Image Processing – Ex5

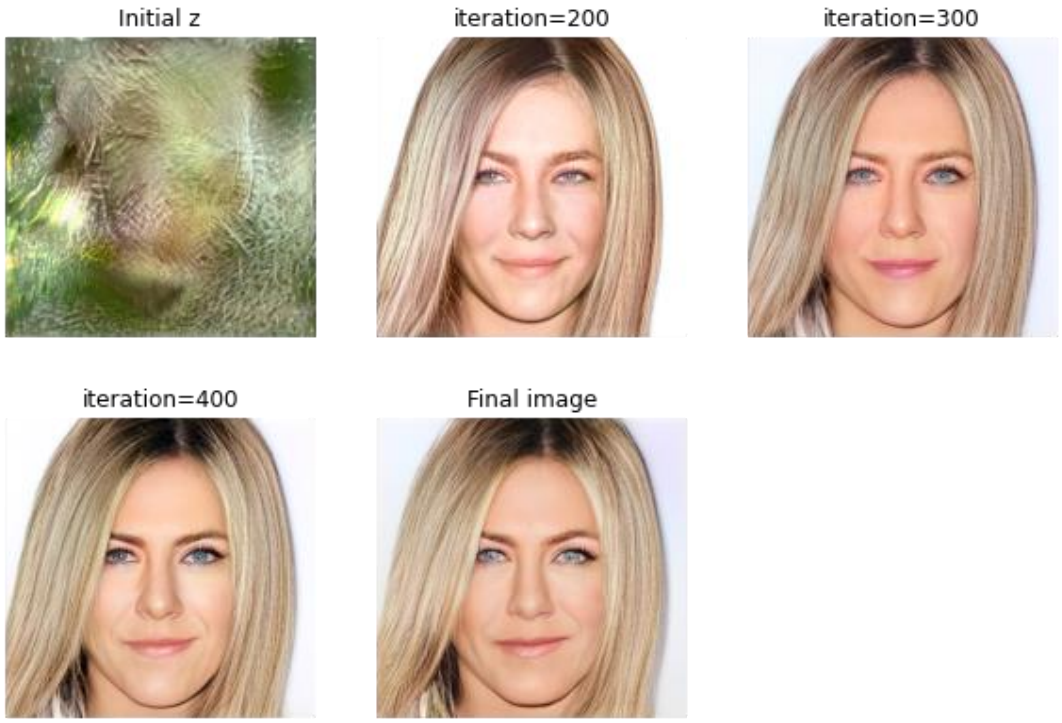
Name: Adam Shtrasner
ID: 208837260

(3.1) Image Alignment

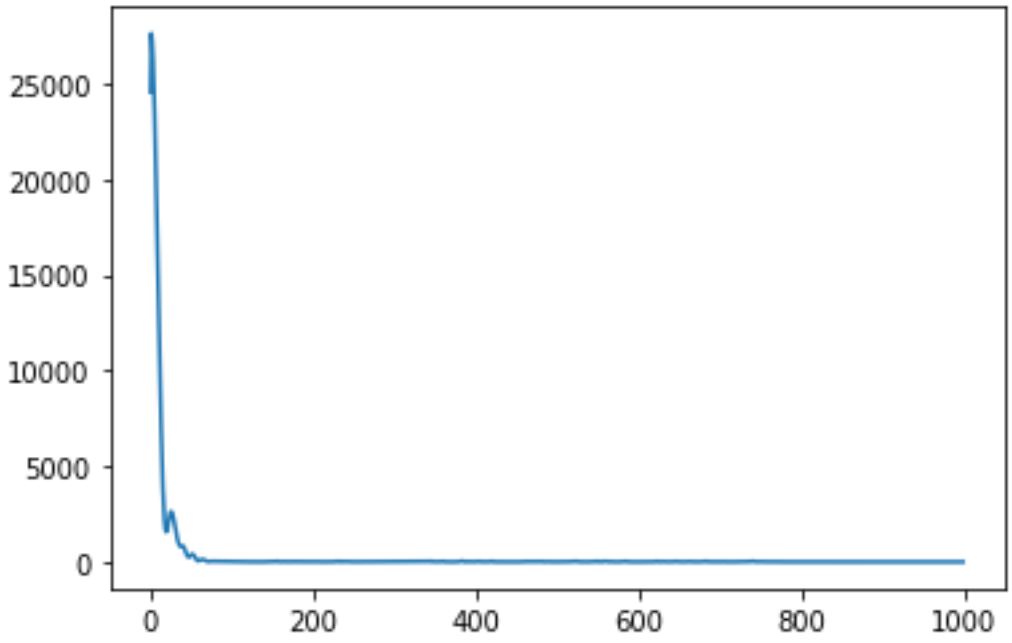
Before Alignment:

After Alignment:

(3.2) GAN Inversion



The loss plot:



- The effect of latent_dist_reg_weight and num_steps on the final results:
    - Increasing the value of latent_dist_reg_weight from 0.001 will Result in a higher loss: in my example, the loss was 0.14 While it increased towards 0.18 when latent_dist_reg_weight=0.01,0.08,0.5 and does not change much when increasing it more until 1 (when the weights are in the range (0,1) the loss is low and the generated image is similar to the original one). On the other hand, when the weights are greater than one, the loss gets higher and the generated image is much different

than the original one: when latent_dist_reg_weight=5 for example, the loss was 0.38, in contrast to 0.14 with lower weights.

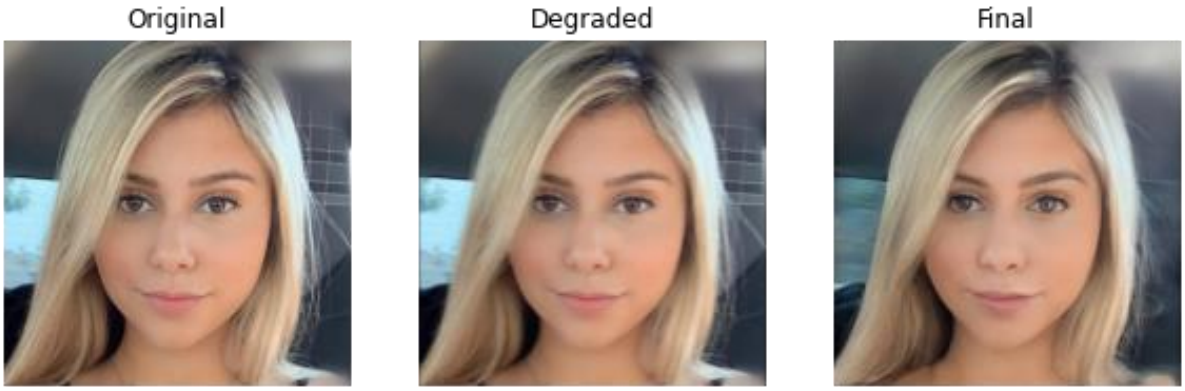
decreasing num_steps will result in a bigger loss than
When num_steps is bigger (which make sense – the process ends earlier and the loss might not converge). When changing num_steps by 100-200 the loss was pretty much the same while changing the num_steps to 500, the loss was higher and the generated image was less similar to the original image.

(3.3) Image Reconstruction Tasks

(3.3.1)

Photo 1:

Photo 2:



Yann Lacun:

Submitted Related Code:

```
from scipy import signal

def filter_construction(filter_size):
    base_filter_vec = np.array([0.5, 0.5])
    if filter_size == 2:
        return np.array([base_filter_vec])
    filter_vec = base_filter_vec
    while len(filter_vec) != filter_size:
        filter_vec = signal.convolve(base_filter_vec, filter_vec)
    return np.array([[[filter_vec]],
       [[filter_vec]],
       [[filter_vec]]])
```

```
def gaussian_blur(im, filter):
    im_blur = conv2d(im, filter, padding='same', groups=3)
    return im_blur
```

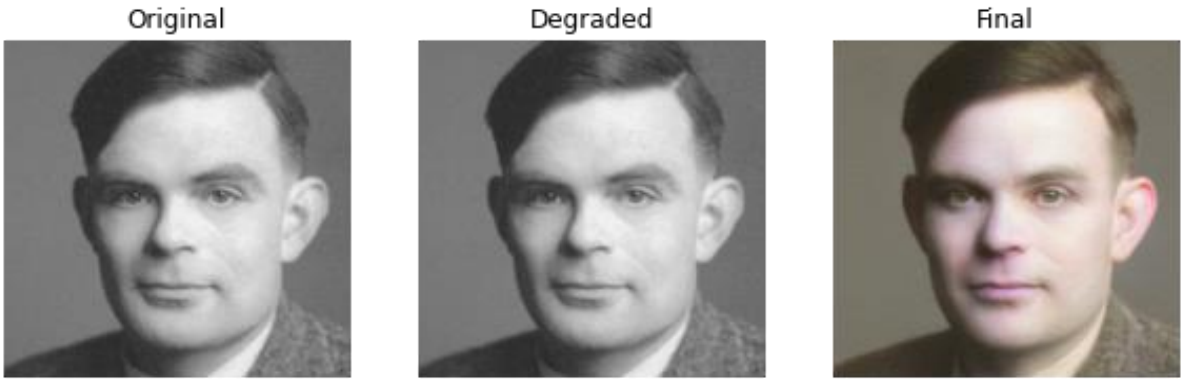
- My solution was to blur the images with conv2d. I found that kernels of sizes around 60 gave the best result (blurs that are not too strong and not too low).
- A brief discussion: I don't thing I got good result on the Yann Lacun image. I'm honesly not sure what I did wrong as I had tried many kernel sizes along with different values for latent_dist_reg_weight. On the other hand, the other images that are not originally blurred did get a good result, with pretty low weights (0.1 and lower). With greater weights, the faces began to look much different than the original ones.

(3.3.2) Image Colorization

Photo 1:

Photo 2:

Alan Turing:



Submitted Related Code:

```
def grayscale(im):
    return T.Grayscale(3)(im)
```

- The solution was to simply transform the input image (if it's RGB) to grayscale by using the torchvision.transforms function – Grayscale. To Achieve better result when painting the image, I increased the weights and stopped when the weights gave a reasonable painting + similisar face to the input.
- A brief discussion: The higher the latent_dist_reg_weight is the more the painting will be better (the colors will be reasonable to a human being), but the face it self will be less similar to the original than when the weights are smaller.

(3.3.3) Image Inpainting

Photo 1:

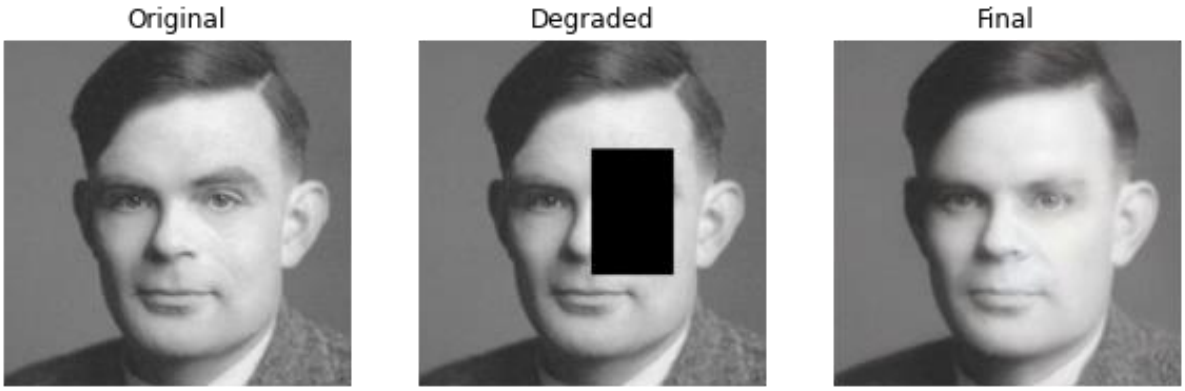
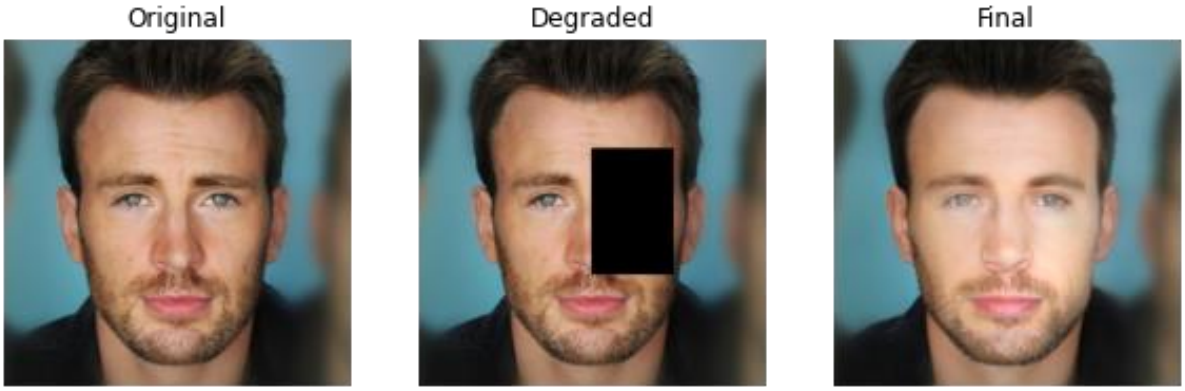
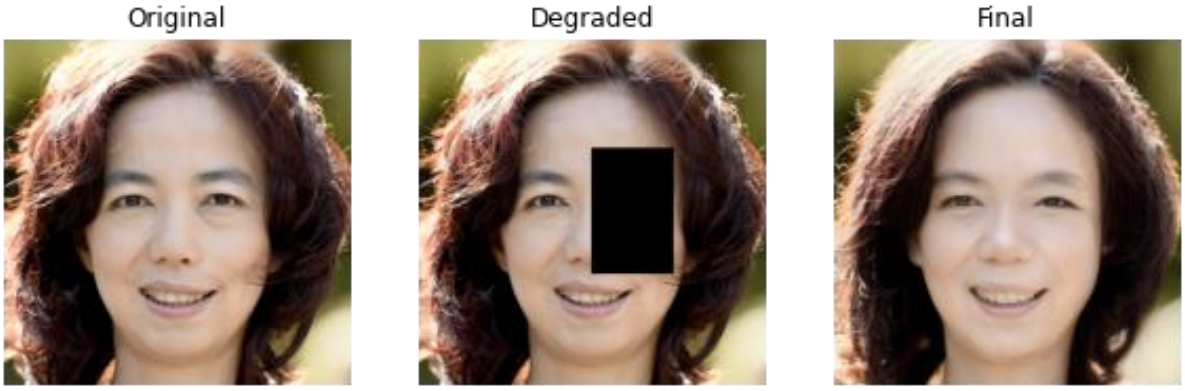


Photo 2:



Fei Fei Li:



Submitted Related Code:

```
def inpaint(im, mask):
    return im * mask
```

- The solution was to multiply the given photo by the mask, when the mask is represented in binary (in that way we get the specific black area in the input picture, and all other pixels remain the same).
- An issue I ran into was that the mask in the final result turned gray the area where the mask was in the final non degraded image was also a little bit gray. That was due to a normalization that you made on the image in the run_latent_optimization function so that the degraded generated image would be less dark and have similar colors to the original image. That normalization turned the mask gray (The normalization was adding 1 to each pixel and then multiplying it by 255/2). I changed the code by executing the normalization process only when the degrading is not inpainting, and when it is – multiply the values by 255 (because the values of the pixels are initially ranged between 0 and 1).
- A brief discussion: at each photo, there was a different value for latent_dist_reg_weight, call it x, and for every value greater than x, the missing part of the photo "looked better", as in the eye looked more like a human's eye (I used the same mask which hid the right eye of each face), but the face begins to look much different than the original face. On the other hand, for every value lower than x, the face in general will have more similar features to the original face, but the eye would look bigger, and different than the original's one.