Computational Photography Homework 4

– Deconvolution

20170243 SimJaeYoon

1. Overview

This assignment 4 is about deconvolution techniques. What I need to implement is two deconvoltuion techniques. One is wiener deconvolution, and another is TV deconvoltuion. And for the second one, I also need to derive an energy function for a classical non-blind deconvolution method, and implements its optimization. In the case of non-blind deconvolution, the blur kernel is already known. I rewrited given sample codes written in MATLAB given by the professor. The followings are the images and kernels for testing.

Testing image and blur kernels

2. Implements and Results

Image deburring is one of the image restoration techniques that removes blur from the blurred image and makes it a sharp image as follows. In this problem, the goal is to restore it to the latent sharp image when given the blurred image. In the latent sharp image, there should be no camera shaking or object shaking.

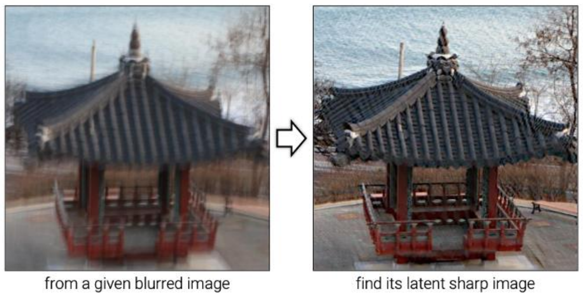
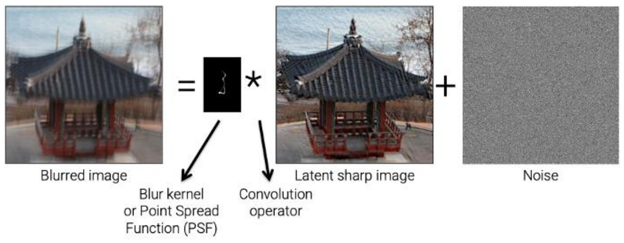


Image deblurring or deconvolution

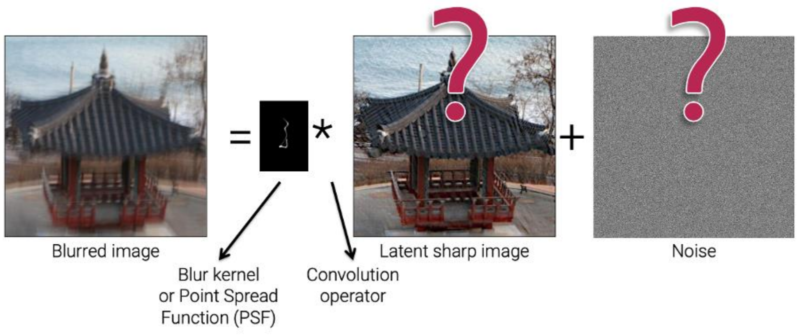
There are several approaches to solve image deblurring, but there is a way to approach the most effective classical approach before deep learning with inverse problem. An image degradation model is required to approach with inverse problem. In this problem now, image degradation is image blur. So, this will be called the image blur model. So, the following is the most commonly used image blur model.



General image blur model

This model explains how blurred images are created. In this model, given the blurred image as above, this blurred image is made by blur kernel, or by the convolution operation of point spread function (PSF) and latent share image. Basically, the latent sharp image assumes that there is no blur, and this sharp image is changed to the blurred image by the blur kernel. If we look closely at the blur kernel, we can see that there is some trajectory. This blur kernel explains how shaken the camera is. In the above case, it can be seen that the camera shook vertically from top to bottom or from bottom to top. So this kernel tells us how shaken the camera is or how much the object has moved. In the latent sharp image, blur is created by this blur kernel and additional noise is combined.

In the case of non-blind deconvolution, we know about blur kernel. That's why we know the existence of the blur kernel and want to restore it to the latent sharp image. So, whether it is blind or non-blind can be determined according to whether we know the blur kernel or not.



Non-blind deconvolution

The most naïve approach to non-blind deconvolution is the inverse filtering. We can derive the inverse filtering by applying the Fourier transform to the blur model. In the inverse filtering, we use the element-wise multiplication opeartor and pixel-wise divison in the frequency domain. Then, by computing the inverser Fourier transform, we can obtain a deconvolution result. As this technique directely applies the inverse of a blur kernel to the input blurred image in the frequency domain, it is called inverse filtering. However, it cannot produce satisfying result because blur kernel in the frequency domain may have elements whose values are zero or close to zero, and dividing blurred image in frequency domain with such elements leads to totally corrupted images with significnatly amplifed noise.
In this assignment, I need to implement two non-blind deconvolution algorithms that remove blur from a given blurred image(b) and a blur kernel(k). And to overcome the issue of inverse filtering, we need proper handling of noise and regularization.

2.1. Derivation of the energy function that MAP formulation

The following are methods of blind deconvolution that are mainly used. Blind deconvolution is a more difficult problem because the number of unknowns is more than other image restoration techniques such as image denoising or super-resolution. So many methods emerged to solve blind deconvolution.

(1) Maximum posterior(MAP)
(2) Variational Bayesian
(3) Edge prediction

MAP should find a solution that maximizes the following poster distribution. b is a given blurred image, *l* is a latent sharp image, and k is a blurred kernel.

$$(\hat{l}, \hat{k}) = \underset{l,k}{\operatorname{argmax}}\, p(l, k|b)$$

So, given the blurred image, we want to find the most likely latent sharp image and blur kernel. To solve this problem, the image degradation model must be defined first. Therefore, the posterior distribution can be formulated based on this model. The simplest way is to use the Bayes' rule.

$$p(l, k|b) \ \propto p(b|l, k)p(l)p(k)$$

Therefore, likelihood and prior must be formulated. Therefore, energy function can be derived using data term and regularization term as follows. So it turns into finding *l*, *k* that minimizes energy function. Here, *l* and *k* have two unknowns.

$$(\hat{l}, \hat{k}) = \underset{l,k}{\operatorname{argmin}}[\|b - k * l\|^2 + \rho_l(l) + \rho_k(k)]$$

When input blurred image *b* is given, the latent image *l* estimation is first performed. Assuming that a specific *k* is given at this stage, the above energy function is minimized for *l*. That is, the initial value of *k* is given and fixed, and then the energy function with respect to *l* is minimized. In the next step, update the latent image *l* and fix it again to minimize the energy function for *k*. The latent image and blur kernel can be improved by repeating the process of updating *l* and *k* in this way. After repeating this process several times, we will finally get a sharp image without blur. In the latent image estimation step, *k* is fixed, which corresponds to non-blind deconvolution which what we have to do.

We can formulate the non-blind deconvolution problem as a maximum-a-posteriori(MAP) problem, and derive the energy function $E(l)$ described above. We are given blurred image(b) and blur kernel(*k*), so by using these, we can obtain optimized latent sharp image(l^*).

$$p(l \mid b, k) \propto \underbrace{p(b \mid l, k)}_{\text{likelihood}} \; \underbrace{p(l)}_{\text{prior}}$$

$$\boxed{\text{energy function } E(l) = \|b - k * l\|^2 + \alpha E_{TV}(l)}$$

$$\begin{aligned}
l^* &= \underset{l}{\operatorname{argmax}}\; p(l \mid b, k) \\
&= \underset{l}{\operatorname{argmax}}\; \frac{p(l, b \mid k)}{p(b)} \\
&= \underset{l}{\operatorname{argmax}}\; \frac{p(b \mid l, k)\, p(l)}{p(b)} \\
&\approx \underset{l}{\operatorname{argmax}}\; p(b \mid l, k)\, p(l) \quad (\because p(b) \text{ is not relevant to } l)
\end{aligned}$$

To solve the MAP problem, we need the negative logarithm first.

$$\begin{aligned}
l^* &= \underset{l}{\operatorname{argmax}}\; p(l \mid b, k) \\
&= \underset{l}{\operatorname{argmax}} \left\{ \log p(l \mid b, k) \right\} \\
&= \underset{l}{\operatorname{argmin}} \left\{ -\log p(l \mid b, k) \right\} \\
&= \underset{l}{\operatorname{argmin}} \left\{ -\log p(b \mid l, k)\, p(l) \right\} \\
&= \underset{l}{\operatorname{argmin}} \left\{ \underset{①}{-\log p(b \mid l, k)} \underset{②}{- \log p(l)} \right\}
\end{aligned}$$

① Assume that noise distribution follows normal distribution with 0 mean and σ standard deviation.

$$\begin{aligned}-\log p(b|l,k) &= -\log N(b-k*l\,|\,0,\sigma^2)\\ &= -\log \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{\|b-k*l\|^2}{2\sigma^2}\right)\\ &= \log \sigma\sqrt{2\pi} + \frac{\|b-k*l\|^2}{2\sigma^2}\end{aligned}$$

② Assume that ∇l follows Laplace$(0,b)$ to solve L1-norm.

$$\begin{aligned}-\log p(l) &= -\log \frac{1}{2b}\exp\left(-\frac{\|\nabla l\|_1}{b}\right)\\ &= \log 2b + \frac{\|\nabla l\|_2}{b}\\ &= \log 2b + \frac{1}{b}\left(\|\partial x*l\|_1 + \|\partial y*l\|_1\right)\end{aligned}$$

From ① and ②, $\partial_x = [0,-1,1]$, $\partial_y = [0,-1,1]^T$

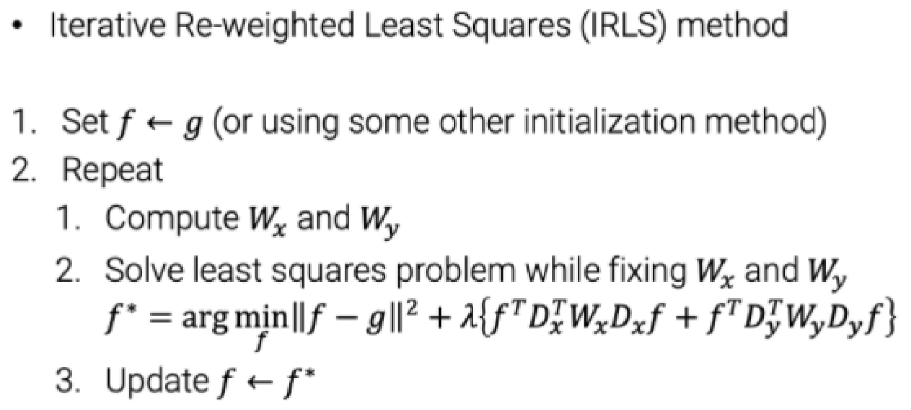
$$\begin{aligned}l^* &= \underset{l}{\arg\max}\ p(l|b,k)\\ &= \underset{l}{\arg\min}\left\{-\log p(b|l,k) - \log p(l)\right\}\\ &= \underset{l}{\arg\min}\left\{\frac{\|b-k*l\|^2}{2\sigma^2} + \frac{1}{b}\left(\|\partial x*l\|_1 + \|\partial y*l\|_1\right)\right\}\\ &= \underset{l}{\arg\min}\left\{\|b-k*l\|^2 + \underbrace{\frac{2\sigma^2}{b}}_{\alpha}\underbrace{\left(\|\partial x*l\|_1 + \|\partial y*l\|_1\right)}_{E_{TV}(l)}\right\}\end{aligned}$$

(total variation prior)

2.2. TV deconvolution

Non-blind deconvolution is an ill-posed problem. Due to the loss of information caused by the noise *n*, even if we know *k*, there can be an infinite number of solutions for *l*. To resolve such ill-posedness, most methods adopt a regularization term or a prior on the latent image *l*. One of the widely-sued prior is the total variation. The L1-norm of the image gradient is called total variation. Total variation is often used in digital image processing. Although total variation is similar to L2-norm-based smootheness term, clean image can be obtained by minimizing it.

In this assignment, I need to implement an algorithm to optimize the energy function. The energy function involves $\mathrm{E}_{TV}(l)$, which is non-quadratic. Thus, we need to implement a non-linear optimization algorithm. We can implement the iterative-reweighted-least-squares(IRLS) method. This algorithm is discribed as follow.



- Iterative Re-weighted Least Squares (IRLS) method

```
1. Set $f \leftarrow g$ (or using some other initialization method)
2. Repeat
    1. Compute $W_x$ and $W_y$
    2. Solve least squares problem while fixing $W_x$ and $W_y$
       $f^* = \arg\min_f \|f - g\|^2 + \lambda\{f^T D_x^T W_x D_x f + f^T D_y^T W_y D_y f\}$
    3. Update $f \leftarrow f^*$
```

At each iteration of the IRLS method, we need to solve a least-squares problem, which is formulated as follow where **b** and **l** are vector representations of *b* and *l*, respectively. **K** is a matrix representing the convolution operation with *k*.

$$\boldsymbol{l}^* = \underset{l}{\mathrm{argmin}}\{\|\boldsymbol{b} - \boldsymbol{K}\boldsymbol{l}\|^2 + \alpha(\boldsymbol{l}^{\boldsymbol{T}}\boldsymbol{D}_{\boldsymbol{x}}^{\boldsymbol{T}}\boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{D}_{\boldsymbol{x}}\boldsymbol{l} + \boldsymbol{l}^{\boldsymbol{T}}\boldsymbol{D}_{\boldsymbol{y}}^{\boldsymbol{T}}\boldsymbol{W}_{\boldsymbol{y}}\boldsymbol{D}_{\boldsymbol{y}}\boldsymbol{l})\}$$

If we organize this equation well, differentiate it, place it as 0, and organize it as follows.

$$(\boldsymbol{K}^{\boldsymbol{T}}\mathbf{K} + 2\alpha(\boldsymbol{D}_{\boldsymbol{x}}^{\boldsymbol{T}}\boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{D}_{\boldsymbol{x}} + \boldsymbol{D}_{\boldsymbol{y}}^{\boldsymbol{T}}\boldsymbol{W}_{\boldsymbol{y}}\boldsymbol{D}_{\boldsymbol{y}}))\mathbf{l} = \boldsymbol{K}^{\boldsymbol{T}}\mathbf{b}$$

This equation has the same form as $\mathbf{Ax} = \mathbf{b}$, which means that we can find a solution **l** using a linear solver. But, the size of the matrix is really huge, so it is hard to directly solve the linear system.

Instead, we can use an iterative solver such as the conjugate gradient method. We can change it to optimization problem which approximates *x*. In conjugate gradient method, by using the orthogonal vectors, we can perform optimization at most n times of gradient descent. This is the iterative process of conjugate gradient method for $\mathbf{Ax} = \mathbf{b}$.

1. Initialization random estimation $\vec{x}_0$
2. Initialization residual $\vec{r}_0 = \vec{b} - \mathrm{A}\vec{x}_0$
3. Initialization learning direction $\vec{p}_1 = \vec{r}_0$
4. Iteration

   4.1.Search the line $\vec{\alpha}_k = \frac{\vec{p}_k^T \vec{r}_{k-1}}{\vec{p}_k^T A \vec{p}_k}$

   4.2.Update estimation $\vec{x}_k = \vec{x}_{k-1} + \alpha_k \vec{p}_k$

   4.3.Update residual $\vec{r}_k = \vec{b} - A\vec{x}_{k-1}$

   4.4.Update direction $\vec{p}_k = \vec{r}_{k-1} - \frac{\vec{p}_{k-1}^T \vec{r}_{k-1}}{\vec{p}_{k-1}^T A \vec{p}_{k-1}} \vec{p}_k$

Following code is about conjugate gradient optimization according to the process above.

```
function x=conjgrad(x,b,maxIt,tol,Ax_func,func_param,visfunc)

    r = b - Ax_func(x,func_param);
    p = r;
    rsold = sum(r.*r);

    for iter=1:maxIt
        Ap = Ax_func(p,func_param);
        alpha = rsold/sum(p.*Ap);
        x=x+alpha*p;

        r=r-alpha*Ap;
        rsnew=sum(r.*r);
        if sqrt(rsnew)<tol
            break;
        end
        p=r+rsnew/rsold*p;
        rsold=rsnew;
    end
end
```

The following codes are non-blind deconvolution using a regulrization term based on an L2 norm on the image gradient. We can solve the $\mathbf{Ax} = \mathbf{b}$ by using following functions and perform the non-blind deconvolution. $\mathbf{Ax}$ is corresponding to $(\boldsymbol{K^T}\mathbf{K} + 2\alpha(\boldsymbol{D_x^T W_x D_x} + \boldsymbol{D_y^T W_y D_y}))\mathbf{l}$ and $\mathbf{b}$ is corresponding to $\boldsymbol{K^T}\mathbf{b}$. When we use the convolution operation, then we can obtatin this formulation. As Fourier transform in MATLAB is complex type, we need to take only real part type and psf2otf function is used to make Fourier transformed result of psf in frequency domain.

```
function ret = fftconv2(img, psf)
  ret = real(ifft2(fft2(img) .* psf2otf(psf, size(img))));
end
```

And we can calculate $\mathbf{Ax}$ in the following function. For the detail, in here we can calculate $(\boldsymbol{K^T}\mathbf{K} + 2\alpha(\boldsymbol{D_x^T W_x D_x} + \boldsymbol{D_y^T W_y D_y}))\mathbf{l}$ using fftconv2 function and matrix representation.

```
function y = Ax(x, p)
    x = reshape(x, p.img_size);
    x_f = fft2(x);
    y = fftconv2(fftconv2(x, p.psf), p.psf_conj);
    y = y + p.reg_strength*imfilter(p.weight_x.*imfilter(x, p.dxf, 'circular'), p.dxf, 'conv',
'circular');
    y = y + p.reg_strength*imfilter(p.weight_y.*imfilter(x, p.dyf, 'circular'), p.dyf, 'conv',
'circular');
    y = y(:);
end
```

We can save the psf which is represented by $\mathbf{K}$ in p.psf and compute $\boldsymbol{K^T}\mathbf{Kl}$ in frequency domain. 2α is saved in p.reg_strength and p.dxf is equal to [0, -1, 1] and p.dyf is equal to $[0\ -1\ 1]^T$. By computing $(\boldsymbol{D_x^T W_x D_x} + \boldsymbol{D_y^T W_y D_y})\mathbf{l}$, we finally get the result of $(\boldsymbol{K^T}\mathbf{K} + 2\alpha(\boldsymbol{D_x^T W_x D_x} + \boldsymbol{D_y^T W_y D_y}))\mathbf{l}$ as $\mathbf{Ax}$.

Through the calculation of the functions and formulas we have seen so far, deconvolution can be comprehensively performed in the following function.

```
function latent = deconv_L2(blurred, latent0, psf, reg_strength, weight_x, weight_y)
    img_size = size(blurred);

    dxf=[0 -1 1];
    dyf=[0 -1 1]';

    latent = latent0;

    psf_conj = rot90(psf, 2);
    b = fftconv2(blurred, psf_conj);
    b = b(:);
    x = latent(:);

    cg_param.psf = psf;
    cg_param.psf_conj = psf_conj;
    cg_param.img_size = img_size;
    cg_param.reg_strength = reg_strength;
    cg_param.weight_x = weight_x;
    cg_param.weight_y = weight_y;
    cg_param.dxf = dxf;
    cg_param.dyf = dyf;
    x = conjgrad(x, b, 25, 1e-4, @Ax, cg_param);

    latent = reshape(x, img_size);
end
```

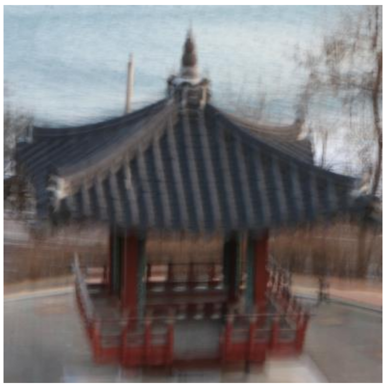
From now on, I will explain the main code that receives blurred image and kernel as inputs and proceeds with deconvolution. We prepare blurred image as jpg file and point spread function image as png file. First of all, we should rerange each pixel intensity from 0 to 1. Because psf is consist of one pixel and sets the sum of all the pixel values to 1. We should make some marices to solve the problem. One of them is diagonal matrix, which denoted by $\boldsymbol{W_x}$, $\boldsymbol{W_y}$. To make these matrices, we have to set the size of matrix. So, we will choose the smaller size between height and width of input image. And we need the smaller size of input matrix between width and height to compute the weighted matrix after.

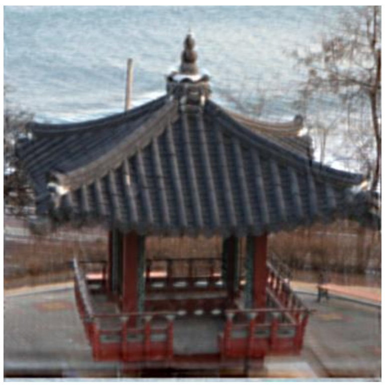
```
blurred = imread('summerhouse.jpg');
blurred = im2double(blurred);

psf = imread('summerhouse_out.jpg.psf.png');
psf = im2double(psf);
psf = rgb2gray(psf);
psf = psf / sum(psf(:));

weight_x = ones(size(blurred));
weight_y = ones(size(blurred));

latent = blurred;
input_size = size(blurred);
diagonal = min(input_size(1), input_size(2));
```

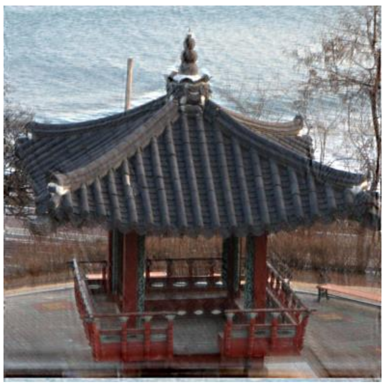
The following code is what we want to focus on. Because the IRLS optimization for non-blind deconvolution needs to compute convolution operations with a blur kernel several times. And we can use IRLS methods iteratively. In here, setting some parameters is needed. We set the 2α as $4\sigma^2/b$ and W_x, W_y as diagonal matrix. Assuming the distribution follows the normal distribution with 0 mean and σ standard deviation and gradient of l follows Laplace distribution(0, b). Thus we divide a gradient of l of standard deviation by square root of 2. All of these are used and we will be able to find optimal solution with conjgrad function.

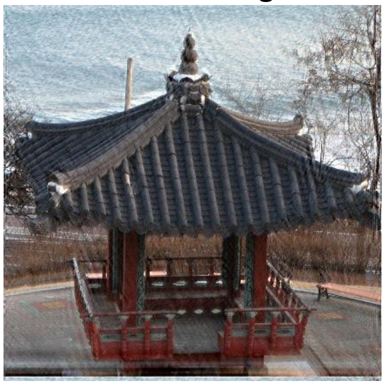
```
prev_alpha = 1;
for i=1:15
    sigma = std2(real(ifft2(fft2(latent) .* psf2otf(psf,input_size)) - blurred));
    [fx, fy] = gradient(latent);
    b = std2(fx+fy)/sqrt(2);

    cur_alpha = (2*sigma*sigma)/b;

    if abs(prev_alpha - cur_alpha) < 5e-4
        break;
    end

    latent = deconv_L2(blurred, latent, psf, cur_alpha, weight_x, weight_y);

    weight_x = zeros(size(blurred));
    weight_y = zeros(size(blurred));

    for idx=1:(diagonal-1)
        for ch=1:3
            weight_x(idx,idx,ch) = 1/max(1e-4,abs(latent(idx,idx+1,ch)-latent(idx,idx,ch)));
            weight_y(idx,idx,ch) = 1/max(1e-4,abs(latent(idx+1,idx,ch)-latent(idx,idx,ch)));
        end
    end

    prev_alpha = cur_alpha;
    imwrite(latent, strcat('deblurred_summerhouse',num2str(i),'.jpg'));
end
```

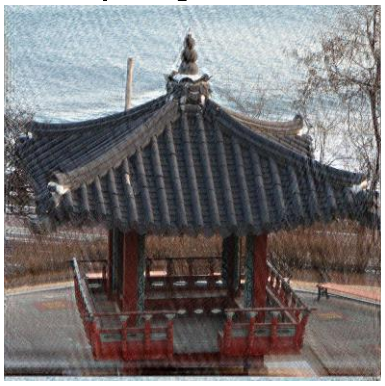
The following is the result of imae deblurring using optimization repeatedly.

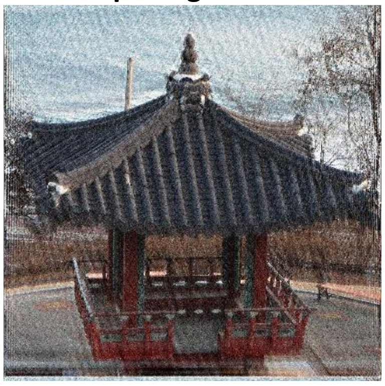

Blurred image


Sharp image with 1 iter


Sharp image with 3 iter


Sharp image with 5 iter


Sharp image with 10 iter


Sharp image with 15 iter

2.3. Wiener deconvolution

The wiener deconvolution is one of the classical approach to non-blind deconvolution. The wiener deconvolution is dervied by minimizing the expected error in the frequency domain. Specially, we want to find a deconvolution filter *H* in the frequency domain.

$$\min_{H} E[\|L - HB\|^2]$$

HB is the pixel-wise multiplication between *H* and *B* in the frequency domain. By solving the equation above, we can obtatin a deconvoltuion filter *H* such that *HB* is a deconvolution result with the least expected squared error.

After organizing several formulas, we can obtain a simplified version of the wiener deconvolution where *c* is a constant. By setting *c* = 0, the following equation reduces to the inverse filtering.

$$l = F^{-1}\left(\frac{|F(k)|^2}{|F(k)|^2 + \frac{1}{SNR(\omega)}} \cdot \frac{F(b)}{F(k)}\right) \approx F^{-1}\left(\frac{|F(k)|^2}{|F(k)|^2 + c} \cdot \frac{F(b)}{F(k)}\right)$$

From the equation above, we can see the constant *c* and originally this *c* comes from the $\frac{1}{SNR(\omega)}$.

$SNR(\omega)$ is the signal-to-noise ratio at each frequency ω, which is required to be estimated.

$$SNR(\omega) = \frac{signal\ variance\ at\ \omega}{noise\ variance\ at\ \omega}$$

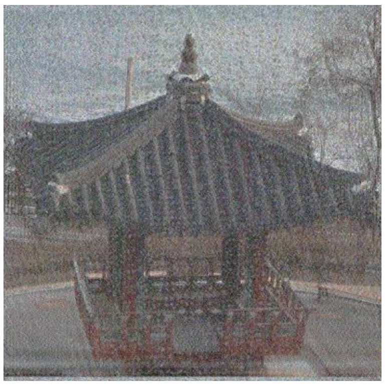
Natural images tend to have signal variance that scales as $\frac{1}{\omega^2}$. Noise variance tends to be a constant, i.e., $\sigma_N(\omega) = \text{constant for all } \omega$. This is called white noise. As a result, we can assume the following relation.

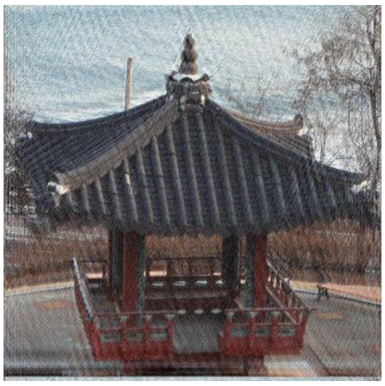
$$SNR(\omega) = \frac{1}{c\omega^2}$$

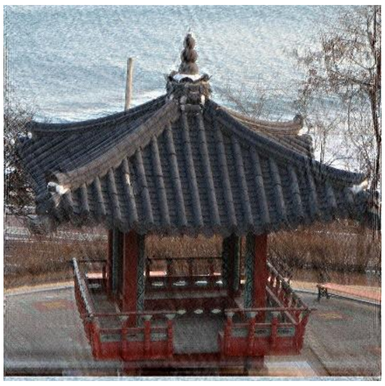
Therefore, *c* can be considered as the ratio of noise to signal, and the sensitivity of deconvolution can be set while adjusting it. The following code is about wiener deconvolution. In this code, to make sharp image, we can use the formulation above.

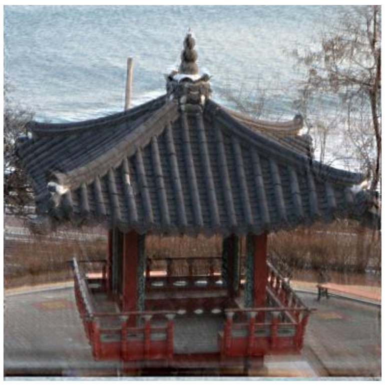
```
blurred = imread('summerhouse.jpg');
blurred = im2double(blurred);

psf = imread('summerhouse_out.jpg.psf.png');
psf = im2double(psf);
psf = rgb2gray(psf);
psf = psf / sum(psf(:));

F_b = fft2(blurred);
F_k = psf2otf(psf,size(blurred));

c = 0.1;

latent = real(ifft2(((abs(F_k) .* abs(F_k))./(abs(F_k) .* abs(F_k) + c)).*(F_b./F_k)));

imwrite(latent, 'summerhouse_deblurred.jpg');
```

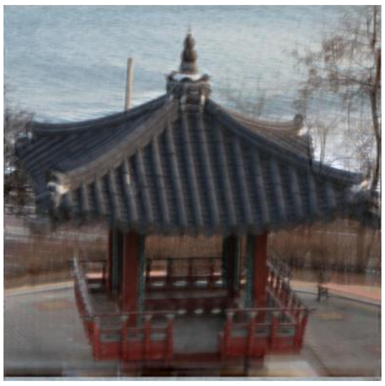
The following is the result of imae deblurring using wiener deconvolution according to *c* value.

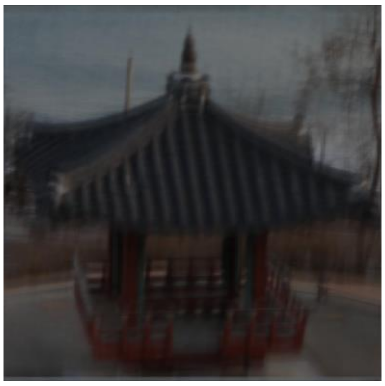

Sharp image wich c = 0


Sharp image wich c = 0.00001


Sharp image wich c = 0.001

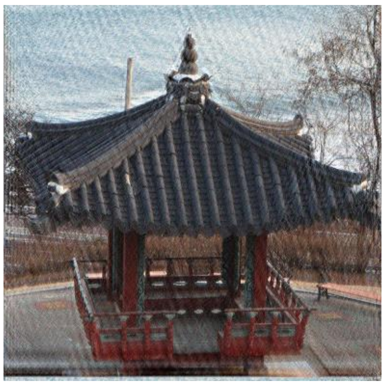

Sharp image wich c = 0.01


Sharp image wich c = 0.1


Sharp image wich c = 1

3 Discussion

When we apply TV deconvolution to blurred image and kernel, we can find out the detail of images is restored after repeating iterations. But there is some limitations I thought which deteriorate image quality when the number of iteration increases. Image deconvolution may tries to sharpen the blurred input image by making gradient sparser. This enhances variance of image gradients even including smooth region of image. Thus it ends up with performing with some noises. In a word, it can be excessive. So, I think it is okay that the IRLS method would stopped when the α is quite small and the result would be the best when the number of iterations about between 3 and 6. So, I added the stop conditon in the iteartion loop with proper limit.


Sharp image with 15 iter

Wiener deconvolution is an application of the wiener filter to the noise problems inherent in deconvolution. It works in the frequency domain, attempting to minimize the impact of deconvolved noise at frequencies which have a poor signal-to-noise ratio(SNR). An equation is configured so that it can be adjusted to a constant c while using SNR. When there is zero noise, which means that the wiener filter is simply the inverse of the system, as we might expect. However, as the noise at certain frequencies increases, the signal-to-noise ratio drops. This means that the wiener filter attenuates frequencies according to their filtered signal-to-noise ratio. If you set the *c* value to zero and look at the results, we can see that there is a lot of noise on the image. As we increase *c* value, noise decreases, but if it is more than 1, there is a problem that the image is not visible at all.

Sharp image with c = 10

Since it's a ratio, we can adjust the noise by setting an appropriate value between 0 and 1, and I think that setting c to 0.01 in the process of setting the value resulted in a clean result.
All of the implementations so far are methods used in image deblurring before deep learning. Classical approach relies on the blur model and the specified blur kernel, which creates several limitations. If the camera rotates, different blurs may occur in one scene. This is because the blur gets worse as the distance increases from the rotating point. This blur is called a spatially vibrating blur. And this blur cannot be explained by a single convolution kernel. In reality, most blurs are these spatially variable blurs. Therefore, classical approach has limitations in eliminating most blurs in reality.