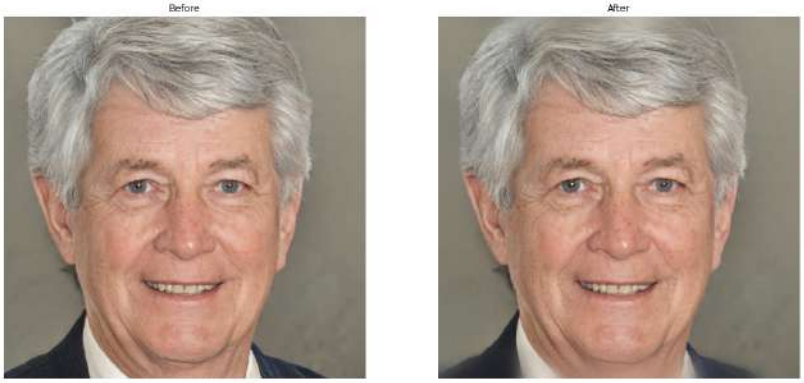
EXERCISE 5
Image Processing
morganizm
342523461

All the images used in my exercise are generated by “This person doesn’t exist”

Image Alignment Part



Gan Inversion 3.2

A figure with the optimization progressions with over 5 images.

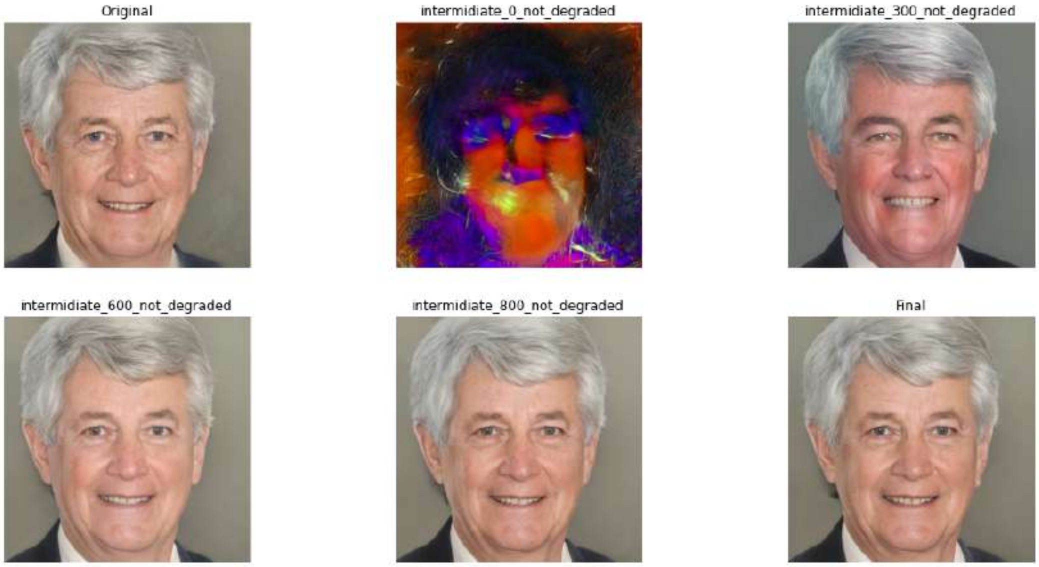
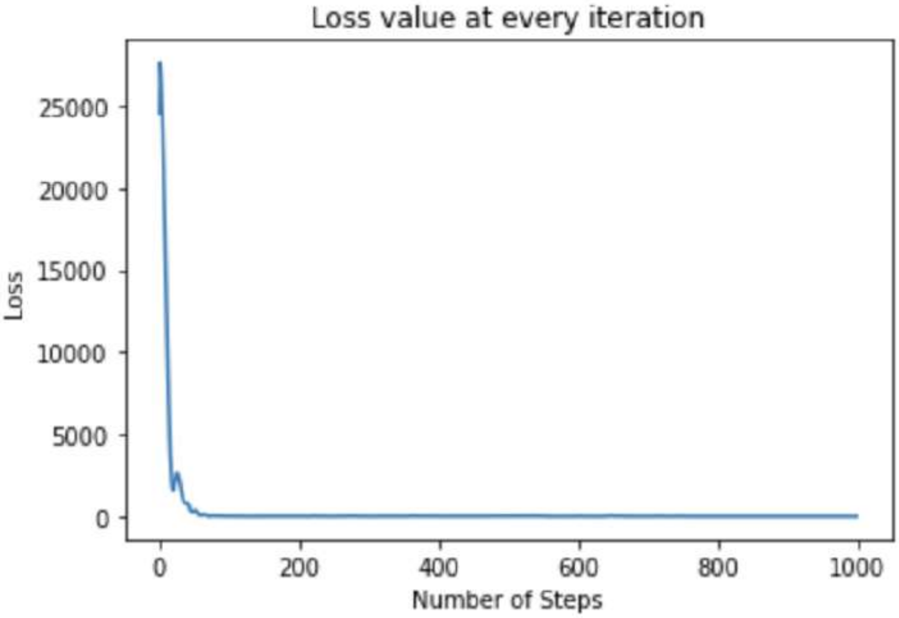
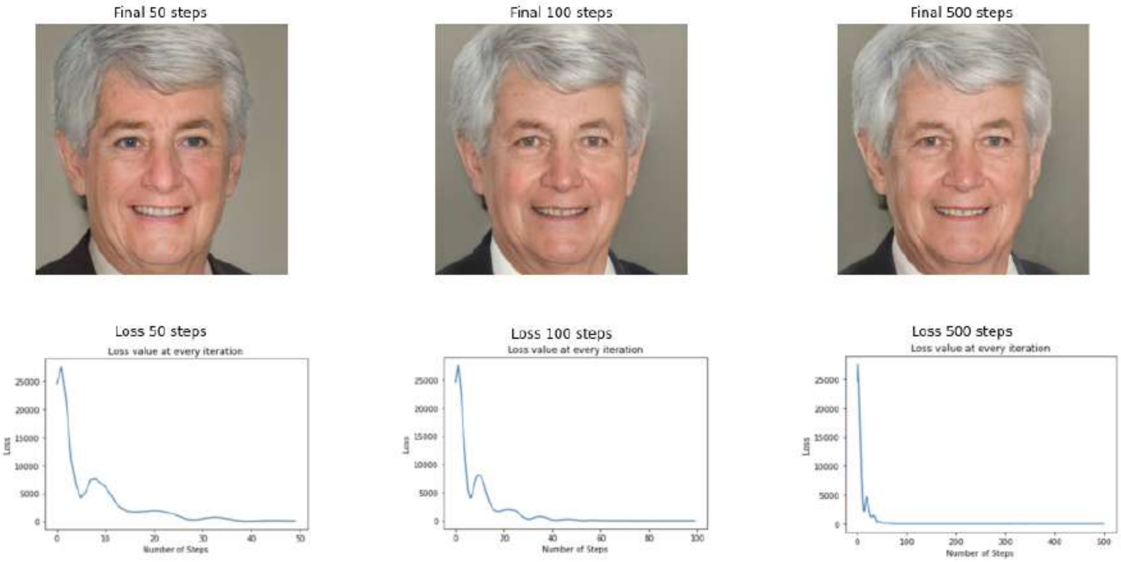

Original
intermidiate_0_not_degraded
intermidiate_300_not_degraded
intermidiate_600_not_degraded
intermidiate_800_not_degraded
Final

figure 1

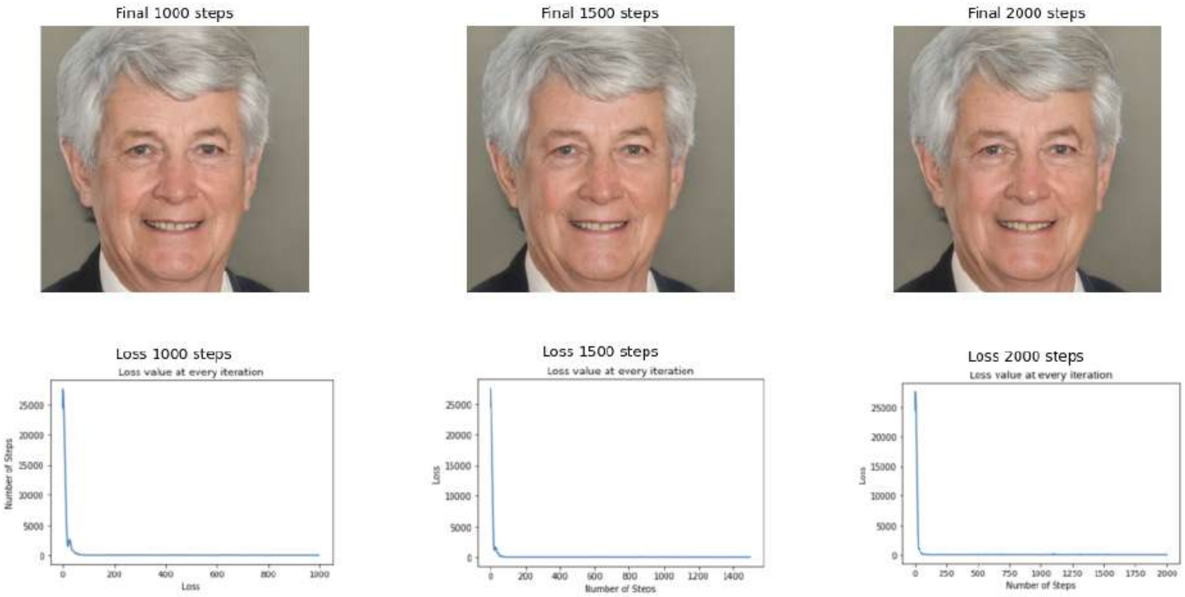
Plot for the optimization loss:



Number_of_steps:
num_steps is the number of algorithm iterations. But also, it's the number of latent vectors that GAN produces during the runtime. Every such guess is supposed to minimize the loss -- the difference in features of the original image and the one synthesized from the latent vector. So, logically, the higher num_steps is, the closer the inverted image is to the original. This can be seen in the figure from below:



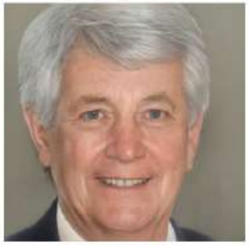
I would like to emphasize that during the first steps of the algorithm, the loss converges to zero much more rapidly. This happens because the initial latent vectors z are out of the domain (G(z) are far from being real human, far from the original image and the loss is high) and the next GAN's guess (even if it guess z2 s.t G(z2) is far from the original image but close to some real human) will make a dramatic difference. While when the loss is already small <1 (the inverted image is already similar to the original) the network can't make a sudden positive dramatic change, and at some point can't minimize the loss. Let's take a look at the following figure -- after 1000, 1500, 2000 steps the results produced
by the network almost identical:

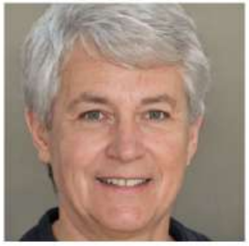


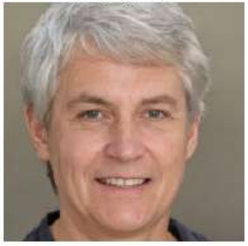
latent_dist_reg_weight
As it was explained in the exercise, latent_dist_reg_weight regularizes the distance z can be from the average latent, so as the value of latent_dist_reg_weight gets bigger the produced inverted image gets closer to the average image of the dataset it was trained on. For example, when latent_dist_reg_weight=20, the inverted image is closer to the mean image than to the original

mean image

Original


latent_dist_reg_weight=0.001


latent_dist_reg_weight=5


latent_dist_reg_weight=10

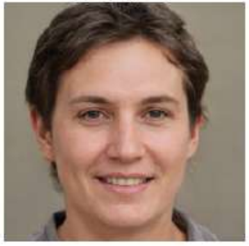
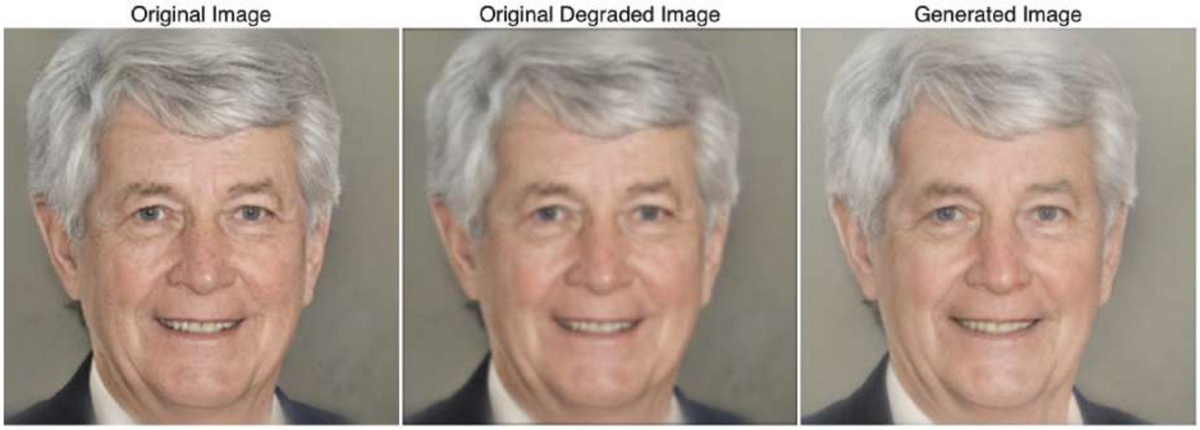

latent_dist_reg_weight=20

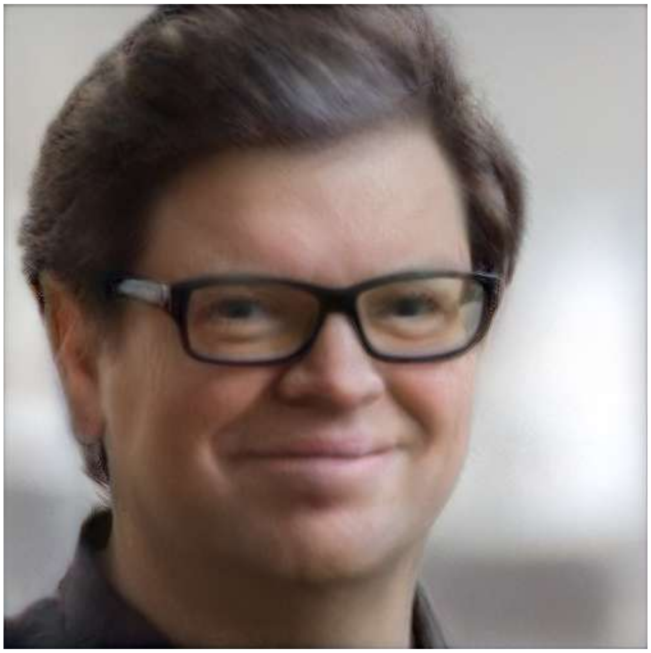
Image Reconstruction Task

Image Deblurring 3.1

Results of blurring and then deblurring of the image of my choice.



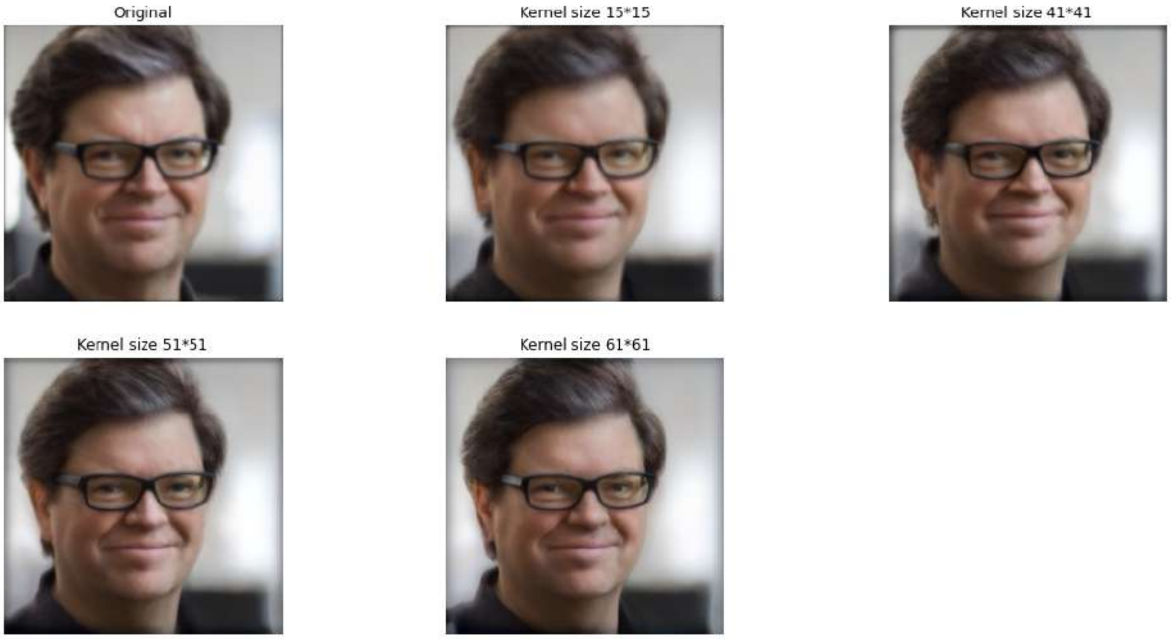
Results of the deblurring on the given blurry image of Yann LeCun



Interestingly, the deblurring of the image I blurred on my own worked much better than the deblurring of Yan Lecun. I believe that the reason behind this is that my image was degraded with the same kernel that was applied to G(z) to compute the loss. The image of Yann LeCun was degraded with a different kernel. So “tricking” the inversion process didn’t go as well as supposed. Even if I apply my kernel to the original image (not blurred), the loss between it and the supplied blurred image won’t be zero.

Kernel-Size

I believe that the bigger the kernel size is, the sharper the image we get. The drawback of that is that the inverted image gets more different from the original. Also, the kernels of the size smaller than 31 produce almost no difference -- the inverted image remains blurred. I believe it happens because the degraded image is pretty similar to the image generated from the latent vector, so the loss between (G(z), original image) and (degraded(G(z)), original image) is almost the same and we don’t actually succeed in tricking the inversion process.



Original

Kernel size 15*15

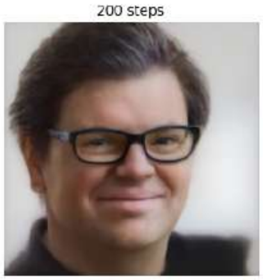
Kernel size 41*41

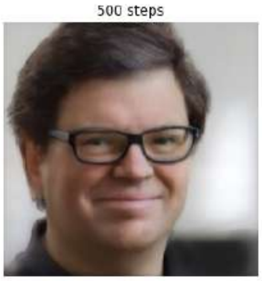
Kernel size 51*51

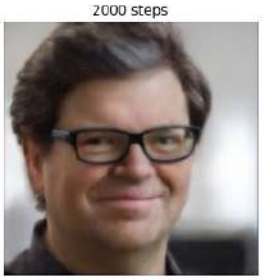
Kernel size 61*61

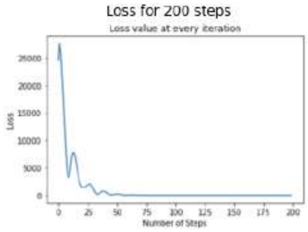
Hyper-parameters. I believe that by increasing latent_dist_reg_weight we can get a better sharpening effect, but we will lose similarity to the original image. On the example below latent_dist_reg_weight = 1

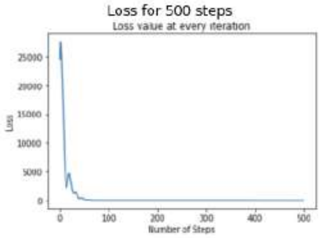
num_stesp influences the result in the same manner it influences the result of regular image inversion (with no degradation).

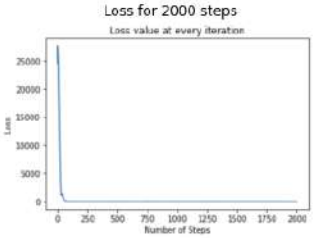










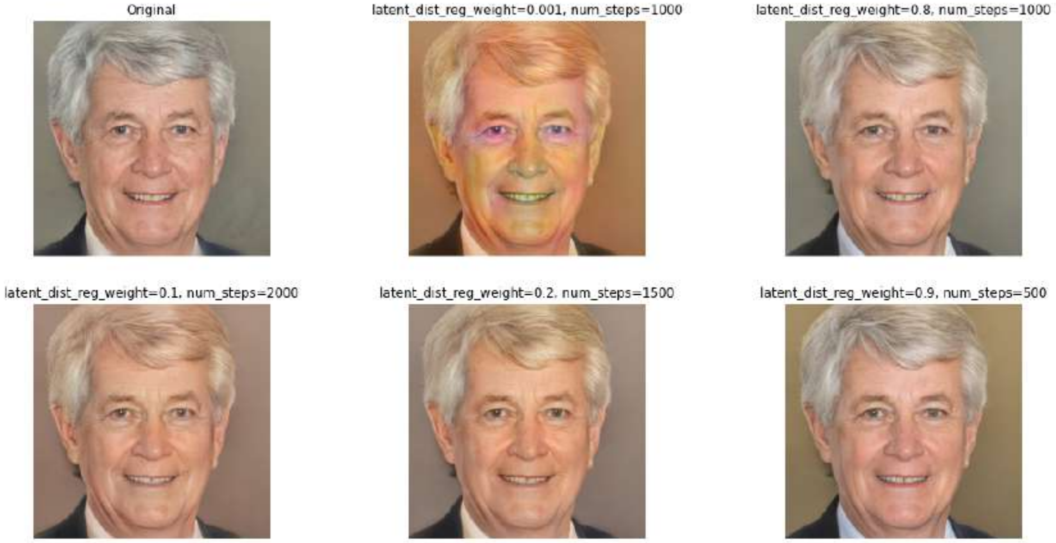


What I’ve tried and failed:

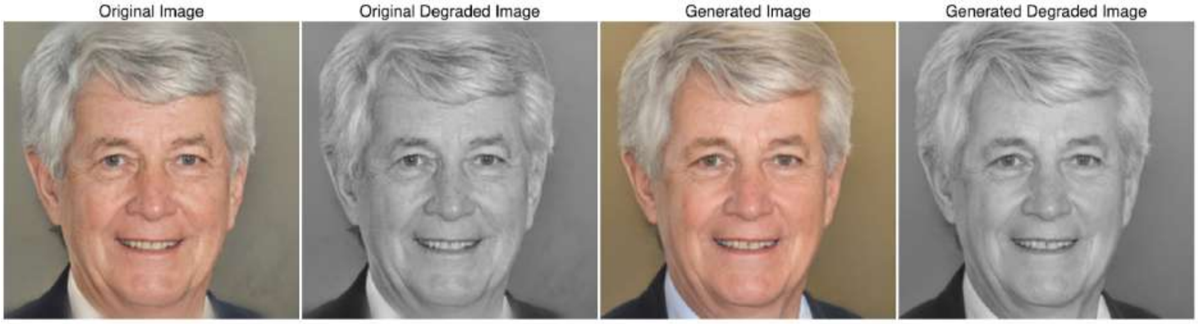
At the beginning I tried to implement the Gaussian Blur according to this function $w(n) = e^{-\frac{1}{2}\left(\frac{n}{\sigma}\right)^2}$
But it didn’t work out well, since the corners of the image were far more blurred than the center, and what I ve got is sharpening of the image background while the face on the inverted image remained almost the same -- blurred.

Image Colorization 3.1

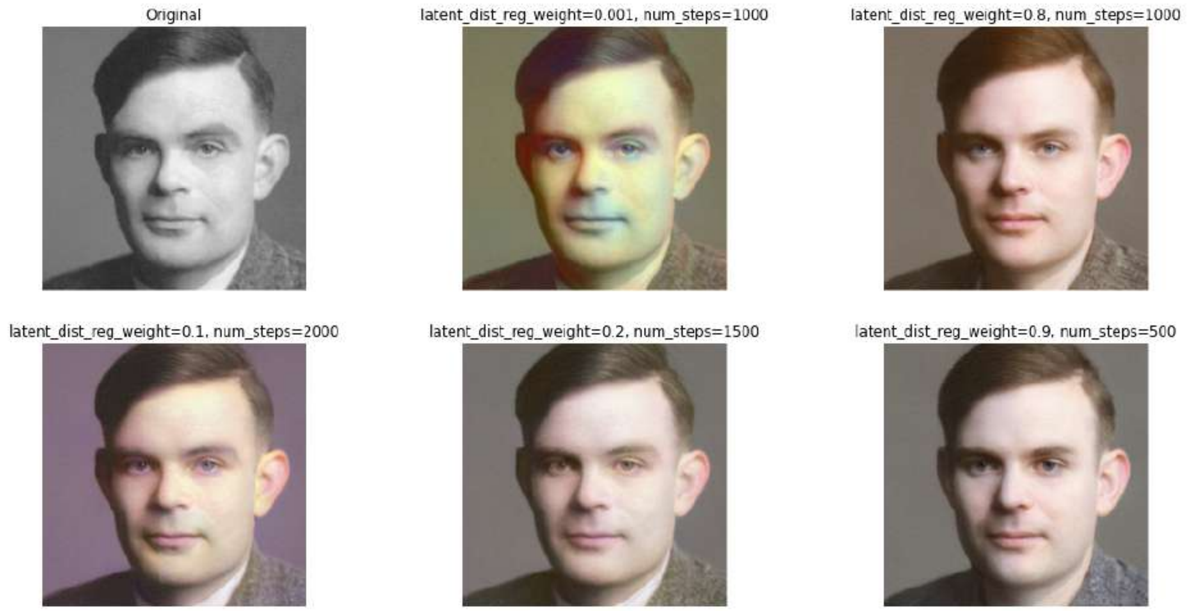
Results of grayscale and then colorization of the image of my choice.


Original
latent_dist_reg_weight=0.001, num_steps=1000
latent_dist_reg_weight=0.8, num_steps=1000
latent_dist_reg_weight=0.1, num_steps=2000
latent_dist_reg_weight=0.2, num_steps=1500
latent_dist_reg_weight=0.9, num_steps=500

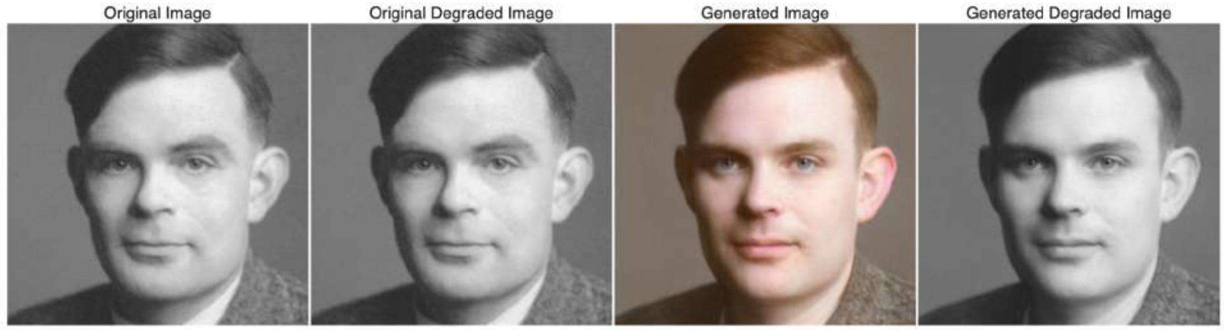
Full output for latent_dist_reg_weight=0.9, num_steps = 500



Results of results of the colorization on the given grayscale image of Alan Turing


Original
latent_dist_reg_weight=0.001, num_steps=1000
latent_dist_reg_weight=0.8, num_steps=1000
latent_dist_reg_weight=0.1, num_steps=2000
latent_dist_reg_weight=0.2, num_steps=1500
latent_dist_reg_weight=0.9, num_steps=500

Output for latent_reg_weight = 0.8, num_steps = 100



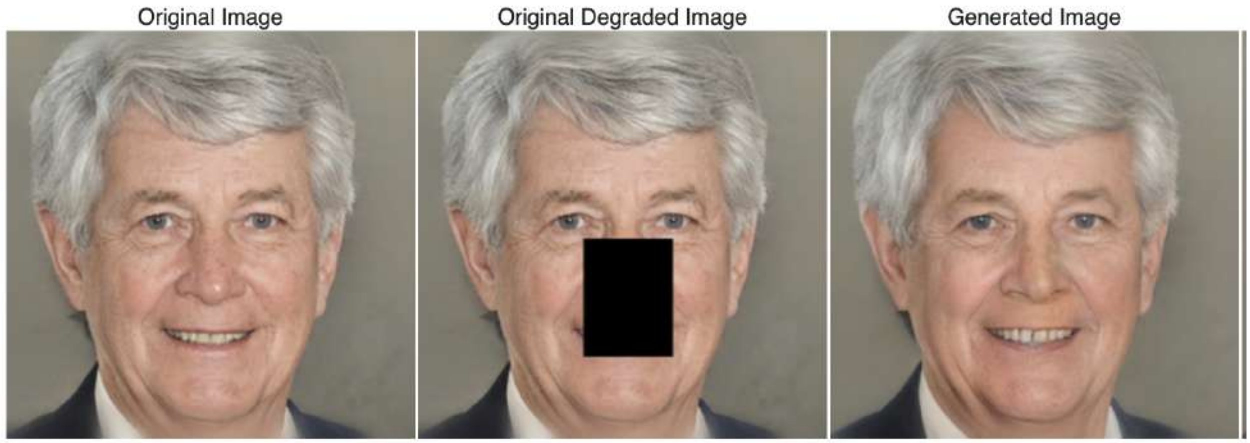
What I've done and why?
I wrote a degradation function that multiplies the tensor which consists of an image by the grayscale matrix tensor generated according to the formula below.

```
0.2989 * R + 0.5870 * G + 0.1140 * B
```

GAN produces latent vectors according to which the image from the data set is generated. This image is in RGB, like all the other images in the data set [the network can't produce a grayscale image, since it was trained on RGB]. By applying grayscale transformation to G(z) we make GAN compare the grayscale version of G(z) to the input, so at the end of the inversion process we will get latent vector z the grayscale of G(z) of which is closest to the input.

Image Inpainting 3.3

The results of the inpainting for an image of your choice

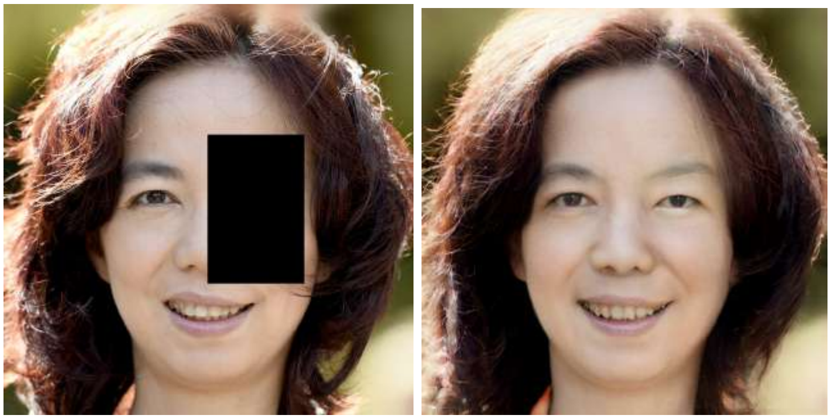


Original Image

Original Degraded Image

Generated Image

Your results of the inpainting on the given masked image of Fei-Fei L



Discuss your solution, explain what you did and the motivation for it:
The idea is the same – to “trick” the inversion process, so it thinks that it produces degraded images. By the end we will get the image the degraded version of which is closest to the original input.

Issues I have run in:

At the beginning, I wanted to erase at each iteration of `run_latent_optimization` random parts of the image, in analogy to the function from torchvision.transforms.RandomErase(). But the results were pretty bad – the inverted image was too bright and even those details that were not erased in the input image were not inverted perfectly. The increase in the number of iterations didn’t really make it better – on the contrary, the borders of the mask were on the inverted image, like a patch.

I believe that it happened because random erasing made GAN confused about what is a good result and what is a bad result. I mean, if you have z10 at 10’s iteration and z200 at 200’s iteration, it’s not necessarily that the loss of (f(G(z10)), input image) < the loss of (f(G(z200)), input image)

So I tried to apply the same mask at every iteration.