USC Viterbi
School of Engineering

DEPARTMENT
Electrical and Computer Engineering

EE 569
Homework #1: Image Demosaicing, Denoising and Histogram Manipulation

Name: Zhiwei Deng
Date: 1/24/2020

Contents

1. Problem 1

1.1 Bilinear Demosaicing

1.1.1 Abstract and Motivation

Cameras use Bayer sensor array to image to save expense and storage. For one reason, an image captured by Color Filtered Array (CFA) occupies only 1/3 of an RGB Image because of every pixel of the image only contains one channel of R, G or B. The Bayer Pattern is shown as Figure 1. Two types of rows it has, red row and blue row. In red rows, pixel channel starts from green and green and red appears alternately. In blue rows, pixel channel starts from blue and blue and green appears alternately.

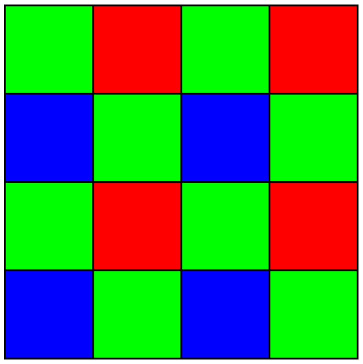


Figure 1. Bayer Pattern

In this case, how to recover RGB color image from a color filtered image is a necessary problem to be considered. There are many ways to demosaic a color filtered image and I used bilinear interpolation method first. This method only uses the pixels that located in a 3 * 3 window that centered by current pixel to evaluate the missing channels' values of current pixel. In details, green pixels use adjacent four pixels to evaluate channel R and B values. Red or blue pixels use four adjacent pixels to evaluate G values and use four corner pixels to evaluate B or R values. The method's calculation is shown by Figure 2 and Table 1.

F(-1,-1)	*F(-1,0)*	*F(-1,1)*
F(0,-1)	*F(0,0)*	*F(0,1)*
F(1,-1)	*F(1,0)*	*F(1,1)*

Figure 2. The window of Bilinear Interpolation method

Table 1. The calculation of Bilinear Interpolation method

F(0,0)	*F(0,0)'s R value*	*F(0,0)'s G value*	*F(0,0)'s B value*
R	F(0,0)	(F(0,-1) + F(0,1) +F(-1,0) + F(1, 0)) / 4	(F(-1,-1) + F(1,1) +F(-1,1) + F(1, -1)) / 4
G in R row	(F(0,-1) + F(0,1))/2	F(0,0)	(F(-1,0) + F(1, 0)) / 2
G in B row	(F(-1,0) + F(1, 0)) / 2	F(0,0)	(F(0,-1) + F(0,1))/2
B	(F(-1,-1) + F(1,1) +F(-1,1) + F(1, -1)) / 4	(F(0,-1) + F(0,1) +F(-1,0) + F(1, 0)) / 4	F(0,0)

Figure 3 shows the original gay image of dog, I used bilinear method to recover the color image.

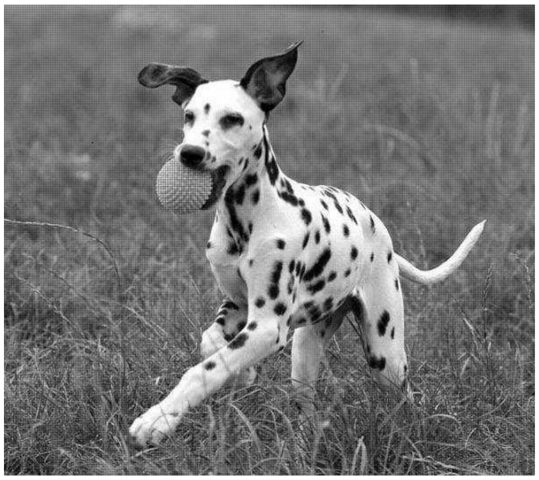


Figure 3. Dog.raw

1.1.2 Approach and Procedures

Firstly, to ensure the pixels on the edge can recover closer color to ground true values. The image should be extended on the edge. I use reflection method to extend the image edges. At the same time, the image should remain the Bayer Array features which green pixels and blue or red pixels appear alternately. To ensure this, instead of extending the edges from the edge pixels, I extended the image from the second row and second column to preserve the same features as original image. The difference is shown below as Figure 4. In Figure 4(a), there is apparently one column pixels missed which means the reflection starts at the second column.

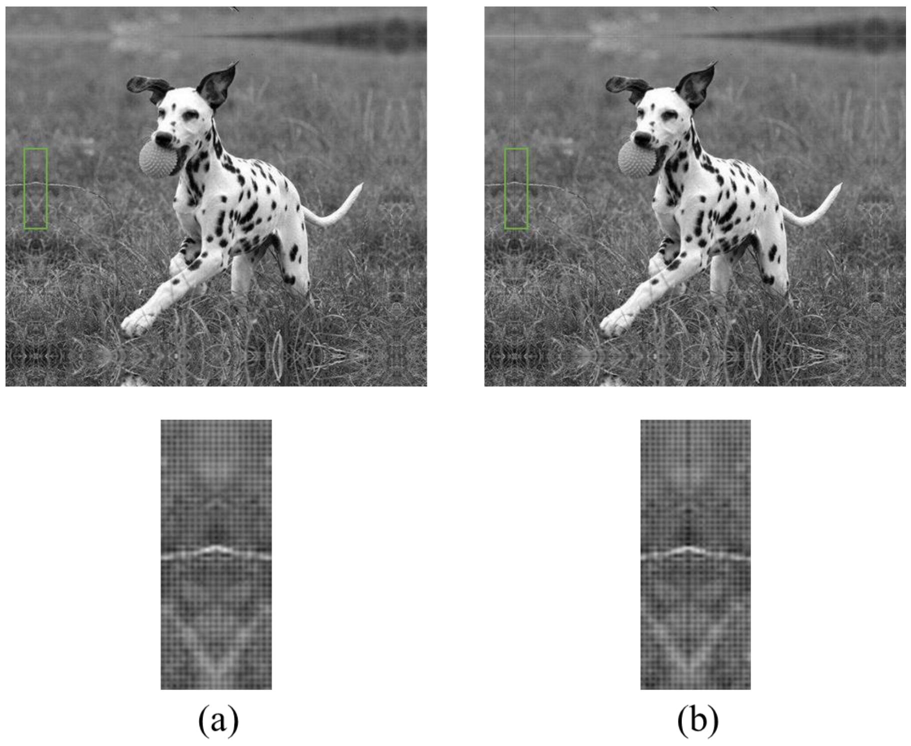


Figure 4. Two different types of edge extension by 50 pixels width

To demosaic a color filtered image by bilinear method, four types of pixels should be divided and handled separately as shown in Table 1. From the Bayer pattern, we can easily get that an odd row is a red row, an even row is a blue row. And a red pixel always located in the even column and a blue pixel always located in odd column. So, I looped all the pixels for *F (i, j)* and use i index to find the type of green pixels and use j index to divide blue pixels from red ones.

Basically, demosaicing is a process to recover the color of the image. It uses surrounding pixel values to estimate the true value. To measure the quality of recovery, it is reasonable to evaluate the similarity between the result and the original image. So, I used PSNR as a metric to evaluate the quality of demosaicing. The equations are shown below as (1-2), where F is result image, K is original image and MAX is maximum pixel intensity 255.

$$MSE = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}[F(i,j) - K(i,j)]^2 \tag{1}$$

$$PSNR = 10\log_{10}\left(\frac{MAX^2}{MSE}\right) \tag{2}$$

1.1.3 Experimental Result.

The demosaiced image by bilinear interpolation method is shown as below in Figure 5.

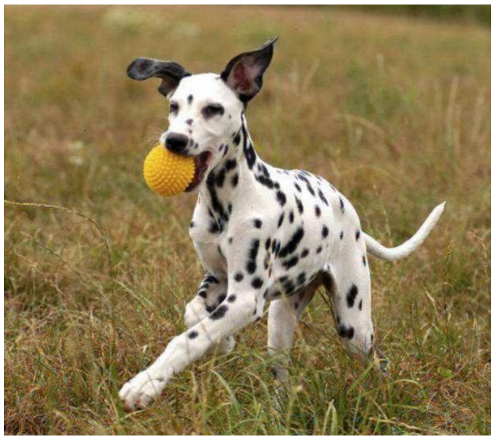
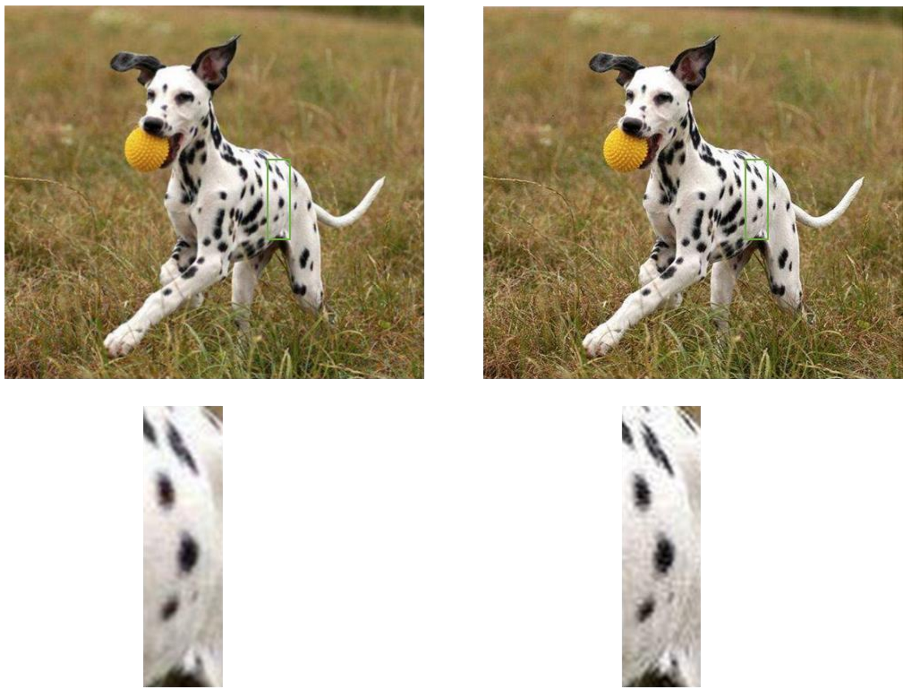


Figure 5. Dog_demosaic.raw PSNR = 28.1169dB

1.1.4 Discussion

Bilinear Interpolation is a fast and easy mean to realize demosaicing**, the computational complexity is O(n) (n is the number of total pixels), we only need to loop whole image pixels once. Spcifically, the time complexity is O(9n), cause the for every pixel, there is a 3 * 3 window to do the convolution.** But its method only takes 4 to 8 pixels in to consider, which is not sufficient enough and causes some problems like **blurry edges, false colors, detail disappearance and 'Zipper Effect'.**

Figure 6 shows a zoomed in detailed difference between demosaiced imgae (a) and original image (b). We can see that the speckle edges of (a) is much blurry than ones of (b). Also, (b)'s color of black is darker and the white color is brighter.



(a) (b)

Figure 6. (a) Dog_demosaic.raw (b) Dog_ori.raw

There might be **three reasons** for thses shortcomings of bilinear interpolation. **First, bilinear interpolation only uses the same channel values to estimate the current pixel's certain channel value and discard other channels' effects.** This assumption ignores the connections between different channels which might cause false color pixels, but the three channels are effected by each other with different weights in the real imaging process. **To solve this problem, maybe setting different weights to different channels' values could improve the performance of the algorithm. Second, the pixels that it considered might be too few, only four pixels are considered for G pixel and eight pixels are considered for R and B pixel.** This assumption may made the edge pixels take consider too few pixels to estimate the clear value, which might cause a blurry edge proble. **To combat this problem, a bigger window than Figure two could be applied to estimate more accurate values. Third, the artifact of "Zipper Effect" is caused by averaging, the averaging method in bilinear interpolation discards the distance of pixels.** But in real world, closer the pixels are, larger effects it would have with each other. **Rather than using a uniform filter, we can use wighted filters like Gaussian Kernel for this. Also, there is an more sophiscated algorithm called 'Newton Gregory interpolation' is propose recently for demosaicing WRGB image and achieved performance of 42dB using the difference of pixels to calculate pixel values changing rate and estimate pixle values more precisely[1].**

To combat the shortcomings of bilinear interpolationmethod, another method is propsed as Malvar-He-Cutler (MHC) Demosaicing below.

1.2 Malvar-He-Cutler (MHC) Demosaicing

1.2.1 Abstract and Motivation

Bilinear interpolation demosaicing is simple and easy to have obvious artifacts. As I mentioned the improvement ways above in discussion part, a new algorithm is proposed. MHC algorithm has a bigger window than bilinear, which is 5 * 5. And also, in MHC method, the three channels have different weights in each other and in the same window. In addition, the MHC filters have bigger coefficients if the pixels in the window is closer to the center. The eight filters are shown as Figure 7[3].

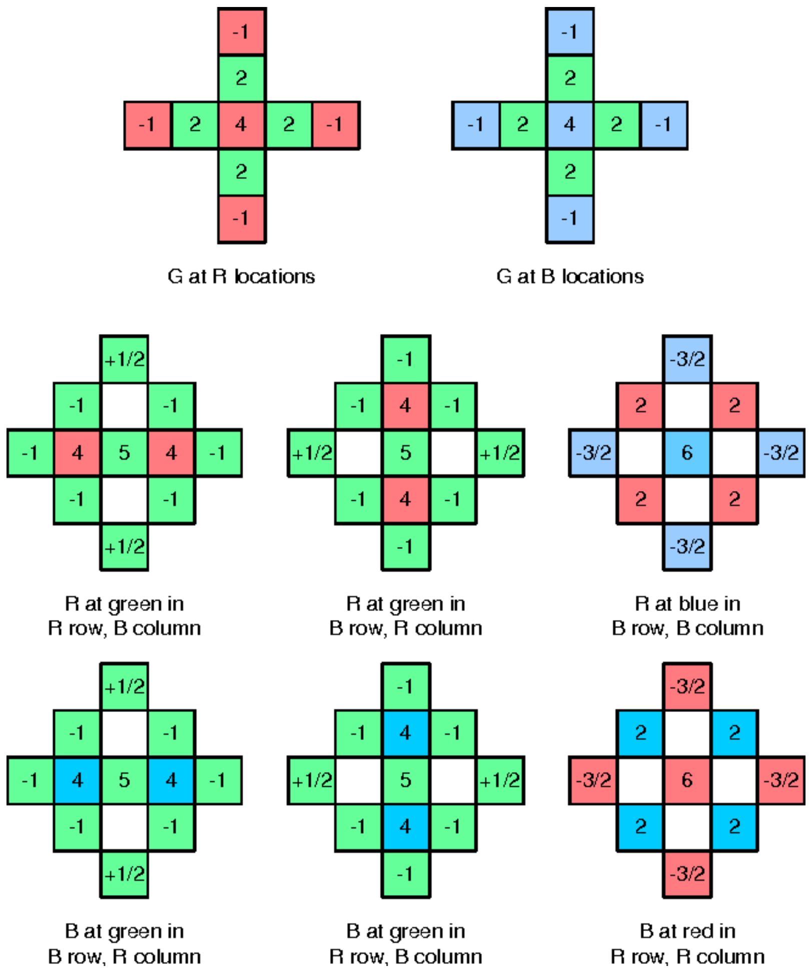

G at R locations
G at B locations
R at green in R row, B column
R at green in B row, R column
R at blue in B row, B column
B at green in B row, R column
B at green in R row, B column
B at red in R row, R column

Figure 7. MHC Filters Coefficients

1.2.2 Approach and Procedures

As same as bilinear demosaicing, we need to pad the first, too. Instead of pading with 3 * 3 window, I padded the image with 5 * 5 window which means the extended image has a boundary with 2 pixels width. Then I divided the pixels into four groups as Table 1 lists and multiplied the corresponding values with certain coefficients – convolution. The diagram of the preocedure is shown as below in Figure 8.

The division method of pixels and extension method is same as bilinear interpolation method.

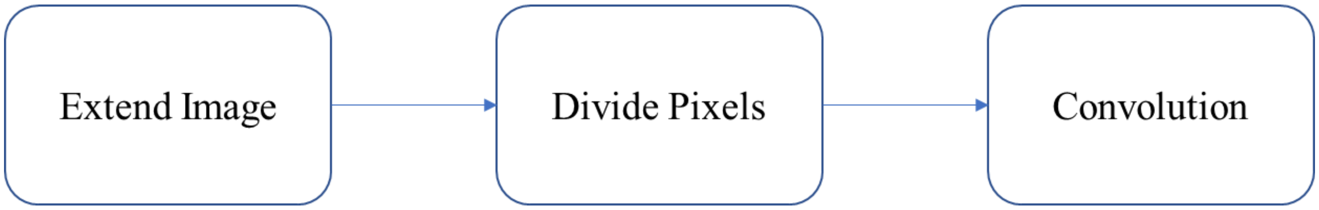


Figure 8. Diagram of MHC Procedure

1.2.3 Experimental Result.

The demosaiced image by Malvar-He-Cutler method is shown as below in Figure 9.

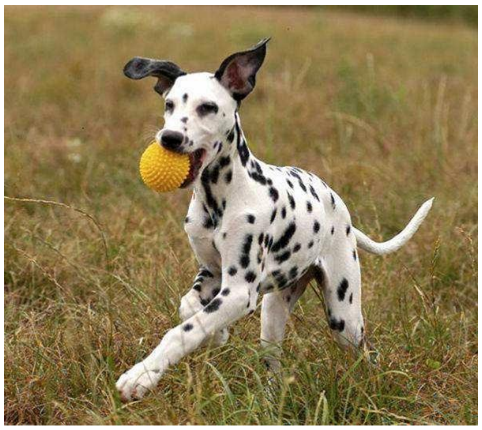


Figure 9. Dog_demosaicMHC.raw PSNR = 34.1863dB

1.2.4 Discussion

MHC demosaicing is a more complicated method compared to bilinear demosaicing due to it larger window size and more complex coefficients. **But the computational complexity is as same as bilinear method, which is O(25n) and the spatial complexity is larger cause there are more space we need to do the convolution. By using other channels' pixel values to estimate current pixel's corresponding channel value, it corrects some of the false color pixels. Also, it uses different coefficients of filter pixels rather than uniform coefficients to avoid edge corruption as possible.**

Figure 10 shows a detailed difference among each other. We can tell that the MHC image's detail is almost same with the original ones and obviously better than (a)'s condition. In addition, the PSNR of this two different methods are **28.1169dB and 34.1863dB** respectively. The MHC's performance is munch better than bilinear method.

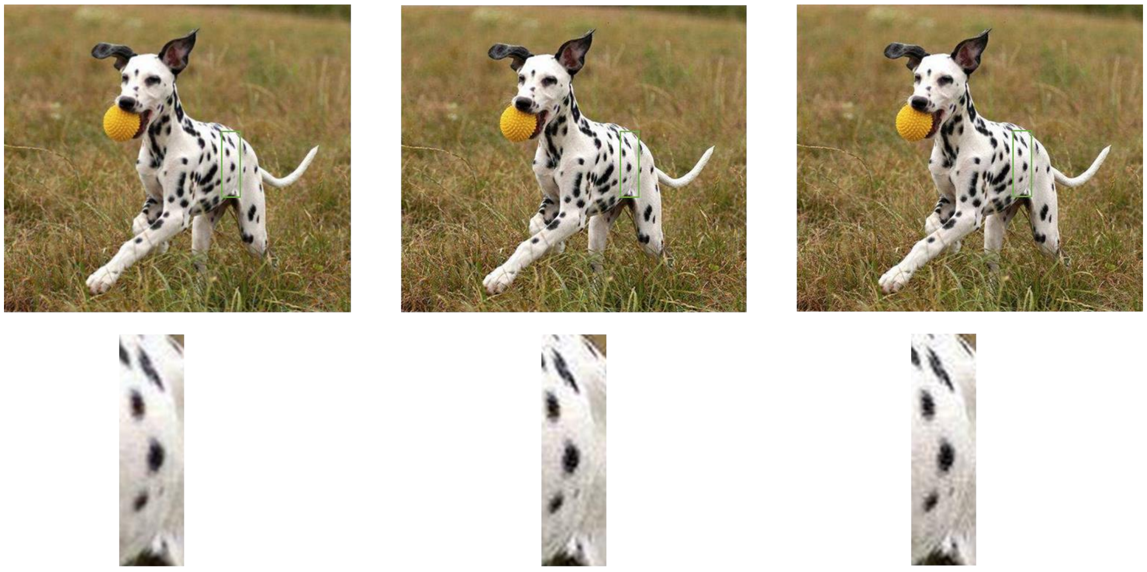
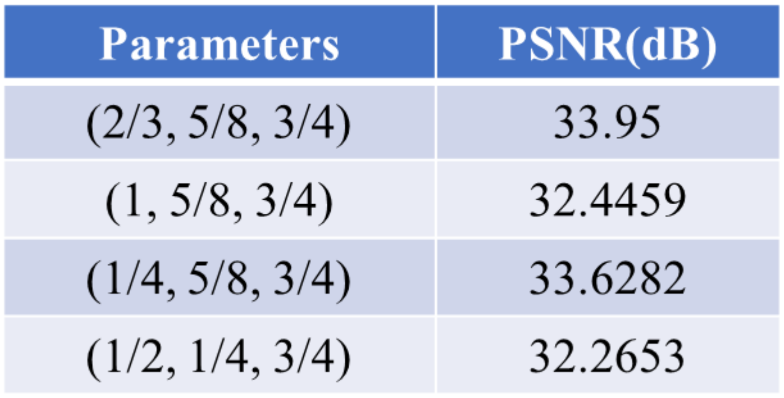


Figure 10. (a) Dog_demosaic.raw (b) Dog_demosaicMHC (c) Dog_ori.raw

The correlation between RGB is the key point that MHC's performance is better than bilinear method. Also, instead of using the default weight of α= 1/2, β = 5 / 8, γ = 3 / 4, I tried some value of my own to find a better performance with a different set of parameter. The result is shown in Table 2. The parameters are grouped by (α,β,γ).

Table 2. Different Parameters with MHC Demosaicing



Parameters	PSNR(dB)
(2/3, 5/8, 3/4)	33.95
(1, 5/8, 3/4)	32.4459
(1/4, 5/8, 3/4)	33.6282
(1/2, 1/4, 3/4)	32.2653

I found that the default parameters have been tested and can achieve best performance of demosaicing. Whether I tuned any parameter, the result cannot achieve beter performance. **So, we can say that the correlations between R, G and B channels are α= 1/2, β = 5 / 8 and γ = 3 / 4.**

1.3 Histogram Manipulation

1.3.1 Abstract and Motivation

Histogram is an important feature of an image, it shows the distribution of the pixel values which reprsents the image brightness of certain channel. When the distribution is centered around smaller value, the image tends to be brighter. In real world, there are many pixtures are taken in unexpected environment, such as too dark or too bright. In these cases, the histogram equalization is an extremely important technique to enhance the image to get more information from it.

In this part, two methods of histogram equalization will be discussed and realized. One is based on the transfer function and the other is based on statistic probability.

1.3.2 Approach and Procedures

The first of both methods is getting the histogram of an image. In this step, I just created a 3 * 256 array to store the counts of pixels of each channel.

For method A, after getting the histogram data, transfer function need to be caculated based on the equation (3) as called cumulative distribution function. In equation(3), N(x) represents the new grayscale value, px(t) is the distribution of pixel values. After computing the transfer function, I looped over the image to change the old grayscale values to the new ones. This map based on (3) is called Transfer Funtion.

$$N(x) = 255 * \int_0^x px(t)dt \tag{3}$$

The diagram of method A is shown in Figure 11.

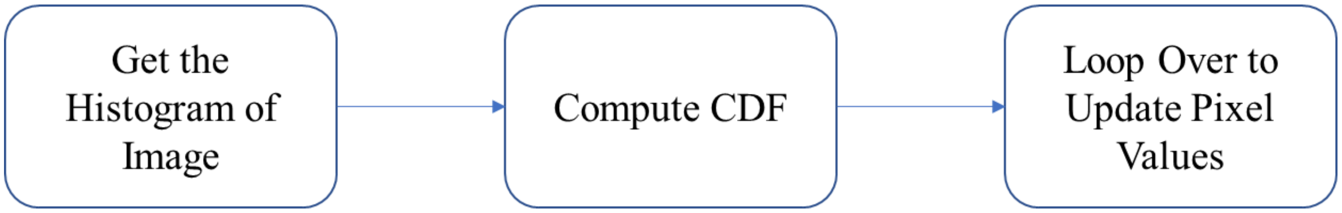


Figure 11. Diagram of Method A Procedure

For method B, the 'Bucket Filling Method', the next step should be sorting the pixels from small to large. And then set equal number of pixels to each value between 0~255. The number of each value should be computed by equation (4).

$$\mathrm{Num} = \mathrm{Width} * \mathrm{Height} / 256 \tag{4}$$

The diagram of method B and Figure 12.

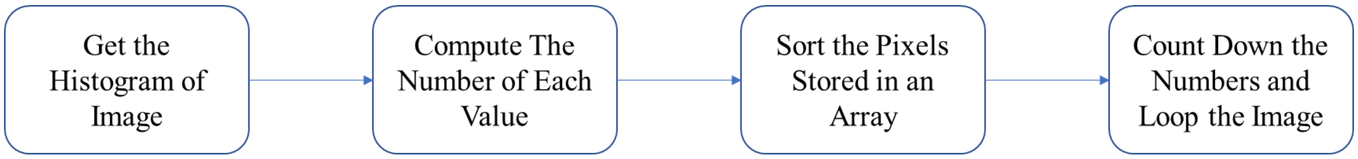
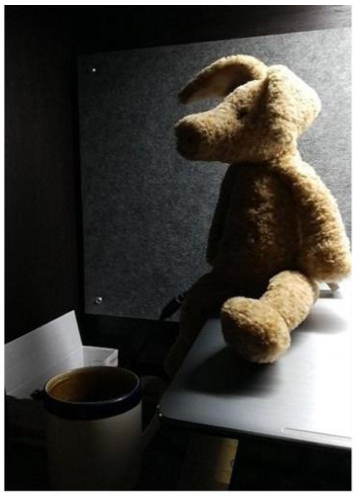
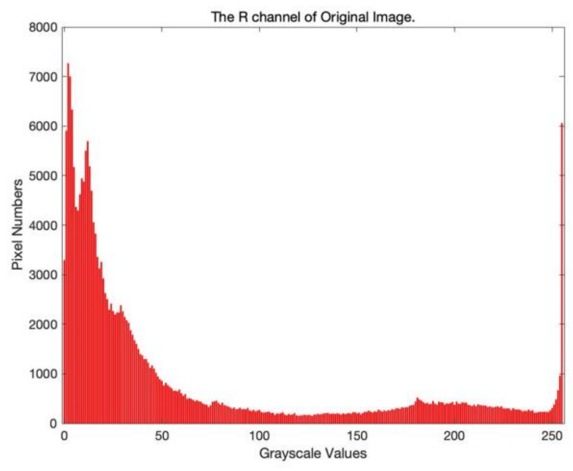

Get the Histogram of Image
Compute The Number of Each Value
Sort the Pixels Stored in an Array
Count Down the Numbers and Loop the Image

Figure 12. Diagram of Method B Procedure

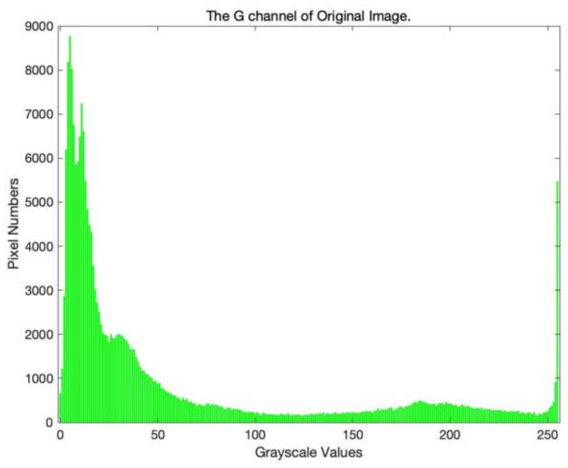
1.3.3 Experimental Result

The original image *Toy.raw* and its histograms of each channel is shown in Figure 13 below. The transfer function of each channel of method A is shown in Figure 14 below. The equalized by method A image *Toy_equalizedA.raw* and its histograms of each channel is shown in Figure 15 below. The equalized by method B image *Toy_equalizedB.raw* and its histograms of each channel is shown in Figure 16 below.

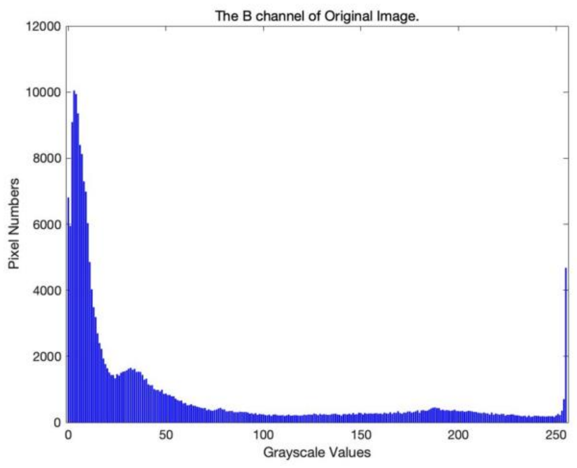

(a)



(b)



(c)



(d)

Figure 13. (a) Toy.raw (b) R channel's Histogram of Toy.raw (c) G channel's Histogram of Toy.raw (d) B channel's Histogram of Toy.raw

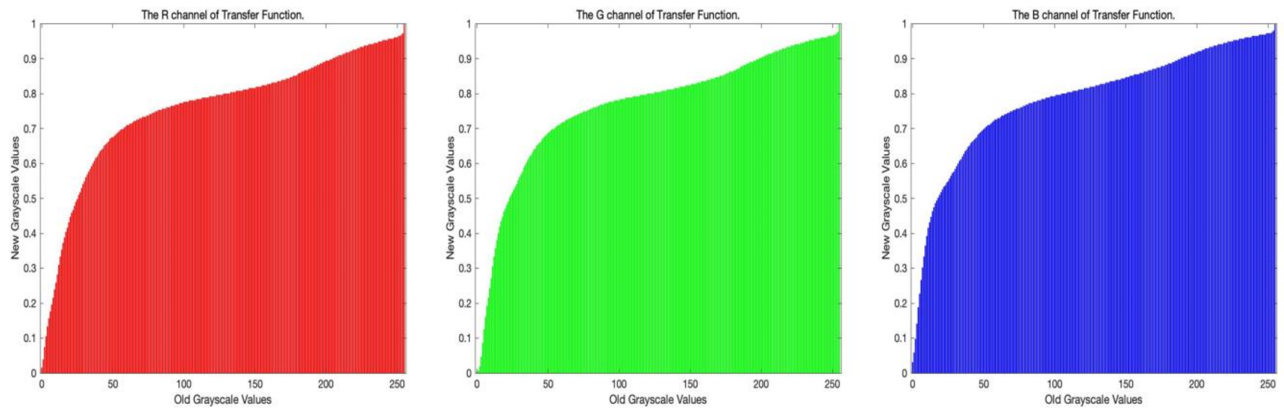
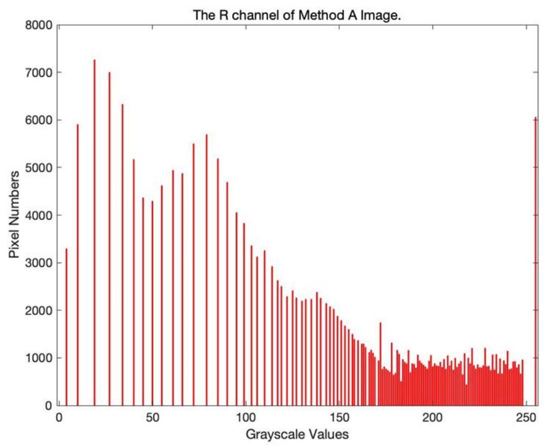
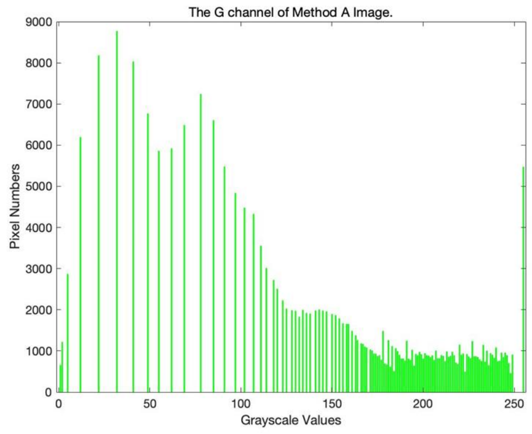


Figure 14. RGB channels' Transfer Functions

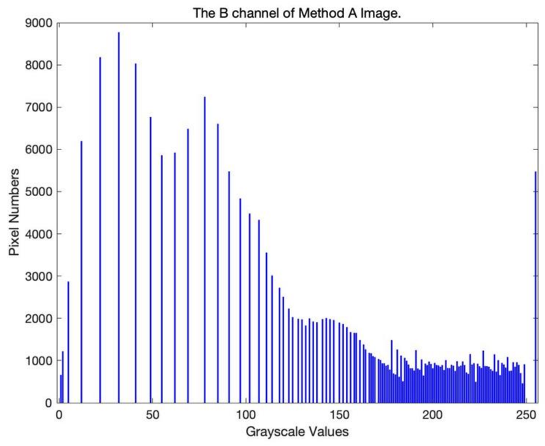
(a)



(b)



(c)



(d)

Figure 15. (a) Toy_equalizedA.raw (b) R channel's Histogram of Toy_equalizedA.raw (c) G channel's Histogram of Toy_equalizedA.raw (d) B channel's Histogram of Toy_equalizedA.raw

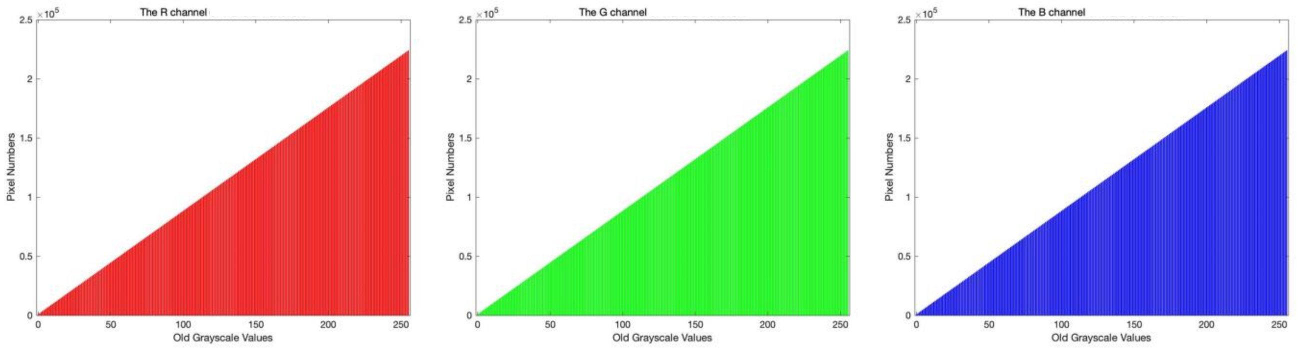
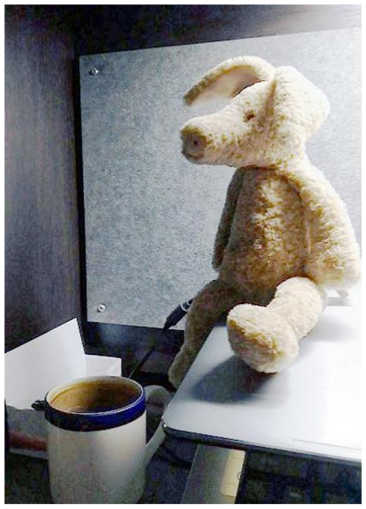
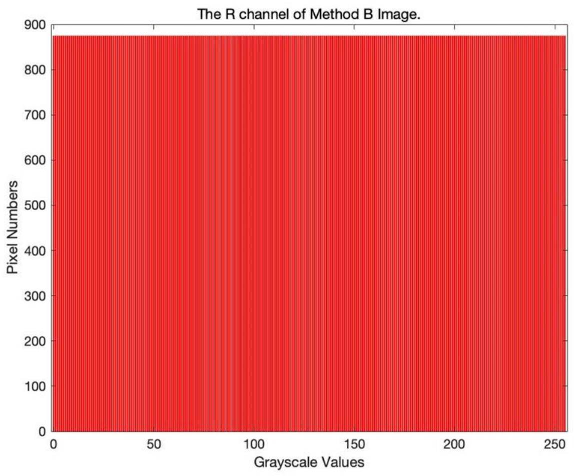


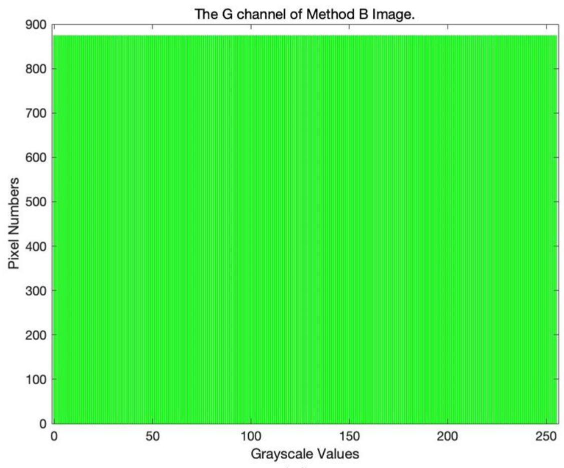
Figure 16. RGB channels' Cumulative Histograms



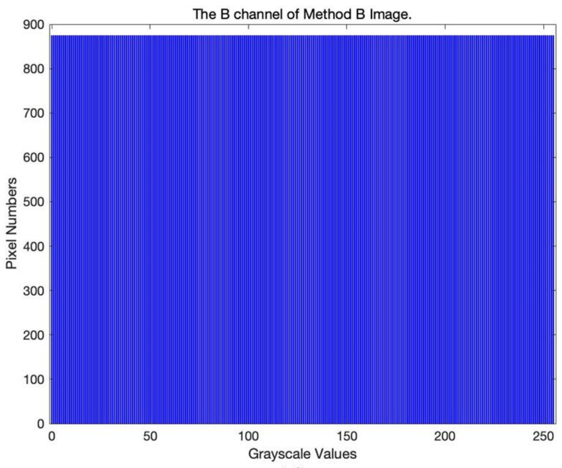
(a)



(b)



(c)



(d)

Figure 17. (a) Toy_equalizedB.raw (b) R channel's Histogram of Toy_equalizedB.raw (c) G channel's Histogram of Toy_equalizedB.raw (d) B channel's Histogram of Toy_equalizedB.raw

1.3.4 Discussion

From Figure 15 and Figure 16, we can see that the result image from either method is quite similar. Both methods did a great job enhancing image by removing the darkness in the left half image. The box and trash can in the left corner are shown much more clearly than original image in both result images as shown in Figure 17. But also, we can tell that there are some fake colors constructed a texture on the trash can.

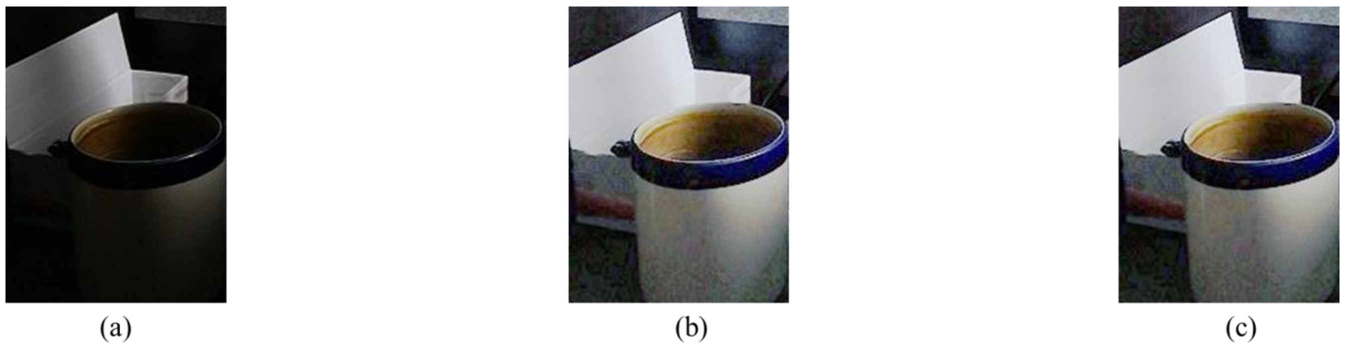


Figure 18. (a) Toy.raw (b) Toy_equalizedA.raw (c) Toy_equalizedB.raw

From Figure 18, (a)'s edge is smoother than (b) and (c)'s. And the right side of two result images are almost too bright to clearify the edges of the toy. **The edge degredation and noise generaion might come from the loss of some grayscal values. Because from Figure 15, we see that there are many gray values that doesn't possess a sigle pixel, which might cause the neighboring pixels change more drasticly to become noisy pixels and generate unknown textures.** Also, some of the relatively white pixels are strengthened to be too bright to ensure the increase of whole contrast.

To improve method A's performance, we could use some post processing techniques such as denoising and linear interpolation. These means can generate more pixel values into the image, which might get lost in the enhancement procedure. To method B, every grayscale values possess some pixels, so the algorithm preserves the range of the pixel values. But we can tell from Figure 16, the 'bucket filling' algorithm discarded the distribution feture of the image. **It might be useful to use a more sophisticated strategy rather just average the number of each value to improve the method B's**

performance. Like we for the values that possess to much pixels we can pour out certain number of pixels from the 'bucket' and use these pixels to fill in other 'buckets' that relatively small. In this case, we can balance the pixel numbers with each value but still preserve some of the distribution feature.

About the back ground, we can see aparently more unknwon textures are generated, this should be caused by the false pixel values from certain channel. To reduce this pattern and make the equlaized picture less noisy, we can **apply some filters like uniform filter or Gaussian filter to smooth the changed pixel values. Also, we can use an more accurate color space, like a 512-scale-palette to separate colors more detailed and reduce some false pixels**.

2. Problem 2

2.1 Basic Denoising Methods

2.1.1 Abstract and Motivation

Noises are widely existed in all kinds of images. How to extract the useful information from noisy image is a persistent interests of researchers. We want to keep details as many as possible to make the denoised image clean and sharp. In this part of the discussion and experiment, I used two different filters to denoise the noisy image, compare them with each other and then discuss the performances that we achieved.

There are different kinds of noise such as Pepper and salt noise, Uniform noise, Guassian noise and Thermal noise, etc. In this problem, we maily focused on the denoise of uniform noise. Two types of filter will be used – Uniform filter and Gaussian filter. Also, the PSNR will be used as a metric to measure the performances of these methods. Figure 19 shows the original image and the noisy image as below.

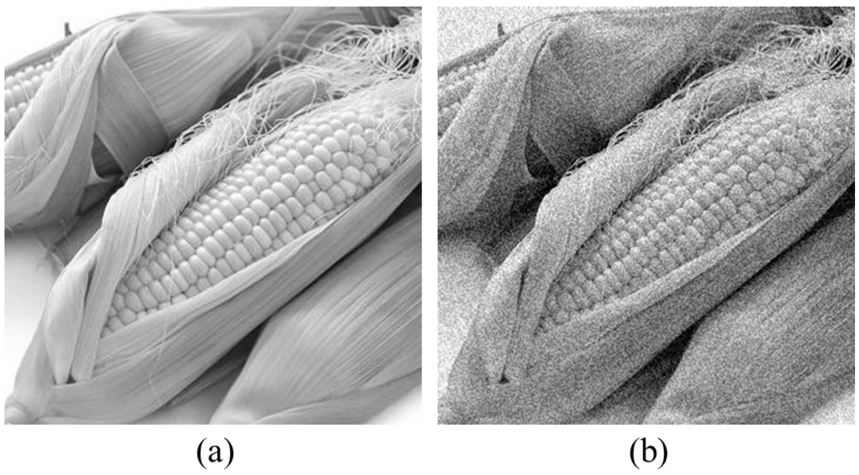

(a) (b)

Figure 19. (a) Corn_gray.raw (b) Corn_noisy.raw

2.1.2 Approach and Procedures

As same as the 1.1 and 1.2 states, to proceed the denoising, we need to extend the image in right way. But don't skip the first raw and colume this time(as Figure 4(b) shows). Also, we need to construct the proper filter arrays and value arrays to do the convolutional computation. The detailed proceedure is shown as Figure 20.

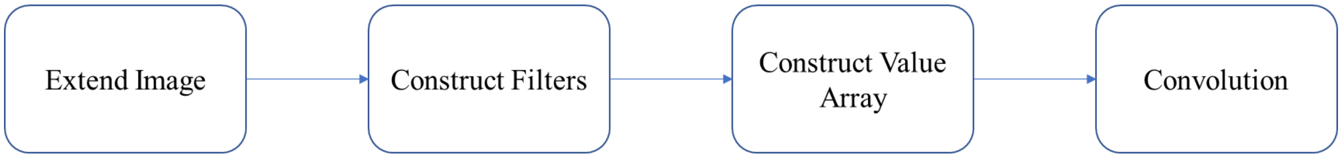
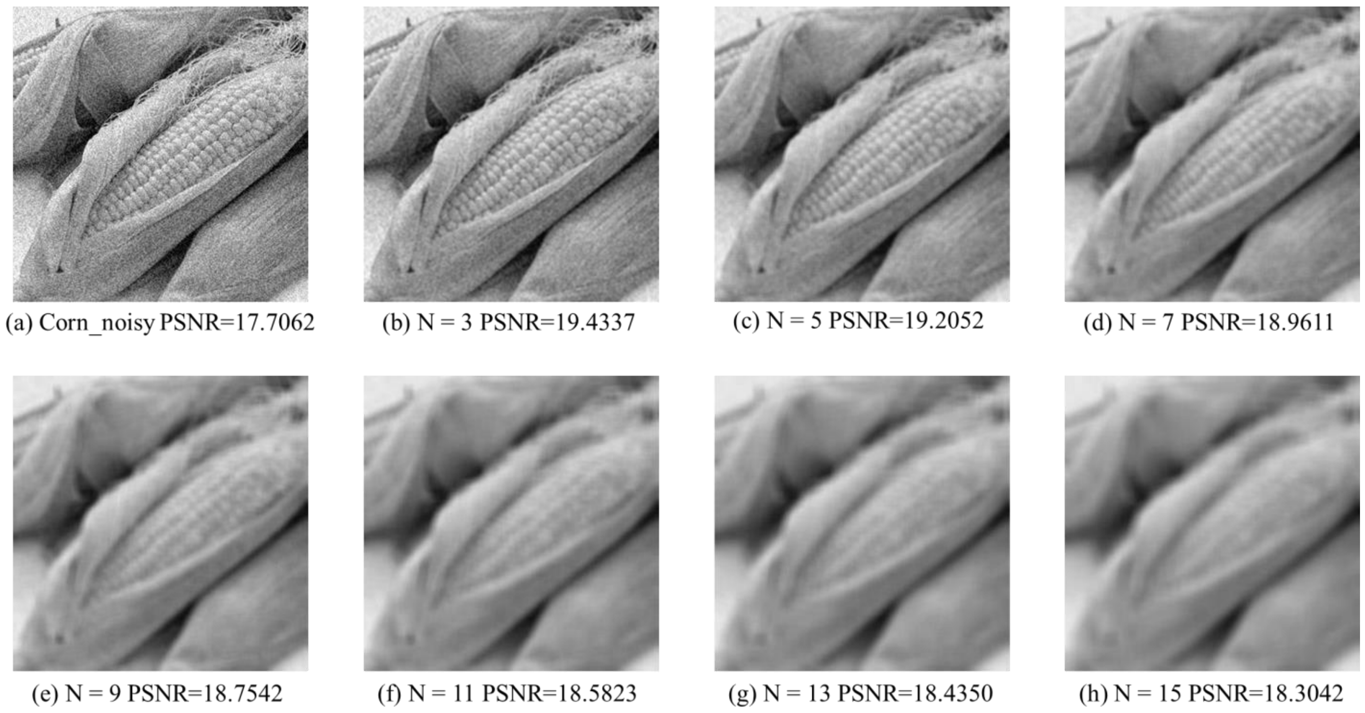


Figure 20. The procedure of Uniform and Gaussian Denoising

2.1.3 Experimental Result

The result images after uniform denoising is shown in Figure 21. The result images after Gaussian denoising is shown in Figure 22.

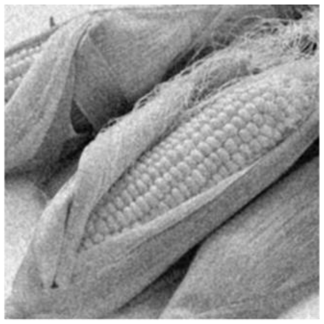


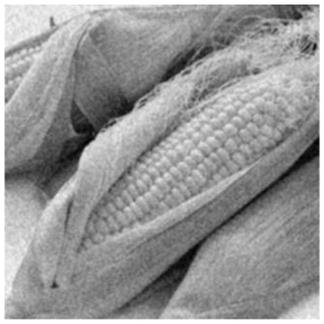
(a) Corn_noisy PSNR=17.7062 (b) N = 3 PSNR=19.4337 (c) N = 5 PSNR=19.2052 (d) N = 7 PSNR=18.9611

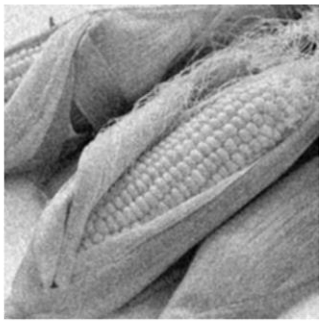
(e) N = 9 PSNR=18.7542 (f) N = 11 PSNR=18.5823 (g) N = 13 PSNR=18.4350 (h) N = 15 PSNR=18.3042

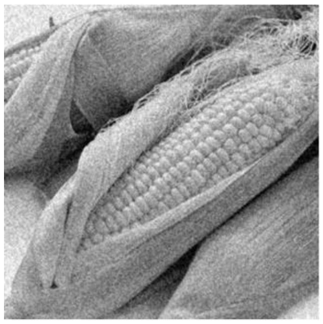
Figure 21. Result Images After Different Window Size of Uniform Filtering

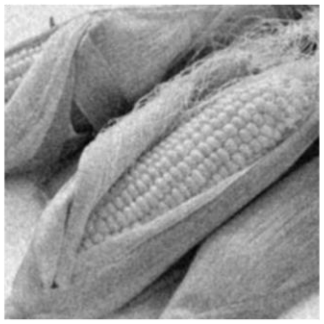
(a) PSNR=19.0305
σ = 0.5
N = 3

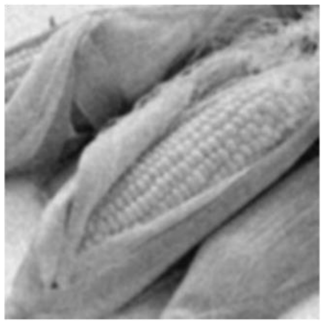

(b) PSNR=19.5262
σ = 1.0
N = 3

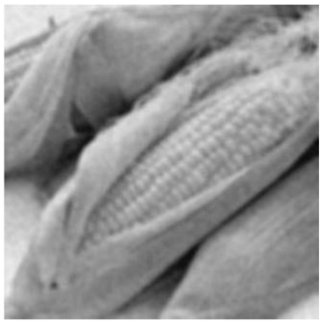

(c) PSNR=19.4630
σ = 2.0
N = 3


(d) PSNR=19.4467
σ = 3.0
N = 3

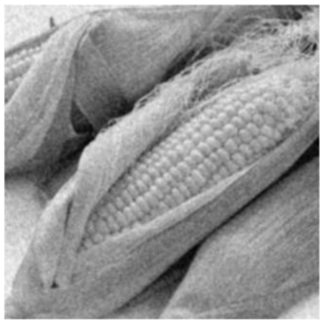

(e) PSNR=19.0323
σ = 0.5
N = 7

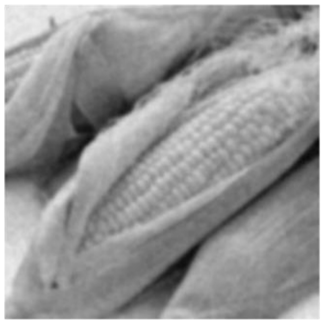

(f) PSNR=19.5271
σ = 1.0
N = 7


(g) PSNR=19.2009
σ = 2.0
N = 7


(h) PSNR=19.0743
σ = 0.5
N = 7

(i) PSNR=19.0323
σ = 0.5
N = 11


(j) PSNR=19.5269
σ = 1.0
N = 11


(k) PSNR=19.1224
σ = 2.0
N = 11

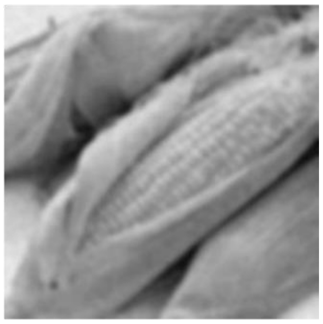

(l) PSNR=18.8662
σ = 3.0
N = 11

Figure 22. Result Images Gaussian Filtering N is window size

2.1.4 Discussion

The noise of Corn_noisy.raw is uniform noises because when I subtract the original image from the noisy image then count the pixels valuse of the matrix. I get the noise distribution has shrap cut-offs and has simillar frequencies, as shown in Figure 23.

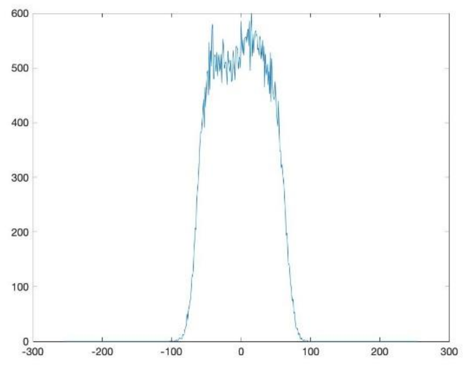
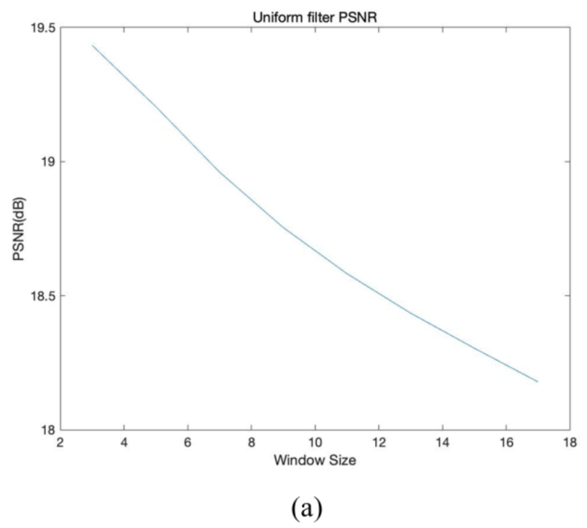


Figure 23. Distribution of Noises

For the uniform filter, we can see from Figure 21 that along the window size increases, the image losses more details and gets more blurry. The PSNR value is still better than the noisy image, but as long as the window size increases PSNR decrease (as shown in Figure 24 (a)). This artifact is caused by the uniform kernel (as shown in Figure 24 (b), N = 3), which just averaging all pixels' values in the windw to smooth the pixels changes to denoise. As we all know, the edges and details in the image may also contain big changes and high frequencies that might be filtered by the kernel.



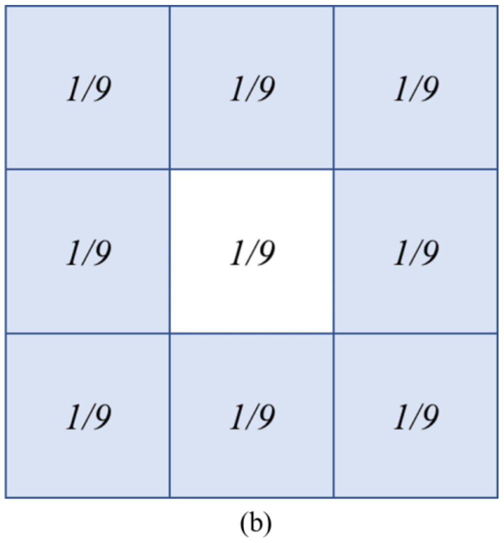


Figure 24. (a) Uniform Filter PSNR related with N

(b) The uniform Kernel

For the Gaussian filter, we can see that PSNRs are higher than uniform filter with the same window size, and more detials and edges are shown clearly. Also, we can find that the max PSNR value appears at the condition of σ = 1, N = 3, and as long as we increase the sigma value of window size, the PSNR decreases. And when the window size is too big, the result images becomes simillar to the uniform filter ones.

The better performance of Gaussian filter comes from the Gaussian Kernel, which gives different weights to each pixel in the filtering window based on the sigma. By this way, every pixel can possess some frequencies of its own and so preserve some detailed features in whole image. **But from Figure 22, we can tell that when the window size is too big the PSNR starts to fall. That is because when the window is too big, the center weight get smaller, which means the current pixel can possess less values of its own but more values from others. So, the detail starts getting lost.** The relations of PSNR with window size and sigma are shown in Figure 25.

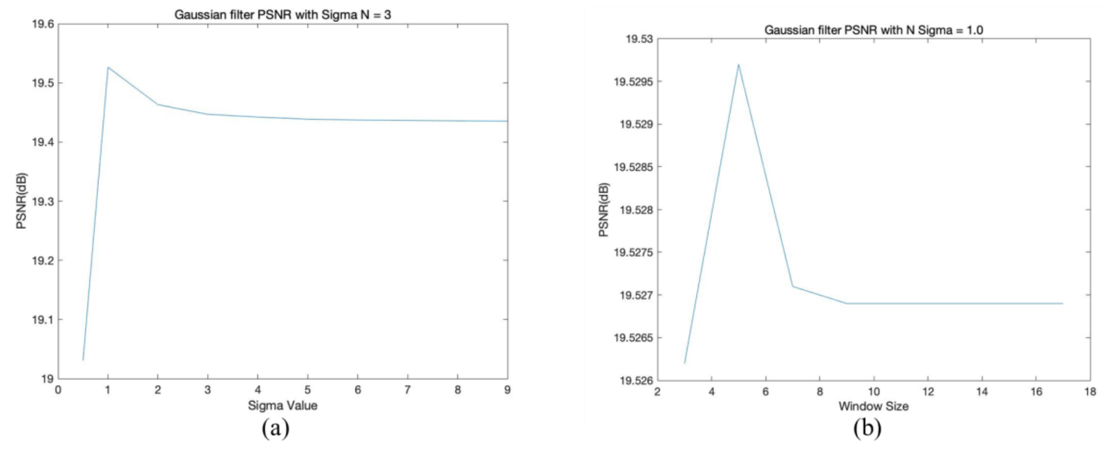


Figure 25. (a) Gaussian Filter PSNR related with Sigma

(b) Gaussian Filter PSNR related with N

2.2 Bilateral Filtering

2.2.1 Abstract and Motivation

The linear low-pass filter like uniform filter and Gaussian filter works and easy to inplement, but the filtered image usually has some artifacts like edge degradation, detail loss and so on. As Gaussian filter and uniform filter did above, we can see the filtered image still containing some noise or getting blurry after the process. These artifacts are the main problem I am trying to solve in this part.

While some nonlinear filters like bilateral filter can remove the noises as well as preserve the details and edges clearly. In this part, the bilateral filter is inplemented and discussed. PSNR is also used as a metric to measure the performance of the algorithm.

2.2.2 Approach and Procedures

The bilateral filter can be constructed by equation (5 - 6). The basic steps are shown in Figure 26, and picking parameters is more complex than basic filtering.

$$Y(i,j) = \frac{\sum_{k,l} I(k,l)w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)} \tag{5}$$

$$w(i,j,k,l) = exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_c^2} - \frac{\|I(i,j)-I(k,l)\|^2}{2\sigma_s^2}\right) \tag{6}$$

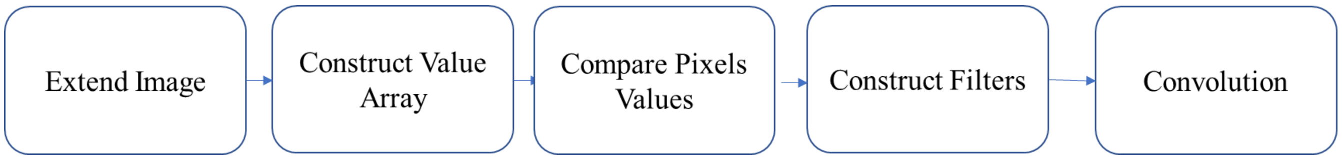
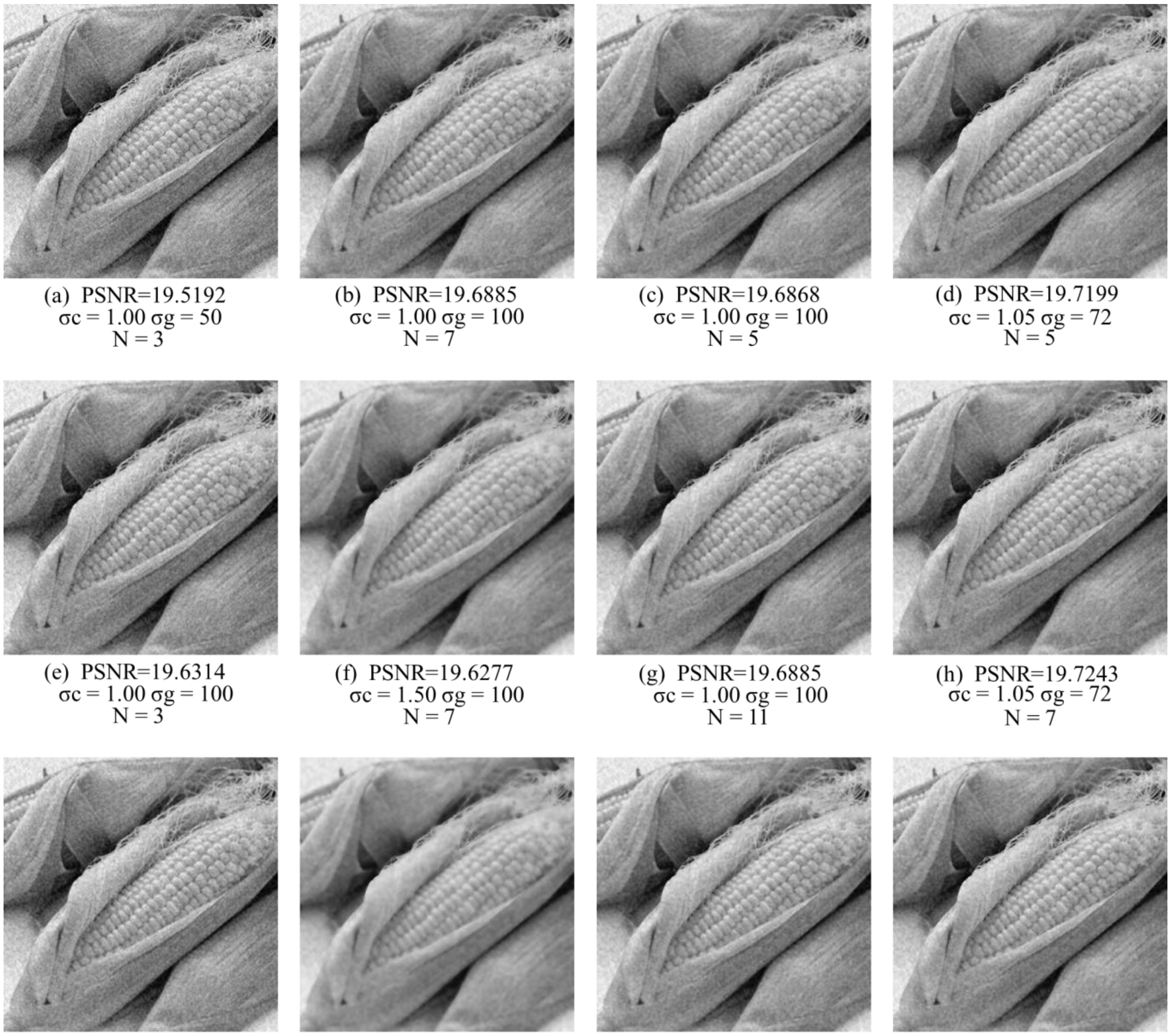


Figure 26. The procedure of Bilateral Denoising

2.2.3 Experimental Result

The result images after bilateral denoising is shown in Figure 27.



(a) PSNR=19.5192 σc = 1.00 σg = 50 N = 3

(b) PSNR=19.6885 σc = 1.00 σg = 100 N = 7

(c) PSNR=19.6868 σc = 1.00 σg = 100 N = 5

(d) PSNR=19.7199 σc = 1.05 σg = 72 N = 5

(e) PSNR=19.6314 σc = 1.00 σg = 100 N = 3

(f) PSNR=19.6277 σc = 1.50 σg = 100 N = 7

(g) PSNR=19.6885 σc = 1.00 σg = 100 N = 11

(h) PSNR=19.7243 σc = 1.05 σg = 72 N = 7

(i) PSNR=19.5938 σc = 1.00 σg = 150 N = 3

(j) PSNR=19.5464 σc = 2.00 σg = 100 N = 7

(k) PSNR=19.6885 σc = 1.00 σg = 100 N = 17

(l) PSNR=19.7244 σc = 1.05 σg = 72 N = 9

Figure 27. Result Images Bilateral Filtering N is window size

2.2.4 Discussion

Parameter Tunning: Bilateral filtering is more advanced than Gaussian filtering or unifrom filtering because of it takes both spatial distances and pixel similarities into account. In Gaussian filtering, the only parameter that affects the weights of certain pixel is σ . **But in bilateral filtering, the weights of certain pixels are influenced by the pixel distances and similarities. There are two parameters that we can control the spatial weight by Sigmac and control the range weight by Sigmas. In other words, in the region that pxiel values doesn't change much, the *w(i, j, k, l)* function is more simillar to the Gaussian filtering. But in the edge region or detailed region where pixel values change drasticly, the *w(i, j, k, l)* function has more weight in the values weight, so that the edge's information can be preserved as possible. For Paramter tunning, we can see that there is a balance between sigmac and sigmas, which means we should balance the weights between spatial part and similarity part. Only when two parameters tunned properly, the PSNR value peaks.**

In the Figure 27, we can tell from the last column that the best values of Sigmac and Sigmas are 1.05 and 72, which achieves highest PSNR value. Also, From the first three columns we can see that there are thresholds for both two parameters. Below this threshold, when we increase the value of the Sigmac and Sigmas, the performance gets better. But from Figure 27(j), we can see that the image starts to get blurry if the Sigmac is too big. This is because when Sigamc gets bigger, the weight of spatial features are larger, the filter is more like a Gaussian filter which is not good at preserving the details and edges. Also, if the range weight gets too large, it will be hard for algorithm to denoise efficiently.

Comparison: In general, the bilateral filtering's performance is better than uniform or Gaussian, from Figure 21, Figure 22 and Figure 27 we can see the PSNR valuse of bilateral filter is generally higher **(higest PSNR = 19.7244)**. But of course, it is harder to compute and implement. **About the complexity, the uniform filter and Gaussian filter has the time complexity with O(n), where n is total number of pixels. While the complexity of bilateral filter with brute force is O(n^2), but with the help of local histgram and seperable kernel, we can lower the time complexity to O($\mathrm{n}log^{sigmas}$) and O(n*sigmas)[5], which are comparable with uniform and Gaussian filter.**

2.3 Non-Local Means (NLM) Filtering

2.3.1 Abstract and Motivation

Although the bilateral filter has taken the pixel distance as well as value changes into account. The pixels similarity is defined only by their values' difference. While an image can contain multiple similar or same regions or textures like the corn kernels in Corn_gray.raw. To make these details, edges and textures to be preserved more clearly, Non-Local Means filter uses regions differences to represent the differences between pixels.

In this part, NLM filter is implemented and the result images are discussed. Also, the PSNR value is used for measure the performances of the algorithm with different parameters.

2.3.2 Approach and Procedures

The NLM filter can be constructed by equation (7 - 10). The basic steps are shown in Figure 28, and picking parameters is more complex than basic filtering.

$$Y(i,j) = \frac{\sum_{k=1}^{N'} \sum_{l=1}^{M'} I(k,l) f(i,j,k,l)}{\sum_{k=1}^{N'} \sum_{l=1}^{M'} f(i,j,k,l)} \tag{7}$$

$$f(i,j,k,l) = \exp\left(-\frac{\left\| I(N_{i,j}) - I(N_{k,l}) \right\|_{2,a}^{2}}{h^2}\right) \tag{8}$$

$$\|I(Ni,j)-I(Nk,l)\|2,a2=\Sigma Ga(n1,n2)(I(i-n1,j-n2)-I(k-n1,l-n2))n1,n2\in\aleph 2 \tag{9}$$

$$G_a(n_1,n_2) = \frac{1}{\sqrt{2\pi a}} \exp\left(-\frac{n_1^2+n_2^2}{2a^2}\right) \tag{10}$$

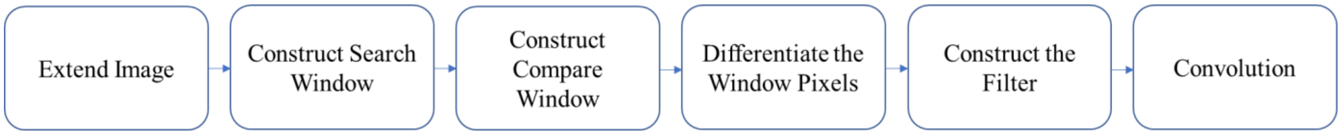


Figure 28. The procedure of NLM Denoising

2.3.3 Experimental Result

The result images after NLM denoising is shown in Figure 29. N is comparing window size, M is searching window size.

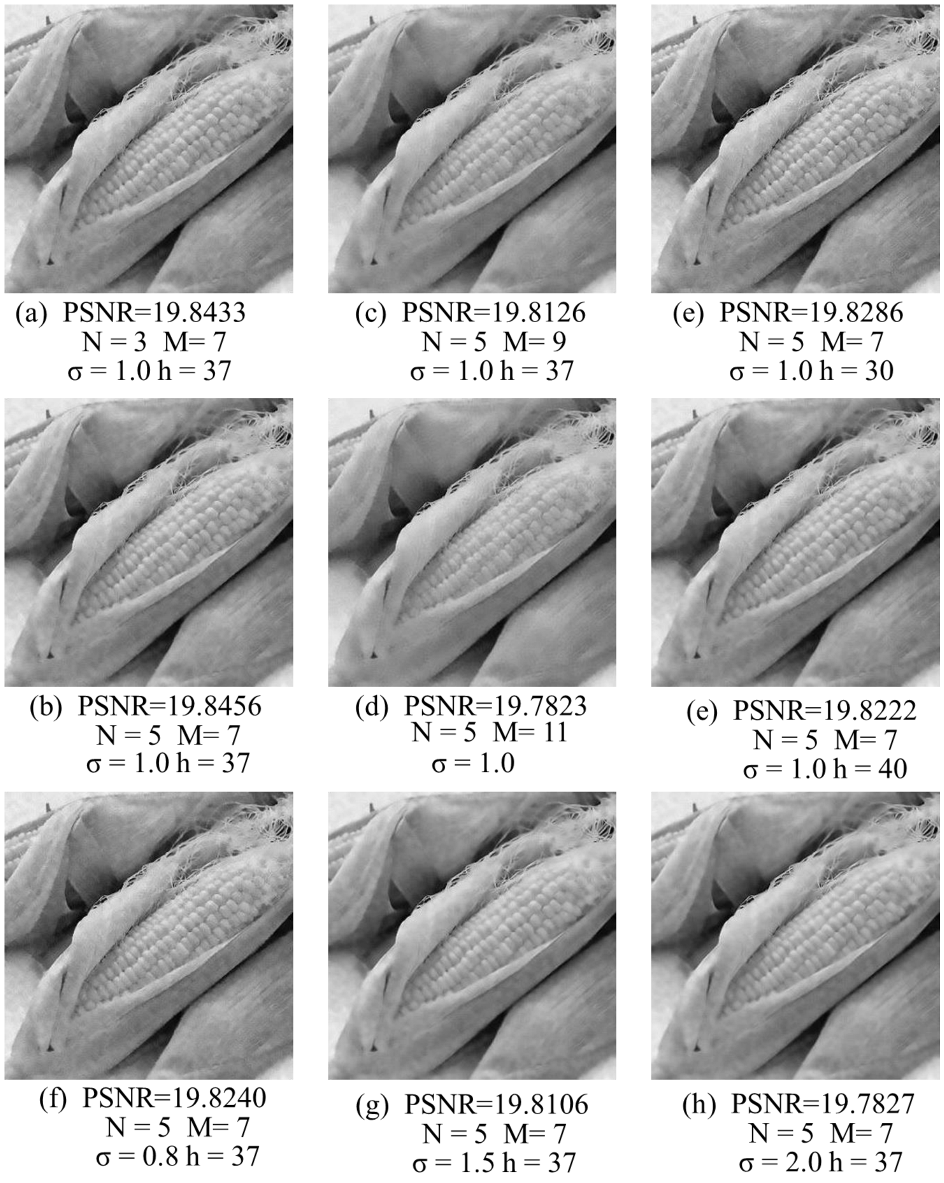

(a) PSNR=19.8433 N = 3 M= 7 σ = 1.0 h = 37
(c) PSNR=19.8126 N = 5 M= 9 σ = 1.0 h = 37
(e) PSNR=19.8286 N = 5 M= 7 σ = 1.0 h = 30
(b) PSNR=19.8456 N = 5 M= 7 σ = 1.0 h = 37
(d) PSNR=19.7823 N = 5 M= 11 σ = 1.0
(e) PSNR=19.8222 N = 5 M= 7 σ = 1.0 h = 40
(f) PSNR=19.8240 N = 5 M= 7 σ = 0.8 h = 37
(g) PSNR=19.8106 N = 5 M= 7 σ = 1.5 h = 37
(h) PSNR=19.7827 N = 5 M= 7 σ = 2.0 h = 37

Figure 29. Result Images NLM Filtering

2.3.4 Discussion

As experimented repeatedly, I find the parameters with N = 5, M = 7, h = 37 and sigma = 1.0 can produce the best performance with PSNR = 19.8456.

The idea of NLM is simillar with bilateral filter, focusing on the spatial weight and simialarity weight. But NLM algorithm has a more scientific scheme of calculating the similatity of pixels using regions rather than sigle pixels. This provide a more robust evaluation to the relations of pixels. And it did provide a better performace as we can see the lowest PSNR in Figure 29 is higher than the highest PSNR in Figure 27, Figure 21 and Figure 22. **While we can still see from Figure 29 (d) that the image can get blurry if the search window size is too big which means the spatial weight is too large.**

Comparison: In addition, even though the NLM algorithm has much better performance than previous methods, the complexity of NLM is larger which means it is harder to implement and compute with O(n^2). There are some schemes are proposed to accelerate the speed of processing like skip pixels and use FFT in transform domain.

In the end of this part, I compared all the mathods' best performance that implemented. The result is as shown in Table 3.

Table 3. Comparison of All Methods Above

Method	Parameters	PSNR(dB)
Uniform	N = 3	19.4337
Gaussian	N = 7, σ = 1.0	19.5271
Bilateral	N=9, σc = 1.05, σs = 72	19.7244
Non-Local Means	N=5, M=7, h=37, σ =1.0	19.8456

2.4 Block matching and 3-D (BM3D) transform filter

2.4.1 Abstract and Motivation

Although the NLM algorithm has achieved great performance with PSNR = 19.8456dB, it still has some space for improvement. Because it only uses the spatial information but discard the transform information. Based on the NLM idea and use the transform domain filtering, it might be better for denoising, as the BM3D method presented below.

BM3D is one of the most advanced denoising technology. In this part, an BM3D algorithm is implemented, the result images are presented and discussed, and the steps of algorithm are explained. Also, the PSNR value is used for measure the performances of the algorithm with different parameters.

2.4.2 Approach and Procedures

There are two main steps in BM3D algorithm – basic estimation and final reconstruction. The flow chart is shown as below in Figure 30[2].

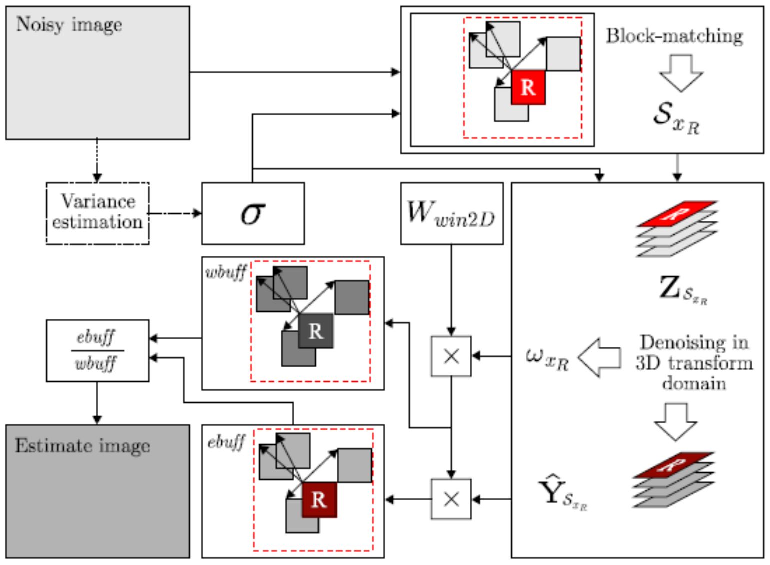
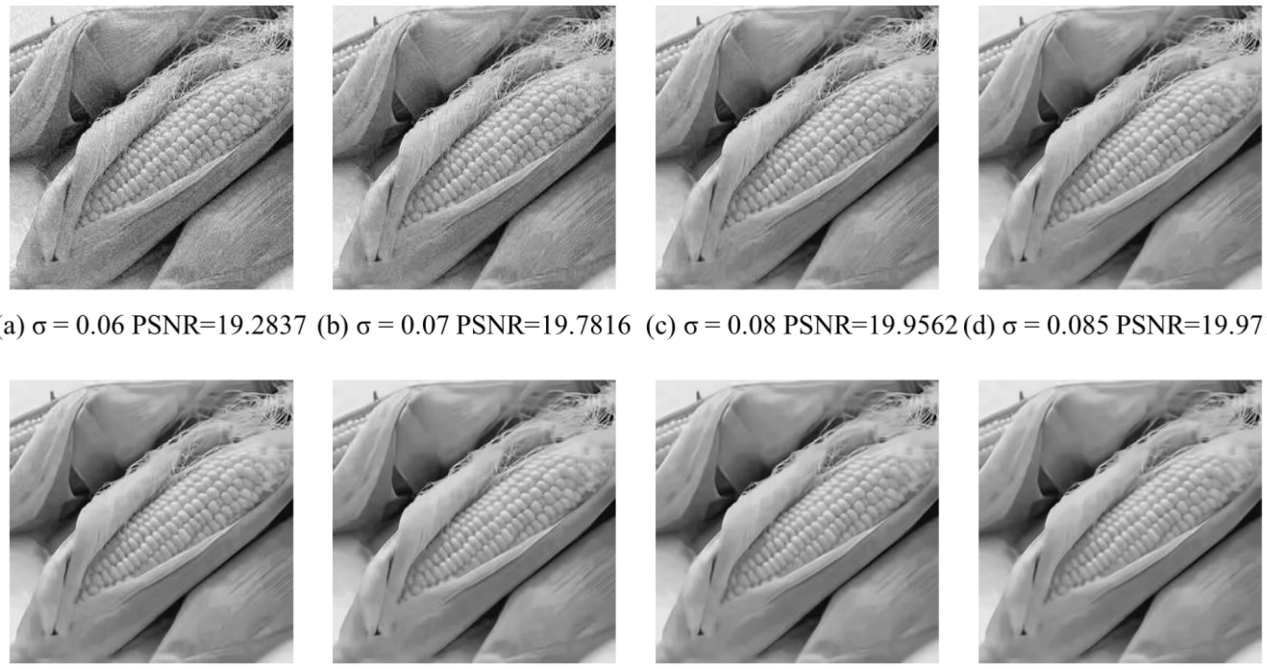


Figure 30.Flow chart of BM3D Procedure

2.4.3 Experimental Result

The result images after BM3D denoising is shown in Figure 31.



(a) σ = 0.06 PSNR=19.2837 (b) σ = 0.07 PSNR=19.7816 (c) σ = 0.08 PSNR=19.9562 (d) σ = 0.085 PSNR=19.9718

(e) σ = 0.10 PSNR=19.9259 (f) σ = 0.11 PSNR=19.8757 (g) σ = 0.12 PSNR=19.8253 (h) σ = 0.13 PSNR=19.7764

Figure 31. Result Images after BM3D Denoising

2.4.4 Discussion

The BM3D method gets the highest PSNR value with 19.9718. Which means it extracts more information than NLM because its spatial processing procedure gets more information from the transform domain.

Explanation for BM3D: BM3D's idea combines the idea of NLM and wavelet transform. It also uses one pixel to find simillar regions. But instead of processing them directly like NLM does in spatial domain. BM3D pile them in a stock and transform them into transform domain and do the denoising job then.

There are two steps in BM3D procedure. In first step, the algorithm searches for simillar regions in the image, and pile these image like a stock in a 3-D matrix. This sub-step is called ‘Grouping’. Then use DWT to transform the whole stack into another domain. In transform domain, it sets a threshold to do the denoising job by setting every value below this threshold to zero. This sub-step is called ‘Collaborative Filtering’. After the filtering in the transform domain, it transforms the regions back into 2-D spatial domain image. To these 2-D image, it averages them together to get the basic estimate regional image. It can be either uniformly averaging or weighted averaging, although weighted averaging might perform better. To the end of ‘Aggregation’, the basic estimated image is generated, which will be used in the next step.

In the second step, the ‘Grouping’ operation is used in both noisy image and basic estimated image. Also, these stacks from both image are going to be transformed into another domain. The stack of the regions from basic estimated image is used as a pilot signal to denoise the noisy stack signal. Both stacks are going through a Wiener-filter for denoising in the transform domain. After the second time filtering, it transforms the stacked regions back into spatial domain for ‘Aggregation’. Same as step1, the aggregation could be weighted or uniformly average.

After twice of ‘Grouping’, ‘Collabrative Filtering’ and ‘Aggregation’, the denoised image is generated.

2.5 Mixed noises in color image

2.5.1 Type of the noises

From the image, we can clearly state that there are many balck pixels distributed uniformly in the whole map. So, the impulse noise is contained. Also, we can see a lot of yellow-like pixels, chich means the range of the noises doesn't vary from 0~255. In conclusion, I think the image may contain impulse noises and uniform noises or Gaussian noises with a small sigma value, which dosen't has large range variaties.

2.5.2 Filter each channel separately

I think whether we can use filters to denoise the colorful noisy image depends on the noise type and features. If the noise pixels' RGB channel are correlated, it would be less efficient and unessesary to filter each channel separately. We have to consider other channel's effects on current channel in this context. For this situation, filter the colorful image directly might be more efficient and productive. On the other hand, if the noise is channel-correlated free like the noises I stated above. Then it might be better to proceed filtering in each channel, because the differnce between pixels might be presented more accuratly in three channels than in only one channel.

In addition, some advanced filters can calculate the differences and distances between two pixels. These filters can either process the noise in color domain or in each channel separately. Such as NLM and Bilateral filters.

2.5.3 Select filters and cascades

Since there are are many impulse noises, a filter that can smooth the impulse pixels should be included. So, a median filter or uniform filter can be used to denoise the impulse noises. Also, the pepper image contains many details and edges. So, a filter that can preserve most details should be included, such as NLM filter or bilateral filter.

In conclusion, the cascaded filters are shown as below in Figure 32.

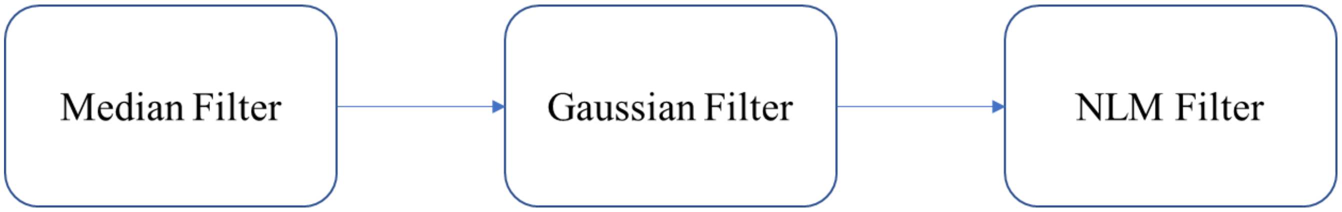


Figure 32. Cascaded Filters

The first median filter can denoise the impulse noises, and the second Gaussian filter with small window size can preserve some details and denoise some Gaussian or uniform nosies. At last, use an NLM filter to denoise the rest of the uniform noises and preserve the details that left.

In addition, we can use the method 'Aggregation' from BM3D, we can apply different filters to the same image and aggregate them together then. To this question, we can use a NLM filter to denoise the Gaussian (Uniform) Noises, and separately use a median filter to denoise the impulse noises. But apparently, the second image would lost many details and the frst image would still contain impulse noises. Then aggregate these two images like BM3D does, might generate a image that preserves details without impuls noises.

References

[1] Kannan E. Paul, Vinsley S. Saraswathibai. Maximum accurate medical image demosaicing using WRGB based Newton Gregory interpolation method, *Measurement*, 2019, 135(3):935-942.

[2] 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.

[3] Malvar, Henrique S., Li-wei He, and Ross Cutler. "High-quality linear interpolation for demosaicing of Bayer-patterned color images." *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 3. IEEE, 2004.

[4]Online resource: http://www.cs.tut.fi/~foi/GCF-BM3D/

[5] Sylvain Paris, Pierre Kornprobst, Jack Tumblin, Fr´edo Durand4 Bilateral Filtering: Theory and Applications. *Foundations and Trends in Computer Graphics and Vision*, Vol. 4, No. 1 (2008) 1–73.