Assignment 1: Image Filtering

Wei Huang

1 Problem

Continuous Fourier Transform is defined below:

$$\mathcal{F}(x(t)) = \hat{x}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} x(t)e^{-i\omega t}dt$$

When shifting in time, we have:

$$\mathcal{F}(x(t-\delta)) = e^{-i\delta\omega}\mathcal{F}(x(t))$$

Because:

$$\begin{aligned}\mathcal{F}(x(t-\delta)) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} x(t-\delta)e^{-i\omega t}dt \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} x(\nu)e^{-i\omega(\nu+\delta)}d\nu \\ &= e^{-i\delta\omega}\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} x(\nu)e^{-i\omega\nu}d\nu \\ &= e^{-i\delta\omega}\mathcal{F}(x(t))\end{aligned}$$

where $\nu = t - \delta$.

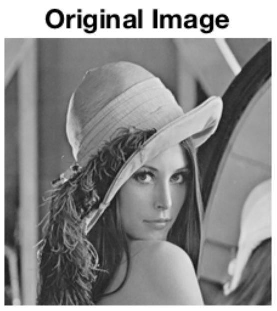
2 Problem

2.1 a

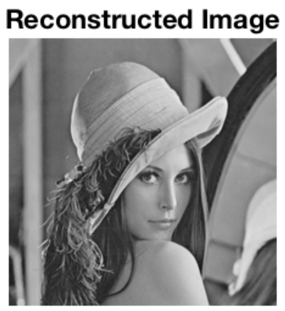
- Zero pad original image to size of next power of 2.
- apply FFT for each row
- Then, apply FFT for each column
- Apply inverse FFT for each column
- Apply inverse FFT for each row
- Take the real part

- Clipping into image with original size

2D Cooley-Tukey FFT is implemented in *FFTIM.m* file, and inverse 2D Cooley-Turkey is implemented in *invFFTIM.m* file. The main function is *fft_tranform.m* file. And the MSE is almost 0. Original image and reconstructed image is stated below.



(a) Origin



(b) Reconstructed

Figure 1: Comparison

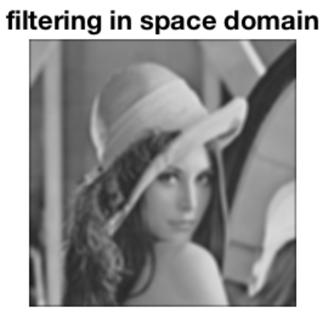
2.2 b

- Space Domain:
  - using *fspecial('gaussian',5,2)* to generate Gaussian filter of size 5×5 with standard deviation of 2.
  - using *conv2(I,gf,'same')* to get filtered image.
- Frequence Domain:
  - pading image with zeros to size of

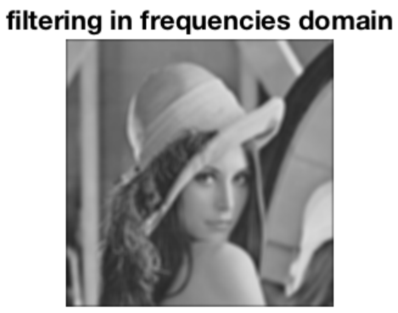
    $image\_num\_rows + filter\_num\_rows - 1 \times image\_num\_cols + filter\_num\_cols - 1$
  - pading filter with zeros to size of

    $image\_num\_rows + filter\_num\_rows - 1 \times image\_num\_cols + filter\_num\_cols - 1$
  - calling $ifft2(fft2(padding\_image). * fft2(padding\_filter))$
  - clipping the image

The main file is low_pass_filter.m. By running MATLAB code, MSE is 1.838e-32 and two filtered images are stated below.



(a) Space



(b) Frequence

Figure 2: Comparison

2.3 c

For image:

$$(220 + 5 - 1) \times (220 + 5 - 1) - 220 \times 220 = 1776$$

For filter:

$$(220 + 5 - 1) \times (220 + 5 - 1) - 5 \times 5 = 50151$$

2.4 d

The result will not be consistent between filtering in space and frequence domain. By running MATLAB code, MSE is 0.0069972 which is much larger than MSE in (c), and tow images are stated below.

In order to avoid space domain aliasing due to circular convolution and get the consistent result, we should padding filter and image to size of proper number before calling 2D FFT.

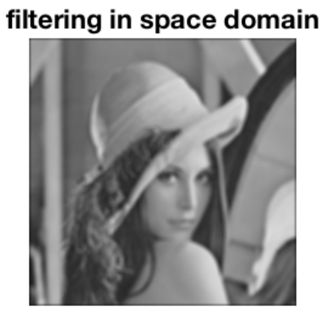
3 Problem

3.1 a

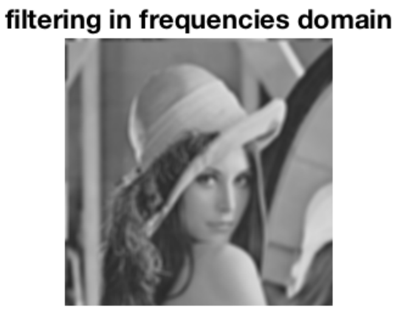
BF could not only reduce noise but also preserve edge by incorporating range filter which average pixels with coefficients whose magnitude is inversely proportional to the distance present in the color-map.

3.2 b

Brute force implementation of BF needs two loops, so the time complexity is $\mathcal{O}(n^2)$, where n is the number of pixels. Durand and Dorsey proposed a faster algorithm by

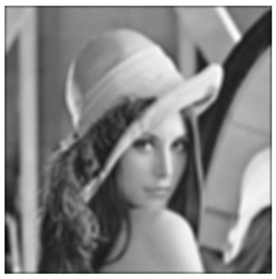


(a) Space

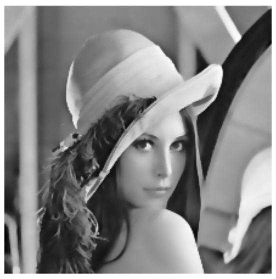


(b) Frequence

Figure 3: Comparison



(a) Classical



(b) BF

Figure 4: Comparison

replacing $f(t)$with a sequence of fixed pixels value $\{i_0, \cdots, i_j, \cdots, i_m\}$, and calling FFT. Firstly, Downsampling f to $f^{'}$, and compute the product:

$$h^j(p) = s(f^{'}(p) - i_j)f(p)$$

Secondly, Calling FFT to compute convolution and normalizing

$$g^j(t) = k^{-1}\sum_p c(t-p)h^j(p)$$

Thirdly, upsampling $g^j(t)$ to $g_*^j(t)$, and now we have layers $\{L_0, \cdots, L_j, \cdots, L_m\}$ each of which contains the exact results of bilateral filter for pixels with intensity equal i_j. Finally, finding two closest value i_{j1}, i_{j2}, and then Interpolating for each pixel p with intensity $f(p)$

$$BF[f](p) = \frac{f(p) - i_{j1}}{i_{j2} - i_{j1}}L_{j2} + \frac{i_{j2} - f(p)}{i_{j2} - i_{j1}}L_{j1}$$

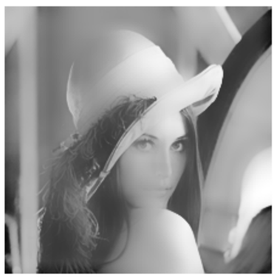
3.3 c

MATLAB code of Bilateral Filter is in *bf.m* file, and main code is in *bf_main.m*, result is stated in Figure 4.we should input four parameters for *bf* function. In this case I set windows size as $5 \times 5, \sigma_d = 2, \sigma_r = 25$

Remark: In order to reduce time complexity, for each pixel, I do not compute weight over all pixel but over small size of window centered at it. For example, 5×5window.

3.4 d

In this problem, I set window size as 21×21, and $\sigma_d = 1000, \sigma_r = 50, \sigma_d = 2, \sigma_r = 50$ respectively. Two filtered images are stated below: we could see when we set σ_d as a


(a) $\sigma_d = 1000, \sigma_r = 50$

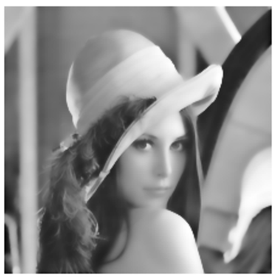

(b) $\sigma_d = 2, \sigma_r = 50$

Figure 5: Comparison

large number, the filtered image will become more smooth and loss some edge. This is because increasing smooths larger features. Although filtered image with large σ_d is more smooth, it still preserve most edges. This is because increasing the spatial σ_d will not blur an edge as long as the range σ_r is small enough.

4 Problem

A. Buades, B. Coll and J. Morel proposed Non-local-means algorithm in which the distance between two pixels not only depend on value themselves but also theis surrounding. The algorithm is summarized below:

$$NLM(f)[i] = \sum_j w(i,j)f(j)$$

where

$$w(i,j) = \frac{\exp(-\frac{d(i,j)}{2h^2})}{\sum_j \exp(-\frac{d(i,j)}{2h^2})},$$

and

$$d(i,j) = \int_t (f(i+t) - f(j+t))^2 G_a(t)dt$$

G_a is Gaussian kernel function.
Remark: In the code implementation, we avoid G_a.
MATLAB code is in *nlm.m.* The result is stated below:

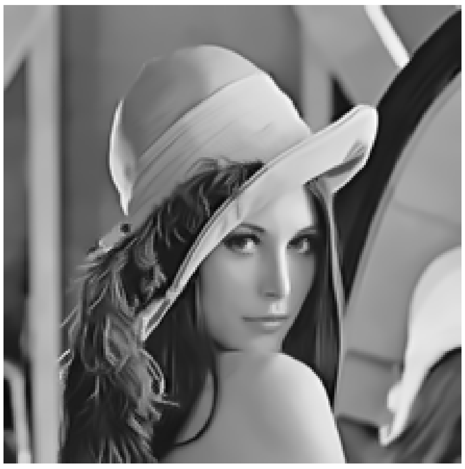


Figure 6: NLM