

2 DEEP IMAGE PRIORS

Figure 1. Example of Encoder-Decoder using image priors to generate target image from random noise. The far left image is the target image while the right images represent the increments of training

I tested several model architectures, depth levels, and sigma values. The first thing I noticed is that depth level affected the amount of artifacts in the generated image. In architecture 1 results, sigma = 0.1 and depth = 2 parameters produced the sharpest image, yet when depths was changed to 4 images produced were blurrier. This was also the case when sigma was changed to 0.5 and with 2nd architecture. In summary, depth level seemingly impacts the clarity and sharpness of the image. I also created two Encoder-Decoder architectures varying in convolution layers (i.e. architecture one had two conv2d operations performed at each depth of the encoder and decoder, while only one conv2d operation was used in architecture two). The difference between the result of architecture one and two were minuscule but I noticed architecture two produced fewer artifacts than architecture one but results are inconclusive. I suggest that architecture affects the speed a global minima is obtained, but more experiments are needed.

Lastly, I tested the effect of sigma values (0.5,0.1), and saw that when sigma=0.5, more image artifacts were found in the training results then when sigma=0.1. This suggests that the amount of random noise has an effect on the model training. I imagine that a noisier input or the furthest away an image is from the target image, the harder it is for the model to produce a target image. I performed further experiments to support my hypothesis'.

I tested how the loss function can impact the final results of the model, comparing L1, L2, and KL divergence loss functions. I found the best results, although blurry, were produced when L2 loss was used. L1 loss produced good results yet had additional artifacts that were not present when using L2 loss. Lastly the KL divergence performed poorly in comparison, only able to produce global features like a human outline and edges of the background. With KL divergence loss, the model failed to produce a human face. My intuition is that since KL divergence is only interested in matching the distribution of the generated image to the target image, the model doesn't care for local features like eyes and hair since it's not explicitly showcased in the distribution. The L1 and L2 shows the difference in target and generated image, which subjects local and global information are preserved.

2.1 Loading Images

Figure 2. The above images were chosen for the Deep Image Prior experiment

2.2 Implementing CNNs
2.3 Loading Images

```
"""Create Encoder Block"""
for level in range(self.depth):
  self.in_c, self.out_c = self.channels[level], self.channels[level+1]
  self.encoder_block.append(nn.ModuleList([nn.Conv2d(self.in_c, self.out_c, stride=2, kernel_size=3, padding=1),
                            nn.BatchNorm2d(self.out_c),
                            nn.LeakyReLU()]))
```

Figure 3. The above images shows how I defined the Encoder in the Encoder-Decoder class

```
def forward(self, x):
    """Encoder"""
    # try:
    for encoder in self.encoder_block:
        x = encoder[0](x) # conv2d
        x = encoder[1](x) # Batch Normalization
        x = encoder[2](x) # Leaky ReLU activation
```

Figure 4. The above images shows how I implemented the Encoder in the Encoder-Decoder class

3 DENOISING

The two architectures chosen for this experiment were encoder-decoders differing in number of convolutional blocks. Architecture 1 had double convolutions while architecture only had one convolutional block.

```python
# Architecture 1
class ENCODER_DECODER(nn.Module):
    def __init__(self, depth = 4, in_channels = 3, out_channels = 3, conv_channels = 128):
        super(ENCODER_DECODER, self).__init__()
        self.depth = depth
        self.in_channels = in_channels # Number of channels of input image
        self.out_channels = out_channels # Number of channels of output image
        self.conv_channels = conv_channels # Number of output channels per convolution

        """Define channels at each layer"""
        self.encoder_block, self.channels = nn.ModuleList(), [in_channels]
        for level in range(self.depth):
          self.channels.append(self.conv_channels)
          self.conv_channels = self.conv_channels + self.conv_channels  # i.e. if depth = 4, n_channels = [3, 128, 256, 512, 1024]

        """Create Encoder Block"""
        for level in range(self.depth):
          self.in_c, self.out_c = self.channels[level], self.channels[level+1]
          self.encoder_block.append(nn.ModuleList([nn.Conv2d(self.in_c, self.out_c, kernel_size=3, padding=1),
                                    nn.Conv2d(self.out_c, self.out_c, kernel_size=3, stride=2, padding=1),
                                    nn.BatchNorm2d(self.out_c),
                                    nn.LeakyReLU()]))

        """Create Decoder Block"""
        self.channels.reverse();
        self.channels[-1] = self.out_channels
        self.decoder_block = nn.ModuleList()
        for idx in range(len(self.channels)-1):
          self.in_c, self.out_c = self.channels[idx], self.channels[idx+1]
          self.decoder_block.append(nn.ModuleList([nn.Upsample(scale_factor=2, mode='bicubic'),
                                                   nn.Conv2d(self.in_c, self.out_c, kernel_size=3, padding=1),
                                                   nn.Conv2d(self.out_c, self.out_c, kernel_size=3, padding=1),
                                                   nn.BatchNorm2d(self.out_c),
                                                   nn.LeakyReLU()]))
```

Figure 5. The above images shows the full definition of the encoder decoder class

Figure 6. The above images shows affect of adding noise to chosen images

4 INPAINTING

I chose to run the inpainting experiment with noise = 0.1 and on same images from denoising example. I used L1 Loss function for training.

Short questions to answer in text (1 paragraph total is sufficient)

Figure 7. Denoising Results (Image 1 Architecture 1): **First Row**: Architecture 1, noise = 0.1, depth = 2. **Second Row**: Architecture 1, noise = 0.5, depth = 2. **Third Row**: Architecture 1, noise = 0.1, depth = 4. **Fourth Row**: Architecture 1, noise = 0.5, depth = 4

Figure 8. Denoising Results (Image 1 Architecture 2): **First Row**: Architecture 2, noise = 0.1, depth = 2. **Second Row**: Architecture 2, noise = 0.5, depth = 2. **Third Row**: Architecture 2, noise = 0.1, depth = 4. **Fourth Row**: Architecture 2, noise = 0.5, depth = 4

Figure 9. Special Cases: I used MSE loss in the previous examples, however here is the effect of other loss functions. **First Row**: L1 loss **Second Row**: KL Divergence Loss

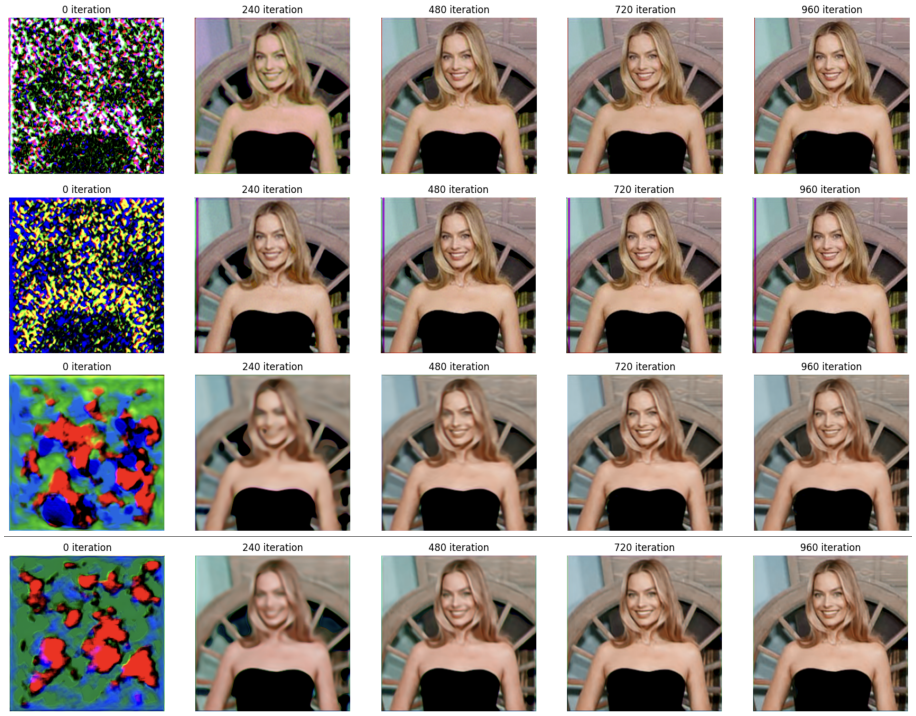

Figure 10. Denoising Results (Image 2 Architecture 1): **First Row**: Architecture 1, noise = 0.1, depth = 2. **Second Row**: Architecture 1, noise = 0.5, depth = 2. **Third Row**: Architecture 1, noise = 0.1, depth = 4. **Fourth Row**: Architecture 1, noise = 0.5, depth = 4

How do your models perform with different region sizes?
I noticed a relationship between the amount and color of artifacts found in output images and the region size of the occlusions. **Seemingly when the region size was set to 64x64 more artifacts were found in the output images than when the region size was set to 32x32.** I also notice that darker artifacts were more persistent, or harder to get rid of, during training when region size was set to 64x64.

How does CNN architectural choice affect performance?
It was evident that the CNN architecture performed better than the second. The first architecture consisted of two convolutional blocks at each layer as opposed to the second with only one convolutional block at each layer. There was noticebly more artifacts when less convolutional blocks were applied. More convolutions or more features extracted did result in better performance.

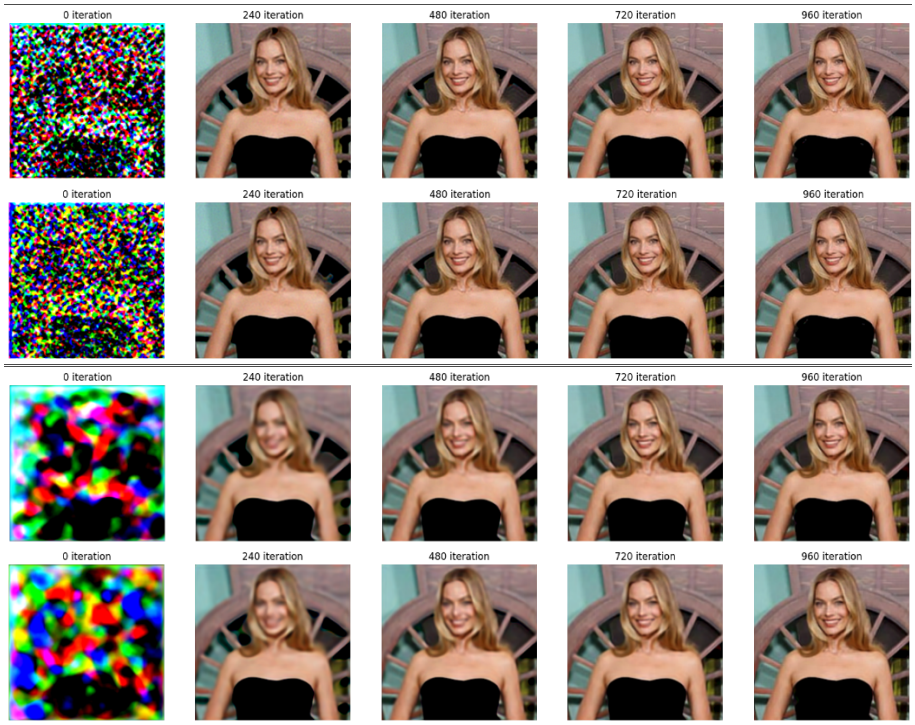

Figure 11. Denoising Results (Image 2 Architecture 2): **First Row**: Architecture 2, noise = 0.1, depth = 2. **Second Row**: Architecture 2, noise = 0.5, depth = 2. **Third Row**: Architecture 2, noise = 0.1, depth = 4. **Fourth Row**: Architecture 2, noise = 0.5, depth = 4

```
def black_out_region(img, n, x, y):
  mask = torch.Tensor(img2.shape[0],img2.shape[1], img2.shape[2], img2.shape[3]).fill_(1)
  new_img = torch.tensor(img1)
  for a in range(n):
    for b in range(n):
      new_img[0][0][x+a][y+b] = 0
      new_img[0][1][x+a][y+b] = 0
      new_img[0][2][x+a][y+b] = 0
  return new_img, mask
```

Figure 12. Black Out Region Code: This is my implementation of the Black Out Region function for section 2.3.b

How do depth levels of the CNNs affect performance?
Increasing the depth level resulted in more artifacts found in output images. While more convolutional block increased performance, a deeper architecture did not yield in better performance or was too deep. It hard to say why this is the case, but I believe too many features made the model struggle in generating images. The model is tasked with finding relevant features and generating an output based on learned features. Too many features make it hard for the model to generate and image that included all the features. **The addition of skip connections in the architecture may help with this problem, as it appears there is information loss the deeper the architecture gets**.

Figure 13. Impainting Results (Image 1): **First Row**: Architecture 1, N = 32 depth = 2. **Second Row**: Architecture 1, N = 64 depth = 2. **Third Row**: Architecture 1, N = 32 depth = 4. **Fourth Row**: Architecture 1, N = 64 depth = 4

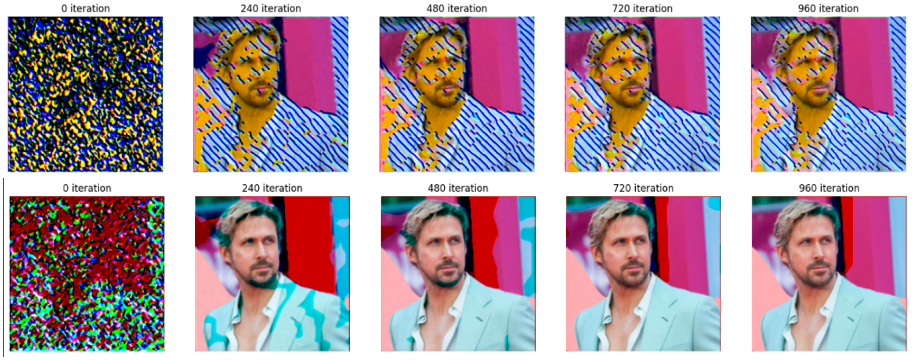

Figure 14. Impainting Results (Image 1): **First Row**: $Noise = 0.5$ **Second Row**: $Noise = 0.5$, $Niterations = 1500$ previous results included $Niterations = 1000$

5 STYLEGAN2

Where does the classifier change its prediction from under 21/over 21 or over 21/under 21 in your sequences?
While interpolating in the noise space the classification model changes its prediction from under 21 to over 21 earlier than when interpolating in the style space. It took only two steps for the predictions to change in the z space, while it took five steps when interpolating in the style space.

What visual features, if any, can you observe changing that cause the classifier's prediction to change?
When interpolating the noise space, gender identity, hair color, and accessories are visual features that change along with the model prediction. When interpolating in the style space, hair color and style were some of the visual features that changed along with the classification model prediction. I believe the gender change and hair change and the main visual features linked with change in age prediction. Eye

Figure 15. Impainting Results (Image 1 Architecture 2): **First Row**: Architecture 2, N = 32 depth = 2. **Second Row**: Architecture 2, N = 64 depth = 2. **Third Row**: Architecture 2, N = 32 depth = 4. **Fourth Row**: Architecture 2, N = 64 depth = 4

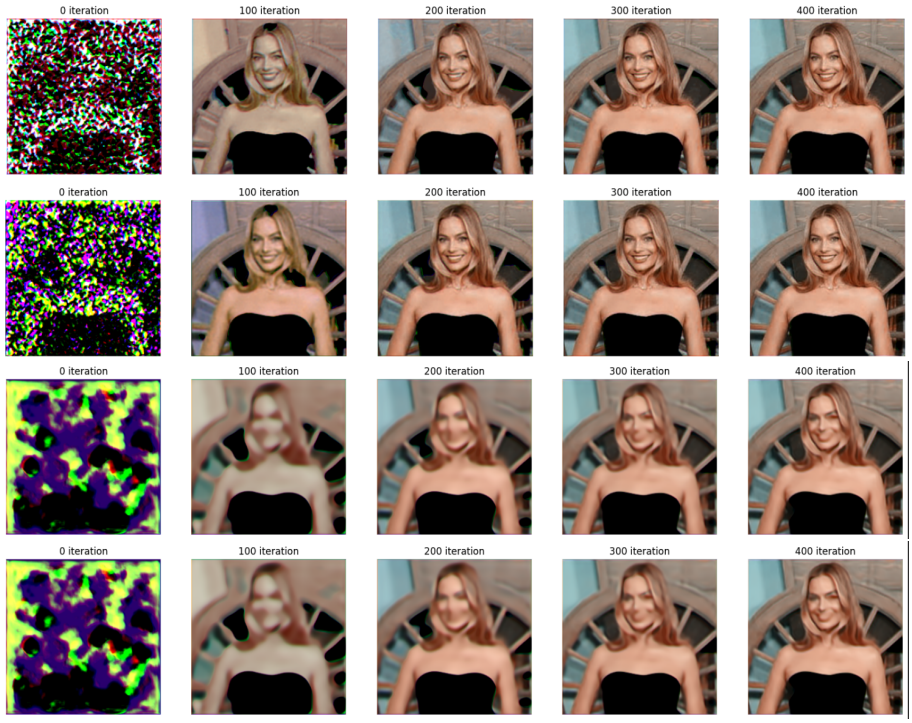

Figure 16. Impainting Results (Image 2 Architecture 1): **First Row**: Architecture 1, N = 32 depth = 2. **Second Row**: Architecture 1, N = 64 depth = 2. **Third Row**: Architecture 1, N = 32 depth = 4. **Fourth Row**: Architecture 1, N = 64 depth = 4

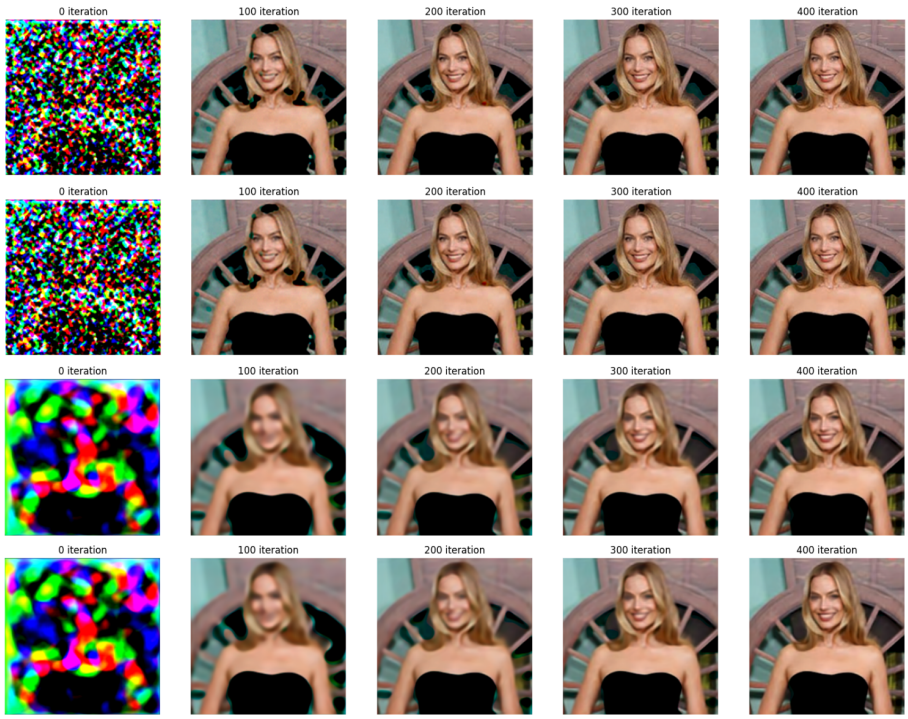

Figure 17. Impainting Results (Image 2 Architecture 2): **First Row**: Architecture 2, N = 32 depth = 2. **Second Row**: Architecture 2, N = 64 depth = 2. **Third Row**: Architecture 2, N = 32 depth = 4. **Fourth Row**: Architecture 2, N = 64 depth = 4

Figure 18. First Row: Example of interpolating from random z vector 1 and z vector 2, textbfSecond Row: Example of interpolating from random z vector 3 and z vector 4

Figure 19. Example of interpolating from random style vector (W) 1 and W 2

color remained constant when interpolating in the style space.

Do you agree with how the classifier changes its prediction?
I don't agree with the classifier predictions in Figure 18 because the classifier seems to struggle with classifying females under 21. I do agree with the prediction in figure 19 when interpolating in the style space. **Overall its a challenging thing to identify gender because images can be take with certain conditions, like makeup or glasses, that make the person look your or older**.

5.1 Latent Space Traversals

Figure 20. First Row: Images from 5 random z vectors **Second Row**: Images from 5 random w vectors

Is identity preserved when moving along these latent directions, and why/why not?
No, in the noise space, there are many examples where the subjects in the outputs look very different from the original image sample. In some image sequences, younger to older, the subject changes gender, hairstyle, and outfit. Although this occurs in the style space, the style space seemingly makes less dramatic steps from younger to older. Gender, facial hair, and accessories changed while interpolating in the style space. **Identity is not preserved when moving along these latent directions**.

Does noise space or style space work better?
Style space works better than noise space, by maintaing more visual features from the original latent vector. Style space takes less drastic steps when interpolating.

Are the traversals always successful?
The traversals were successful at generating an older and younger version of the original latent vector.

Do you notice any facial attributes that seem to commonly change when moving between two age groups?
Facial attributes that commonly change when moving between age groups are facial hair, eyeglasses, hairstyle and hair loss.

Why do you think that occurs?
I think this occurs because these are common elements found in younger and older examples. When training stylegan, the model received training examples that included these facial attributes and decided to associate it with an area in the latent space.

REFERENCES

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2018). Deep image prior. *IEEE conference on computer visions and pattern recognition*.

Figure 21. Interpolation between younger and older results: I sampled 5 random z vectors and used my learned latent space directions, from the trained SVM, and nudged the vectors towards and away from younger and older.

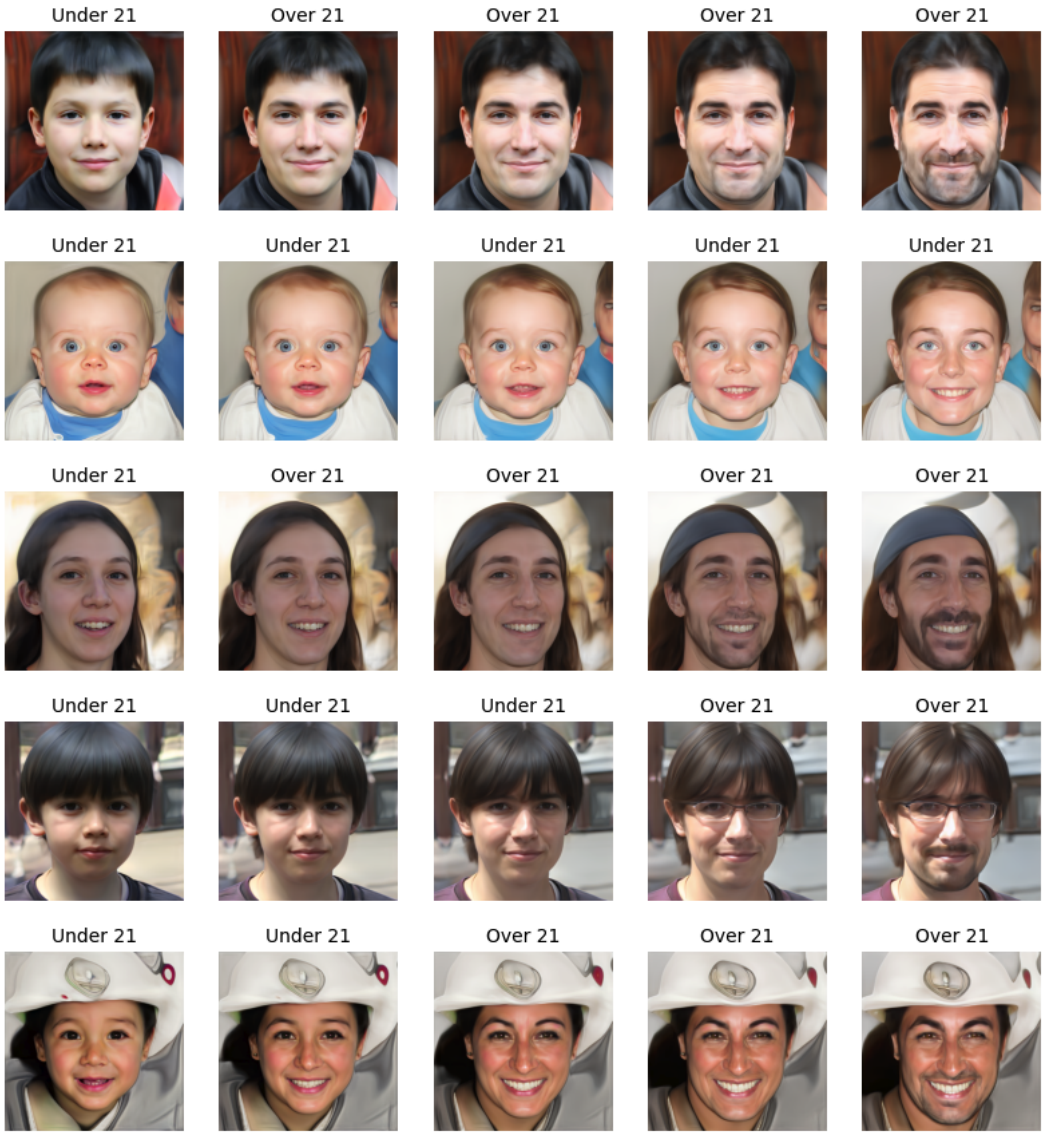

Figure 22. Interpolation between younger and older results: I sampled 5 random w vectors and used my learned latent space directions, from the trained SVM, and nudged the vectors towards and away from younger and older.