עיבוד תמונה | תרגיל 5

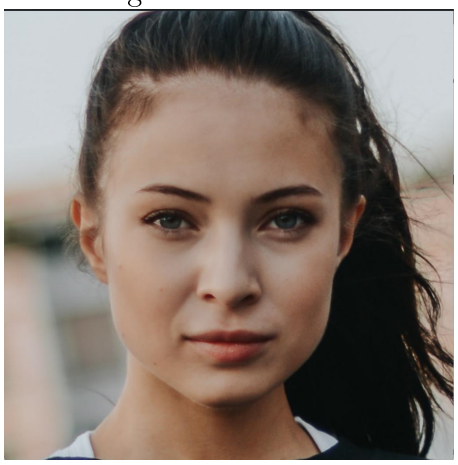
שם: תמוז גיטלר | ת`ז: 312243439

26 בינואר 2023

Image Alignment - 3.1

Before Aligment

After Aligment

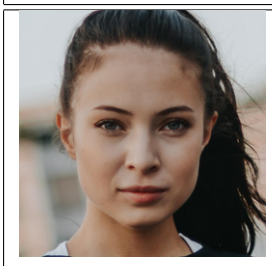
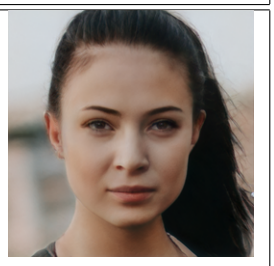


the folliwing files are found inside “Image Alignme” folder:

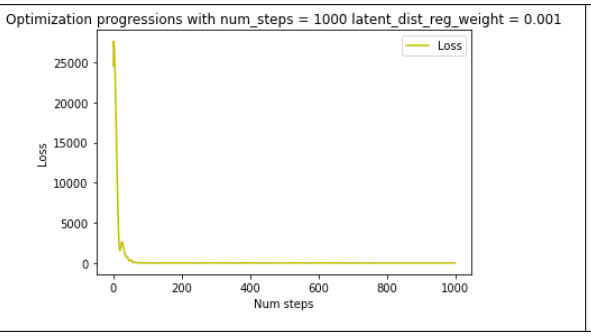
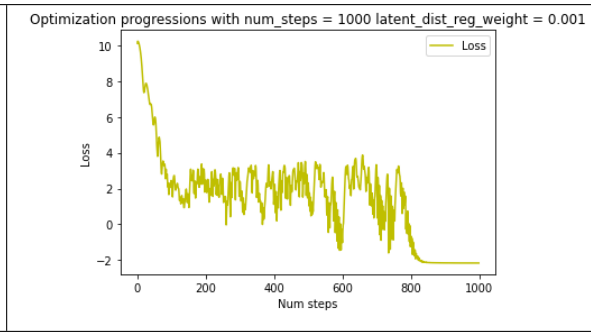
- degraded_woman.png
- afterAlign.jpg

Gan Inversion - 3.2

The pictures i got using 1000 nums of steps and 0.001 latent dist reg:

original image	step 100	step 200	step 300	step 1000
				

I decided to aplly Log on the loss to better understed the obtained loss over 1000 steps:

Loss	after applying log on each loss
	

latent dist reg weight discussion:

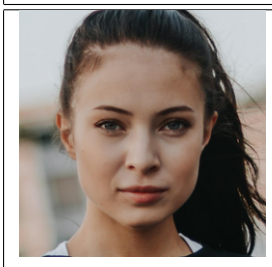
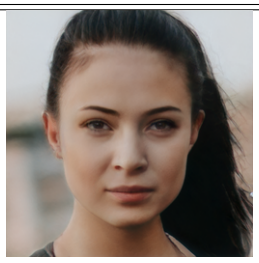
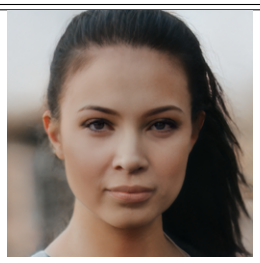
Latent dist reg weight - LDR is a regularization hyper paramater used in the training procces.

With it we are able to control the smoothness of transition between the generated images in different num steps, and make the generated images diverse.

when the generator is given small LDR weight he produces images that aren't simillar to the average latent space - we will get a generated image that is closer to the original image. while given higher LDR weight the generator produces images that are more closer to the average latent space, thus the generator will generate an image that will look more different than the original image.

As seen in the table above, with small LDR (0.001) i was able to generate an image that is very close to the origin.

After increasing the LDR to 1, the generator produced an image that in first sight it looks very close to the origin, but when i dived into small details like lips i was able to notice the difference.

Original image	Produced image using 0.001 - LDR	Produced image using 1 - LDR
		

Num steps discussion:

Num steps is a hyper paramater used in the training procces, it decides how many steps are taken in the generator network.
Using small number of steps value will make the generator to produce less detailed images but the training proccess will be short.
While using higher number of steps value will make the generator to produce high detailed images but the training proccess will be longer.

In the chart below we can see the produced images using different number of steps.
when i used small number of steps (100) the generator produces an image which is very far from the origin
after 600 number of steps the generator produces an image which is very close to the origin
I found it really interesting that when i used more than 600 number of steps the produces image didnt changed alot, so using big number of steps isnt always usefull.

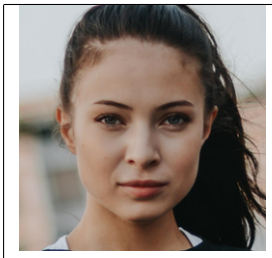
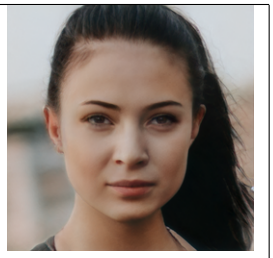
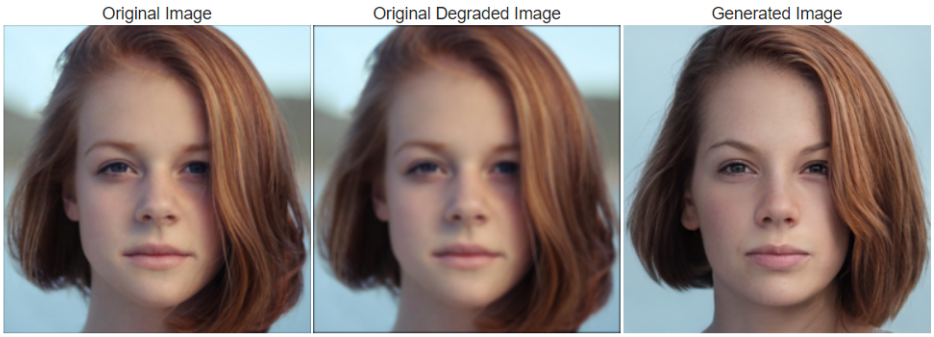
Original image	num steps 100	num steps 600	num steps 1000
			

Image Reconstruction Tasks - 3.3

Image Deblurring - 3.3.1

for this part i decided to use a different image
my result of the deblurring for an image of your choice
using :

- num_steps = 1000
- latent_dist_reg_weight = 1
- kernel_size = 9

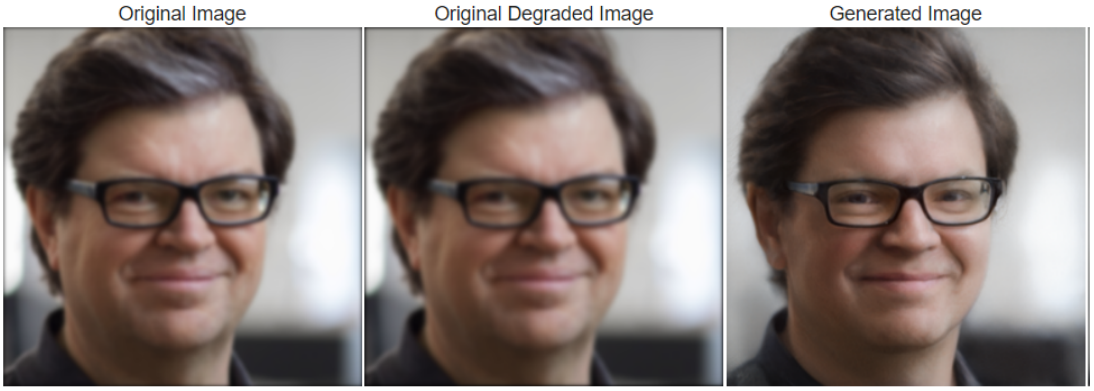


the folliwing files are found inside "my_debluring" folder:

- original.png
- degraded.png
- deblurred.png
- inverted_latent.npz
- Copy_of_IMPR_Ex5_Deep_Style_Image_Prior_2022_2023_.ipynb - my code

my result of the deblurring on the given blurry image of Yann LeCun
using :

- num_steps = 1000
- latent_dist_reg_weight = 1
- kernel_size = 9



the folliwing files are found inside "yan debluring" folder:

- yann_lecun_blur.png
- final_inverted_image.png

- inverted_latent.npz
- Copy_of_IMPR_Ex5_Deep_Style_Image_Prior_2022_2023_.ipynb - my code

Discuss your solution

First i had to blur the original image and only after bluring i ran invert_image with the blured image as the original image.
for bluring the image i had to implement a method that creates a 2D gaussian filter, i decided to use my method create_filter_vec which i implemented in exercise 3 but i needed to change it to create 2D vec, and i also had to return torch filter and not numpy because i used conv2D from torch to perform the convolution. I decided to create 1 filter instead of 3 filters which are exactly the same, and i used it to convulate on each channel (RGB) of the image, at the end i used torch.cat to concatenate all the blurred chanel images.

Issues i ran into and how i solved them

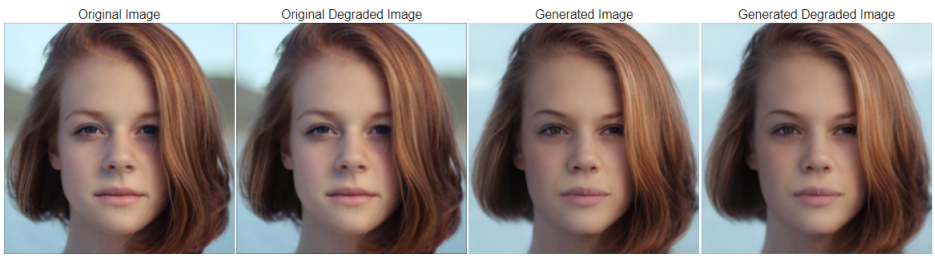
creating the 2D gaussian filter wasn't easy. first i transfered the image to numpy array and applied 1D convolution on image rows and then on the blured columns as we did in previous exercies, at the end i transformed the blured image back to torch type, but by doing so i detached the gradients from the image and i got bad results.
After deciding not to convert the image to numpy, (as i mentioned above) i implemented a method that creates 2D filter by creating 1D filter and multiplying it with himself.
Then i created a new image with (torch.zeros(img.shape)) and tried to fill it with the 3 blured channels, but as before, the gradients were detached from the image. so i had to append the blured chanels into array and then concatenate them, this solution saved the gradients and because of that i got good results.

Analysis of the effect of the blur kernel size

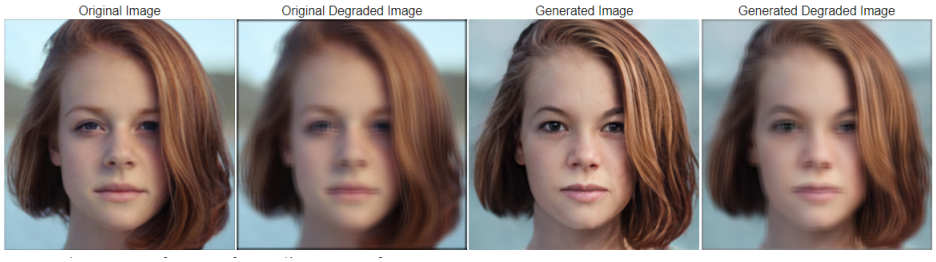
Kernel Effect

I found out that **increasing** the kernel size doesn't contribue for generating a better images. on the contrary as you can see below, increasing the kernel size made the generator to produce images which look **less similar** to the original image.

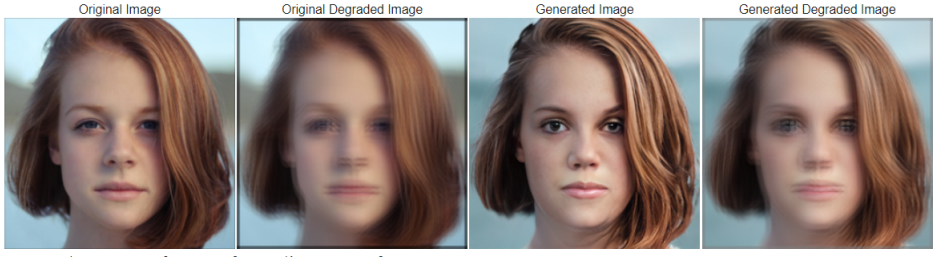
kernel_size = 5



kernel_size = 19



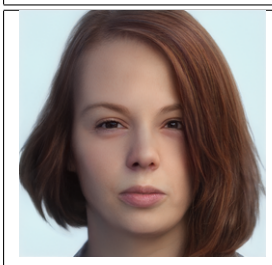
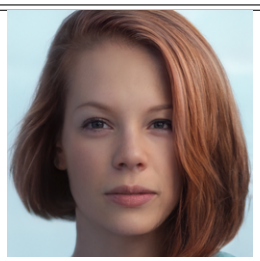
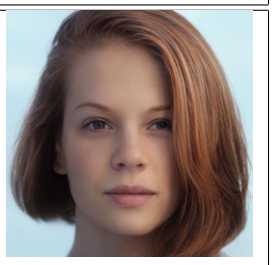
kernel_size = 23



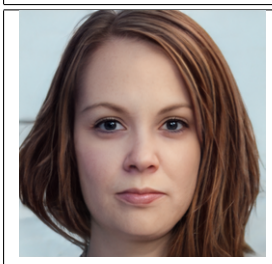
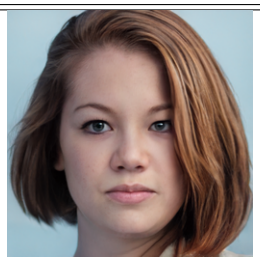
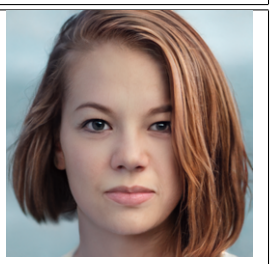
Num Steps Effect

As before, when i used **small** num_steps with the same kernel i got images which were **less similar** to the original picture:

Kernel size = 5, latent dist regweight = 1

num steps = 50	num steps = 100	num steps = 200
		

Kernel size = 23, latent dist regweight = 1

num steps = 50	num steps = 100	num steps = 200
		

latent dist reg weight Effect

as seen in the examples below, **smaller** latent dist reg weight produces images which are more simillar to the origin image

Kernel size = 5, num steps = 100

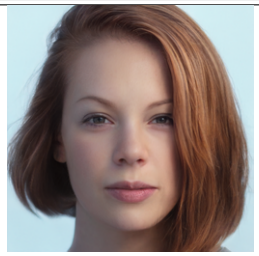
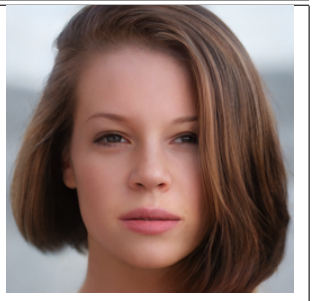
latent dist reg weight = 1	latent dist reg weight = 5	latent dist reg weight = 10
		

Image Colorization - 3.3.2

My example:

num steps = 1000, latent dist reg weight=1

original	grayscale	colorized
		

the folliwing files are found inside “my_colorization” folder:

- colorized.png
- original.jpg
- grayscale.png

- inverted_latent.npz
- Copy_of_IMPR_Ex5_Deep_Style_Image_Prior_2022_2023_ .ipynb - my code

Discuss your solution

first of all i had to run invert_image to get the grayscale image from my original image, then i ran the code with the grayscale image.

because i used the grayscale as my original image i didnt want to apply degradation and to produce grayscale image from my grayscale image so i had to blacken the following code in invert_image method:

```
# elif degradation_mode == GRAYSCALE_DEGRADATION:

#     target_images = rgb_to_grayscale(target_images)
```

for transforming the rgb image to grayscale i multiplied each channel:

0.2989 * r + 0.5870 * g + 0.1140 * b

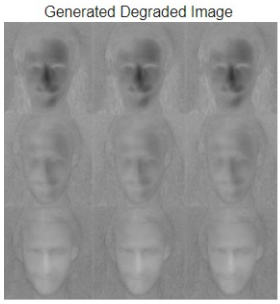
Issues i ran into and how i solved them

In exercise 1 we implemented a function named "rgb2yiq", because the Y component represents the luma information, and is the only component used by black and white television

i thought i can use this function to transfer my RGB image to YIQ representation, get the Y component and treat it as my grayScale image.

because grayScale has 1 channel and we needed 3 channels i added them (each channel is a copy of the one channel).

but when i tried to produce color image from my grayScale my generator start generating duplicated image as you can see below:

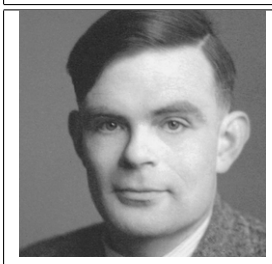
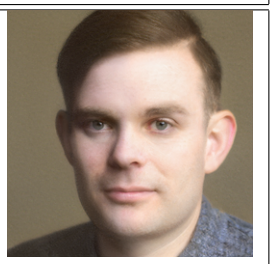


producing grayScale image with YIQ representation didnt work, so i decided to search for another solution.

I found out that i can multiple each channel of the RGB image by different scalar and produce grayScale image.

Alan Turning:

num steps = 1000, latent dist reg weight=1

Original Image	Generated Image
	

the folliwing files are found inside “Alan Turing” folder:

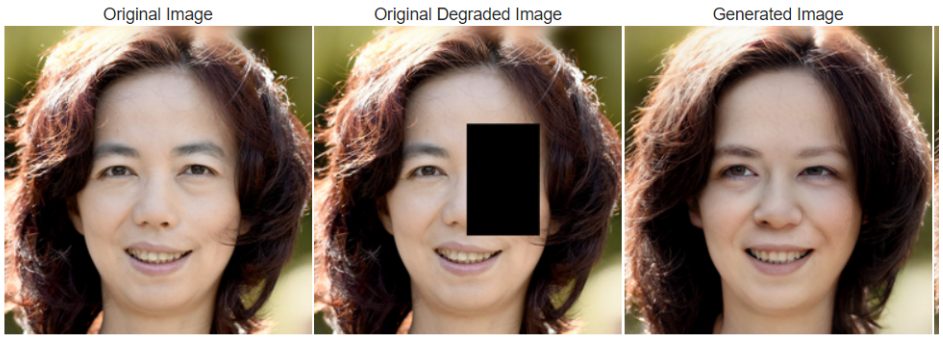
- alan_turing_grayscale.png
- final_inverted_image.png
- inverted_latent.npz
- Copy_of_IMPR_Ex5_Deep_Style_Image_Prior_2022_2023_ .ipynb - my code

The latent npz file responsible for the reconstruction result and my code for running is found in the submitted directory inside “Alan Turing” directory.

Discuss your solution
like i explained before in invert image method i had to blacken the code.

Image Inpainting - 3.3.3

inpatining Fei-Fei Li:
num steps = 1000, latent dist reg weight=0.5

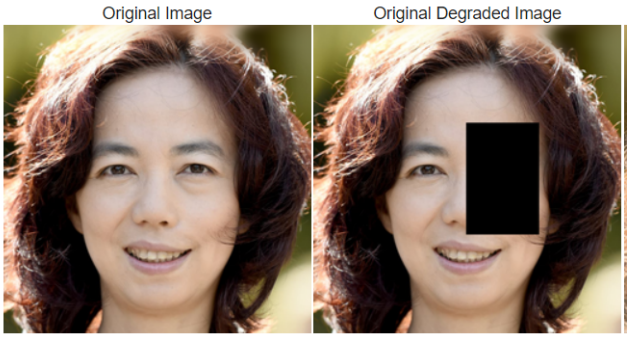


the folliwing files are found inside “fei-fei li inpaiting” folder:

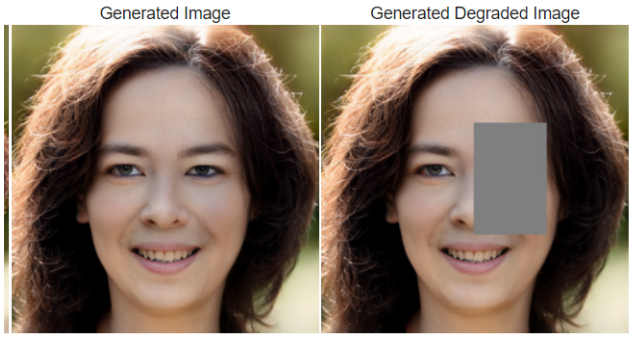
- fei_fei_li_original.png
- final_inverted_image.png

- fei_fei_li_inpainting_mask.png
- inverted_latent.npz
- Copy_of_IMPR_Ex5_Deep_Style_Image_Prior_2022_2023_ .ipynb - my code

Issues i ran into and how i solved them and Discuss your solution (for fei and my example)
In the beggining i multiplied the original image with the masked image, and i got the degredad image as expected.



But when i applied the same method on the "synth_images" (the genertated image) i got the gray mask instead of black

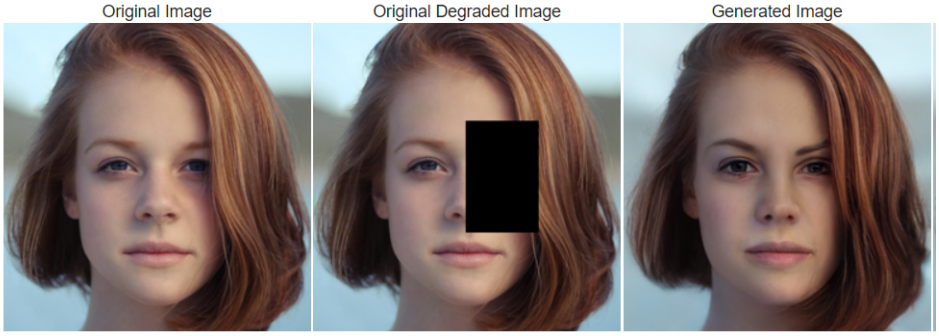


After reading online on the stylegan2 generator, i found out that the generated images are in range (-1,1)
so when i multiplied the original image with the masked image, the synth image got values equal to zero in the mask area.
zero value is right between -1 and 1 (black and white values), which means that zero value corresponds to gray color.
after undresting that, i decided to add (mask-1) to the multiplied image, now all the masked area contains -1 values instead of 0 values which correspond to -1.

Another issue i ran into was which latent dist reg weight should i use, i used different values to compare the different results i got, here are 3 of them:

latent dist reg weight = 0.01	latent dist reg weight = 0.1	latent dist reg weight = 1
		

My inpatining:



the folliwing files are found inside "my_inpaiting" folder:

- afterAlignGirl.jpg
- final_inverted_image.png
- fei_fei_li_inpainting_mask.png
- inverted_latent.npz
- Copy_of_IMPR_Ex5_Deep_Style_Image_Prior_2022_2023_ .ipynb - my code