

Problem 1 – Part A: Image Demosaicing

I. Abstract and Motivation

In the past we used film to get analog images. The color film contains sensitizers sensitive which are sensitive to red, green and blue respectively. These sensitizers are applied to the film in several layers. When light hits the film, light can penetrate the layers, and the sensitizers react separately to get a color photograph. Digital imaging technology now plays a pivotal role in our lives. When we use digital photosensitive chips, the method of obtaining color images, that is, the method of getting RGB components is different. Firstly, the digital chip has a fixed resolution, and each pixel has a light collecting element that measures the number of photons that arrive at that location within a fixed time. However, the element cannot distinguish between different wavelengths of light, so it will include all of the light that is incident on it into the measurement result. In order to obtain an R or B or G monochrome image, a single wavelength filter is placed in front of the element to filter out unwanted light. But in this way, it is impossible to stack photosensitive elements together to get RGB value in a same point, because the light cannot penetrate photosensitive elements. So current photosensitive chips, each pixel can only capture the amount of light of one of the RGB colors. *Bayer Array* is commonly used to arrange which pixel is responsible for capturing which color of light. Bayer Array contains 2 green, 1 red, and 1 blue filters in each 2*2 unit. The amount of green is twice that of other colors because the human eye is more sensitive to green, so use more green pixels to provide brightness information. After obtaining the original image data that each pixel contains only one color data, we need to estimate the values of the two missing color values. This article implements and discusses two of the most basic image demosaicing methods, which are *Bilinear Demosaicing* and *Malvar-He-Cutler (MHC) Demosaicing*.

II. Approach and Procedures

i. Convolution

Convolution is very common in image processing. Here I will first introduce how to implement image convolution.

The convolution operation of the image requires two parameters, an original image and a convolution kernel.

$$Y(x, y) = \sum_{i=-a}^a \sum_{j=-b}^b K(x, y) I(x - j, y - i) \quad (1)$$

Where Y is the result 2-D image, I is the original 2-D image, K is the $2a + 1$ by $2b + 1$ kernel.

We can design different convolution kernels and perform convolution on the input image to get the image with the specified effect. Equation (1) requires that the width and height of the convolution kernel should be odd.

In implementation, I implement the convolution process as a function in the program, call this function, pass in the input image and convolution kernel, you can get the output image. The convolution kernel can be hard coded in the program or automatically generated at runtime. It should be noted that in order to obtain an output image with a reasonable range of values, the convolution kernel here should be normalized, that is, the cumulative sum of each element in the convolution kernel is 1, that is

$$\sum_{i=-a}^a \sum_{j=-b}^b K(x, y) = 1 \quad (2)$$

ii. Border Extension

When performing a convolution operation, we are unable to perform a convolution operation on the pixels at the edges of the input image. Since, according to Equation (1), when convolution is performed on these pixels, some pixel indices exceeds the range of the input image. This also causes that after convolution operation is performed, and the output image is smaller than the original image.

$$\begin{aligned} H' &= H - 2a \\ W' &= W - 2b \end{aligned} \quad (3)$$

Where H and W are the height and width of input image, H' and W' are the height and width of output image.

After doing this, the information at the edge of the input image is lost, which is not what we want. In order to satisfy the condition of performing convolution on edge pixels, we need to extend each side of the input image by filling in some extra pixels. This procedure is called *Boundary Extension*. What kind of pixel is filled is a problem, I chose a mirror-reflecting extension method, so that the filled pixels are more correlated with original pixels than simply copying the outermost pixels. Specifically, selecting boundary row or column of pixels as the center of symmetry, or selecting the edge of the image as the center of symmetry is two implementation manners. Considering the specific pattern of the Bayer Array, we choose the boundary row or column of pixels as the symmetry center of the mirror-reflecting.

To hide this implementation detail for high-level programming, I implemented the process depending by whether coordinate indices are beyond the boundaries of the image when addressing in the data structure. If not exceeded, it is addressed in the normal way, and if it is exceeded, the reflected address is calculated according to Equation (4).

$$i' = f(i) = f(x) = \begin{cases} -i, & i < 0 \\ 2(Max - 1) - i, & i \geq Max \\ i, & else \end{cases} \quad (4)$$

Where i and i' are the input and reflected indices start from 0, Max is the height or width of the input image.

iii. Bilinear Demosaicing

Bilinear demosaicing uses only the color of the pixels around the specified pixel to estimate missing colors of the current pixel. The red pixels in the original image using

$$\begin{aligned} R_{x,y}^R &= R_{x,y}^R \\ G_{x,y}^R &= \frac{1}{4}(G_{x+1,y}^G + G_{x-1,y}^G + G_{x,y+1}^G + G_{x,y-1}^G) \\ B_{x,y}^R &= \frac{1}{4}(B_{x-1,y-1}^B + B_{x+1,y-1}^B + B_{x-1,y+1}^B + B_{x+1,y+1}^B) \end{aligned} \quad (5)$$

Similarly, for the blue pixels we have

$$\begin{aligned} R_{x,y}^B &= \frac{1}{4}(R_{x-1,y-1}^R + R_{x+1,y-1}^R + R_{x-1,y+1}^R + R_{x+1,y+1}^R) \\ G_{x,y}^B &= \frac{1}{4}(G_{x+1,y}^G + G_{x-1,y}^G + G_{x,y+1}^G + G_{x,y-1}^G) \\ B_{x,y}^B &= B_{x,y}^B \end{aligned} \quad (6)$$

Because the arrangement of green pixels is different from other colors, it has only 2 neighbour pixels for each of red and blue, so the following equations are used.

$$\begin{aligned} R_{x,y}^G &= \frac{1}{2}(R_{x,y+1}^R + R_{x,y-1}^R) \\ G_{x,y}^G &= G_{x,y}^G \\ B_{x,y}^G &= \frac{1}{2}(B_{x+1,y}^B + B_{x-1,y}^B) \end{aligned} \quad (7)$$

The above Equation (5) (6) (7) can be rewritten into the following five 3*3 convolution kernels.

0 0 0 0 1 0 0 0 0	0.25 0 0.25 0 0 0 0.25 0 0.25	0 0.25 0 0.25 0 0.25 0 0.25 0	0 0.5 0 0 0 0 0 0.5 0	0 0 0 0.5 0 0.5 0 0 0
a. for R^R , G^G and B^B	b. for R^B and B^R	c. for G^R and G^B	d. for R^G	e. for B^G

For each pixel of the input image, its own color is determined according to its row and column coordinate indices, and then the corresponding convolution kernel is used for each of the RGB channels according to the above rules to obtain the final color of the pixel.

iv. Malvar-He-Cutler(MHC) Demosaicing

MHC demosaicing was proposed by Malvaretal.[1], which is superior to Bilinear demosaicing. Considering that the luminance values of different channels are related, the value of the other channel neighbours can be used to adjust the mean of the neighbour colors. MHC demosaicing is an algorithm that takes this correlation into account based on bilinear demosaicing. Each color is calculated as follows

$$\begin{aligned} R_{x,y}^R &= \overline{R_{x,y}^R} \\ G_{x,y}^R &= \overline{G_{x,y}^R} + \alpha \Delta_{x,y}^R \\ B_{x,y}^R &= \overline{B_{x,y}^R} + \gamma \Delta_{x,y}^R \end{aligned} \quad (8)$$

$$\text{where } \Delta_{x,y}^R = R_{x,y}^R - \frac{1}{4}(R_{x+2,y}^R + R_{x-2,y}^R + R_{x,y+2}^R + R_{x,y-2}^R)$$

$$\begin{aligned} R_{x,y}^B &= \overline{R_{x,y}^B} + \gamma \Delta_{x,y}^B \\ G_{x,y}^B &= \overline{G_{x,y}^B} + \alpha \Delta_{x,y}^B \\ B_{x,y}^B &= \overline{B_{x,y}^B} \end{aligned} \quad (9)$$

$$\text{where } \Delta_{x,y}^B = B_{x,y}^B - \frac{1}{4}(B_{x+2,y}^B + B_{x-2,y}^B + B_{x,y+2}^B + B_{x,y-2}^B)$$

$$\begin{aligned} R_{x,y}^G &= \overline{R_{x,y}^G} + \beta \Delta_{x,y}^G \\ G_{x,y}^G &= \overline{G_{x,y}^G} \\ B_{x,y}^G &= \overline{B_{x,y}^G} + \beta \Delta_{x,y}^G \end{aligned} \quad (10)$$

where $\Delta_{x,y}^G$ is a "discrete 9 point Laplacian of green channel"

$\overline{G_{x,y}^R}, \overline{B_{x,y}^R}, \overline{R_{x,y}^B}, \overline{G_{x,y}^B}, \overline{R_{x,y}^G}, \overline{B_{x,y}^G}$ are the results getting by using bilinear demosaicing. α , β and γ are parameters, here we take

$$\alpha = \frac{1}{2}, \beta = \frac{5}{8}, \gamma = \frac{3}{4}$$

Therefore, five normalized 5*5 convolution kernels can be written in a similar way.

$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & -\frac{1}{8} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 & 0 \\ -\frac{1}{8} & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & -\frac{1}{8} \\ 0 & 0 & \frac{1}{4} & 0 & 0 \\ 0 & 0 & -\frac{1}{8} & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & -\frac{3}{16} & 0 & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 \\ -\frac{3}{16} & 0 & \frac{3}{4} & 0 & -\frac{3}{16} \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 \\ 0 & 0 & -\frac{3}{16} & 0 & 0 \end{matrix}$
a. for R^R , G^G and B^B	b. for G^B and G^R	c. for R^B and B^R
$\begin{matrix} 0 & 0 & \frac{1}{16} & 0 & 0 \\ 0 & -\frac{1}{8} & 0 & -\frac{1}{8} & 0 \\ -\frac{1}{8} & \frac{1}{2} & \frac{5}{8} & \frac{1}{2} & -\frac{1}{8} \\ 0 & -\frac{1}{8} & 0 & -\frac{1}{8} & 0 \\ 0 & 0 & \frac{1}{16} & 0 & 0 \end{matrix}$		$\begin{matrix} 0 & 0 & -\frac{1}{8} & 0 & 0 \\ 0 & -\frac{1}{8} & \frac{1}{2} & -\frac{1}{8} & 0 \\ \frac{1}{16} & 0 & \frac{5}{8} & 0 & \frac{1}{16} \\ 0 & -\frac{1}{8} & \frac{1}{2} & -\frac{1}{8} & 0 \\ 0 & 0 & -\frac{1}{8} & 0 & 0 \end{matrix}$
d. for R^G (R row, B column) and B^G (B row, R column)		e. for R^G (B row, R column) and B^G (R row, B column)

By performing convolution in a similar way to bilinear demosaicing, we can determine all missing colors.

III. Experimental Results

Shown below are the results for image demosaicing.



Fig1. Raw input image with Bayer RGB pattern



Fig2. Reference color image



Fig3a. Output image of bilinear demosaicing (Color)



Fig4a. Output image of MHC demosaicing (Color)



Fig3b. Output image of bilinear demosaicing (Red channel)



Fig4b. Output image of MHC demosaicing (Red channel)



Fig3c. Output image of bilinear demosaicing
(Green channel)



Fig4c. Output image of MHC demosaicing
(Green channel)



Fig3d. Output image of bilinear demosaicing
(Blue channel)



Fig4d. Output image of MHC demosaicing
(Blue channel)

IV. Discussion

- i. [Question (a)-(2)] Do you observe any artifacts? If yes, explain the cause of the artifacts and provide your ideas to improve the demosaicing performance.

Yes, I can see two kinds of artifacts clearly.

The first kind is checkerboard texture ("zipper effect"). I think the reason for this effect is, estimated pixel values and real color pixel values are different, and the differences are large enough that we

can notice the discontinuous pixels. This effect is stronger in the red and blue channels, which are the channels have less sensing pixels. Especially, straight borders will have sawtooth edge in the result image, because the estimated pixel values are average of true color pixel values from both sides of the boarder, which makes the difference between estimated value and real value larger. The second kind is "false color". I can see unnatural color at light and dark boundaries. And the unnatural color are yellow, cyan, magenta, which are the complementary colors of red, green and blue. Most of them are yellow and cyan. That means at those pixels, some kind of RGB channel miss the value (the value is very low). If we get yellow false color, that means blue is missing, and if we got cyan, that means red is missing. Red and blue are just channels have fewer sensing pixels. As a result, on light and dark boundaries, when pixels estimate the blue or red value, they can only average pixel values on dark side of the boundary which make the average value lower. And all the yellow false color occurs on horizontal boundaries and cyan false color occurs on vertical boundaries, that is because of the arrangement of Bayer array. Actually, there are red and blue false color which are not that obvious than yellow and cyan. The reason of red and blue false color is opposite to yellow and cyan false color.

ii. [Question (b)-(2)] Compare the MHC and the bilinear demosaicing results and explain the performance differences between these two algorithms in your own words.

Results from MHC is better than that from bilinear demosaicing.

At first, the color Fig4a is more vivid than Fig3a. Colors in Fig4a seems real, while colors in Fig3a is dominated by green which make people feel cold.

Then, results from MHC do not have checkerboard pattern and false color artifacts. While some small details with only one pixel width or height blurred.

Overall, MHC demosaicing does better than bilinear demosaicing.

iii. Color filter array

I wondered if there is another technique to replace or improve Bayer filter. When we are taking photos at night, we need to adjust the camera to let more light come into the camera. If a Bayer filter is used, some light is wasted. So, I searched some information.

There are some other filter arrays, such as RGBW, CYGM. Arrays can also be larger than 2×2 . Some of them do reduce the waist of light or better theoretical effects, but the most basic way of working has not changed. I think maybe we can separate the light using some kind of lens and

Zongjian Li
6503-3789-43
zongjian@usc.edu

EE 569 Homework #1
January 22nd, 2019

remove the color array filters.

Problem 1 – Part B: Histogram Manipulation

I. Abstract and Motivation

When we take pictures, we adjust the size of aperture, shutter speed, and light sensibility ordinance (ISO) to adjust the overall brightness of the exposure image, so that the photos are not too bright or too dark to make it difficult to see the details. These are somethings we can adjust when creating a digital image. Of course, after we have got an digital image, we can still make some adjustments to highlight the details of the over-bright or over-dark parts of an image. When the exposure of an image is too bright or too dark, the difference between pixel values is small, which is the reason why we can not see details clearly. We can use Histogram Manipulation to evenly distribute the over-concentrated color values over the entire numerical range, so that the difference in pixel values becomes larger and we can see the details. Here we introduce two methods. In order to obtain a simple and intuitive effect, in this article I use grayscale images here, and each pixel value represents only the brightness of the pixel.

II. Approach and Procedures

i. Histogram

Since this article only discusses grayscale images, there is only one channel per pixel. The input image uses 8 bits to represent the gray level of one pixel, so the gray level is taken as all integers from 0 to 255. A histogram is the number of times each gray value appears in the entire image. The x-axis is the value of all possible gray levels, and the y-axis is the number of occurrences. The process of getting histograms is very simple. We only need to traverse all the pixels and add 1 to the gray value counter of the pixel. Finally, we can get the number of times each gray value appears in the image. We can also replace the y-axis with the percentage of occurrences in the entire image, in which case the data of the histogram needs to be normalized by dividing each item in the original histogram by the total number of pixels. In this article, unnormalized histograms are used according to the requirements.

ii. Histogram Manipulation

When an image is too bright or too dark overall, the values in its histogram will be concentrated

in the high-value or low-value regions. At this time, the details of the over-bright or over-dark areas are difficult to see because the difference of gray levels are small. So we can adjust its gray level and increase the contrast at too dark or too bright regions to highlight the details that are difficult to see. On the histogram, the values that were originally gathered in a small range are expanded to a larger range. The brightness of a suitable image should be moderate, that is, the value of its histogram should be average over the entire range of the x-axis, and a large number of values will not gathered in a small range.

iii. Method A: Transfer-function-based

In this method we need to find a function $f(x)$ that maps each gray value of the input image to a new gray value, and the resulting image is moderately bright.

$$y = f(x) \text{ where } x, y \in \{0, 1, \dots, 255\} \quad (11)$$

We need to calculate the *Cumulative Distribution Function* (CDF) first, as follows:

$$C(x) = \begin{cases} H'(x), & x = 0 \\ C(x-1) + H'(x), & x \in \{1, 2, \dots, 255\} \end{cases} \quad (12)$$

Where $H'(x) = H(x)/S$ is the normalized histogram, S is the number of pixels of the image. Let

$$y = f(x) = x \cdot C(x) \quad (13)$$

This is the transfer function we used in this method. It always gets a fixed output gray value for each given gray value, so we can calculate all the results for all gray values to get a mapping table. Then look up the mapping table directly when processing each pixel, which can reduce the amount of calculation.

iv. Method B: Cumulative-probability-based

In order to make the histogram of the output image uniformly distributed, we can directly calculate how many pixels each gray value in the output image has, that is, a quota of each gray value in the output image.

$$Q = \begin{cases} \frac{S}{256}, & S \bmod 256 = 0 \\ \frac{S}{256} + 1, & S \bmod 256 \neq 0 \end{cases} \quad (14)$$

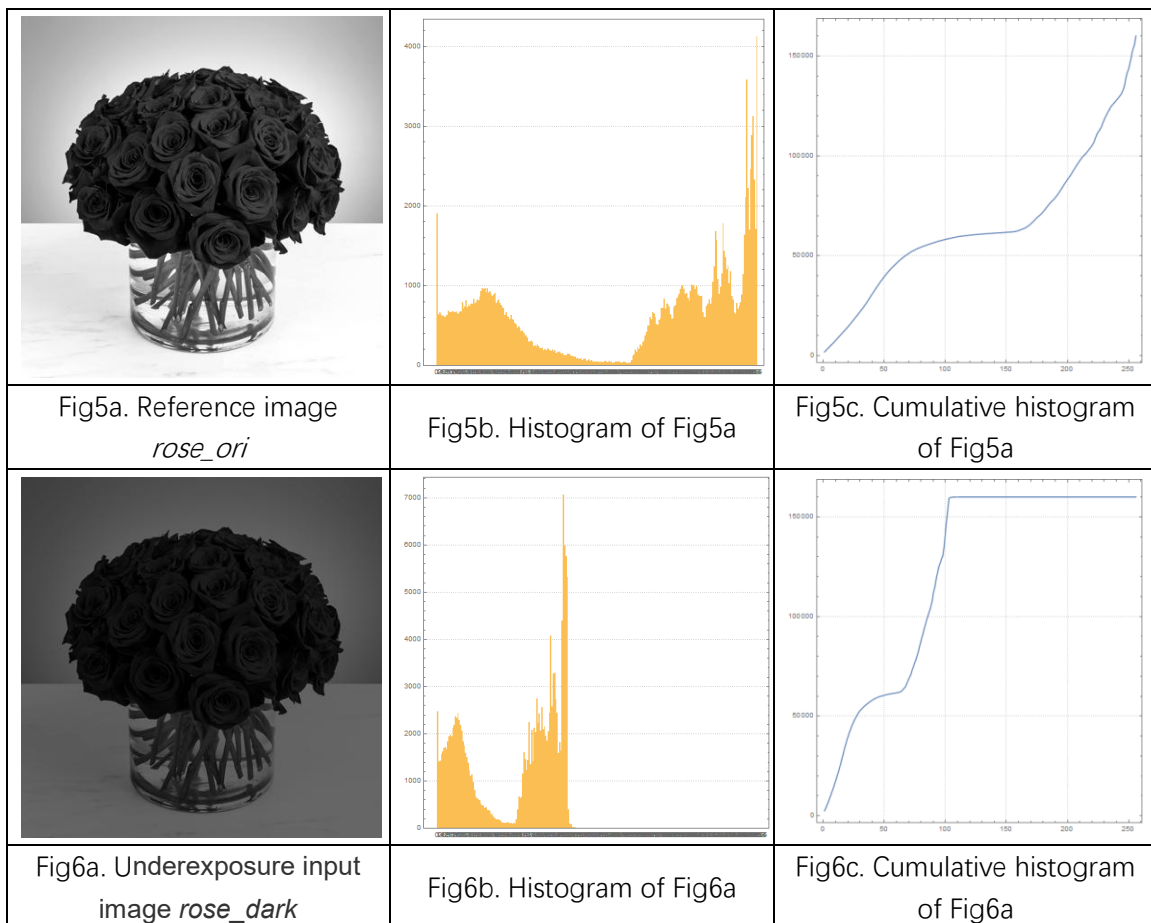
Then we can assign the gray value of the input image to the gray value of the output image with the restriction of the quota. For example, we can traverse from the lowest gray value 0 to highest gray value 255. When the gray value of the pixel is equal to the current gray level, the gray level of its output pixel is allocated as the lowest output gray value that does not reach the quota.

With this method, different pixels of the same gray value in the input image may have different

gray values in the output image, and different pixels of different gray values in the input image may have the same gray value in the output image. There may be many specific mapping methods, but we don't care about these details in this method, as long as we get a mapping that satisfies the above conditions, that is, I can be random in the order of traversing the input image pixels. It can be randomly chosen or it can be traversed from the smallest coordinate index to the largest coordinate index.

III. Experimental Results

Fig5 is the reference image. Fig6, Fig7 and Fig8 are three input images with their histogram.



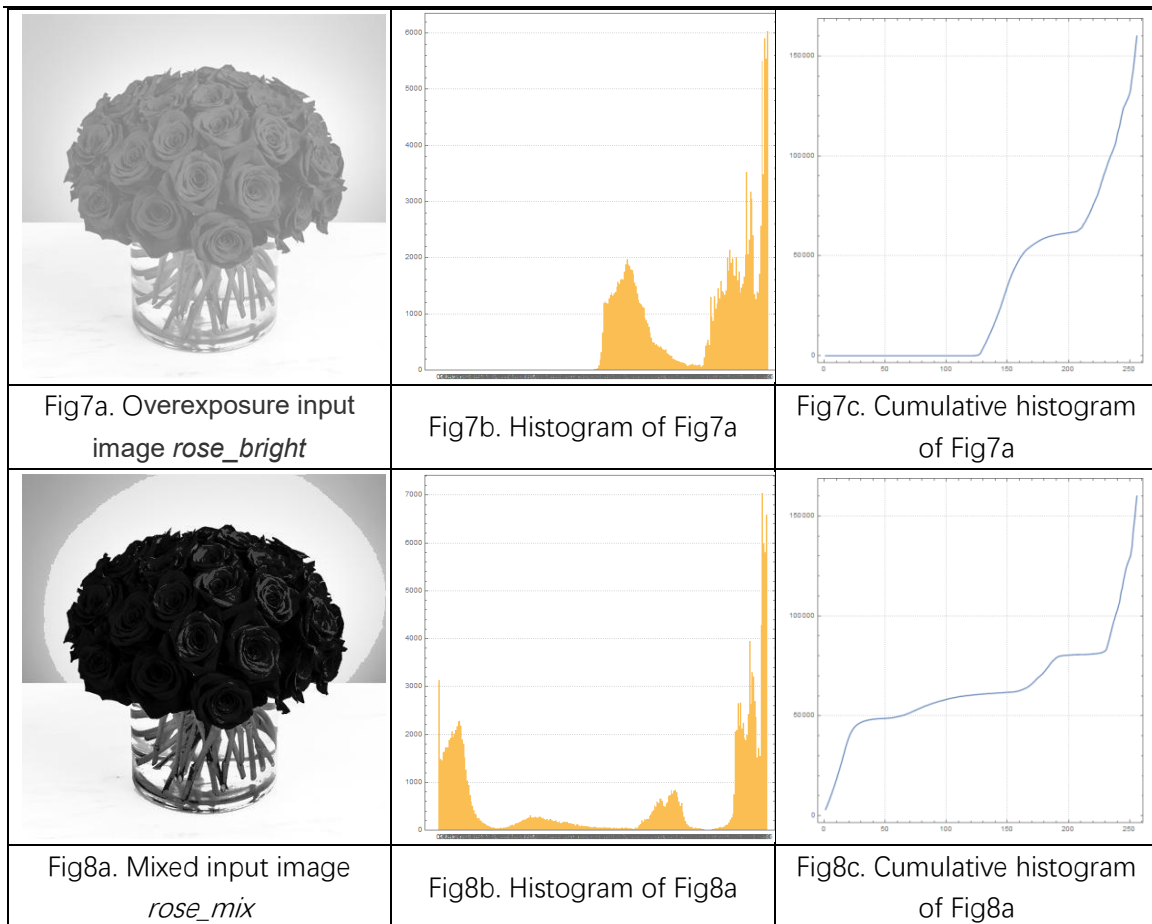


Fig9, Fig10 and Fig11 are the output images with their histogram using method A.


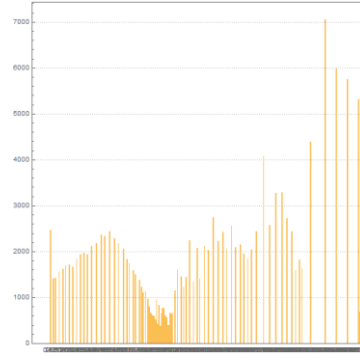
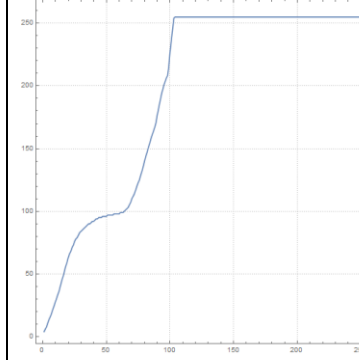

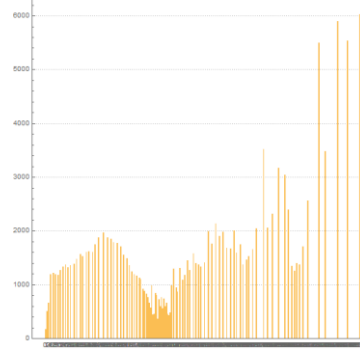
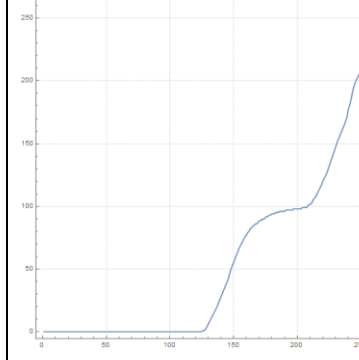

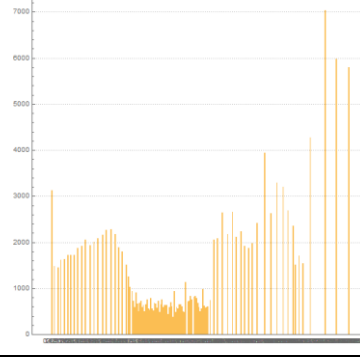
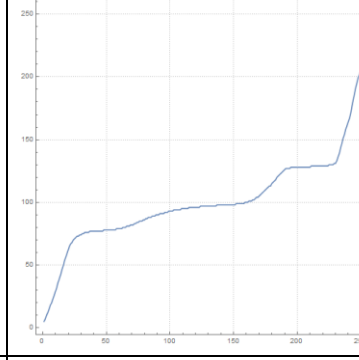

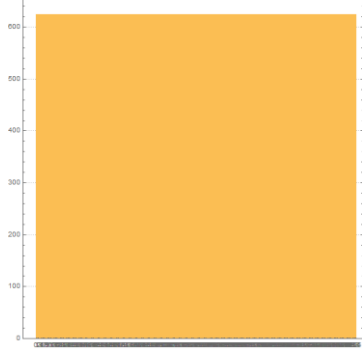
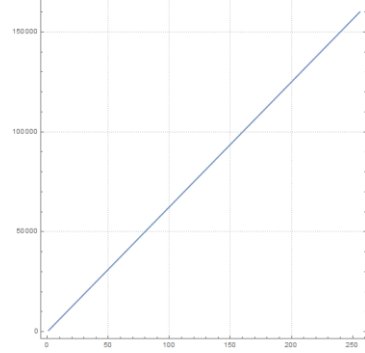

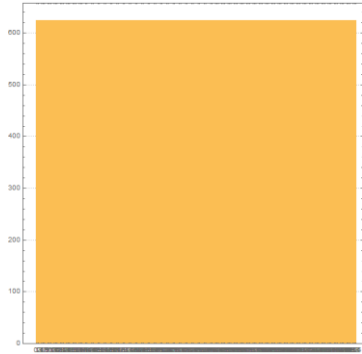
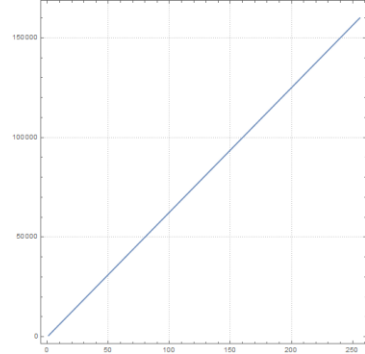

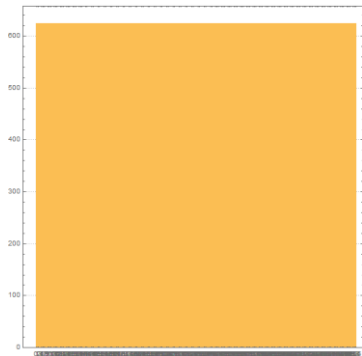
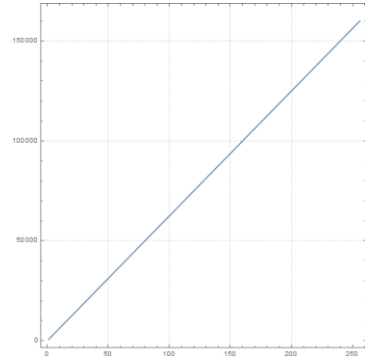
		
Fig9a. Output of method A on <i>rose_dark</i>	Fig9b. Histogram of Fig9a	Fig9c. Transfer function used to get Fig9a
		
Fig10a. Output of method A on <i>rose_bright</i>	Fig10b. Histogram of Fig10a	Fig10c. Transfer function used to get Fig10a
		
Fig11a. Output of method A on <i>rose_mix</i>	Fig11b. Histogram of Fig11a	Fig11c. Transfer function used to get Fig11a

Fig12, Fig13 and Fig14 are the output images with their histogram using method B.

		
<p>Fig12a. Output of method B on <i>rose_dark</i></p>	<p>Fig12b. Histogram of Fig12a</p>	<p>Fig12c. Cumulative histogram of Fig12a</p>
		
<p>Fig13a. Output of method B on <i>rose_bright</i></p>	<p>Fig13b. Histogram of Fig13a</p>	<p>Fig13c. Cumulative histogram of Fig13a</p>
		
<p>Fig14a. Output of method B on <i>rose_mix</i></p>	<p>Fig14b. Histogram of Fig14a</p>	<p>Fig14c. Cumulative histogram of Fig14a</p>

Note: Data of histograms is generated by my program and printed into standard output. Then I copy the text format data into a bar-chart drawing software and get pictures of histogram.

IV. Discussion

- i. [Question (c)-(4)] Discuss your observations on these two enhancement results. Do you have any idea to improve the current result?

For me, it is hard to tell the difference of two enhancement results at the first glance, they both work well.

After careful observation, I found, in the right bottom part of the input images which is a large area with little gradient change, method B can give us a smoother surface than method A. I can see clear boundary of gray level change in Fig9a & Fig10a, however, Fig12a & Fig13a also has this effect but weaker. From the histogram of these results, I can see that the gray levels got by method A are individual values, there are large gaps between values. That is the reason why the gradient effect is very abrupt. While method B do not have these gaps, so the results are smoother. This area in input images has little gradient change, and the number of quantized bits is only 8-bits so it looks not pleasant in this area after we stretched the range of gray level values in both methods.

The main part in this picture is roses in the middle, this area has complex pattern and the gray values changes within a larger range. I found that roses are brighter in results from method B than method A. I think the reason is the part of the rose in the picture is basically dark, while in method B, number of pixels of each gray level is bounded, so to arrange dark gray values in input images into a new value in outputs, values in output got higher and higher. In method A, there is no this kind of procedure, values are just mapped to fixed values. As a result, in this area, method B results in higher contrast, and the image is more vivid.

Two methods both work well in the background area. I can see smooth change in this area, except at the junction with the roses after zooming in. There are water wave patterns. I think this is also caused by insufficient quantization bit number of the input images.

To improve the results, I think smoothing the image with some kind of low pass filter can remove the abrupt color gradient boundaries and water wave patterns. Above this, if using a larger number of quantization bits to store the images, the result should look better.

- ii. [Question (c)-(5)] Apply your implemented Method A and Method B to *rose_mix* in Figure 4 and show the result. Can you get similar result as in previous part? If not, how to modify your implementation? Please justify your answer with discussion.

Yes, I got results similar to other ones. At least from my eyes, I cannot say the result is a failure. However, the gradient change in the background do not seems good, and the result from method A is worse. There is a halo at the back ground. I think it is because in the input image, the gradient of those pixels changes rapidly.

iii. Bits of quantization

Histogram is useful when we use image editing software to edit pictures. When we need to change the tone or brightness, we just pull the triggers. Photographer tutorials always telling me, that I should take and store my pictures in raw formats, because these data formats has larger bits of quantization. When we editing pictures using raw formats, we can get smoother results. I was confused, is 8-bits enough for today? Why those common image formats only using 8-bits for each channel and do not provide any flexibility? I think image processing devices with higher quantization bits will become popular, but there should be some innovations for computer infrastructures. However, digital image processing technology can still work if we have better devices. While some techniques may have lower priority, such as image denoising (Problem 2), if we have better devices.

Problem 2: Image Denoising

I. Abstract and Motivation

During digital image processing, it may be necessary to process images with additional noise. In the image acquisition stage, the low tolerance of the imaging chip and poor working environment may result in noise in images. In the image transmission stage, if the quality of transmission channel is poor or there is strong interference, the digital image signal will also get additional noise. The data that has been stored in certain media may also get noise due to aging of the storage medium or interference. For the noise generated during the acquisition phase, we can use a better imaging device, and we can use the error correction code to eliminate noise during transmission and storage stages. However, replacing a better imaging device can be expensive, and for the transmission and storage of analog image signals, the use of error correcting codes is also difficult. Therefore, denoising images to obtain clear images is a basic technique in the field of image processing. Common types of noise are Gaussian noise, Salt-and-peper noise, Shot noise, Uniform noise. In this article, we are considering denoising digital images.

II. Approach and Procedures

i. Peak signal-to-noise ratio (PSNR)

PSNR(dB) is a quantitative denoising index used in this article. It is often used in the field of image processing to evaluate the quality of image reconstruction result. It represents the ratio of the maximum peak value of the signal to the average error. The specific calculation formula is

$$\text{PSNR(dB)} = 10\log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right) \quad (15)$$

where

$$\text{MSE} = \frac{1}{NM} \sum_{i=1}^N \sum_{j=1}^M (Y(i,j) - I(i,j))^2 \quad (16)$$

and where Y is the filtered image of size $N \times M$, I is the noise-free image, MAX for 8 bit signal is 255.

A larger PSNR(dB) value means better results, of course, we can also judge the quality of the results by our eyes, rather than relying solely on the PSNR(dB) indicator.

ii. Uniform Filter

The uniform filter is a kind of linear low pass filter. It works in a very simple way, averaging the surrounding pixels, which eliminates high-frequency components in the image. It is generally believed that noises are high-frequency components, and a low-pass filter such as a uniform filter can mitigate the effects of such high-frequency noise. While, the details and edges in the image are also high-frequency and will be erased by this filter.

Constructing a uniform filter convolution kernel is very simple. For an $N \times N$ convolution kernel, each element (weight) is the same.

$$Y(i, j) = \frac{\sum_{k=1}^N \sum_{l=1}^N I(k, l) w(i, j, k, l)}{\sum_{k=1}^N \sum_{l=1}^N w(i, j, k, l)} \quad (17)$$

$$w(i, j, k, l) = \frac{1}{N^2} \quad (18)$$

We can change the weight function w in Equation (17) to change the type of the filter.

iii. Gaussian Filter

The gaussian filter is also a kind of linear low-pass filter that takes a weighted average of surrounding pixels. Unlike the uniform filter, the farther away from the center, the lower the weight assigned.

An $N \times N$ gaussian filter convolution kernel can be constructed according to the following equation

$$w(i, j, k, l) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{(k-i)^2 + (l-j)^2}{2\sigma^2}} \quad (19)$$

where Y is the filtered image, I is the input image, σ is a parameter to control the gaussian distribution.

The first part of RHS of Equation (19) is not needed to be calculated, because of the normalize operation.

iv. Bilateral Filter

The Bilateral filter is a kind of non-linear filter. Based on the gaussian filter, it also considers the difference between the value of the surrounding pixels and the value of the center pixel. If the difference of one surrounding pixel is large, that means the value of the pixel changes drastically, it possibly is the edge of the image, and the filter will reduce its weight. Therefore, the bilateral filter can better preserve the image edge information while denoising rather than blur everything.

The difference between pixels values in gray scale image is the difference of their gray channel values, while in color images it can be defined in other ways. Weights of a discrete bilateral filter convolution kernel is given by

$$w(i, j, k, l) = e^{-\frac{(k-i)^2 + (l-j)^2}{2\sigma_c^2} - \frac{\|I(i, j) - I(k, l)\|^2}{2\sigma_s^2}} \quad (20)$$

where σ_c and σ_s are two parameters.

Because bilateral filter is a kind of nonlinear filter, the convolution kernel used to process each pixel is related to the pixel itself, so I generate a bilateral filter convolution kernel for each pixel in my program implementation.

v. Median Filter

The median filter is also a kind of non-linear filter that takes the median of the surrounding pixels as the value of the center pixel. This filter preserves image edge information while removing noise. It is especially suitable for removing impulse noise.

vi. Non-local Mean Filter

The non-local mean filter [2] is also a kind of non-linear filter that uses redundant information in the image. Regions in the image that is similar to surrounding region can provide this kind of information. This filter works well but requires more computation. The weights of its convolution kernel can be obtained by the following equations

$$w(i, j, k, l) = e^{-\frac{\|I(W_{i,j}) - I(W_{k,l})\|_{2,\sigma}^2}{h^2}} \quad (21)$$

$$\|I(W_{i,j}) - I(W_{k,l})\|_{2,\sigma}^2 = \sum_{x,y \in W} G(x, y) \|I(i - x, j - y) - I(k - x, l - y)\|^2 \quad (22)$$

$$G(x, y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2 + y^2}{2\sigma^2}} \quad (23)$$

where W is the nearby region of a pixel in the input image, I is the input image, σ and h are parameters.

vii. Block-matching and 3D (BM3D) Filtering

BM3D [3] filtering is an alternative algorithm to remove noise from an image. I will use it and explain it in the discussion part. The source code of BM3D is taken from online resources [4].

viii. Color Image

Color images has several channels, such as RGB image has three channels. Removing noise on color images is different from that in gray scale images. We can just remove noise in each channel separately or we can use the relationship information between channels to help us do it better. Some filters take difference of pixels into consideration, after we give a definition of the difference, they can work directly on color image. The definition which is used in this article is *Euclidean Distance*. Such as, in RGB input image I , the difference (distance) between pixels p_1 and p_2 is given by

$$\|I(p_1) - I(p_2)\| = \sqrt{(R(p_1) - R(p_2))^2 + (G(p_1) - G(p_2))^2 + (B(p_1) - B(p_2))^2} \quad (24)$$

where R , G and B are three channels of I . It maybe the one of easiest ways to calculate that, although the output images of using this definition may not be the best.

To calculate PSNR(dB) of color images, we can calculate PSNR(dB) of each channel individually. Here, for getting intuitive results, I am not going to distinguish data from different channels. So that, for each pair of color images, I will only have one PSNR(dB) value rather than three.

ix. Short noise

Short noise is Poisson distributed, I transfer the values in the noise image using following equation

$$D = f(z) = 2 \sqrt{z + \frac{3}{8}} \quad (25)$$

As a result, the additive noise become gaussian distributed with unit variance. Then I can use either a gaussian low pass filter or BM3D filtering to remove the noise. At last, reverse the transformation in Equation (25) using Equation (26).


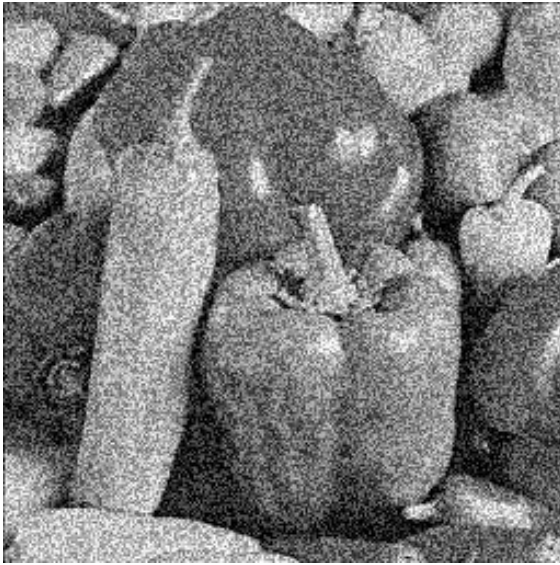
$$\text{Biased: } z' = f^{-1}(D) = \left(\frac{D}{2}\right)^2 - \frac{3}{8} \quad (26)$$

$$\text{Unbiased: } z' = f^{-1}(D) = \left(\frac{D}{2}\right)^2 - \frac{1}{8}$$

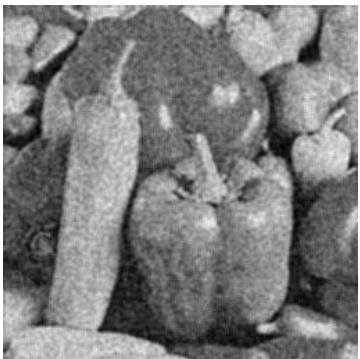
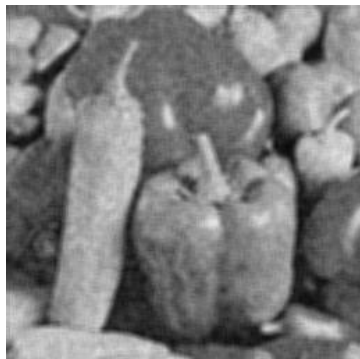
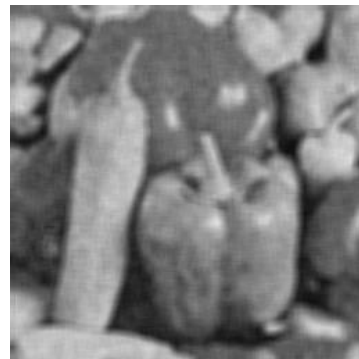
Notice that, to keep details after transformations, I use *float* data type rather than 8-bits integer to store the pixel value.

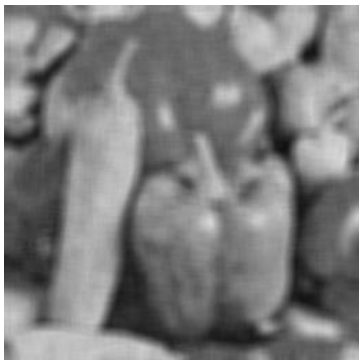


III. Experimental Results

Gray image denoising input images

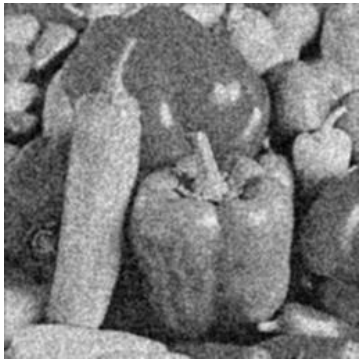

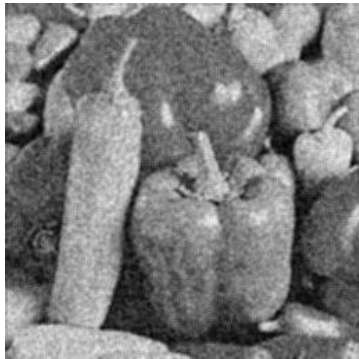



	
Fig15. Noise-free image <i>peper.raw</i>	Fig16. Noise image <i>peper_noise.raw</i> (PSNR=16.922)




Uniform filter

		
Fig16. 3*3 uniform filter (PSNR=24.573)	Fig17. 5*5 uniform filter (PSNR=24.780)	Fig18. 7*7 uniform filter (PSNR=23.694)

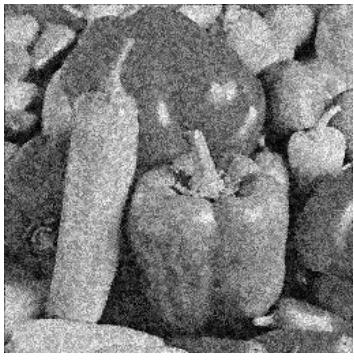
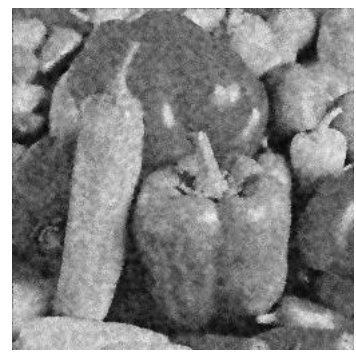

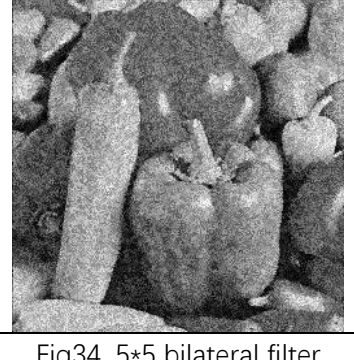
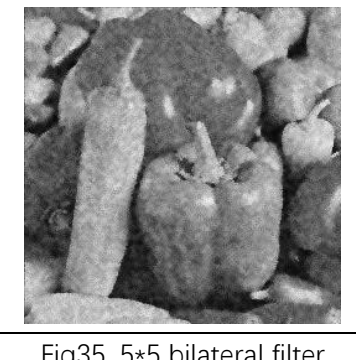
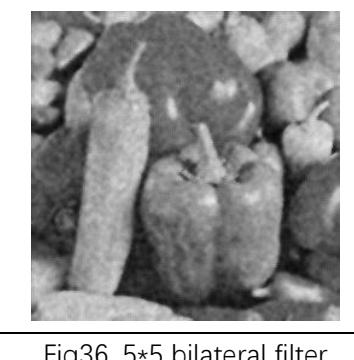
		
Fig19. 9*9 uniform filter (PSNR=22.623)	Fig20. 11*11 uniform filter (PSNR=21.729)	Fig21. 13*13 uniform filter (PSNR=20.996)

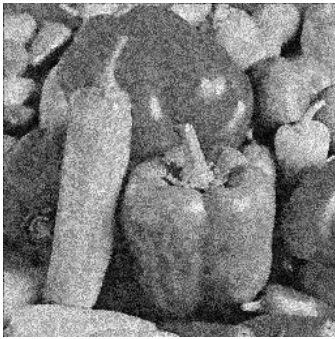
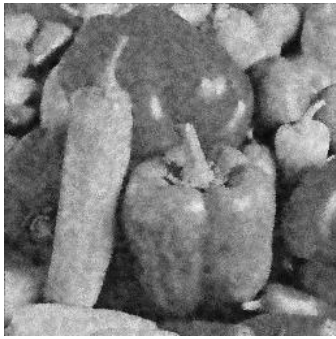

Gussian filter

		
Fig22. 3*3 gaussian filter $\sigma=1.0$ (PSNR=24.607)	Fig23. 3*3 gaussian filter $\sigma=2.0$ (PSNR=24.656)	Fig24. 3*3 gaussian filter $\sigma=4.0$ (PSNR=24.599)
		
Fig25. 5*5 gaussian filter $\sigma=1.0$ (PSNR=25.410)	Fig26. 5*5 gaussian filter $\sigma=2.0$ (PSNR=25.278)	Fig27. 5*5 gaussian filter $\sigma=4.0$ (PSNR=24.922)

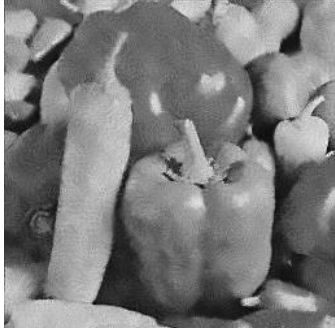

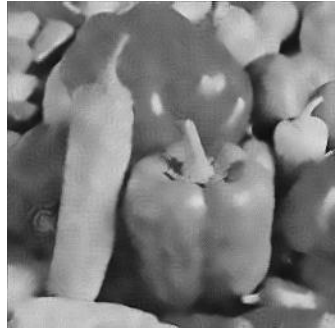
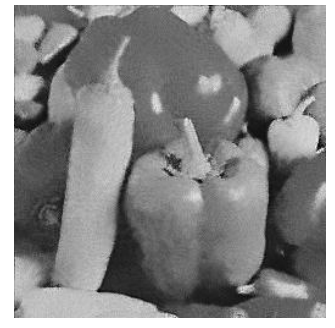


		
Fig28. 7*7 gaussian filter $\sigma=1.0$ (PSNR=25.448)	Fig29. 7*7 gaussian filter $\sigma=2.0$ (PSNR=24.863)	Fig30. 7*7 gaussian filter $\sigma=4.0$ (PSNR=24.203)

Bilateral filter






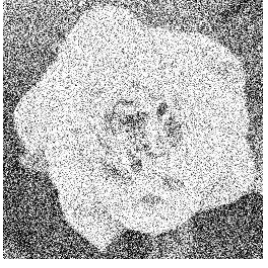
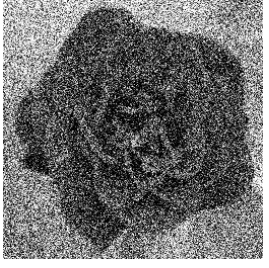
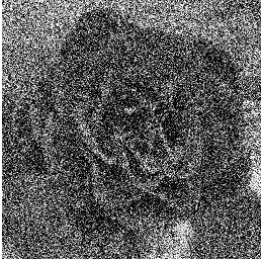
		
Fig31. 5*5 bilateral filter $\sigma_c=2.0$ $\sigma_s=32.0$ (PSNR=19.628)	Fig32. 5*5 bilateral filter $\sigma_c=2.0$ $\sigma_s=64.0$ (PSNR=24.046)	Fig33. 5*5 bilateral filter $\sigma_c=2.0$ $\sigma_s=128.0$ (PSNR=25.795)
		
Fig34. 5*5 bilateral filter $\sigma_c=4.0$ $\sigma_s=32.0$ (PSNR=19.758)	Fig35. 5*5 bilateral filter $\sigma_c=4.0$ $\sigma_s=64.0$ (PSNR=24.199)	Fig36. 5*5 bilateral filter $\sigma_c=4.0$ $\sigma_s=128.0$ (PSNR=25.642)

		
Fig37. 7*7 bilateral filter $\sigma_c=2.0$ $\sigma_s=32.0$ (PSNR=19.720)	Fig38. 7*7 bilateral filter $\sigma_c=2.0$ $\sigma_s=64.0$ (PSNR=24.214)	Fig39. 7*7 bilateral filter $\sigma_c=2.0$ $\sigma_s=128.0$ (PSNR=25.700)

Non-local mean filter




		
Fig40. 11*11 non-local mean filter W is 5*5 $\sigma=2.0$ $h=64.0$ (PSNR=26.297)	Fig41. 11*11 non-local mean filter W is 5*5 $\sigma=2.0$ $h=128.0$ (PSNR=24.829)	Fig42. 11*11 non-local mean filter W is 5*5 $\sigma=4.0$ $h=64.0$ (PSNR=26.512)
		
Fig43. 21*21 non-local mean filter W is 7*7 $\sigma=2.0$ $h=64.0$ (PSNR=25.943)	Fig44. 21*21 non-local mean filter W is 7*7 $\sigma=2.0$ $h=128.0$ (PSNR=23.458)	Fig45. 21*21 non-local mean filter W is 7*7 $\sigma=4.0$ $h=64.0$ (PSNR=26.088)

Color image denoising input images

			
Fig46a. Noise-free image <i>rose_color</i>	Fig46b. R channel of Fig46a	Fig46c. G channel of Fig46a	Fig46d. B channel of Fig46a
			
Fig47a. Noise image <i>rose_color_noise</i> (PSNR=11.729)	Fig47b. R channel of Fig47a	Fig47c. G channel of Fig47a	Fig47d. B channel of Fig47a





Color image denoising: “Median filter & other filters” cascade order

A: Step 1-Median filter

		
Fig48. 3*3 median filter (input Fig47a) (PSNR=17.606)	Fig49. 5*5 median filter (input Fig47a) (PSNR=21.777)	Fig50. 7*7 median filter (input Fig47a) (PSNR=23.613)

Color image denoising: “Median filter & other filters” cascade order





A: Step 2-Other filters



	
Fig49. 5*5 uniform filter (input Fig48) (PSNR=23.708)	Fig50. 5*5 gaussian filter $\sigma=2.0$ (input Fig48) (PSNR=23.393)
	
Fig51. 3*3 uniform filter (input Fig48) (PSNR=21.568)	Fig52. 3*3 gaussian filter $\sigma=2.0$ (input Fig48) (PSNR=21.513)

	
<p>Fig53. 5*5 bilateral filter $\sigma_c=2.0$ $\sigma_s=64.0$ (input Fig48) (PSNR=22.098)</p>	<p>Fig54. 5*5 bilateral filter (perform without separating channels) $\sigma_c=2.0$ $\sigma_s=64.0$ (input Fig48) (PSNR=21.877)</p>
	
<p>Fig55. 11*11 non-local mean filter W is 5*5 $\sigma=2.0$ $h=64.0$ (input Fig48) (PSNR=24.492)</p>	<p>Fig56. 11*11 non-local mean filter (perform without separating channels) W is 5*5 $\sigma=2.0$ $h=64.0$ (input Fig48) (PSNR=20.803)</p>

Color image denoising: “Median filter & other filters” cascade order



B: Step 1-Other filters





	
Fig57. 5*5 uniform filter (input Fig47a) (PSNR=21.554)	Fig58. 5*5 gaussian filter $\sigma=2.0$ (input Fig47a) (PSNR=21.510)
	
Fig59. 5*5 bilateral filter $\sigma_c=2.0$ $\sigma_s=64.0$ (input Fig47a) (PSNR=15.239)	Fig60. 5*5 bilateral filter (perform without separating channels) $\sigma_c=2.0$ $\sigma_s=64.0$ (input Fig47a) (PSNR=14.101)

	
<p>Fig61. 11*11 non-local mean filter W is 5*5 $\sigma=2.0$ $h=64.0$ (input Fig47a) (PSNR=13.325)</p>	<p>Fig62. 11*11 non-local mean filter (perform without separating channels) W is 5*5 $\sigma=2.0$ $h=64.0$ (input Fig47a) (PSNR=11.729)</p>


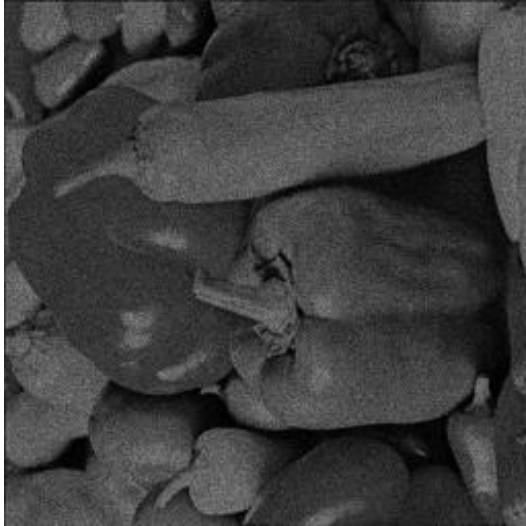
Color image denoising: “Median filter & other filters” cascade order

B: Step 2-Median filter

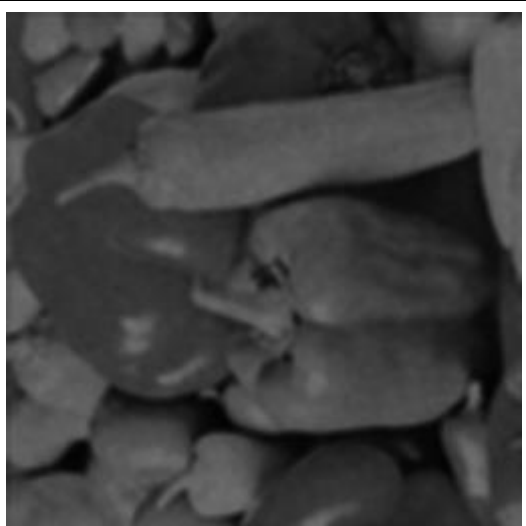
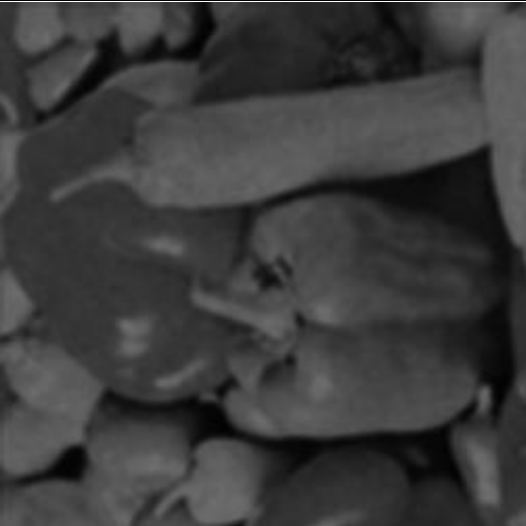
	
<p>Fig63. 3*3 median filter (input Fig57) (PSNR=21.939)</p>	<p>Fig64. 3*3 median filter (input Fig58) (PSNR=21.815)</p>

	
Fig65. 3*3 median filter (input Fig59) (PSNR=19.751)	Fig66. 3*3 median filter (input Fig60) (19.301)
	
Fig67. 3*3 median filter (input Fig61) (PSNR=18.804)	Fig68. 3*3 median filter (input Fig62) (PSNR=17.606)

Short noise denoising input images

	
Fig69. Noise-free image <i>peper_dark</i>	Fig70. Noise image <i>peper_dark_noise</i> (PSNR=30.210)

Short noise denoising using gaussian low pass filter in the middle step

	
Fig71. 5*5 $\sigma=2.0$ Biased (PSNR=32.725)	Fig72. 5*5 $\sigma=2.0$ Unbiased (PSNR=32.754)

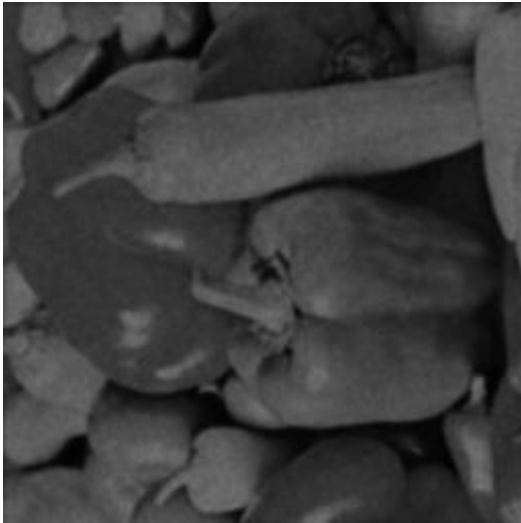


Fig73. 5*5 gaussian filter $\sigma=1.0$ Biased
(PSNR=34.643)

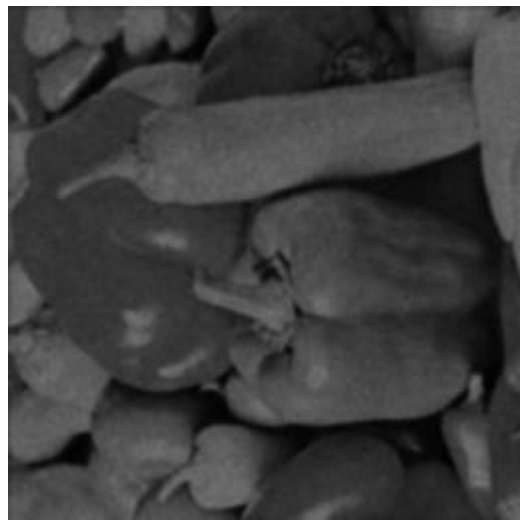


Fig74. 5*5 gaussian filter $\sigma=1.0$ Unbiased
(PSNR=34.675)

Short noise denoising using BM3D in the middle step

Here σ should be 1, because after the transformation is performed the noise will be gaussian noise with unit variation.



Fig75. BM3D $\sigma=1$ Biased (PSNR=39.266)

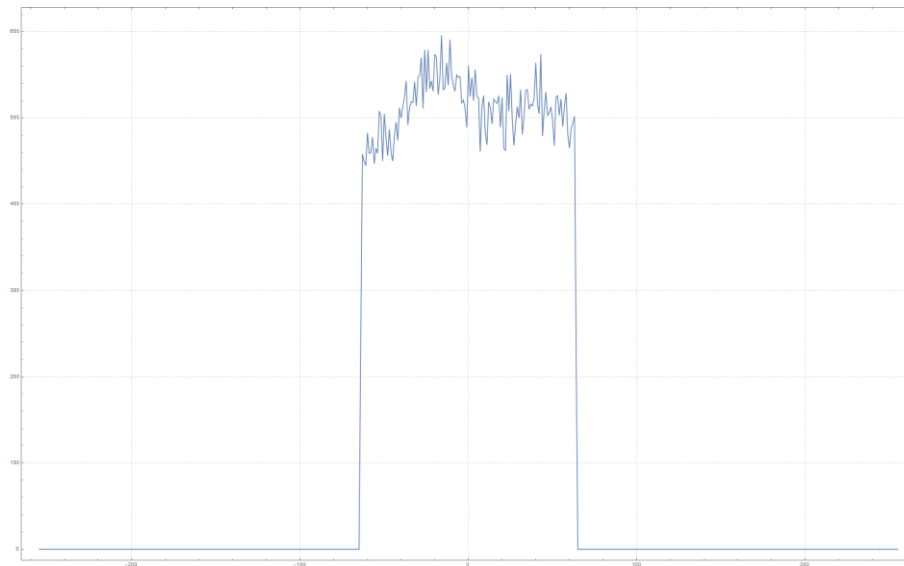


Fig76. BM3D $\sigma=1$ Unbiased (PSNR=39.306)

IV. Discussion

- i. [Question (a)-(1)] What is the type of embedded noise in Figure 7(b)?

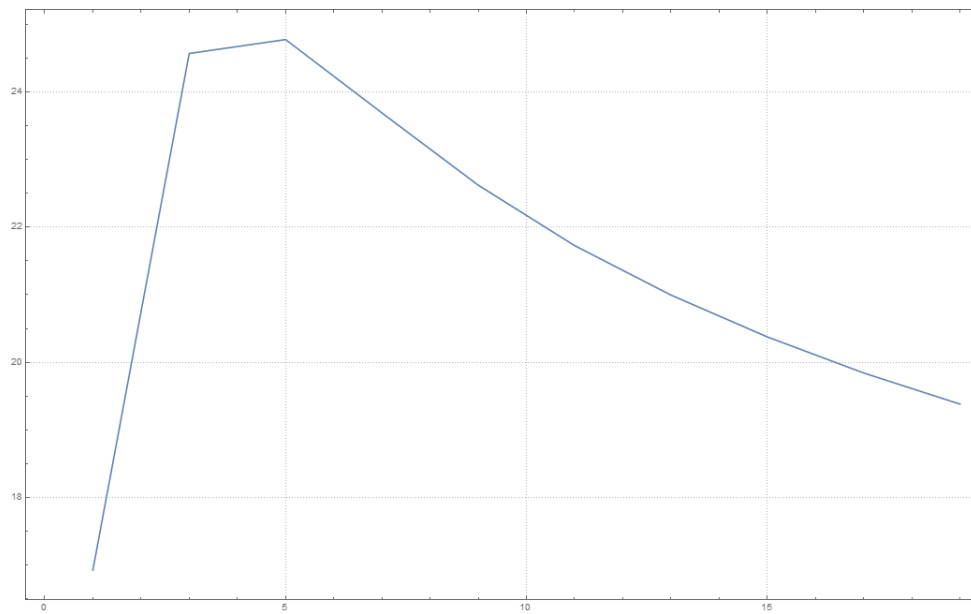
It is uniform noise, which can be verified by plotting the histogram of difference between noise image and noise-free image. Difference here can be calculated by using noise image subtracting noise-free image. The following plot image shows the result.



It can be seen that all the difference values are among $[-64, 64]$, and occurrences of values are basically the same. So we can identify that the type of noise is uniform noise.

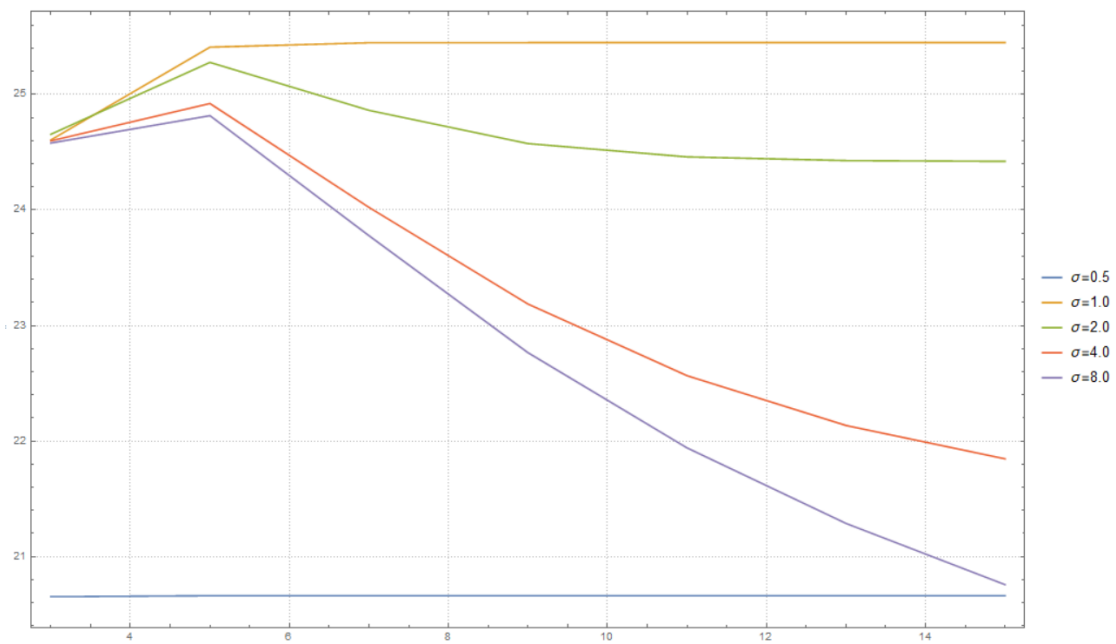
- ii. [Question (a)-(2)] Plot the peak-to-signal-noise value as a function of N for both weighting functions.

For uniform filters:



X-axis is the number of N , and y-axis is the PSNR.

For gaussian filters:

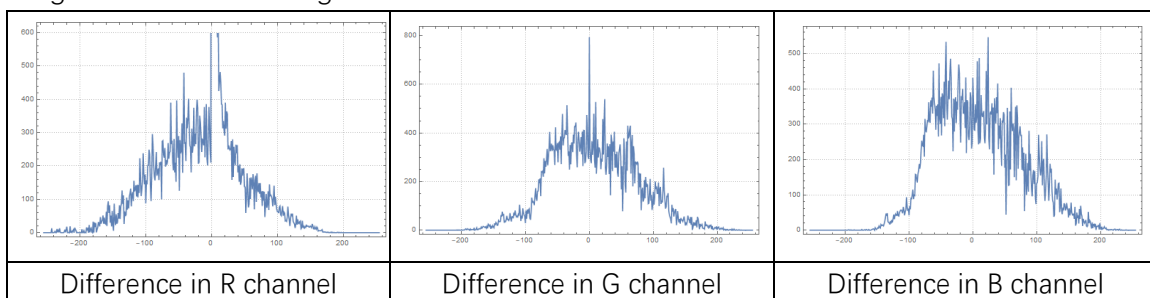


X-axis is the number of N , and y-axis is the PSNR.

iii. [Question (b)-(1)] Should you perform filtering on individual channels separately for both noise types?

Some filters such as bilateral filter and non-local mean filter can perform on color image without separate channels, because their weight functions have already considered the measurement of distance between two color pixels. While a definition of distance show be given at first. In my approach I use the *Euclidean Distance* of RGB channels. Other distance measurements using Euclidean Distance equations but performed on other color spaces are also available. The distance of luminance also can be used, and so on.

While, despite that, I think it is okay to perform filtering on individual channels separately. Because the mix noise in `rose_color_noise.raw` are not correlated between RGB channels, and the noise types and their properties in each channel are basically the same. The difference between noise image and noise-free image in each channel is shown below.



Two kinds of noise are impulse noise and gaussian noise, while the assignment document says they are impulse noise and uniform noise, but that does not matter. As a result, if we perform filtering individually, it will not get color casted results.

In my experiments above, for those filters can be both perform on each channel or on a color image, the PSNR values of individually performed results are better than the others. Therefore, for `rose_color_noise.raw`, I can perform filter on individual channels separately for both noise types.

iv. [Question (b)-(2)] What filters would you like use to remove mixed noise?

Median filter and non-local mean filter (perform filtering on individual channels). First I will use the median filter then I will use non-local mean filter. In my experiments, this kind of combination got the highest PSNR value. After that, median filter & gaussian filter also got a high PSNR value. However, all the experimental results did not seem satisfactory to me.

v. [Question (b)-(3)] Can you cascade these filters in any order?

Justify your answer.

Yes, I can. The median filter can move impulse noise, so it is necessary. This should be at least on median filter.

The experiment results of order are shown above. We can see that using median filter at last is not a good idea based on PSNR or visual experience.

The median filter should be performed at first, it removes impulse noise. Then other filters can be used. While, edges are blurred in all these results.

vi. [Question (b)-(4)] Please suggest an alternative to low pass filter with Gaussian weight function and discuss why such solution can potentially work better.

The question asks me to suggest a low pass filter alternative. If our filter can distinguish edges and uniform noise, the results will be better. Because the uniform noise ranges from -64 to 64. So, the variance of edge may larger than that of noise. We can modify the weight function like this

$$w(i, j, k, l) = e^{-\frac{(k-i)^2 + (l-j)^2}{2\sigma_c^2} - \frac{\text{Max}(0, \|I(i, j) - I(k, l)\|^2 - t^2)}{2\sigma_s^2}}$$

This is a combination of gaussian low pass filter and bilateral filter. Max is a function to get larger parameter from two, t is the threshold, this threshold determines whether the difference of neighbor pixel value is large enough to be accepted as an edge. If a neighbor pixel is edge, then decrease its weight. Here let t equals 65, so my new filter can distinguish the edge from the input noise image. This weight function is only an example, what I mainly want to express here is weight functions can use the limited range of additional noise value to detect edges. Unfortunately, I have no sufficient time to implement it and adjust it. (I enrolled in this course late.)

vii. [Question (c)-(2)] Show the final noise-removed pepper images and compare the PSNR values and visual quality of the resulting images.

The results are shown above. BM3D filtering got better visual experience and a higher PSNR value. The biased and unbiased results look no different, while the PSNR values are slightly different. I think it is because the difference between two equations in Equation (26) is only 0.25, much smaller than quantization step (Δ) 1, so only few pixels will get different values after quantization.

Note: the gaussian low pass filter is implemented by myself and written in C++, and the BM3D function I used was downloaded from [4], and it is written in MATLAB.

viii. Results from 4 kinds of filters

I found that using 5×5 kernels usually can get good results. And a good parameter for gaussian weight function is around 2. The parameter for adjusting distance of pixel values should be larger than above ones, because the pixel value changes by at least one and up to 255 (one 8-bit channel), and 64 and around will be good choices.

Uniform filters and gaussian filters blur the edges, while bilateral filters really preserve edges. So, results from bilateral filters seem better, while they do not get much higher PSNR than those from linear low pass filters. Results from non-local mean filters seem the best, and the PSNR values are also higher if we set fine parameters. However, non-local mean filters are much slower than other ones which is a defect. After I increase the search area of non-local mean filter, I found the PSNR values do not increase while the computational time increases rapidly, so a relatively small searching area, such as 11×11 , is enough for non-local mean filters.

ix. PSNR and visual quality

In sub problem (b). I found that even the filtered image is badly blurred, the PSNR is also high only if the added noise is removed. I can see some details in noise image which has low PSNR, while we cannot find these details from results with high PSNR values generated by low pass filters. That means the visual quality of an image is not only determined by noise level, but also some other indicators. An image with high PSNR value does not mean it looks good.

x. Distance of colors

Actually, I do not think calculating *Euclidean Distance* on RGB channels is a good idea. I think using another color spaces such as *Lab*, *YUV*, *HSL(or HSV)*, may get different results. Unfortunately, I do not have enough time to implement them and do some experiments.

xi. Briefly explain BM3D filtering

BM3D is designed to denoising gray level images. Its mean idea is similar to non-local mean. Non-local mean finds similar regions in 2D area, while BM3D tries to find similar blocks which is 3D.

It has two steps, each step has an estimate procedure with domain transformation. In the first step called basic estimate, to generate one pixel value, find similar 2D regions like non-local mean does. Then combine these regions in to a 3D block, transform this block to another domain, the transform method can be varies. Then using Hard-thresholding to denoising the block. After that, transform the block back. Hard-thresholding is a way to weaken the noise in the block composed by similar regions. Then values from different blocks are aggregated by weighted average to get the value of each pixel. The method to get weights here is also adjustable. Here we have already get a primary denoised image. Next is the second step called final estimate using the output image from the first step as the input. Similarly, for each pixel, find similar regions and combine them into a block. Blocks generated at this procedure will compare with the blocks generated in the first step and get blocks for noise. Then, using Wiener filtering on two kinds of new blocks got in step 2. It needs a domain transform procedure and a inverse transform procedure before and after Wiener filtering. Finally, do the aggregation procedure again, and we can get the denoised image.

References

- [1] Malvar, Henrique S., Li-wei He, and Ross Cutler. "High-quality linear interpolation for demosaicing of Bayer-patterned color images." *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*. Vol. 3. IEEE, 2004.
- [2] Buades, Antoni, Bartomeu Coll, and J-M. Morel. "A non-local algorithm for image denoising." *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 2. IEEE, 2005.
- [3] Dabov, Kostadin, et al. "Image denoising with block-matching and 3D filtering." *Image Processing: Algorithms and Systems, Neural Networks, and Machine Learning*. Vol. 6064. International Society for Optics and Photonics, 2006.
- [4] Online: <http://www.cs.tut.fi/~foi/GCF-BM3D/>