Computer Vision Programming Assignment #1

Kyeong Seop Park[1]

[1]School of Electrical Engineering and Computer Science, GIST, Gwangju, Korea

kyeongspark@gist.ac.kr

Abstract

Image deblurring is an essential technique when there was only one chance to capture a particular moment. There have been many interests in finding the optimal algorithm to recover the sharp image. Over many deblurring techniques, in this programming assignment, I have chosen the iterative approach using the Gradient Descent optimization algorithm to minimize the loss function. I have also added a prior to the loss function and had a slightly better result than the method without using the prior term inside the loss function. For an additional method, I have incorporated the guided filter and the bilateral filter and the PSNR metric has improved by about 0.61.

Introduction

Image blurring occurs frequently when it comes to taking a photograph with a shaking hand motion, a long exposure time, or capturing a fast-moving object. Although this phenomenon is usually solved by taking a new picture, there exist some circumstances when recapturing an image is prohibited. Image deblurring, reconstructing a sharp image from a blurred image, is essential for the situation to record a one and only moment. There have been several deblurring techniques to create a desired sharp image, such as the Richardson-Lucy algorithm, iterative algorithms using a prior knowledge, Wiener filtering, and a neural network approach [1].

Above these approaches, I have performed a super resolution trying to reconstruct a high-resolution image using an iterative approach. I have brought the gradient descent algorithm to optimize the loss function with and without the additional loss term of a prior making the edges inside the image sharper.

Methods

1. Gradient Descent

Gradient descent is an optimization algorithm that finds the local minimum of a particular function f. It first chooses an initial point x_0, calculates the gradient at the initial point $\frac{\partial f}{\partial x}(x_0)$, and then updates the next iteration point by the equation below,

$$x_{t+1} = x_t - \alpha \frac{\partial f}{\partial x}(x_t) \tag{1}$$

where x_t indicates the value of point x at iteration t and α a hyperparameter indicating the step size. On the gradient descent algorithm, fitting the step size α is important as if the step size is too small, it takes a long computation time to converge and has a decent probability to end up in a local minimum instead of the global minimum. Also, for the case, if the step size is too large, the value of point x might diverge not even converging into the local minimum.

2. Iterative Approach using Gradient Descent

For the first method, using the gradient descent, I have optimized the loss function E without using any prior (method 1),

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

where I^l is the low-resolution image, I^h is the high-resolution image candidate, and D is the downsampling function using a bilinear interpolation. I updated the high-resolution image candidate image by

$$I^h_{t+1} = I^h_t - \alpha \frac{\partial E}{\partial I^h_t} \tag{3}$$

$$\frac{\partial E}{\partial I^h_t} = U(D(I^h) - I^l) \tag{4}$$

where t is the iteration of the gradient descent, α is the step size, and U is the upsampling function using a bilinear interpolation. The code can be viewed and run on the file, ‘method1.py’.

3. Image Super Resolution with Prior

For the second method, I have added a prior term on the loss function E (method 2). Adding a prior term on the loss function helps to recover a high-frequency component making the edges in the image to be sharper. The modified loss function E is the following,

$$E = (I^l - D(I^h))^2 + \beta(\nabla I^h - \nabla I^T)^2 \tag{5}$$

where ∇ is the Sobel operator, ∇I^T is the prior that I am aiming for, and β is a hyperparameter, while other notations represent the same meaning of eqn. (2). I

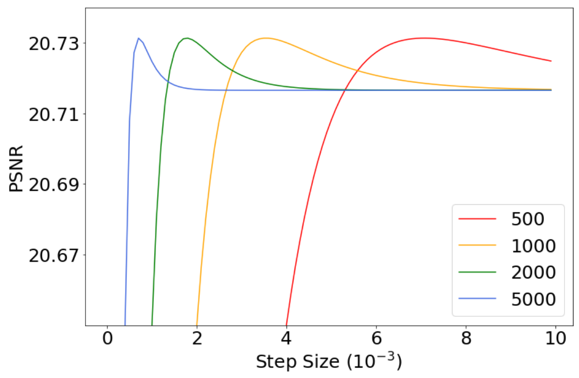

Figure 1. Method 1 results of PSNR regarding the step size α. Red, yellow, green, and blue line indicates 500, 1000, 2000, and 5000 iterations, respectively

updated the high-resolution image candidate image using the eqn. (3) with the gradient $\frac{\partial E}{\partial I_t^h}$ calculated as

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

where ∇^2 is the Laplacian operator, while other notations represent the same meaning of eqn. (4) and (5). To calculate the gradient of prior $\nabla^2 I^T$, it follows

$$\nabla^2 I^T = \gamma \cdot \nabla^2 I^u \cdot \frac{G^T}{G_l^u} \quad (7)$$

$$G^T = G_l^u - |\nabla^2 I^u| \quad (8)$$

where G_l^u is the gradient of I^l, $\nabla^2 I^u$ is the Laplacian of I^l, G^T denotes the sharp edge, and γ is a hyperparameter. The code can be viewed and run on the file, 'method1.py'.

4. Evaluation using PSNR

To evaluate the two methods of quantifying how much the image sharpened, I calculated the mean square error (MSE) with the equation below,

$$MSE = \left(I_{gt} - I^h\right)^2 / H \cdot W \quad (9)$$

where H is the height of the image, W is the width of the image, I_{gt} is the ground truth image, and I^h is the estimated image using the methods incorporated with the gradient descent algorithm. With the MSE, I have computed the peak signal to noise ratio (PSNR) by

$$PSNR = 10 \log_{10}(R^2 / MSE) \quad (10)$$

where R is the maximum value of the pixel of images, which is 255.

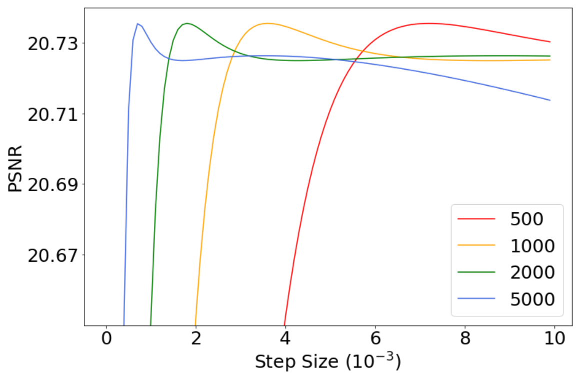

Figure 2. Method 2 results of PSNR regarding the step size α. Red, yellow, green, and blue line indicates 500, 1000, 2000, and 5000 iterations, respectively

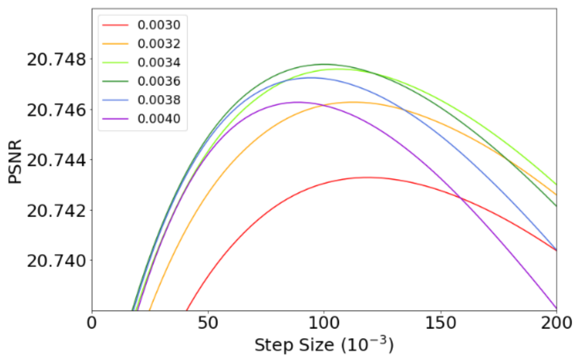

Figure 3. Method 2 results of PSNR regarding the step size β. Red, orange, lime, green, blue, and violet line indicates the step size value of $\boldsymbol{\alpha}$ being 0.0030, 0.0032, 0.0034, 0.0036, 0.0038, 0.0040, respectively

Results & Discussion

1. Iterative Approach using Gradient Descent

As I used the gradient descent algorithm to get to the global minimum of the loss function E, setting an appropriate iteration number and fitting the optimal step size is essential to get the optimal result. I have plotted the PSNR results regarding the step size α from 0 to 0.01 with a step of 0.0001 with setting the iteration number to 500, 1000, 2000, and 5000 (Fig. 1).

By Fig. 1, we can notice that the PSNR increases until it reaches a particular step size value, and then decreases as the value of the loss function diverges. The optimal step size value gets smaller as the iteration number increases since at the same step value, it's more likely to diverge while the iteration number increases. We can also note that the PSNR peak value is the same regardless of the difference in the iteration numbers. So, I have set the iteration number to 500 and the step size to 0.00707 where the derived image's

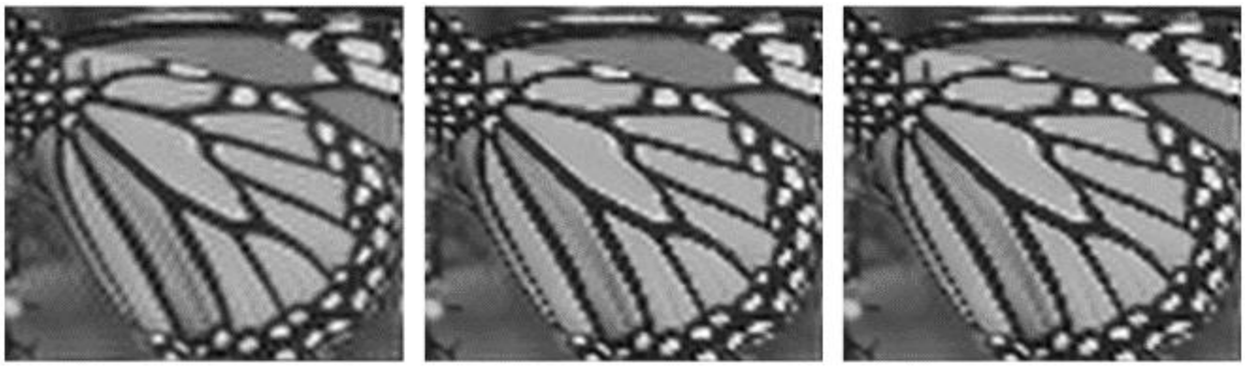
Figure 4. *Left:* Upsampled input image, *Middle:* Result image from method 1, *Right:* Result image from method 2

PSNR value is 20.73 while the PSNR value of the initial unsampled image is 18.01. For reference, I have saved the derived image as 'method1.png'.

2. Image Super Resolution with Prior

For method 2, there exist three hyperparameters α, β, γ which need fitting to get the best result image. First, same as the analysis of the first method, I have plotted the PSNR results regarding the step size α from 0 to 0.01 with a step of 0.0001 with setting the iteration number to 500, 1000, 2000, and 5000 and fixing β to 0.01 and γ to 6 (Fig. 2).

What we can notice from Figure 2 is that the overall trend of the line representing the image's PSNR value regarding the step size is similar towards Fig. 1. Until it reaches a certain step size value, the PSNR increases and then starts to decrease as the step size value becomes overly large. However, there is a small difference in the trend of the lines between Fig.2 and Fig.1 where the PSNR is converged to a higher value at the iterations of 1000 and 2000 and the PSNR value descends rather than converging at the iteration of 5000. The peak of the PSNR value regarding the step size increased slightly compared to method 1, and we can observe that the PSNR peak value is the same regardless of the iterations. So, I have fixed the iteration number to 1000 and set the range of the step size to [0.003, 0.004]. To optimally fit β, I have plotted the PSNR regarding the hyperparameter β value from 0 to 0.3 with a step of 0.003 with fixing γ to 6 and setting the α value to 0.003, 0.0032, 0.0034, 0.0036, 0.0038, and 0.004 (Fig. 3).

In Fig. 3, we can note that the green line, which has the α value of 0.0036 has the highest peak PSNR value. Thus, I have set the α value to 0.0036, β value to 0.1, and γ to 6 with the iteration to 1000, and the result image's PSNR output is 20.75 where the PSNR in method 1 was 20.73 (Fig. 4).

Figure 4 shows the comparisons of the initial upsampled image, the resulting image from method 1, and the resulting image from method 2. The PSNR value was respectively 18.01, 20.73, and 20.75. The PSNR value had a slight increase when including the prior term inside the loss function, but it wasn't dramatic. The reason why is due to the input image itself, as the input image doesn't contain a strictly vertical or horizontal edge and rather contains diagonal edges. The Sobel operator computes the gradient of the image's intensity function regarding the adjacent pixels horizontally and vertically, so there would be limitations to detect the diagonal edges with the prior term in method 2.

Additional Methods

1. Guided and Bilateral Filtering

I have thought of guided filtering to improve the results of the original methods using the optimization of gradient descent. The guided filter has two input

Algorithm 1 Guided Filter

```
Input: filtering input image $p$, guidance image $I$,
       window size $r$, regularization $\varepsilon$
Output: filtering output $q$
$mean_I = blur(I, r)$
$mean_p = blur(p, r)$
$corr_I = blur(I * I, r)$
$corr_{I_p} = blur(I * p, r)$
$var_I = corr_I - mean_I * mean_I$
$cov_{I_p} = corr_{I_p} - mean_I * mean_p$
$a = cov_{I_p} / (var_I + \varepsilon)$
$b = mean_p - a * mean_I$
$mean_a = blur(a, r)$
$mean_b = blur(b, r)$
$q = mean_a * I + mean_b$
```

Figure 5. Algorithm for the guided filter

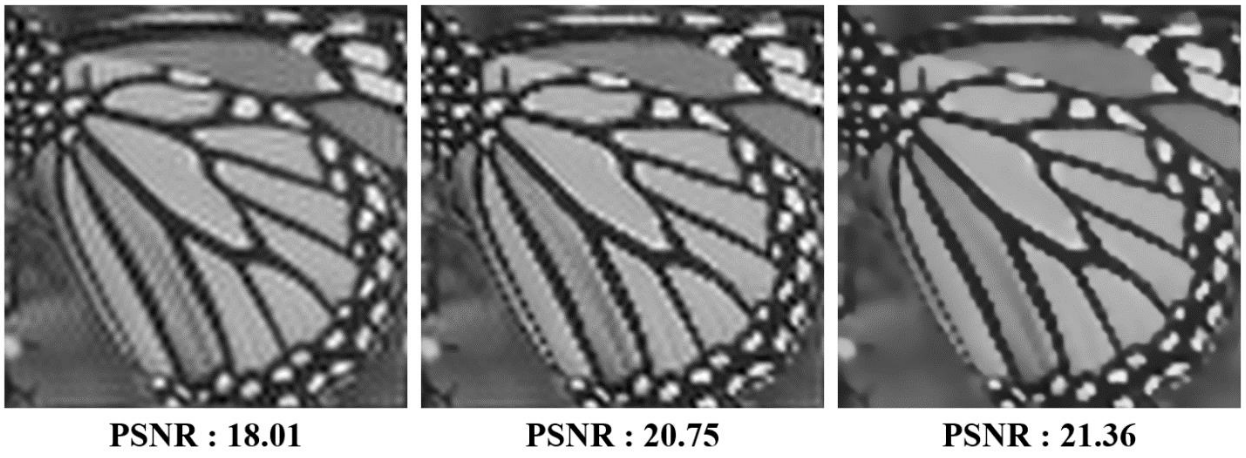

Figure 6. *Left:* Upsampled input image, *Middle:* Result image from method 2, *Right:* Image from additional method

images, the filtering input image and the guidance image with the user having to select the optimal window size and the regularization factor. The guided filter uses the pixel's mean, variance value and the correlation coefficient inside the selected window size. The algorithm for implementing the guided filter is illustrated below (Fig. 5). [2]

Guided filtering shows its effectiveness at low-light conditions with combining the inputs of flash and no-flash picture. As we only have the images of 'upsampled.png', 'method1.png', and 'method2.png', there exist limitations using only the guided filtering. With a small number of inputs, I have found out that inserting the guidance image as 'method2.png' and the filtering input image as 'method1.png' had the best result. For elaboration, I have set the window size to (8, 8), and the regularization value to 0.001.

To achieve a better result, I have also brought the function of bilateral filter. The bilateral filter is similar as to the guided filter that it smooths the image and preserves the edges, but on a single image input. The pixel value is replaced with the average of nearby similar pixel intensity values. Also, the kernel weight is based on the Gaussian distribution, but it also considers the radiometric differences such as the depth distance and the color intensity. [3] With setting the diameter of each pixel neighborhood as 10 pixels, and the filter sigma values for the color space and the coordinate space as 25, I derived the result image, 'addMethod.png' with the PSNR value of 21.36, which is 0.61 greater than the PSNR value from 'method2.png.' (Fig. 6) For reference, the code can be viewed and run at 'addMethod.py'.

References

1. El-Henawy, I., A. Amin, and H.A. Kareem Ahmed, *A comparative study on image deblurring techniques.* International Journal of Advances in Computer Science and Technology (IJACST), 2014. **3**(12): p. 01-08.
2. He, K., J. Sun, and X. Tang, *Guided image filtering.* IEEE transactions on pattern analysis and machine intelligence, 2012. **35**(6): p. 1397-1409.
3. Tomasi, C. and R. Manduchi. *Bilateral filtering for gray and color images*. in *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*. 1998. IEEE.