67829 – Image Processing – Fall 2021

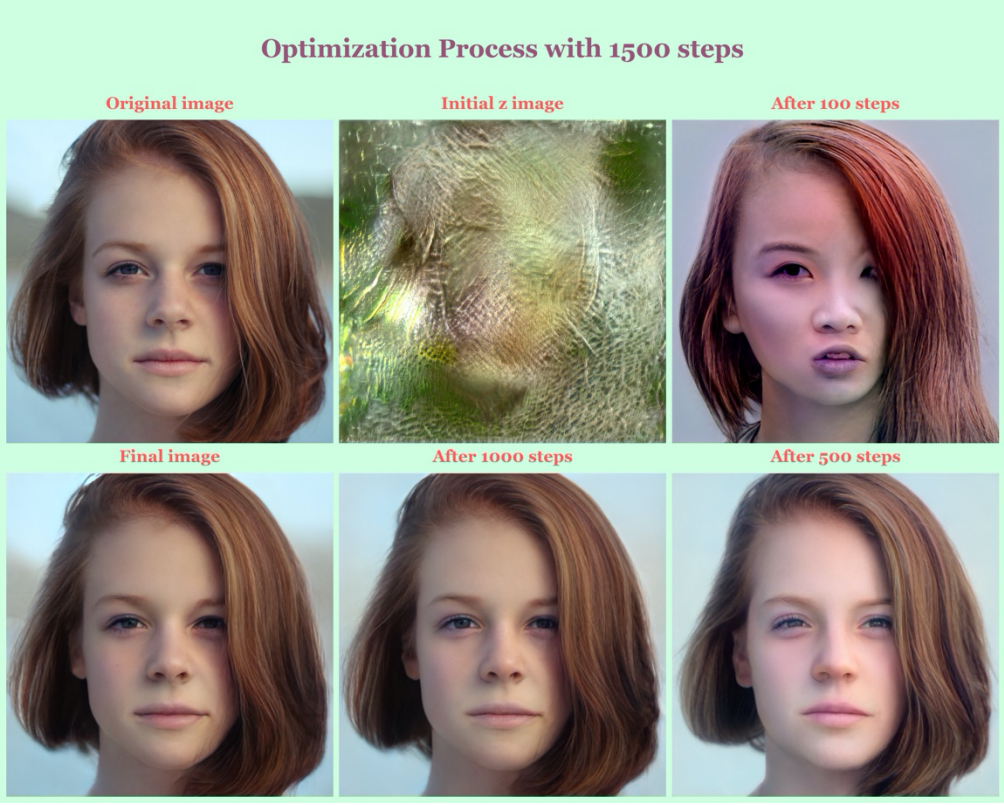
Exercise (5)

Muaz Abdeen – 300575297

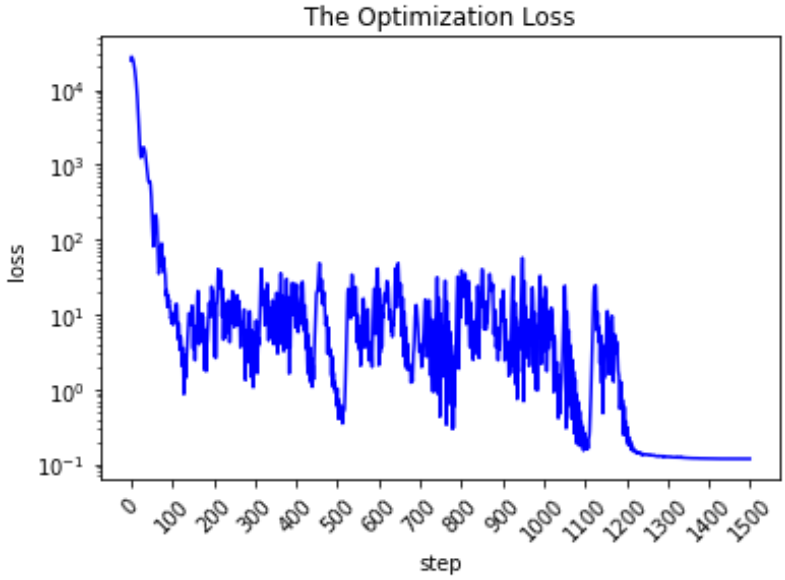
Image Alignment:

Before	After
	

The Optimization Progressions:



A Plot of the Optimization Loss:



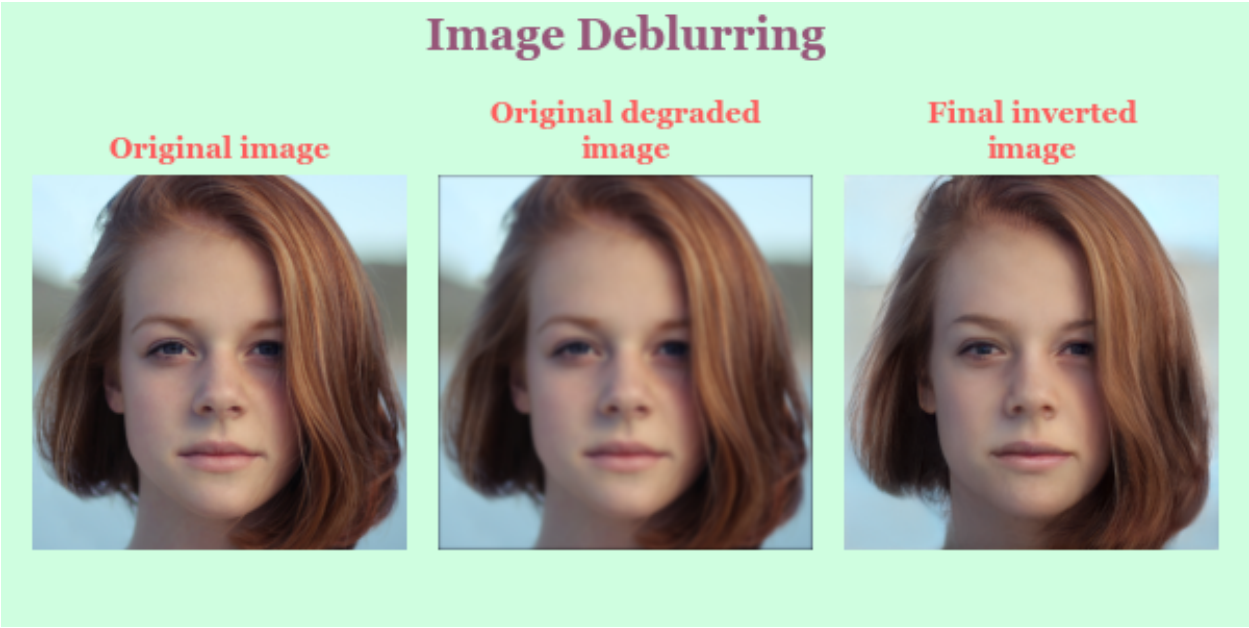
The effect of *latent_dist_reg_weight* and *num_steps* on the results:

I noticed that given a *num_steps*, increasing the *latent_dist_reg_weight* made the optimization process generates good results earlier, that is, closer images to the original one will start to show earlier.

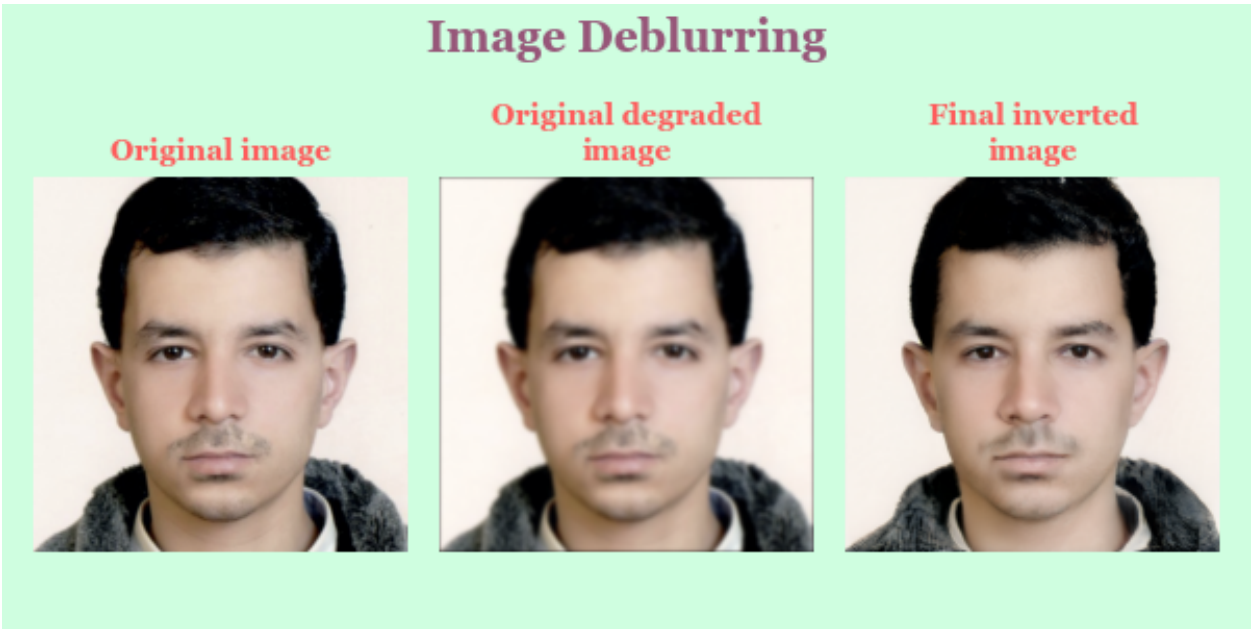
Also, increasing the *num_steps* will give us a better result, to some extent, that is after some threshold, there will be almost no improvement.

Image Deblurring:

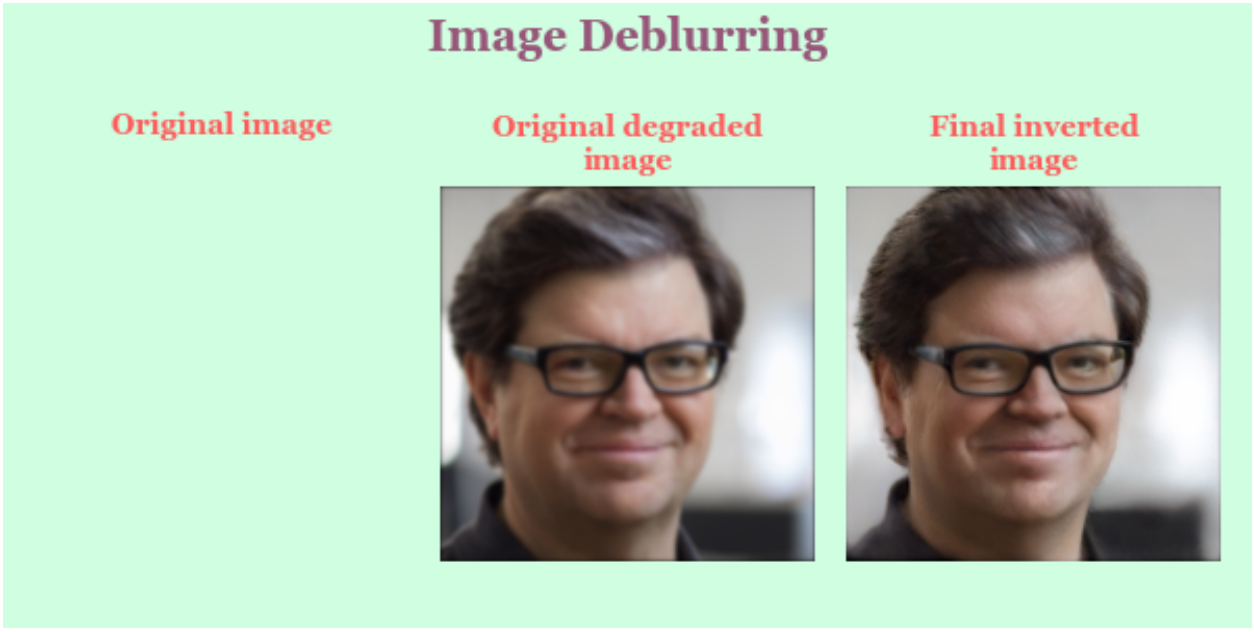
First image



Second image (my personal picture):



Yann LeCun



Solution Discussion:

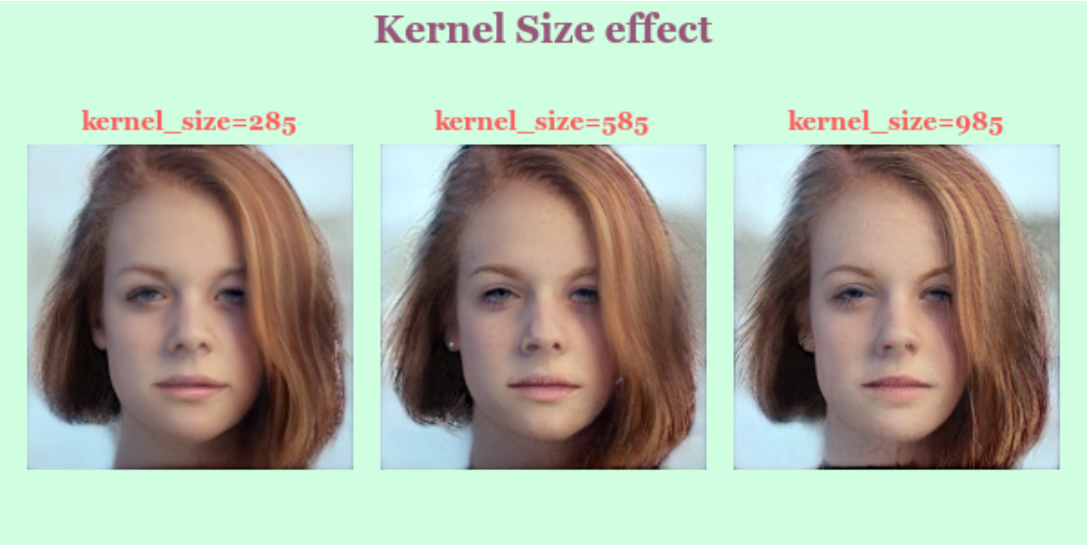
Hyperparameters: $kernel\_size = 85$, $num\_steps = 1000$,

$$latent\_dist\_reg\_weight = 0.001$$

I applied the degrading function on the generated image, that is, I blurred the generated image $G(z)$, before computing the loss.

I defined two functions to blur the image: *gaussian_kernel* and *blur_spatial*, the first one generates a gaussian kernel according to the size passed to it, the second blurs the image passed to it by convolve it with this kernel twice, horizontally and vertically, as we have learned in the class. I made some changes to the API; I passed the kernel to the *run_latent_optimization* function and use it to blur the generated image before computing the loss.

The kernel size affected the results; the smaller the kernel the less blurred the image, and we get a result like the one of undegraded image. Conversely, if we choose a big kernel, that is, more blurred image, the result will be less clear, many fine details will be lost, it will be somehow noisy.

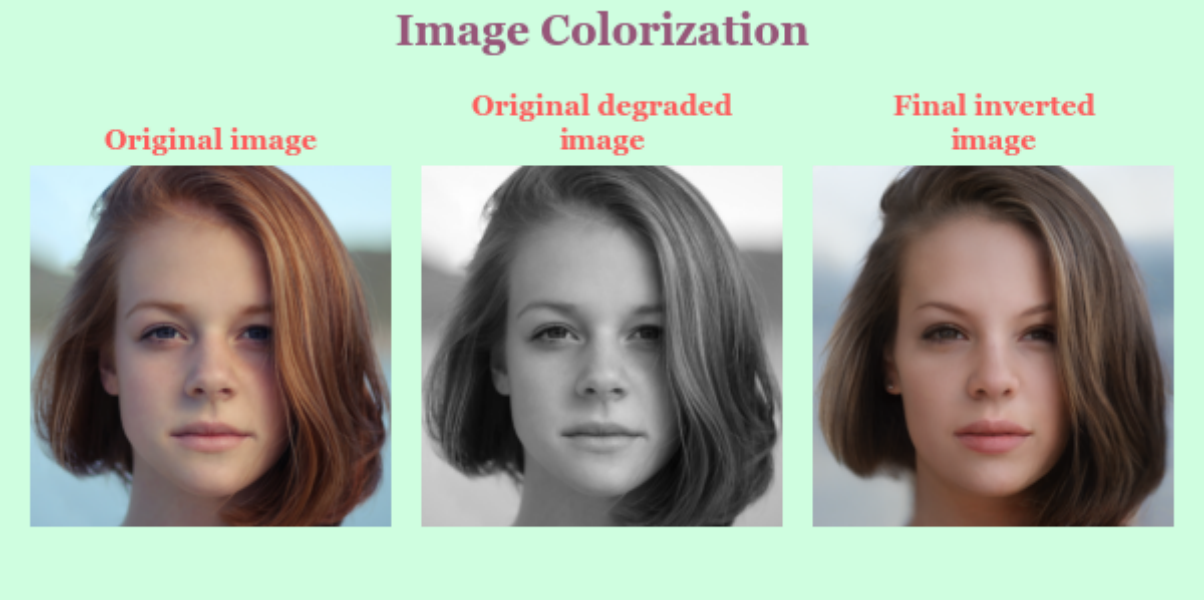


I noticed that after at most 1000 steps, the loss almost converged, and the result did not improve more. Also, and the bigger the *latent_dist_reg_weight* the closer the constructed image to the mean of natural images, and vice versa.

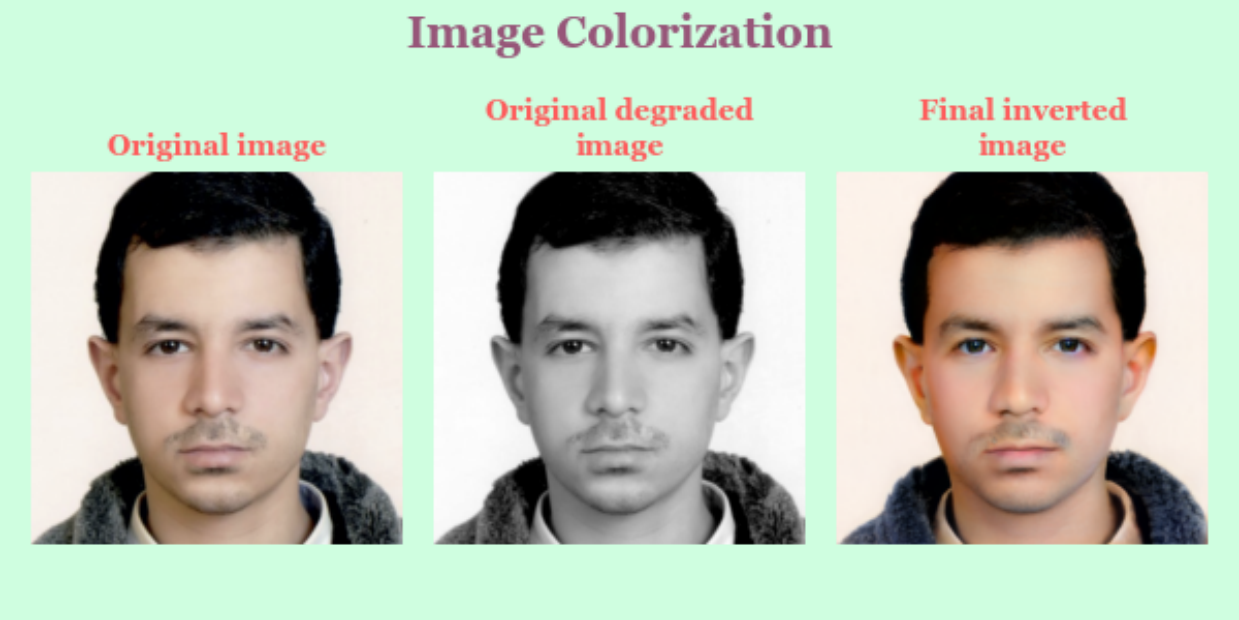
For example, a more blurred image (*kernel_size* = 195) with large *latent_dist_reg_weight* (= 0.2) will give a clear result but deviated relatively from the original one. In the other hand, small *latent_dist_reg_weight* (= 0.01) will give closer result to the original image, but less clear.

Image Colorization:

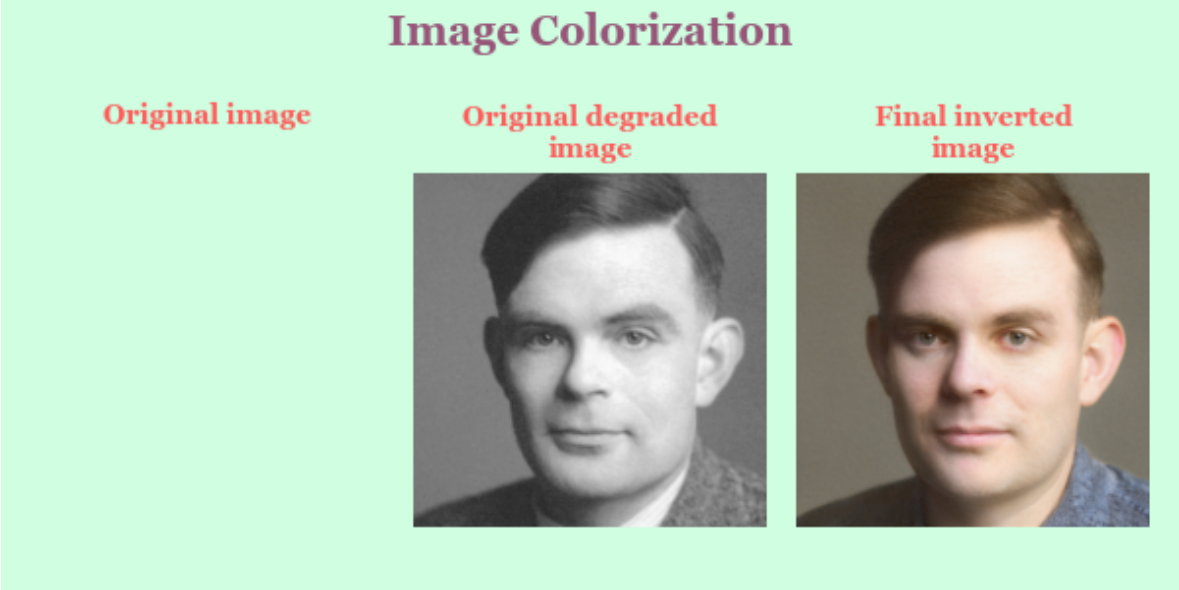
First image



Second image (my personal picture)



Alan Turing



Solution Discussion:

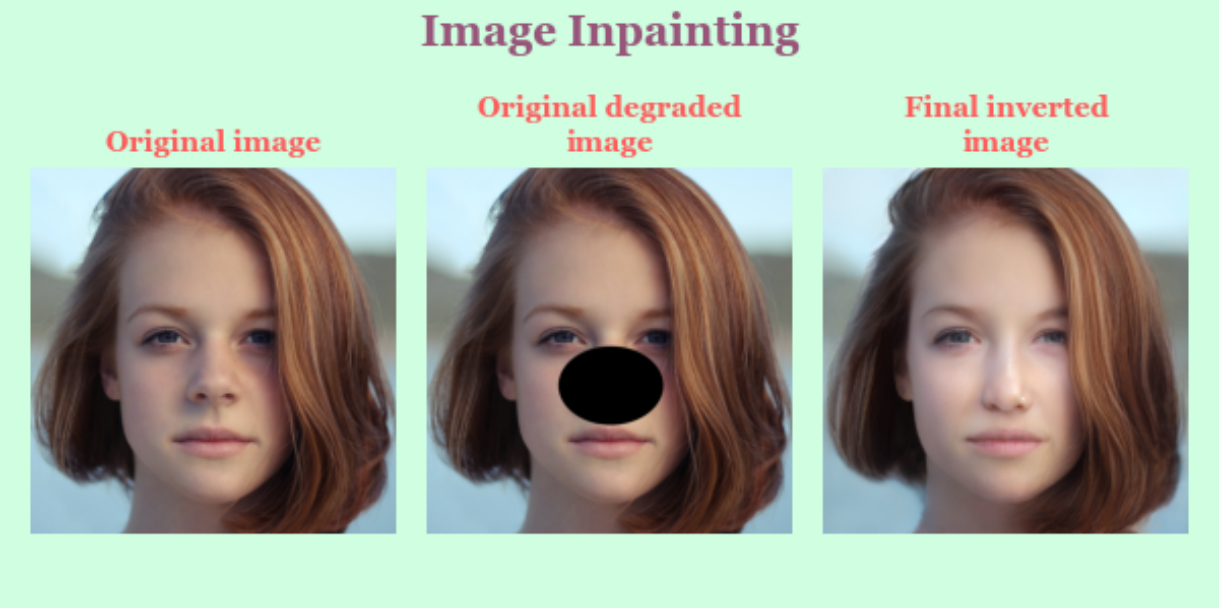
Hyperparameters: $num\_steps = 1000$, $latent\_dist\_reg\_weight = 1$

I applied the degrading function on the generated image, that is, I converted the generated image $G(z)$ to a grayscale one, before computing the loss. I used the `Grayscale` function from `torchvision.transforms` module to convert an image to grayscale mode.

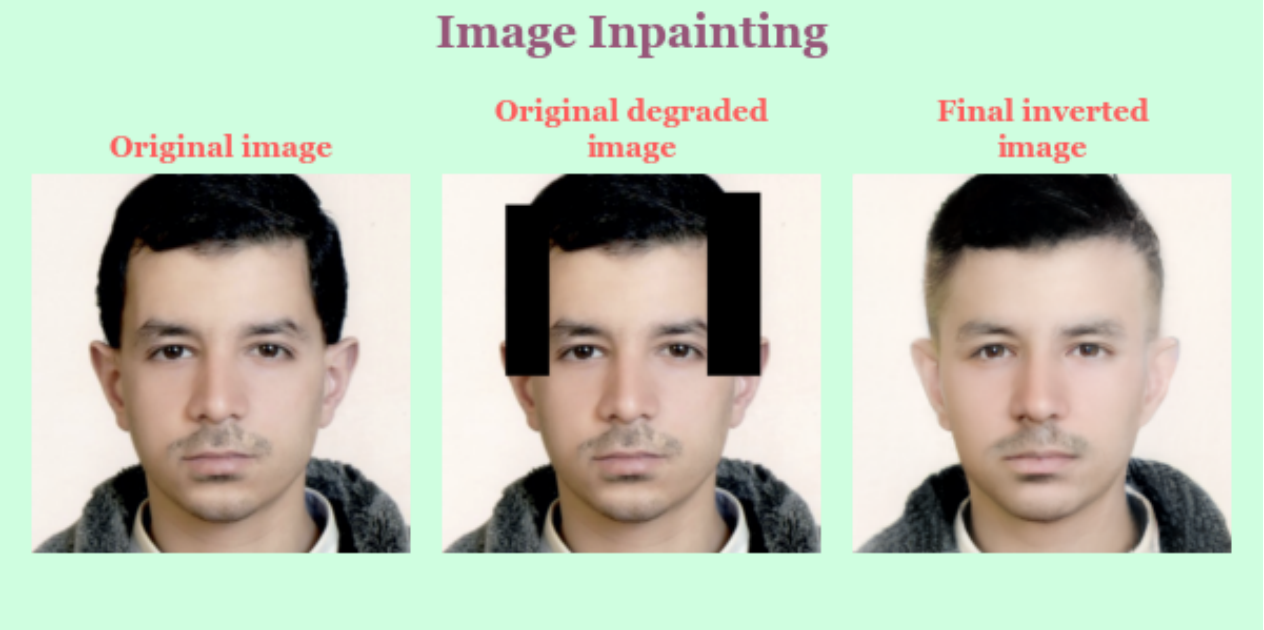
I noticed that a big *latent_dist_reg_weight* gives more realistic colorization, but with a little deviated image from the original one, I think the bigger the *latent_dist_reg_weight* the closer the constructed image to the mean of natural images, and vice versa. Moreover, after at most 1000 steps, the loss almost converged, and the result did not improve more.

Image Inpainting:

First image

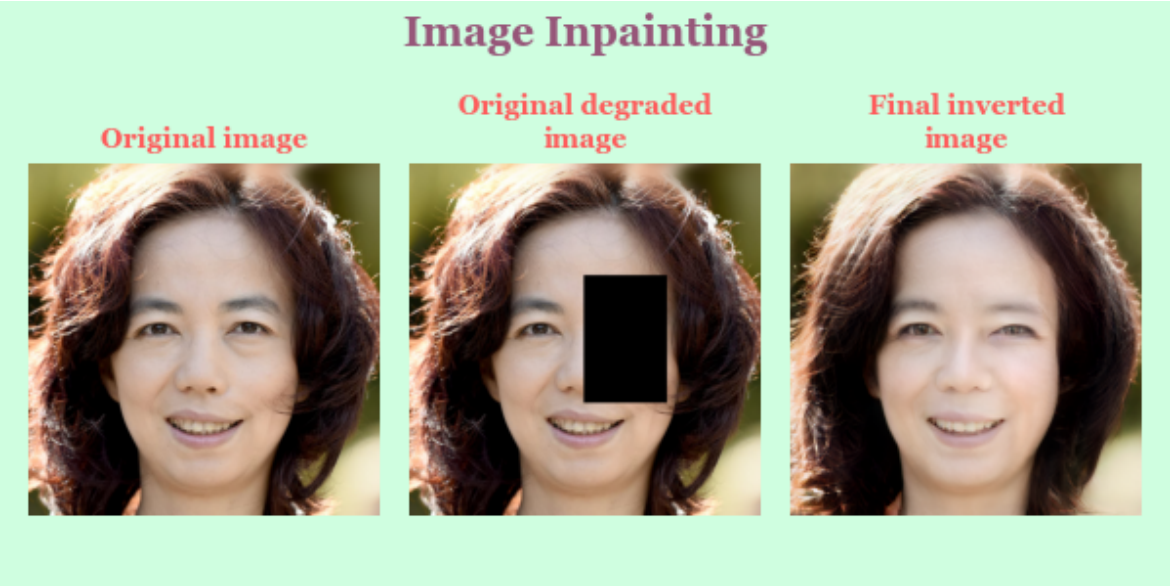


Second image (my personal picture)



* Here I think the difference in the sideburns is because the shape of the mask. *

Fei-Fei Li



Solution Discussion:

Hyperparameters: $num\_steps = 1000$, $latent\_dist\_reg\_weight = 0.2$

I applied the degrading function on the generated image, that is, I multiplied the generated image $G(z)$ with the mask, before computing the loss.

I made some changes to the API, I passed the mask path name ($mask\_fname$) to the $invert\_image$ function, then inside this function I read the mask file, converted it to a tensor, and multiply it with original image, then passed it to the $run\_latent\_optimization$ function and use it to multiply the generated image with it before computing the loss.

I noticed that after at most 1000 steps, the loss almost converged. The optimal $latent\_dist\_reg\_weight$ range for inpainting was between 0.1 – 0.3.

As the previous reconstruction problems, num_steps has improved the result up to 1000 steps, and the bigger the $latent\_dist\_reg\_weight$ the closer the constructed image to the mean of natural images, and vice versa.