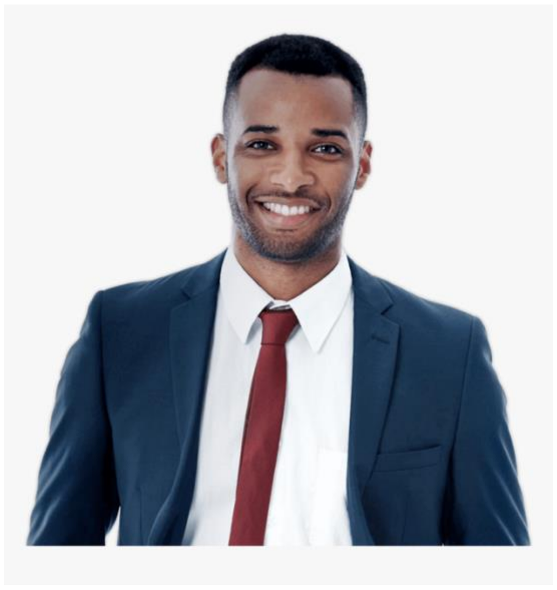
Image Processing - 67829

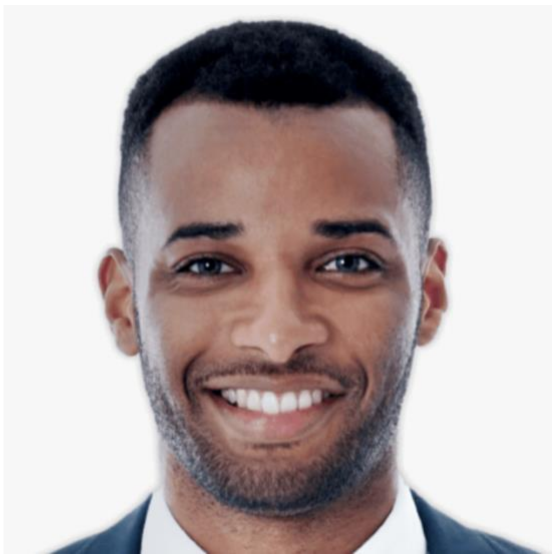
Exercise 5 Write-Up

Seaf Aliyan [redacted]

3.1 Image Alignment:-

- Input image: Output -aligned- image:

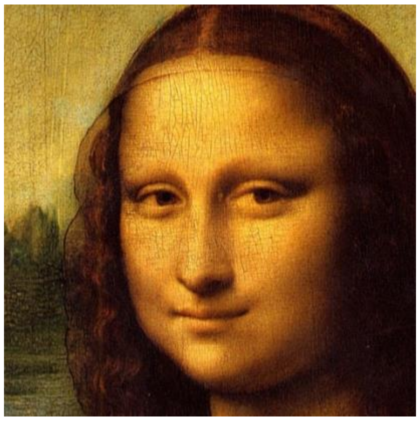
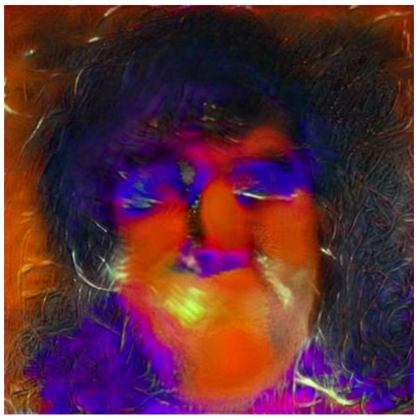
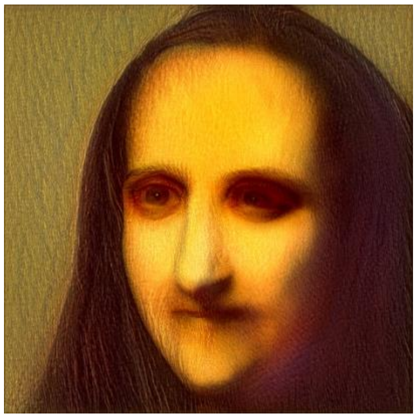
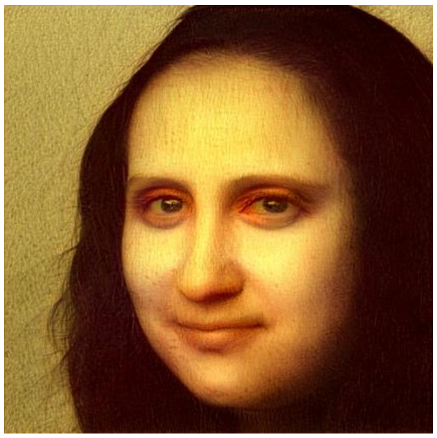
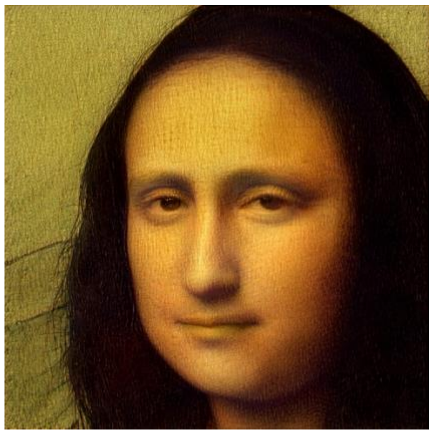
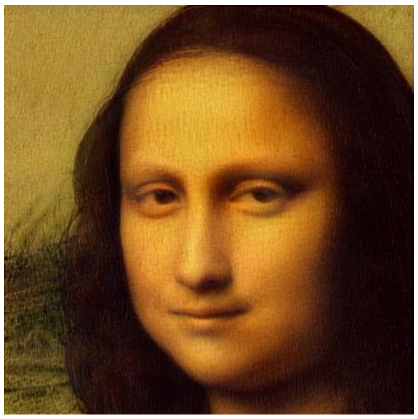
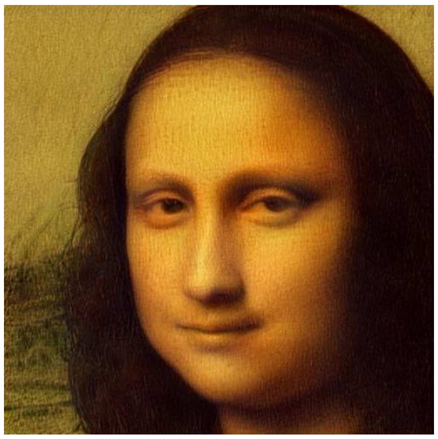
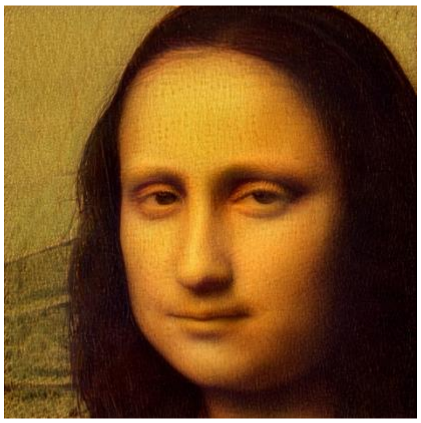
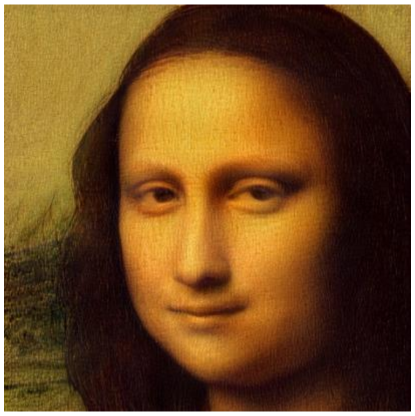




3.2 GAN Inversion:-

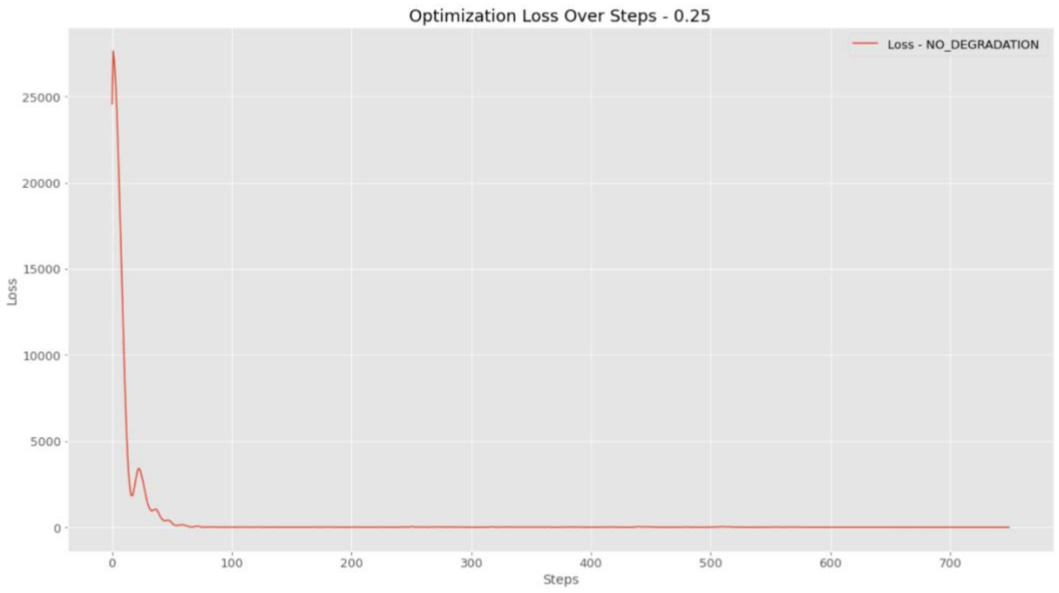
- Optimization progression (5 images at least):

Original image **first generated**

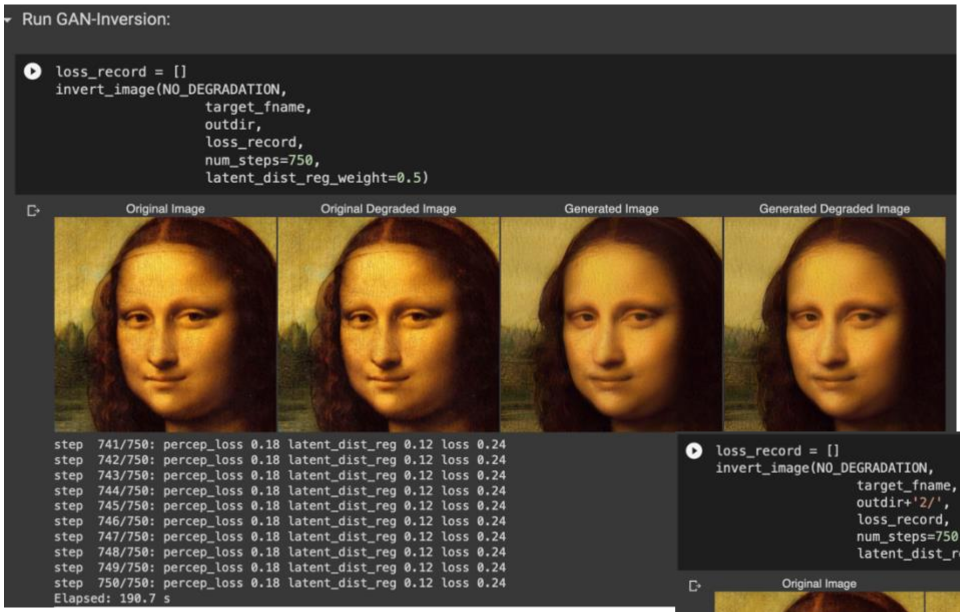










Result

- A plot of optimization loss:

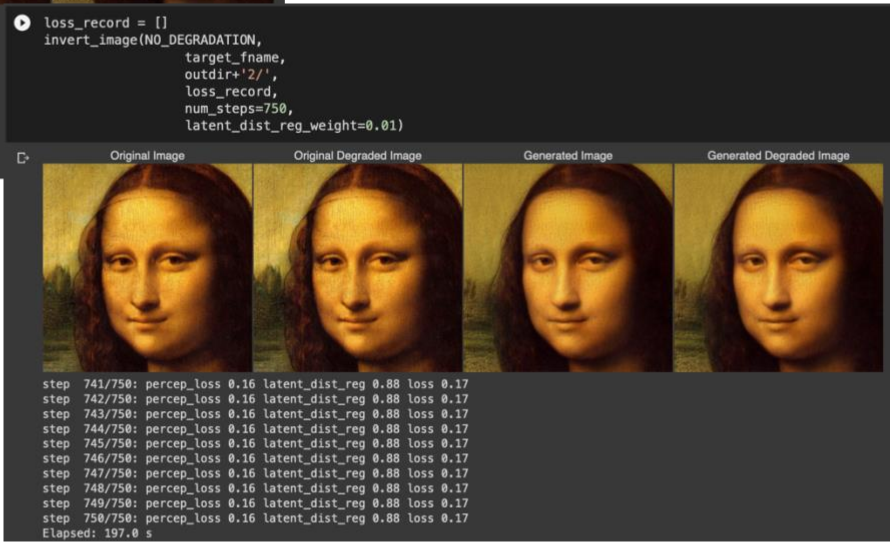


- The effect of latent dist reg weight and num steps on the results:
    - Latent_dist_reg: As we can see, higher values result in higher loss.



```
loss_record = []
invert_image(NO_DEGRADATION,
                    target_fname,
                    outdir,
                    loss_record,
                    num_steps=750,
                    latent_dist_reg_weight=0.5)
```

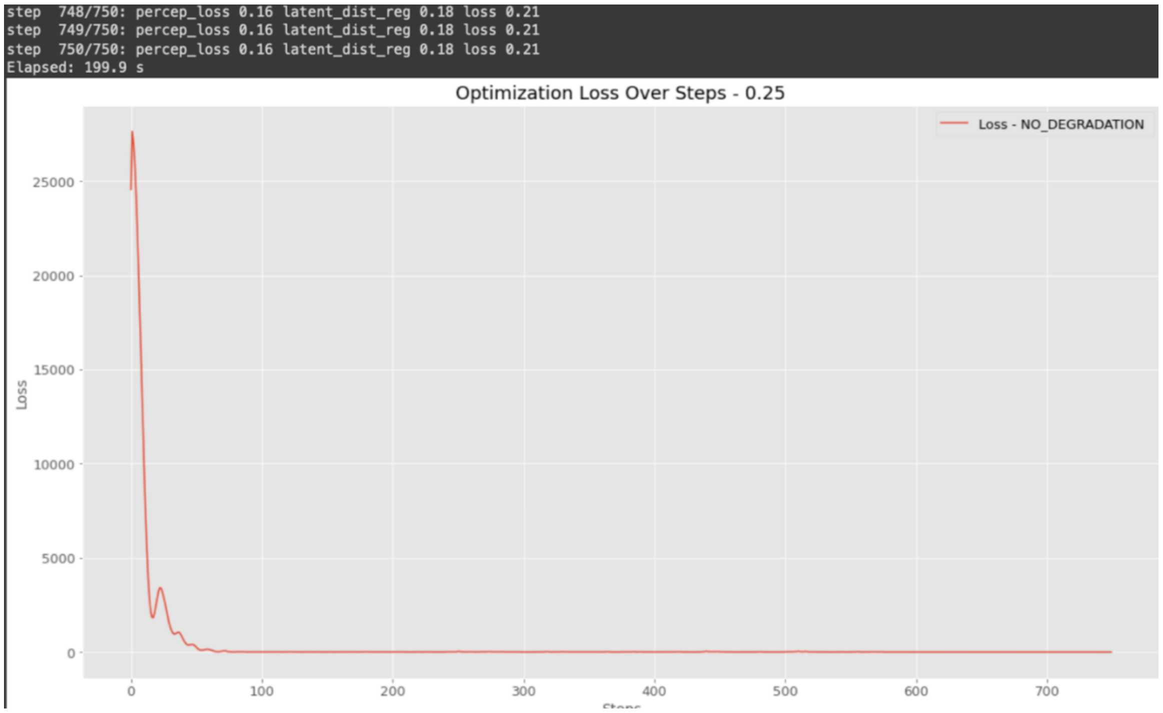
```
step  741/750: percep_loss 0.18 latent_dist_reg 0.12 loss 0.24
step  742/750: percep_loss 0.18 latent_dist_reg 0.12 loss 0.24
step  743/750: percep_loss 0.18 latent_dist_reg 0.12 loss 0.24
step  744/750: percep_loss 0.18 latent_dist_reg 0.12 loss 0.24
step  745/750: percep_loss 0.18 latent_dist_reg 0.12 loss 0.24
step  746/750: percep_loss 0.18 latent_dist_reg 0.12 loss 0.24
step  747/750: percep_loss 0.18 latent_dist_reg 0.12 loss 0.24
step  748/750: percep_loss 0.18 latent_dist_reg 0.12 loss 0.24
step  749/750: percep_loss 0.18 latent_dist_reg 0.12 loss 0.24
step  750/750: percep_loss 0.18 latent_dist_reg 0.12 loss 0.24
Elapsed: 190.7 s
```



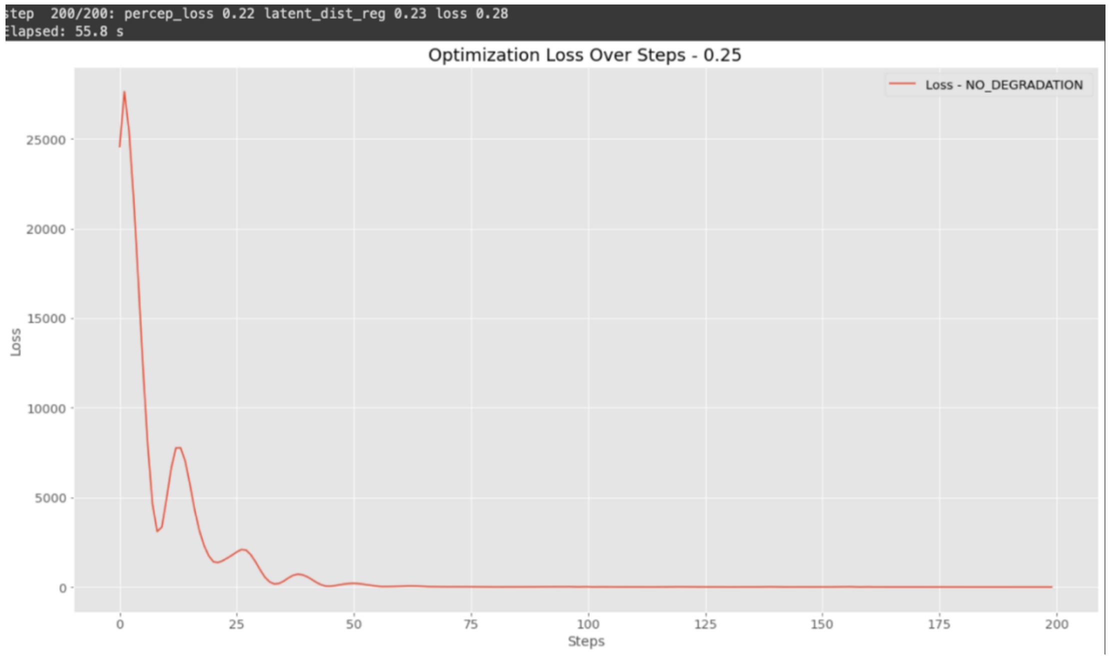
```
loss_record = []
invert_image(NO_DEGRADATION,
                    target_fname,
                    outdir+'2/',
                    loss_record,
                    num_steps=750,
                    latent_dist_reg_weight=0.01)
```

```
step  741/750: percep_loss 0.16 latent_dist_reg 0.88 loss 0.17
step  742/750: percep_loss 0.16 latent_dist_reg 0.88 loss 0.17
step  743/750: percep_loss 0.16 latent_dist_reg 0.88 loss 0.17
step  744/750: percep_loss 0.16 latent_dist_reg 0.88 loss 0.17
step  745/750: percep_loss 0.16 latent_dist_reg 0.88 loss 0.17
step  746/750: percep_loss 0.16 latent_dist_reg 0.88 loss 0.17
step  747/750: percep_loss 0.16 latent_dist_reg 0.88 loss 0.17
step  748/750: percep_loss 0.16 latent_dist_reg 0.88 loss 0.17
step  749/750: percep_loss 0.16 latent_dist_reg 0.88 loss 0.17
step  750/750: percep_loss 0.16 latent_dist_reg 0.88 loss 0.17
Elapsed: 197.0 s
```

- Num_steps: Obviously as this is a learning process the lower the num_steps is the less time we give each adversarial party to learn and improve, hence a lower num_steps give a less accurate image generation, however this doesn't mean that giving a super high num_steps will give exactly the same picture as some details are generated using stochastic operations (more steps gave about 0.16 loss while the less steps gave about 0.23).



```
step  748/750: percep_loss 0.16 latent_dist_reg 0.18 loss 0.21
step  749/750: percep_loss 0.16 latent_dist_reg 0.18 loss 0.21
step  750/750: percep_loss 0.16 latent_dist_reg 0.18 loss 0.21
Elapsed: 199.9 s
```



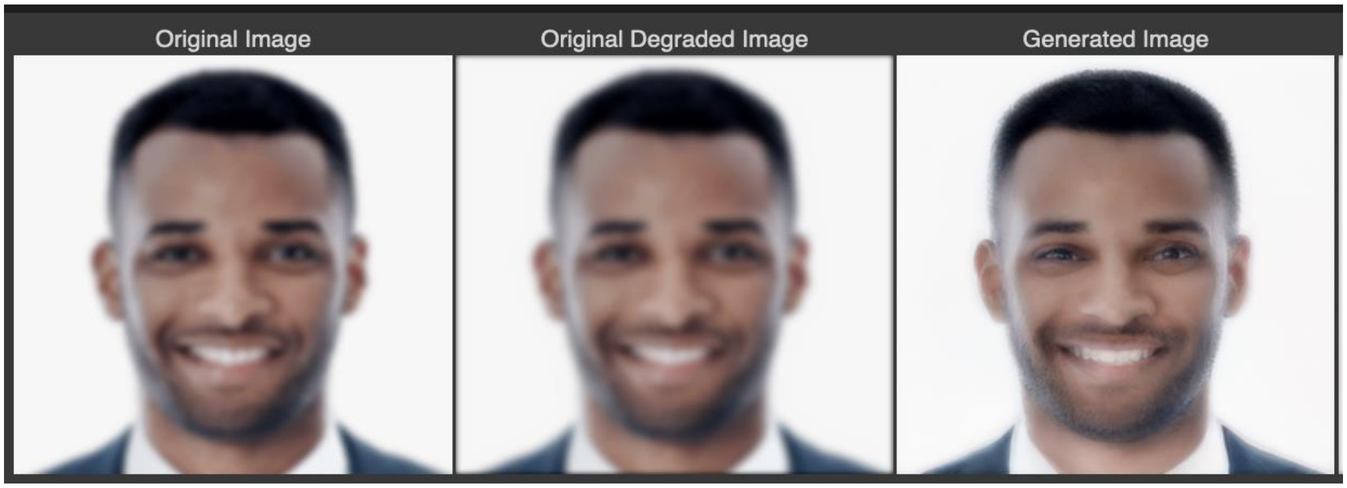
```
step  200/200: percep_loss 0.22 latent_dist_reg 0.23 loss 0.28
Elapsed: 55.8 s
```

3.3 Image Reconstruction Tasks:-

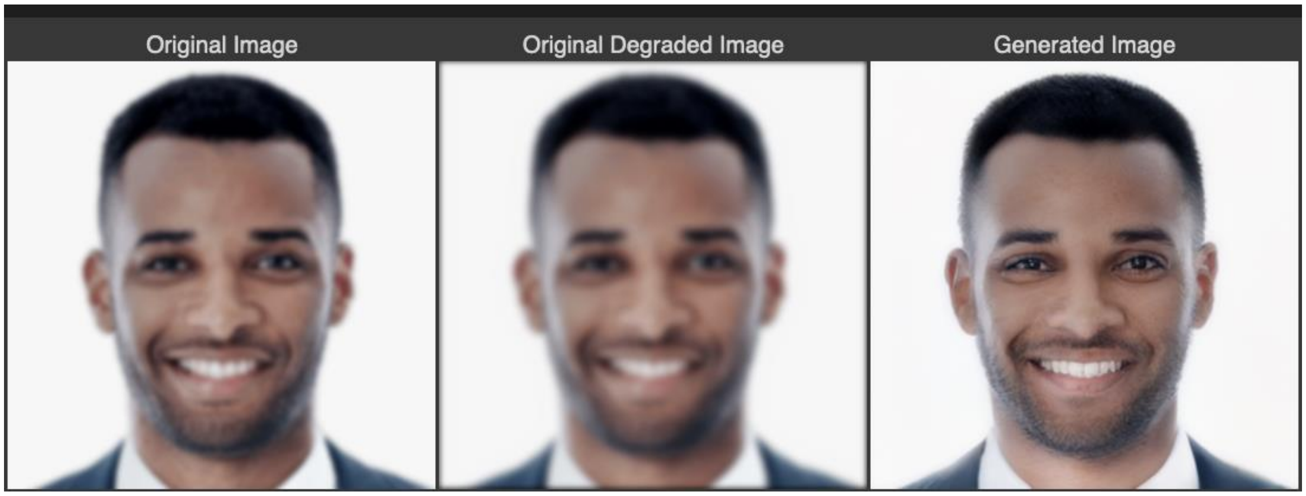
3.3.1 Image Deblurring (npz files and full images are in zip):

- The results of the deblurring for an image of your choice (original, blurred, deblurred). Also, include this image and the latent npz file responsible for the reconstruction result alongside your submitted code (the npz file is a zipped format of np files which are used to save numpy arrays, in the notebook there is already an example of saving the latents to an npz file):

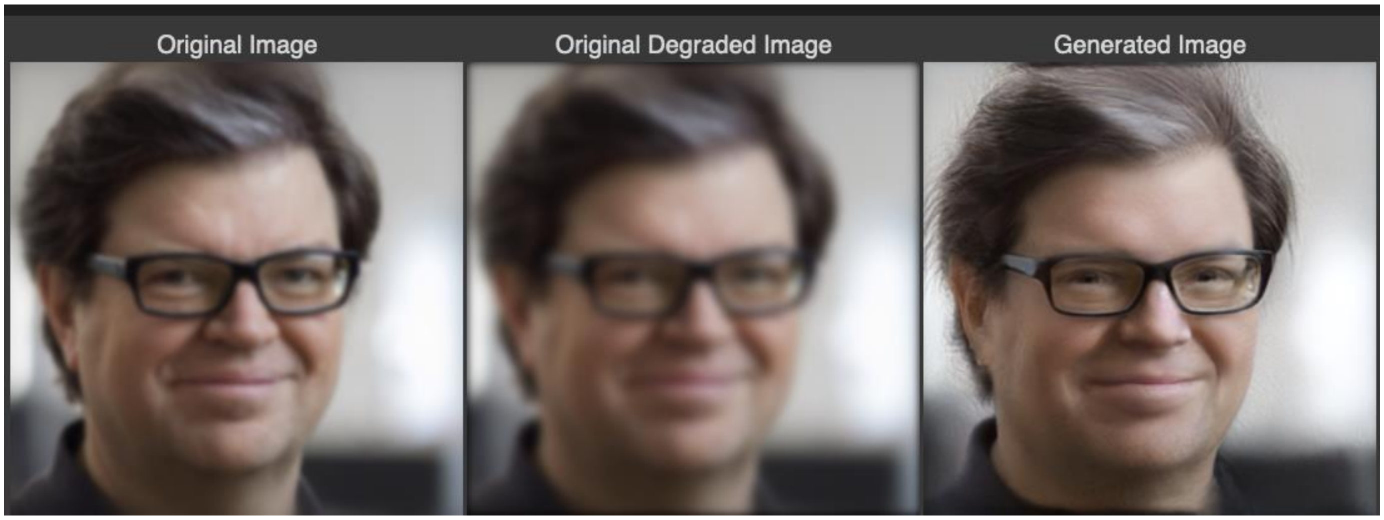
  FILTER SIZE=120



FILTER SIZE=200 -npz file is for this one, the previous is just to assess parameters-.



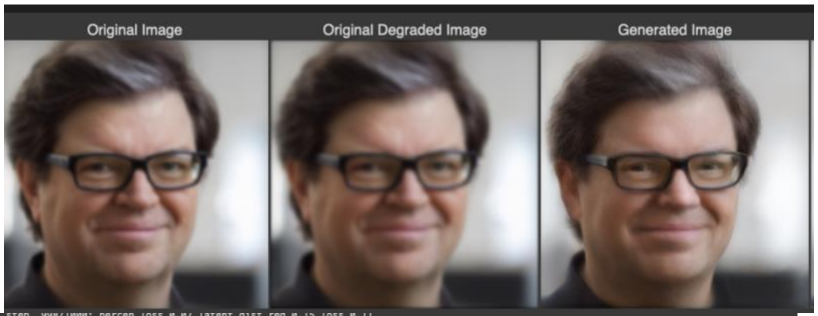
- Your results of the deblurring on the given blurry image of Yann LeCun (the blurry image from Fig. 2 - you can find it in the Moodle in a zip called ”Ex5 Supplementary”). Also, include this result and the latent npz file responsible for the reconstruction result alongside your submitted code:
  Filter size = 400, steps=1000, latent_dist_reg=0.3.



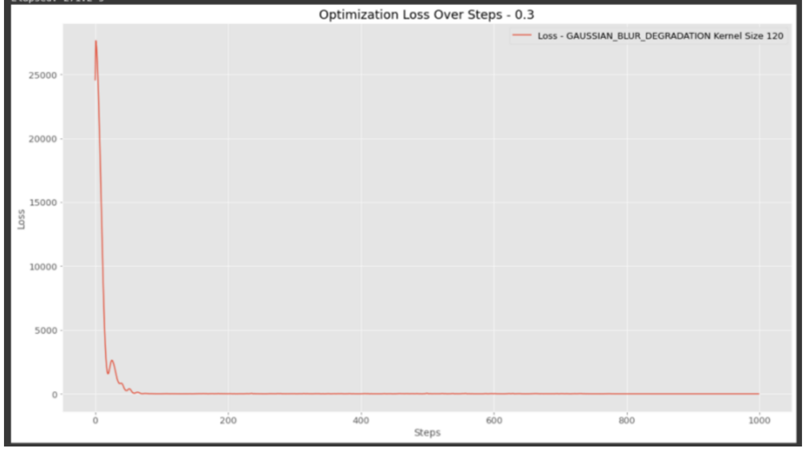
- Discuss your solution, explain what you did and the motivation for it ,if you ran into any issues, elaborate on them and how you solved them:

  I applied a low-pass twice, first in the x-direction, then in the y-direction which results in a smooth blurring effect, also I added a normalization to prevent infinite values on large kernels. I did that before using it with the neural network which improves the results of deblurring, also makes it easier for the network to understand the image and reduces noise and details. Additionally, it makes the network run faster by using less computational power.

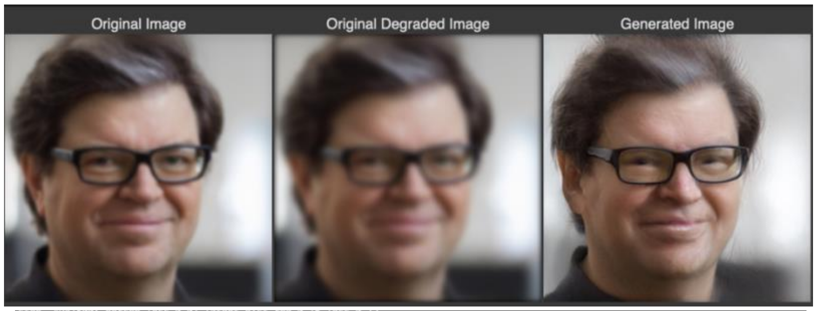
- A short analysis of the effect of the blur kernel size (with examples!) and discuss how the different hyper parameters effect this task:



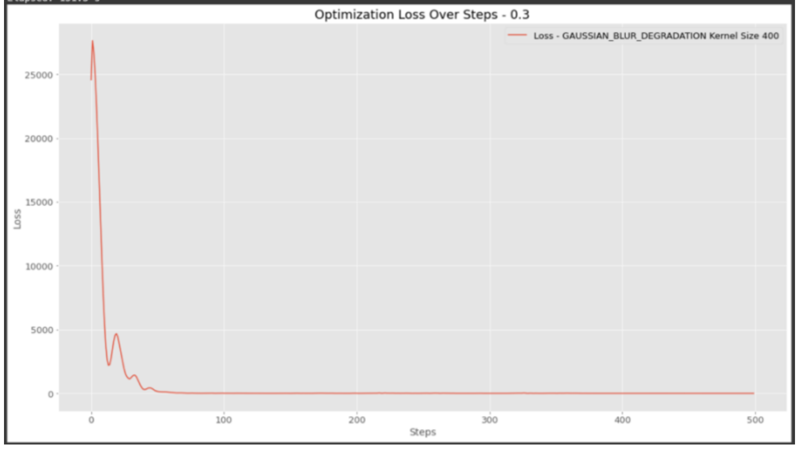
```
step  998/1000: percep_loss 0.07 latent_dist_reg 0.15 loss 0.11
step  999/1000: percep_loss 0.07 latent_dist_reg 0.15 loss 0.11
step 1000/1000: percep_loss 0.07 latent_dist_reg 0.15 loss 0.11
Elapsed: 271.2 s
```



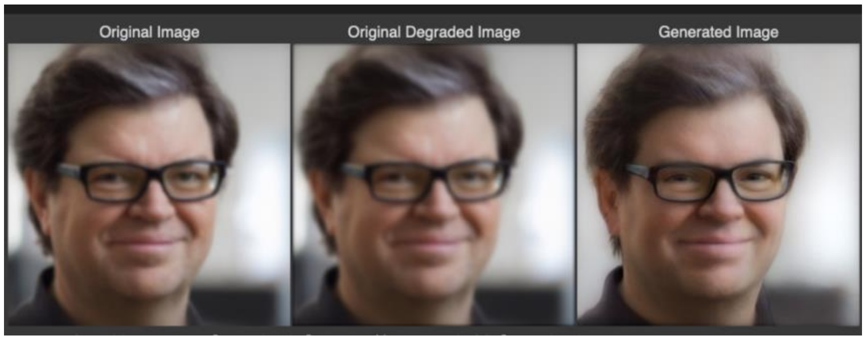
- In the next couple plots, we set iterations to 500, laternt_reg_weight=0.3, we can see that a larger filter size might help giving a better deblur, however it reduces the color accuracy (see face).

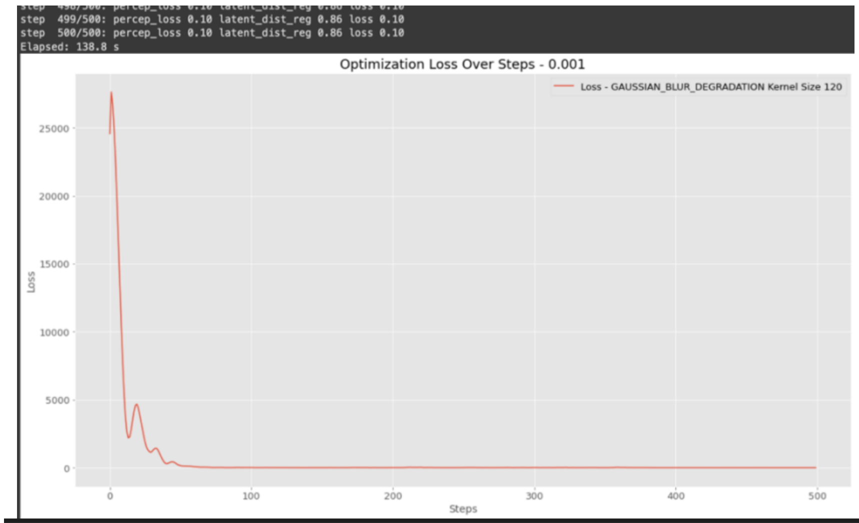


```
step  498/500: percep_loss 0.07 latent_dist_reg 0.16 loss 0.12
step  499/500: percep_loss 0.07 latent_dist_reg 0.16 loss 0.12
step  500/500: percep_loss 0.07 latent_dist_reg 0.16 loss 0.12
Elapsed: 151.3 s
```

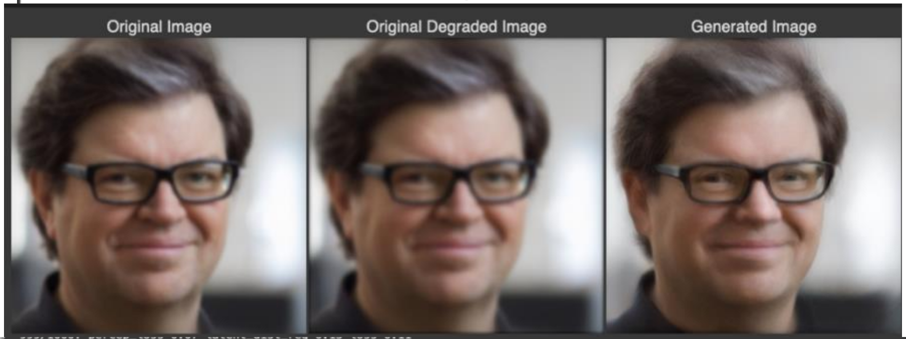


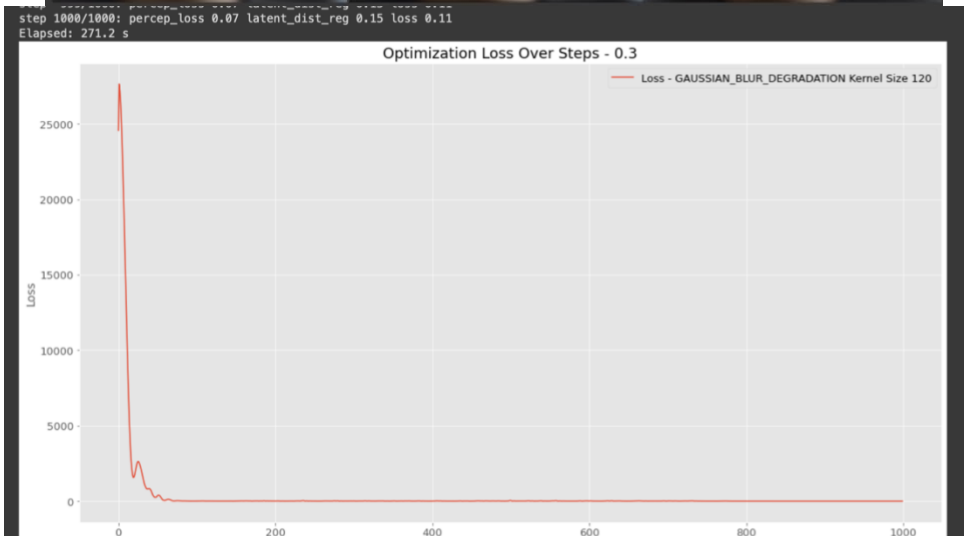
- In the next 2 plots, we're setting 500 iterations, with 120 blur size, and latent_reg_weight 0.001 for the first and 0.3 for the second, the first resulted in a more accurate colors while the second resulted in a slightly better deblur.





```
step  499/500: percep_loss 0.10 latent_dist_reg 0.86 loss 0.10
step  500/500: percep_loss 0.10 latent_dist_reg 0.86 loss 0.10
Elapsed: 138.8 s
```

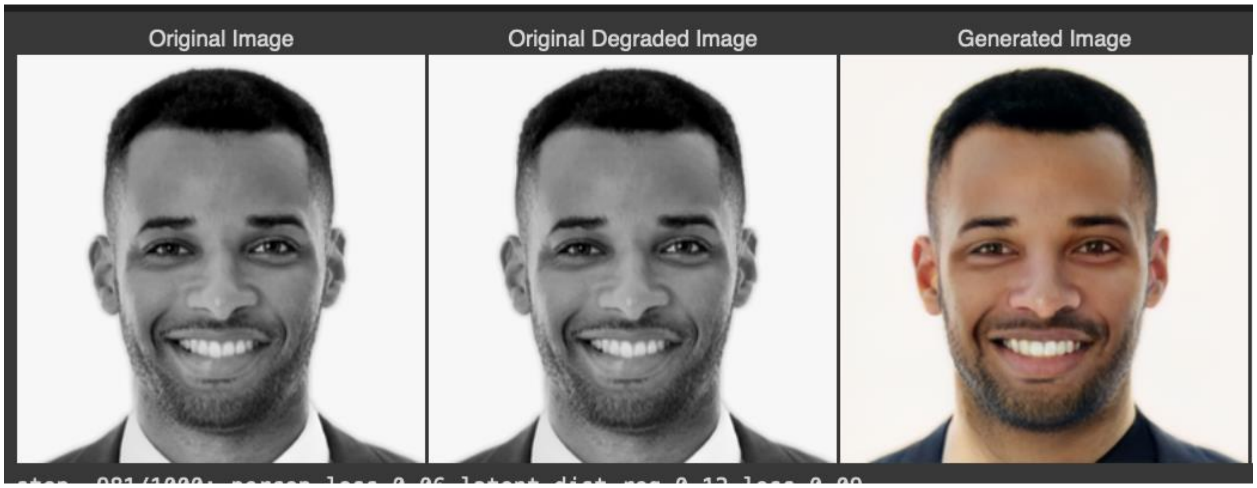




```
step 1000/1000: percep_loss 0.07 latent_dist_reg 0.15 loss 0.11
Elapsed: 271.2 s
```

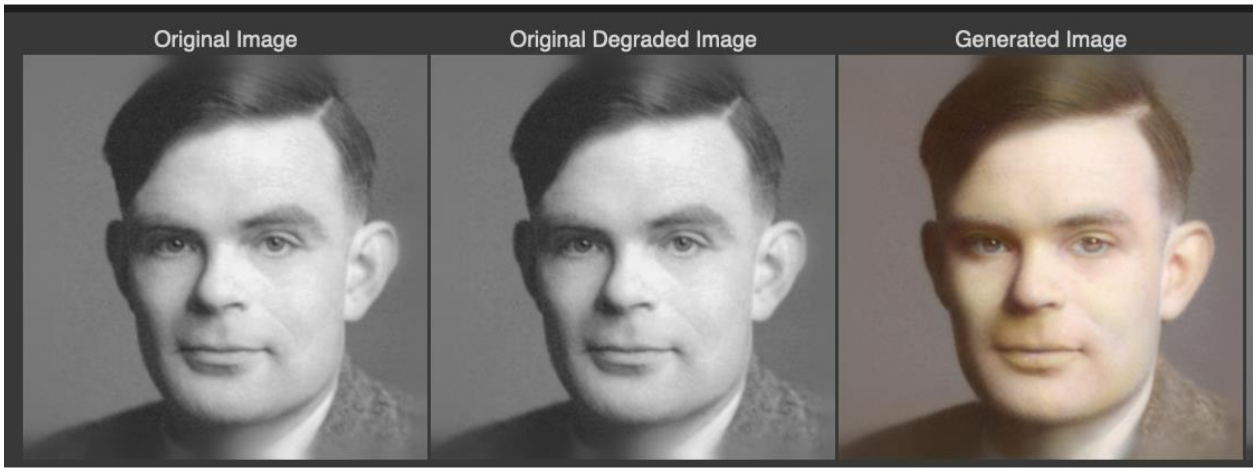
3.3.2 Image Colorization (npz files and full images are in zip):

- The results of the colorization for an image of your choice (original, grayscale, colorized). Also, include this image and the latent npz file responsible for the reconstruction result alongside your submitted code.



```
step  981/1000: percep loss 0.06 latent dist reg 0.12 loss 0.09
```

- Your results of the colorization on the given grayscale image of Alan Turing (the grayscale image from Fig. 3). Also, include this result and the latent npz file responsible for the reconstruction result alongside your submitted code.
  Steps=1000, latent

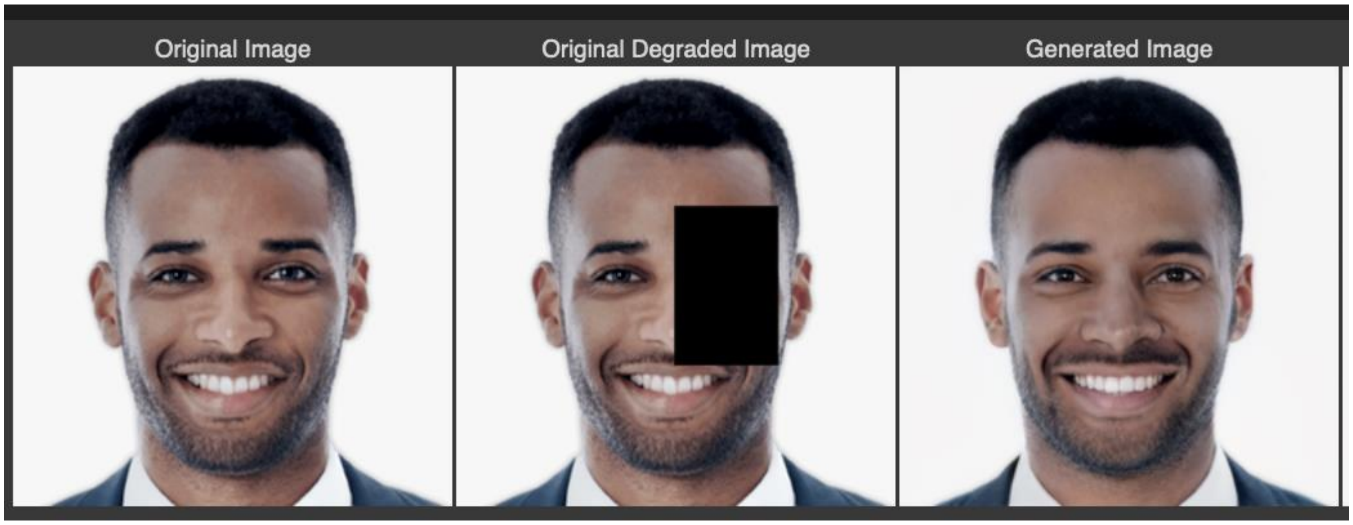


- Discuss your solution, explain what you did and the motivation for it, If you ran into any issues, elaborate on them and how you solved them.

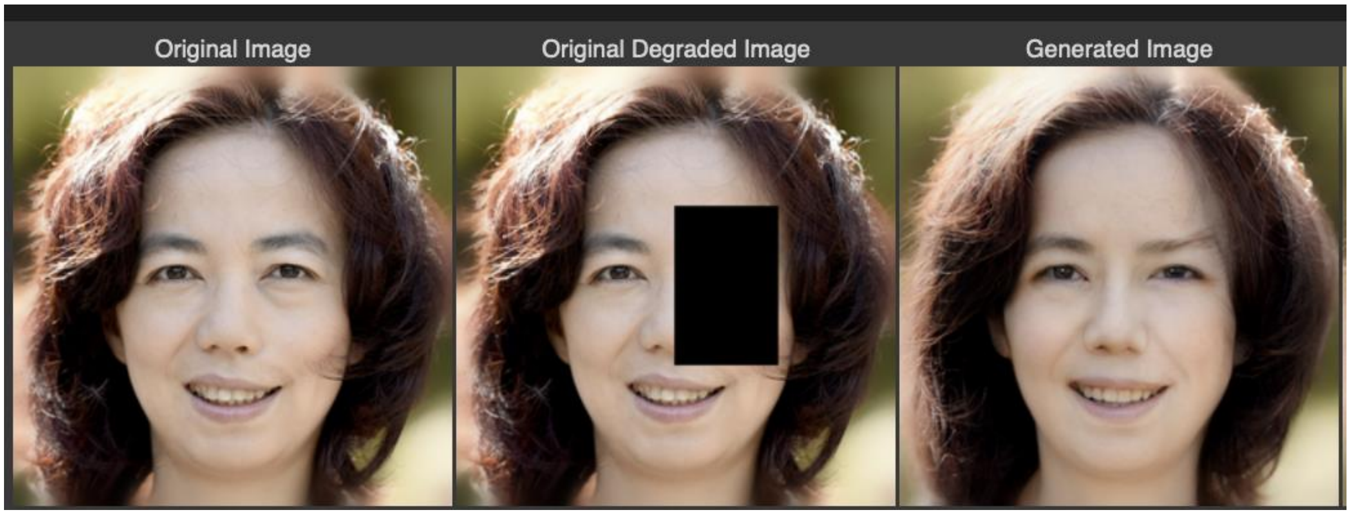
  I converted an RGB image to a grayscale image using a weighted sum of the red, green, and blue channels of the original image according to the formula in class, then I converted the resulting grayscale image into a tensor and stacked three copies of the grayscale image along the third dimension to create a 3-channel (RGB) image.
  A few problems occurred when using tensors to and some other dimensions problems, especially the need to pass the image 3 times in order to create a 3-channel image.

3.3.3 Image Inpainting (npz files and full images are in zip):

- The results of the inpainting for an image of your choice (original, masked, inpainted). Also, include the image and the latent npz file responsible for the reconstruction result alongside your submitted code.



- Your results of the inpainting on the given masked image of Fei-Fei Li (the masked image from Fig. 4). Also, include this result and the latent npz file responsible for the reconstruction result alongside your submitted code.
  Steps=1000, latent_dist_reg=0.25.



- Discuss your solution, explain what you did and the motivation for it, If you ran into any issues, elaborate on them and how you solved them:

  I made image inpainting by element-wise multiplying the original image with a binary mask to zero out the pixels in the missing or corrupted regions of the image.
  A problem was encountered where normalization caused the zeroed-out pixels to take on non-zero values, so the image was masked again after normalization and downsampling to improve the accuracy of the final generated image.