Computer Vision Programming Assignment 1

Sunghyun Kang[1],
[1]Department of Electrical and Computer Science, GIST, Gwangju, 61005 Republic of Korea

In this experiment, we will look for super resolution methods to construct a high-resolution image from low-resolution data. To begin with, we will apply the gradient descent method to obtain a high quality image. From this simple method, I was able to obtain a decent result as PSNR = 20.707. Also, by introducing prior it was able to obtain a compelling result as PSNR = 20.766. Last but not least, I introduced Richardson-Lucy deconvolution with blur kernel to obtain a better result. With bilateral filter, this turned out to be a noble result as PSNR=21.376.

***Index Terms*—gradient descent, step size, prior, MSE, PSNR, blur kernel, Richardson-Lucy Deconvolution.**

I. Introduction

RESOULTION of the images can be deteriorated by various reasons such as hand shaking, bad camera focus, etc,. Hence, there are various ways to manage bad resolution image to a high quality image. In this assignment, we will look for a very simple task that can enable us to obtain super resolution image. That is, gradient descent, prior method, and Richardson-Lucy deconvolution.

II. Methodology

A. Gradient Descent

For the super resolution, we used gradient descent as follows:

$$E = (I^l - D(I^h))^2 \tag{1}$$

$$\frac{\partial E}{\partial I_t^h} = U(D(I_t^h) - I^l) \tag{2}$$

$$I_{t+1}^h = I_t^h - \alpha \frac{\partial E}{\partial I_t^h} \tag{3}$$

Where:

- α: Step size hyperparameter
- E: Loss of the image during update
- I_t^h: Image that restored in t iteration
- $D(I_t^h)$: Image downscale function
- $U(I_t^h)$: Image upscale function

B. Super Resolution with Prior

Upgrading the previous methods, we will introduce a new prior to enable this. Hence the equation will be slightly changed from equation 1.

$$E = (I^l - D(I^h))^2 + \beta(\nabla I_t^h - \nabla I^T)^2 \tag{4}$$

$$\nabla^2 I^T = \gamma \cdot \nabla^2 I_0^h \cdot \frac{G^T}{G_0^h} \tag{5}$$

$$\frac{\partial E}{\partial I_t^h} = U(D(I_t^h) - I^l) - \beta(\nabla^2 I_t^h - \nabla^2 I^T) \tag{6}$$

$$I_{t+1}^h = I_t^h - \alpha \frac{\partial E}{\partial I_t^h} \tag{7}$$

Where:

- α: Step size hyperparameter
- β: Hyperparameter for prior
- γ: Initial image hyperparameter
- ∇: Sobel operator
- ∇^2: Laplacian operator
- G_0^h: Same as ∇I_0^h, while I_0^h refers to initial high resolution image
- ∇I^T: Our aiming gradient

C. Metric for Evaluation

To measure the MSE (Mean Square Error) and PSNR (Peak Signal to Noise Ratio) values as our benchmark, we used the following equation:

$$MSE = \frac{(I_{gt} - I_h)^2}{H \cdot W} \tag{8}$$

$$PSNR = 10 \cdot log_{10}(\frac{R^2}{MSE}) \tag{9}$$

Where:

- I_{gt}: Ground-truth image
- H: Height of ground-truth image
- W: Width of ground-truth image
- R: Maximum value of pixel of images

MSE is better as they show low value while PSNR is opposite.

III. Experiment

For this programming assignment, I submitted the files described in table I. Also, we set our local environment like this:

- python version: 3.10
- OpenCV version: 4.6.0
- numpy version: 1.23.3

To begin with, two hyperparameters β and γ was set as $\beta = 0.001$, and $\gamma = 6.0$. Other hyperparameters and iteration time

is the key for this project, so we tried to find optimal results for that. To analyze the different results due to hyperparameters and iteration time, we tested the step size for both problems between 0.0001 to 0.1 in an interval of 0.0001 value, and the big step size from 0.1 to 2.4 in an interval of 0.001 value. For the iteration, we tested our iteration time from 100 to 10000 times in interval 100 iteration as step size = 0.002. Last but not least, we also measured the tendency between accuracy performance and β and γ values. For β, I looked for the 0.0001 to 0.01 range with an interval of 0.0001, and 1 to 21 with an interval of 0.01.

TABLE I
IMPORTANT FILE LIST IN CODES DIRECTORY

File list	File explanation
problem1.py	Solution for problem 1
problem2.py	Solution for problem 2
additional_problem.py	Solution for additional method
KERNEL.csv	Blur Kernel that retrieved from [1]

IV. RESULTS

In problem 1, we can see decent performance as PSNR=20.707 as we test our step size around 0.1 to 0.9 in 1000 iteration time. To see the optimal step size for problem 1, I tested the step size as the same method that I mentioned in the experiment section. The result is shown in Figure 1, Figure 2.

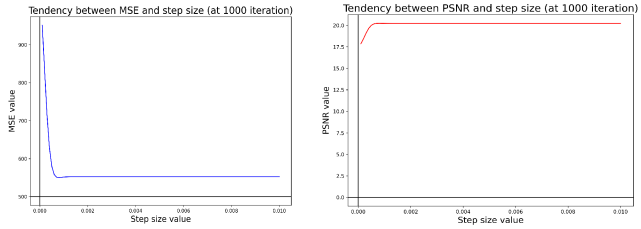

Fig. 1. Step size tendency for Problem 1 in 0.0001 to 0.002 interval. Left: correlation between step size and MSE value, Right: correlation between step size and PSNR value.

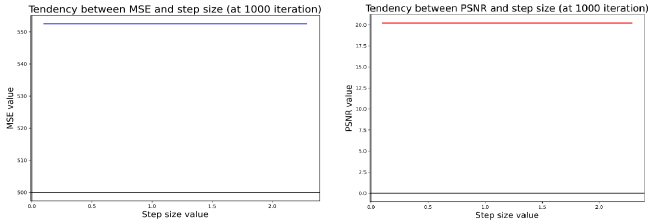

Fig. 2. Step size tendency for Problem 1 in 0.1 to 2.4 interval. Left: correlation between step size and MSE value, Right: correlation between step size and PSNR value.

Here, we can see that a very small step size (< 0.002) has a significant impact on MSE and PSNR values. If the step size is moderate (e.g. $0.002 <$ step size < 2.1), the MSE and PSNR does not affect by it. However, if we choose a large step size (e.g. step size > 2.3), then MSE and PSNR diverge. For the iteration, we analyzed the iteration time from 100 to 10000 times with interval 100 times while setting step size = 0.002. The result is shown in Figure 3.

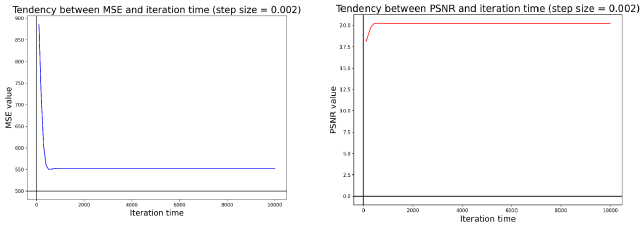

Fig. 3. Iteration tendency for Problem 1. Left: correlation between iteration time and MSE value, Right: correlation between iteration time and PSNR value.

From this iteration time, we can see that moderate amount of iteration (e.g. $1000 <$ iteration time < 10000) we can assure the descent accuracy (PSNR = 20.707). However, if we iterate in less (e.g. around 100 times) or too much (> 100000), then the image will not fully de-blurred or saturate and lead to failure.

In problem 2, we observed performance as PSNR = 20.766 which is improved compared to problem 1. Also, the tendency of step size and iteration time followed differently compared to the previous analysis. Here is the result of a problem 2 based on the step size (Figure 4, Figure 5).

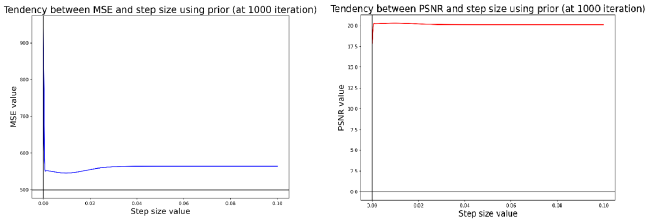

Fig. 4. Step size tendency for Problem 2 in 0.0001 to 0.1 interval. Left: correlation between step size and MSE value, Right: correlation between step size and PSNR value.

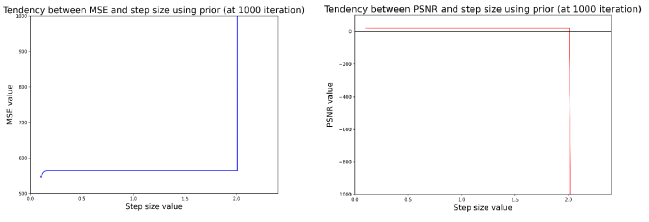

Fig. 5. Step size tendency for Problem 2 in 0.1 to 2.1 interval. Left: correlation between step size and MSE value, Right: correlation between step size and PSNR value.

As we can see, the tendency is may similar compared to problem 1 in small step size. However, the step size that used to be moderate ($0.1 <$ step size < 2.4) are may no longer appropriate since it shows worse MSE relatively compared to step size such as 0.01. Also, we can see that the result diverges faster than problem 1 in a larger step size (> 2.1).

For the iteration time, we did the same scheme as problem 1. The result is shown in Figure 6.

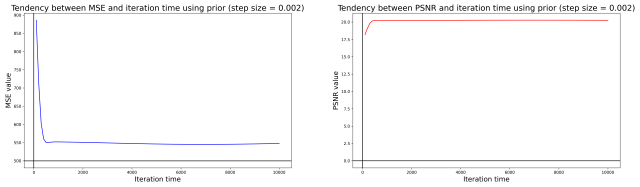

Fig. 6. Iteration tendency for Problem 2. Left: correlation between iteration time and MSE value, Right: correlation between iteration time and PSNR value.

As we can see, the iteration time tendency is quite similar to problem 1. However, we can see little accuracy improvement as we set more iteration times. But this does not imply that it guarantees the accuracy improvement with a very big iteration (as it will diverge).

For the beta and gamma values for problem 2, I checked the dependency between those hyperparameters with PSNR. For this test, I set step size = 0.0002, iteration time = 1000. Here's the result for beta (Figure 7).

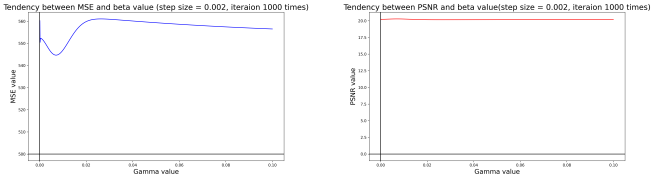

Fig. 7. Beta value tendency for Problem 2. Left: correlation between beta value and MSE, Right: correlation between beta value and PSNR.

Here, we can see that around $\beta = 0.007$, It shows the minimum MSE while showing maximum PSNR. Also, this shows that β values have a significant impact on MSE values. This kind of tendency is also shown in result of gamma (Figure 8).

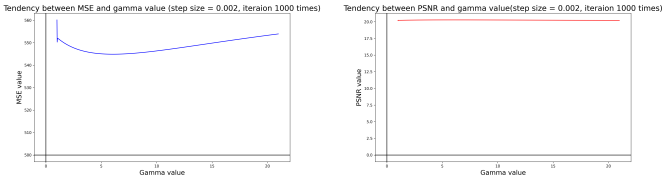

Fig. 8. Gamma value tendency for Problem 2. Left: correlation between gamma value and MSE, Right: correlation between gamma value and PSNR.

We can observe that gamma has huge impact on MSE value, and around value 6 has the most optimal results.

From the previous results, I set all the optimal parameters as like table II. This appends for both problem 1 and 2:

TABLE II
OPTIMAL HYPERPARAMETER VALUES

Hyperparameter	Value
Iteration time	10000
Step size (α)	0.01
Gamma (γ)	6.0
Beta (β)	0.007

Hence the total improvements show PSNR = 20.707 in problem 1 and PSNR = 20.766 in problem 2. The overall PSNR improvements and result image follows (Figure 9, Table III).

Fig. 9. From left to right: upsampled.png, result from gradient descent (problem1.png), result from gradient descent with prior (problem2.png).

TABLE III
IMAGE PSNR RESULTS

Image name	PSNR
upsampled.png (blurred image)	18.009
problem1.png (gradient descent)	20.707
problem2.png (gradient descent with prior)	20.766

V. MODIFIED RICHARDSON-LUCY DECONVOLUTION

To solve this problem as an additional method, I used a blur kernel based on these researches[1][2]. That is, reconstructing the blur kernel from Gaussian prior. For that kernel, I used Richardson-Lucy[3] method to obtain a modest result. The pseudo code is explained in the appendix section 1.

A. blur kernel

Initially, I tried to reconstruct all the methods to make a blur kernel based on in these papers[1][2]. However, I failed both of them due to inaccurate inference implementation and lack of computation resources. Hence I used MATLAB R2022a with a source code from the same author. This code is available at here. I changed the AXIS as [1 256 1 256] and BLUR_KERNEL_SIZE as 50 $\times$ 50 to obtain moderate size blur kernel.

B. Richardson-Lucy deconvolution

Richardson-Lucy deconvolution method is a simple way to restore the image when we have information about the blur kernel. The original equation to implement this follows here10.

$$I^h_{t+1} = I^h_t \cdot \left(\frac{I^h_0}{I^h_t \otimes K} \otimes K^*\right) \tag{10}$$

where:

- K: Blur kernel matrix that we need to estimate.
- K^*: Flipped kernel matrix.
- $\cdot$: Element-wise matrix multiplication.
- $\otimes$: 2-dimensional convolution operator.

Usually, Richardson-Lucy deconvolution is directly used to restore the image. However, in my case the image is directly saturated after very few iterations. Thus I utilized this saturation appropriately to obtain good results. To enable this, I multiplied downsampled blur image every iteration and it worked out as we can see in figure 10.

Fig. 10. From left to right: Blur kernel that obtained from the code[1], appropriately saturated image from Richardson-Lucy method, and utilized filter from left image data.

As you can see in figure 10, if we saturate the image moderately using blur kernel, then the result shows an image that is almost inverted. If we choose the value and make it as a filter, then the outline will show like the very left image of figure 10. Note that this filter-like image is exaggerated to easily inspect the filtered area.

C. results

Hence, I was able to obtain prudent results with iteration time=4 in the Richardson-Lucy method. It was notable that the image is easily saturated if I do a little bit more iteration. Since the direct result from my Richardson-Lucy implementation is inverted, I incorporated it with the result of problem 2 by appropriately choosing the pixel value from the inverted image. Last but not least, I applied a bilateral filter that provided in OpenCV library. Hence I draw the result as PSNR=21.376. Here are the figures (Figure 11):

Fig. 11. From left to right: Image result from problem 2, image result without bilateral filter, final result using bilateral filter (additional.png).

TABLE IV
PSNR RESULTS FOR EACH METHODS

Method	PSNR
gradient descent with prior	20.766
Richardson-Lucy without bilateral filter	20.757
Richardson-Lucy with bilateral filter	21.376

VI. DISCUSSIONS

In problem 1 and 2, we have seen the results that may go similarly as they showed almost the same PSNR. Also, the advantage of iteration time due to prior was not that significant as we can see in figure 3 and figure 6. However, prior hyperparameters β and γ played a significant role in image restoration (Figure 7, 8). Hence we cannot overestimate the importance of the prior. For the blur kernel restoration process, it is observed that the method of the reference article[1] cannot create an original image size blur kernel since it stacks out the kernel based on both the original blur image and Gaussian distribution. Hence I had to make a 50 X 50 blur kernel and resize all the images related to its computation. Using Richardson-Lucy deconvolution, I had to set the iteration time really small or otherwise it will saturate. This phenomenon is also observed in these researches [1][3]. Using Richardson-Lucy deconvolution as a kind of filter is unrepresented in the academic society, so this research has meaningful aspects of it. However, as we can see in the table IV, the outcome is not superior compared to gradient descent with prior. Hence we have to optimize the related values to make it more attractive.

VII. CONCLUSION

Through this lab, we have seen various methods from gradient descent to Richardson-Lucy deconvolution. It turned out that gradient descent with prior shows slightly better results than just using gradient descent. However, it has an advantage compared to normal gradient descent because prior hyperparameters γ and β significantly impact PSNR and output image quality. From this result, I mixed with a blur kernel that was resampled from Gaussian distribution by using the Richardson-Lucy deconvolution method. Adding the bilateral filter, I successfully got a better result compared to previous trials.

APPENDIX A
PSEUDO CODE

Algorithm 1 Additional Method algorithm

Require: blur image B, width w, height h, blur kernel K, width k_w, height k_h iteration time I
Ensure: $0 < I < 10$ ▷ I should be small
  $K_{flip} \leftarrow flip(K)$
  $B_{down} \leftarrow D(B, k_h, k_w)$
  $Estimate \leftarrow B_{down}$
  while iteration time $< I$ **do** ▷ R-L deconvolution
    $X \leftarrow Estimate \otimes K$
    $R \leftarrow B_{down} \div X$
    $e \leftarrow R \otimes K_{flip}$
    $Estimate \leftarrow B_{down} \times e \times Estimate$
  end while
  $Inverted \leftarrow U(Estimate, h, w)$
  $Result \leftarrow B + Inverted^*$ ▷ Choose appropriate value
  $Result \leftarrow BilateralFilter(Result)$

REFERENCES

[1] Rob Fergus, Barun Singh, Aaron Hertzmann, Sam T. Roweis, and William T. Freeman. 2006. *Removing camera shake from a single photograph*. ACM Trans. Graph. 25, 3 (July 2006), 787–794. https://doi.org/10.1145/1141911.1141956

[2] Phong Tran, Anh Tran, Quynh Phung and Minh Hoai. 2021. *Explore Image Deblurring via Encoded Blur Kernel Space*. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). (2021), 11951-11960. https://doi.org/10.1109/CVPR46437.2021.01178

[3] Fish D. A. Brinicombe A. M. Pike E. R. and Walker J. G. 1995. *Blind deconvolution by means of the Richardson–Lucy algorithm*. Journal of the Optical Society of America A, 12 (1): 58–65, https://doi.org/10.1364%2FJOSAA.12.000058