Qidi Han (6814891757)
EE-569
Homework #1
Jan-27-2020

Problem 1: Image Demosaicing

I. Abstract and Motivation

In the recent year, the technology let the word to a better place. From the black and white film to color film, the improvement of image quality gives a better visual effect. In this project, I simulate the digital camera sensors to make a grey scales image to a colorful image. The method called image demosaicing, which using the color filter array. Color filter array (CFA) also called Bayer array that present in a Bayer pattern. In order to make a colorful image, each pixel location should contain red, green and blue color (RGB). Since the sensor only receive on color in each pixel, the remaining two colors will generate by the neighbor pixels according to Bayer pattern. There are two methods to find the remaining two colors in this pixel. The first method called bilinear demosaicing, the second method called Malvar-He-Cutler (MHC) Demosaicing. MATLAB is the only platform that the project will use to generate and modify the image. Finally, I discusses how the methods effect our final result.

II. Approach and Procedures

The first step of the project is reading the image from RAW file to 600 * 532 matrix in MATLAB. There are three different 600*532 matrices that represent red, green and blue (RGB) matrix. Reading the color value from the RAW file to those three matrices according to Bayer pattern. There are many zero in those three matrices. The second step of this experiment is boundary extension, since the edge pixel lost some neighbors to compute the color value.

a. Bilinear demosaicing

The pixel value of Bilinear demosaicing is only according to the neighbors, each three matrices (RGB matrix) need extend only one column and one row to the boundary. Therefore, the matrix will extend to new matrix with 602*534. Placing the original matrix in the center of the new matrix, the original matrix start the point (2,2). Copy the value from the second and second-last row to the first and last row. Copy the value from the second and second-last column to the first and last column. Figure 1 shows the boundary extension for bilinear demosaicing. Then calculate the value start point (2,2) of this new matrix. There are four different type of the calculation for generating each color value from the neighbor pixels. The first two methods use in the pattern of repeating green and red. The last two methods use in the pattern of repeating blue and green. The first type is red and blue color value in the green pixel.

$$R_{(i,j)} = \frac{1}{2}(R_{(i,j-1)} + R_{(i,j+1)}) \quad (1)$$

$$B_{(i,j)} = \frac{1}{2}(B_{(i-1,j)} + B_{(i+1,j)}) \quad (2)$$

The second type is green and blue color value in red pixel

$$G_{(i,j)} = \frac{1}{4}(G_{(i,j-1)} + G_{(i,j+1)} + G_{(i-1,j)} + G_{(i+1,j)}) \quad (3)$$

$$B_{(i,j)} = \frac{1}{4}(B_{(i-1,j-1)} + B_{(i-1,j+1)} + B_{(i+1,j-1)} + B_{(i+1,j+1)}) \quad (4)$$

The third type is green and red value in blue pixel

$$G_{(i,j)} = \frac{1}{4}(G_{(i,j-1)} + G_{(i,j+1)} + G_{(i-1,j)} + G_{(i+1,j)}) \quad (5)$$

$$R_{(i,j)} = \frac{1}{4}(R_{(i-1,j-1)} + R_{(i-1,j+1)} + R_{(i+1,j-1)} + R_{(i+1,j+1)}) \quad (6)$$

The four type is red and blue value in green pixel

$$R_{(i,j)} = \frac{1}{2}(R_{(i-1,j)} + R_{(i+1,j)}) \quad (7)$$

$$B_{(i,j)} = \frac{1}{2}(B_{(i,j-1)} + B_{(i,j+1)}) \quad (8)$$

After calculating each color value in each pixel, the new matrices delete the first row, last row and first column and last column. Combining three 600*532 matrices into a 600*532*3 in order of red, green and blue matrix, because the picture have to follow the order of RGB. The first layer is red, and the last layer is green. Finally, use imshow() function in MATLAB to show the processed image in Figure 4.

b. Malvar-He-Cutler (MHC) Demosaicing

For the MHC demosaicing, it needs a 2nd-order cross channel addition. The original matrix adds two addition rows to the beginning and the last, also the original matrix adds two addition columns to the beginning and the last. Therefore, the boundary extension makes the matrix to 604*536. The matrix does same mirror-reflecting process as bilinear demosaicing. Take an example of this process, the first row copies the value from the fourth row, and the second row copy the value from the third row. Figure 2 shows how MHC demosaicing extend the boundary by using mirror-reflection. Then calculate the value start point (3,3) of this new matrix. The general equation writes as

$$G_{(i,j)} = (G_{(ij)}^{bl} + \alpha\Delta_{R_{(i,j)}}) \quad (9)$$

The first term of $G_{(ij)}^{bl}$ use equation through (1) to (8). The coefficient of second term $\alpha\Delta_{R_{(i,j)}}$have three different types. The component in red pixel had weight coefficient $\alpha = \frac{1}{2}$, the component in green pixel had weight coefficient $\beta = \frac{1}{2}$. The component in blue pixel had weight coefficient $\gamma = \frac{1}{2}$. The term of Δ_R in red pixel had 5 discrete term as

$$\Delta_R(i,j) = 4R(i,j) - (R(i-2,j) + R(i+2,j) + R(i,j-2) + R(i,j+2)) \quad (10)$$

The term of Δ_G in green pixel had 9 discrete term as

$$\Delta_G(i,j) = 5G(i,j) - \big(G(i-1,j-1) + G(i-1,j+1) + G(i+1,j-1) + G(i+1,j+1) + G(i,j-2) + G(i,j+2)\big) + \frac{1}{2}G(i-2,j) + \frac{1}{2}G(i+2,j) \quad (11)$$

There are total eight different type of the calculation for generating each color value from the neighbor pixels. Figure 3 shows the eight different type of the filter.

After calculating each color value in each pixel, the new matrices delete the first two row, last two row and first two column and last two column. Combining three 600*532 matrices into a 600*532*3 in the order of red, green and blue matrix, because the picture have to follow the order of RGB. The first layer is red, and the last layer is green. Finally, use imshow() function in MATLAB to show the processed image in Figure 5.

III. Experimental Results

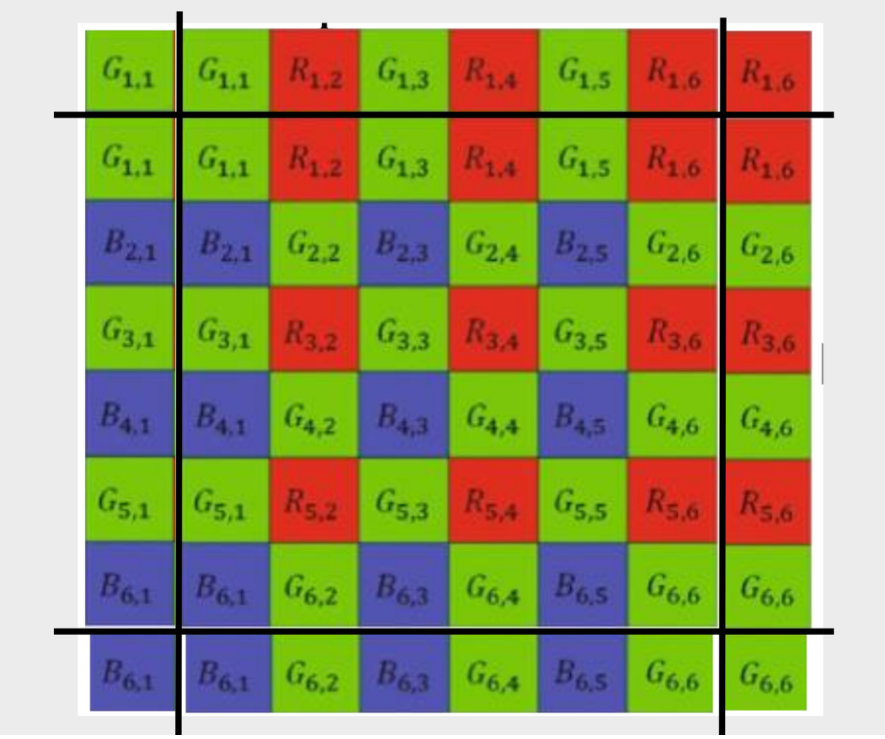


Figure 1: the boundary extension for bilinear demosaicing.

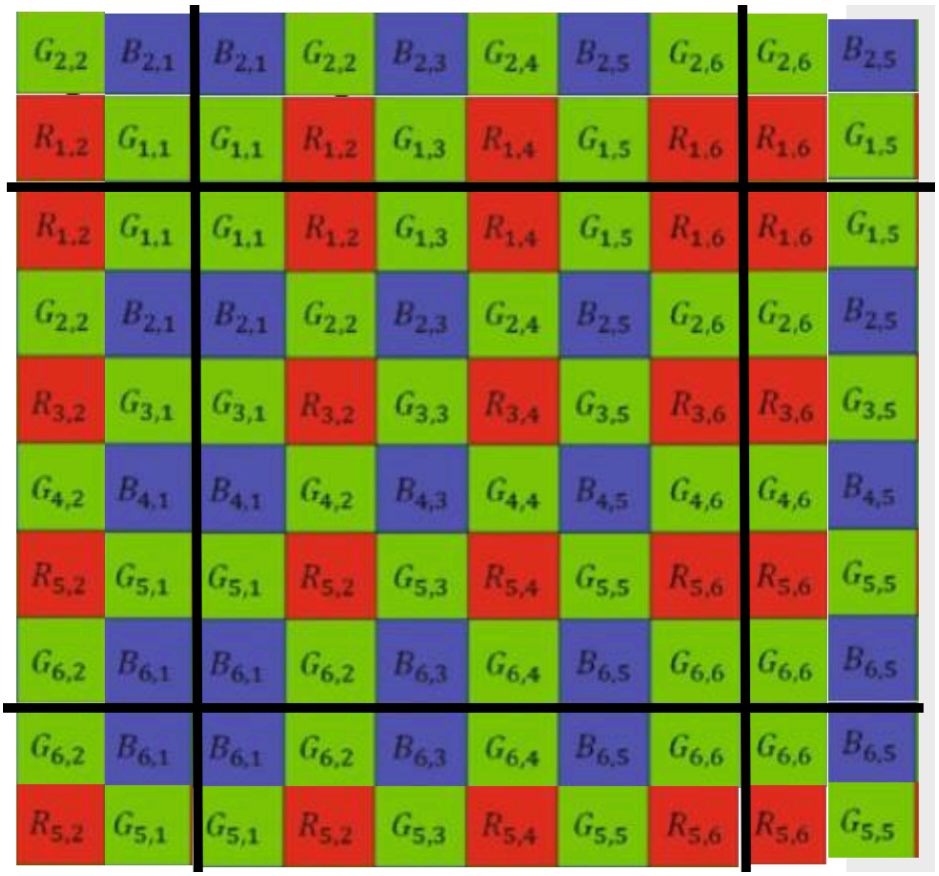


Figure 2: the boundary extension for Malvar-He-Cutler (MHC) Demosaicing.

0 0 -1 0 0 0 0 2 0 0 -1 2 4 2 -1 0 0 2 0 0 0 0 -1 0 0	0 0 -1 0 0 0 0 2 0 0 -1 2 4 2 -1 0 0 2 0 0 0 0 -1 0 0
Green component in red pixel	Green component in blue pixel
0 0 1/2 0 0 0 -1 0 -1 0 -1 4 5 4 -1 0 -1 0 -1 0 0 0 1/2 0 0	0 0 -1 0 0 0 -1 4 -1 0 1/2 0 5 0 1/2 0 -1 4 -1 0 0 0 -1 0 0
Red component in green pixel with repeating red and green row	Red component in green pixel with repeating blue and green row
0 0 -1.5 0 0 0 2 0 2 0 -1.5 0 6 0 -1.5 0 2 0 2 0 0 0 -1.5 0 0	0 0 -1.5 0 0 0 2 0 2 0 -1.5 0 6 0 -1.5 0 2 0 2 0 0 0 -1.5 0 0

Red component in blue pixel with repeating blue and green row					Blue component in red pixel with repeating red and green row				
0	0	1/2	0	0	0	0	-1	0	0
0	-1	0	-1	0	0	-1	4	-1	0
-1	4	5	4	-1	1/2	0	5	0	1/2
0	-1	0	-1	0	0	-1	4	-1	0
0	0	1/2	0	0	0	0	-1	0	0
Blue component in green pixel with repeating blue and green row					Blue component in green pixel with repeating red and green row				

Figure 3: show eight different type of the filter with coefficient 1/8.

Figure 4: Bilinear demosaicing image

Figure 4a: Original image vs Bilinear demosaicing image

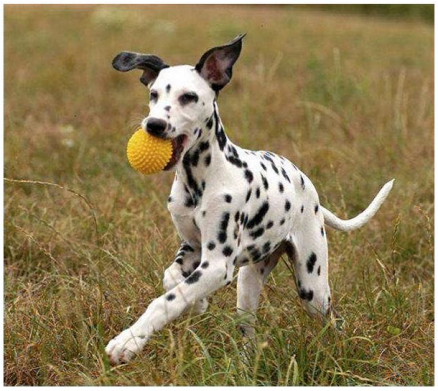


Figure 5: MHC demosaicing image

Figure 5a: Bilinear demosaicing image
Vs MHC demosaicing image

IV. Discussion

Comparing with the original image, the bilinear demosaiced image have some artifacts. The first observation is that the bilinear demosaiced image have some vague outline of each edges of the objects. The second observation is that the contrast of color become weaker. From the Figure 4a, Bilinear demosaiced is not the perfect solution for demosaicing the image. From

own opinion, the reason of both issues is because the pixel value only depends on their neighbor pixel. The color variance is limited by only considering the neighbor pixel. To improve this issue, the best solution is adding more terms to our consideration. For example, the pixel value will not only be adding up the neighbor pixel, also it will add the 2nd neighbor pixel value into the equation. If my observations are correct, I think MHC will be better solution than bilinear demosaicing.
From Figure 5a, I think MHC processing is better solution than bilinear demosaicing. The MHC processed image are fresher and brighter. The image of MHC processed is much distinct on the edge of the objects. The reason of this improvement is that the equation added second term consideration to give a better color value for the pixel. Also, in the class, we discussion the negative and positive term of MHC equation will help us to balance the image value from 0-255 and make the pixel value more advanced. But MHC is still not the most advanced solution. There are still some different between the MHC processed image and the original image.

Problem 2: Histogram Manipulation

I. Abstract and Motivation

When the contrast of image is too high or too low, histogram manipulation is the better solution to make the image normalized. For some pictures, the distribution of all the pixel are not uniform. To improve the quality of image, equalize the distribution of all the pixel give a better image. There are two method that I use in this project. The first method is called the transfer function-based histogram equalization method. The second method is called cumulative probability-based histogram equalization method. Both methods are equalized the distribution of pixels. The MATLAB is the only platform that the project will use to generate and modify the image.

II. Approach and Procedures

The first step is reading the image from the RAW file. Contract the RAW data into 560*400*3 matrix. Use a for loop in the MATLAB to read each value from RAW data to three different matrices, the first data point read into red matrix, the second data point read into green matrix, the third data point read into blue matrix, the fourth data point read into red matrix. Repeated the process to the end of the RAW data. There are 672000 data in the RAW file. Each RGB matrix contain 1x224000 data point. Use the reshape() function in the MATLAB to generate 560*400 matrix. And combine those three 560*400 matrices into 560*400*3. Use the imshow() function to show the original image in the MATLAB. After generating each color matrix. Read each data point from the matrix and use hist() function to plot the distribution of all the pixel value. Figure 6a, 6b, 6c shows the image of histograms of red, green and blue matrix.

a. Method A

To calculate the normalize the histogram, the y-axis data value divide with the total number of pixels. There are 224000 pixels in each matrix. The y-axis data value is the pmf of this

distribution. Then use the new y-axis data value to calculate the CDF. The CDF use the formula of

$$Fx(x) = \sum_{x_k \le x} P_x(x_k) \tag{12}$$

Where $P_x(x_k)$ is the probability of each interval, Fx(x) is the CDF of those value. Then replace the y-axis with new CDF value that generating by equation 11. Now the distribution close to a uniform distribution. The range for the y-axis is from 0-1. Enlarge the y-axis with 255. Use x-axis as the original data point and use y-axis as the new data point. Replace the old data point with new data point by using the mapping rule. And use imshow() to show the image in MATLAB. Figure 7 shows the processed image of transfer-function-based histogram equalization. Figure 7a, 7b, 7c shows the plot of transfer function for each channel.

b. Method B

For the method b, the steps are straight forward. Rearrange the 224000 pixels from smallest value to biggest value. And cut those 224000 pixels with 256 groups. Make it into 875 pixels in each group. Assign each group from 0 to 255. Replace the old data point with new data point by using the mapping rule. And use imshow() to show the image in MATLAB. Figure 8 shows the processed image of the cumulative-probability-based histogram equalization. Figure 8a, 8b, 8c cumulative histogram for each channel.

Experimental Results

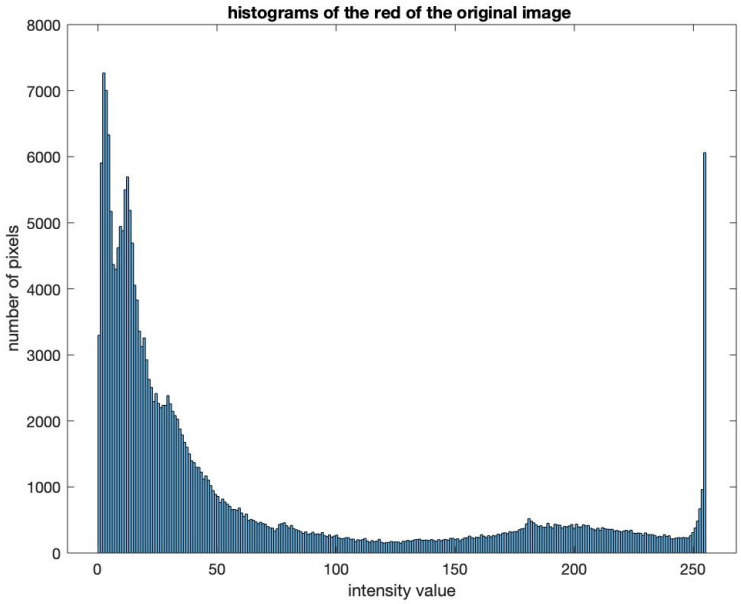


Figure 6a. the image of histogram of red matrix.

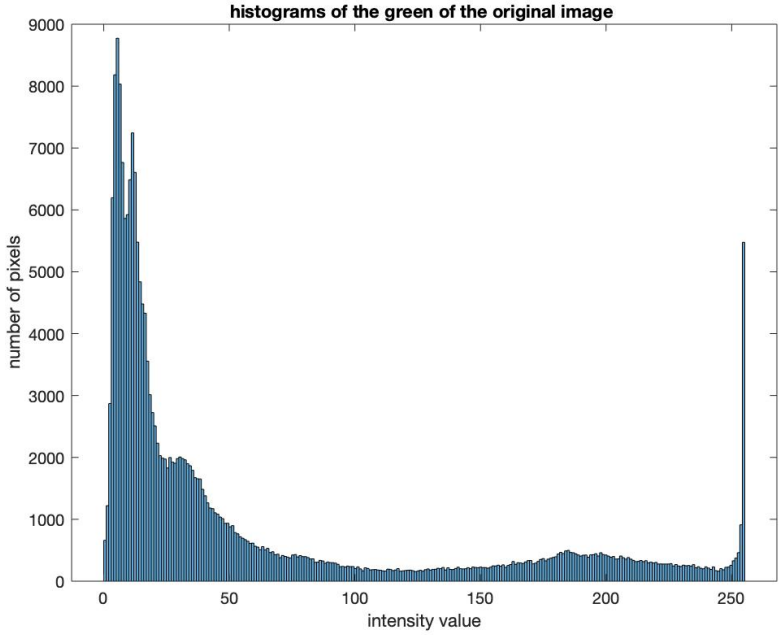


Figure 6b. the image of histogram of green matrix.

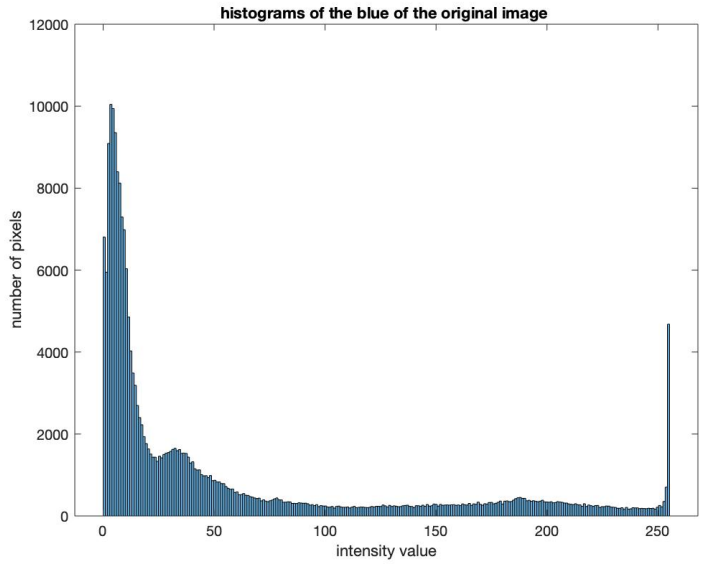


Figure 6c. the image of histogram of blue matrix.

Figure 7. the processed image of transfer-function-based histogram equalization.

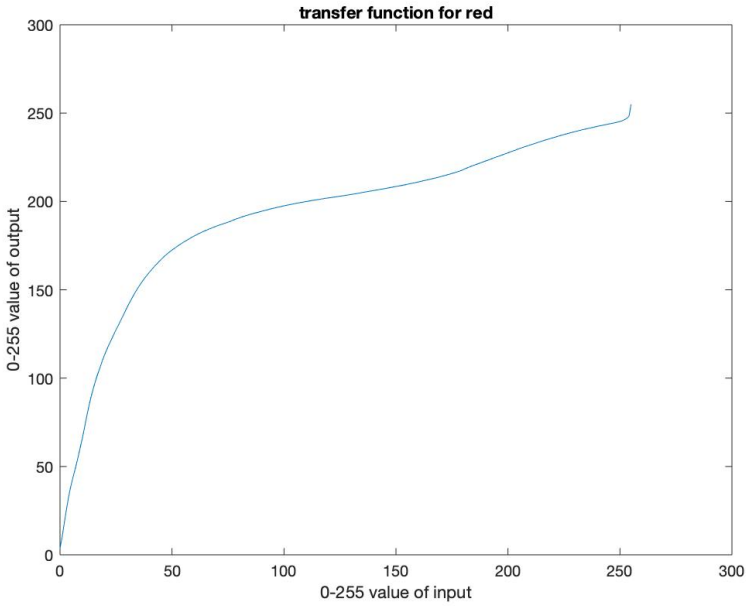


Figure 7a. the plot of transfer function for red channel.

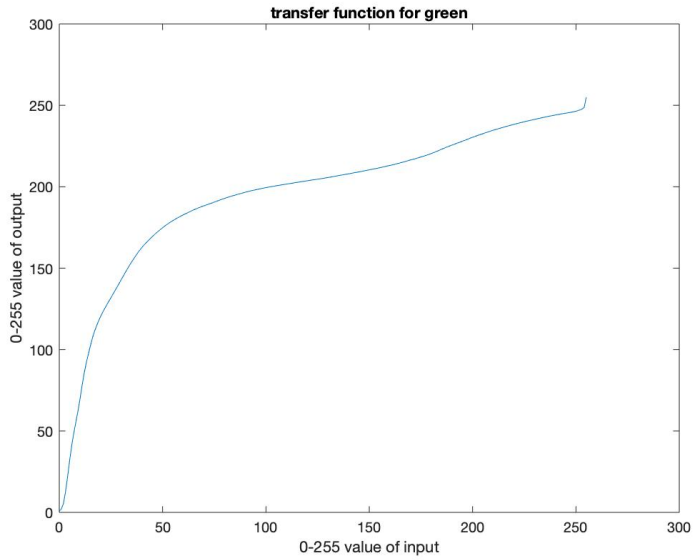


Figure 7a. the plot of transfer function for green channel.

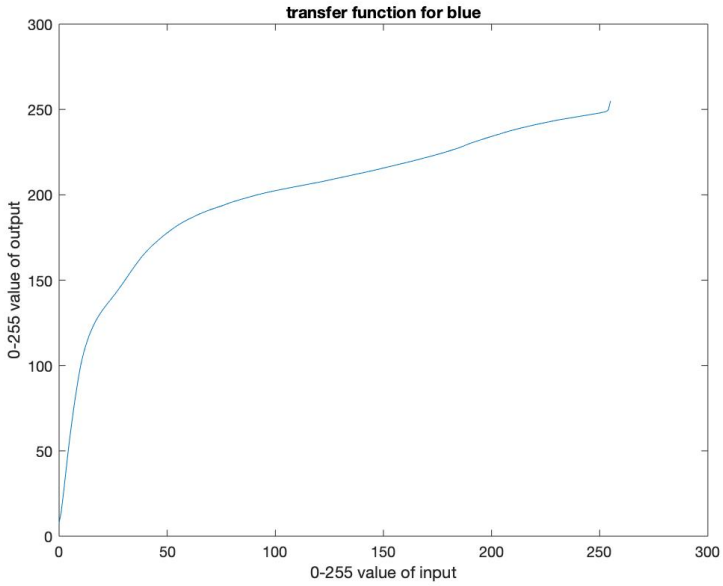


Figure 7a. the plot of transfer function for blue channel.

Figure 8. the processed image of the cumulative-probability-based histogram equalization.

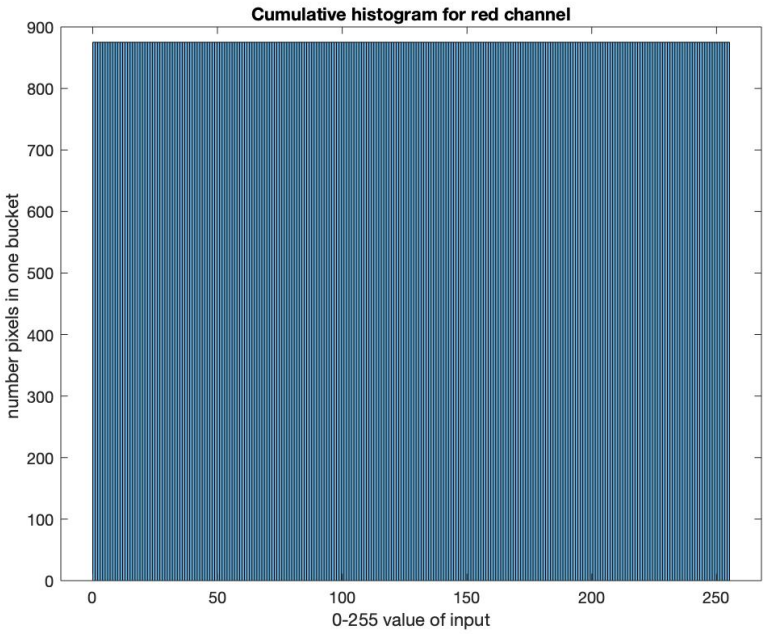


Figure 8a. the plot of cumulative histogram for red channel.

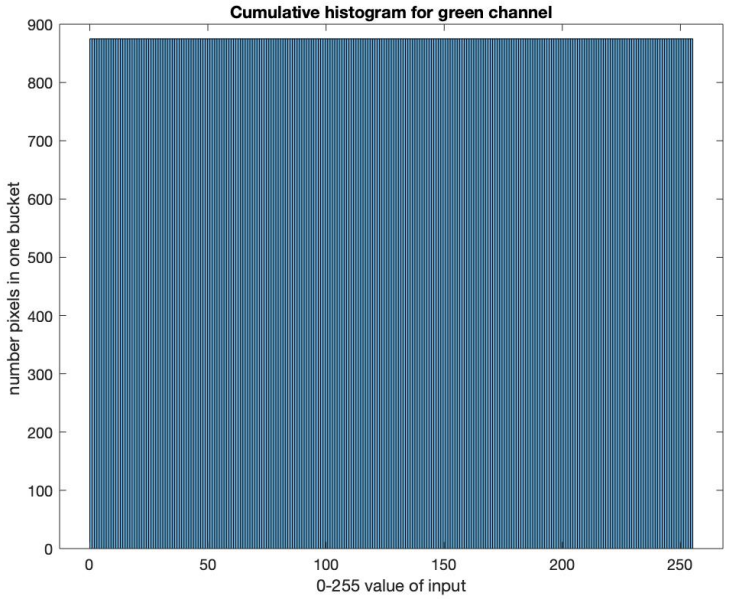


Figure 8b. the plot of cumulative histogram for green channel.

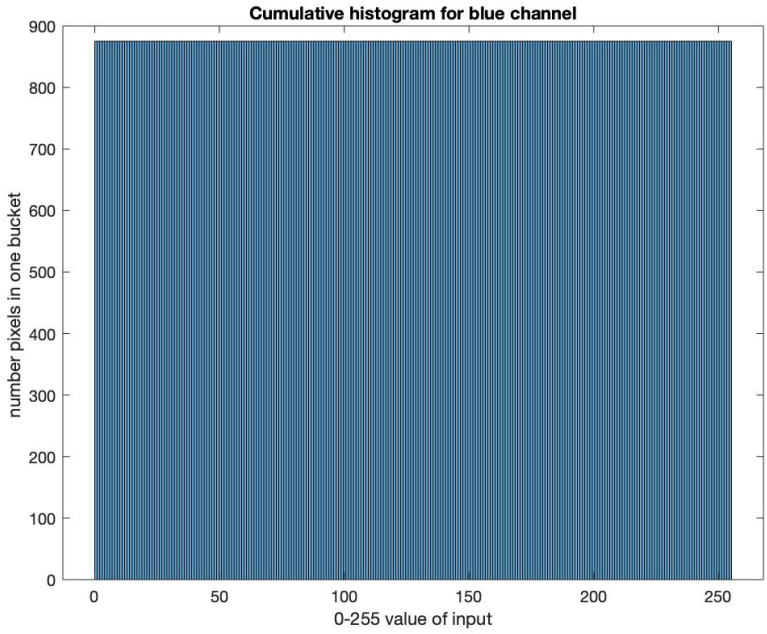


Figure 8c. the plot of cumulative histogram for blue channel.

III. Discussion

From the Figure 7 and 8, the processed images are much bright. As I mentioned before, when the contrast of image is too high or too low, histogram equalization techniques are good

solutions. There are still some different from those two images from my observation. The Figure 7 are a little bit bright. From the right-hand side of image, it is more obvious. The contrast of image on Figure 8 is smoother. For human eye's observation, I think Figure 8 is better than Figure7. But both histogram equalization techniques are good, I cannot tell the big issues or the big different between two images. But the original images are totally different with the processed image. The edge of objects is not smooth when we zoom in the picture. To further improve the result, I think we can use a Gaussian and Bilateral filter to process the image, because those two types of filter can help us to fix the edge problem that I mentioned before. There is a black curve on the head of the bear from the Figure 7 and 8. If we use Gaussian and Bilateral filter, it may get better.

Problem 3: Image Denoising

I. Abstract and Motivation

In the real word, there are many pictures have some noise by the environment change or many other factors. To improve the image quality, I use five method to remove the noise in this project. The first two method call uniform weight function and Gaussian weight function denoising. The third method is similar to the gaussian weight function, but have an extra term, it calls bilateral filtering denoising. The last two are Non-local means filtering and Block matching and 3-D (BM3D) transform filter denoising. For each method, using the PSNR to measure the difference between the original image and processed image is the best way to show the performance of the denoising algorithm. The MATLAB is the only platform that the project will use to generate and modify the image.

II. Approach and Procedures

The first step of this project is learning the convolution for the image processing. Name the filter as w(i, j, k, l). The general convolution process uses the formula below

$$Y(i,j) = \sum_{k,l} w(i,j,k,l) * I(k,l) \qquad (13)$$

Where Y(i,j) is the filter image, I(k,l) is the original image, the convolution start the left top corner for each image.

The second step of this project is learning how to compare the original with processed image. The term of PSNR measure the difference between the original image and processed image to show the performance of the denoising algorithm. The PSNR formula shows below

$$PSNR(dB) = 10log10(\frac{255^2}{1}\frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M}(Y(i,j) - X(i,j))^2) \qquad (14)$$

Where N and M is the width and length of the matrix. In this experiment, the N and M both are 320. Y(i,j) is the filtered image of NxM. X(i,j) is the original image of NxM.

To determine the type of embedded noise, the noise is equal the noisy corn image subtract the original image. Figure 9 shows histogram graph for the noise.

a. Uniform weight filtering

The filter of this uniform weight function is constant. Each component of this filter are constant 1. Therefore, we can generate a fixed filter to do the processing. There are three different type constant filter in the project. There are 3x3, 5x5 and 7x7 filter. After construct the filter, the initial image needs to do the boundary extension, since the filtered image size should match the size of initial image. The size of boundary extension according to the filter size. The formula below shows the relationship between size of boundary extension and the filter size.

$$extended\_x = x + filter_{size} - 1 \tag{15}$$

$$extended\_y = y + filter_{size} - 1 \tag{16}$$

$extended\_x$ and $extended\_y$ represent the number of rows and columns of the extended matrix, x and y represent the original image size. The $filter_{size}$ represent the custom filter size. Finally, the experiment use formula below to do the convolutions.

$$Y(i,j) = \frac{\sum_{k,l} w(i,j,k,l) * I(k,l)}{\sum_{k,l} w(i,j,k,l)} \tag{17}$$

Y(i,j) represent the output pixel, $w(i,j,k,l)$ represent the filter, I represent the current input pixel. k and l represent the location for the current pixel, and i, j is the center of the filter window. Figure 10a, 10b, 10c shows the filter and the PSNR value for each output.

b. Gaussian weight filtering

Gaussian weight filtering has similar process with uniform weight filtering, but the components of this filter is not constant 1. Generating the filter by using the formula below.

$$w(i,j,k,l) = \frac{1}{\sqrt{2\pi}\sigma} \; e^{-\frac{(k-i)^2+(l-j)^2}{2\sigma^2}} \tag{18}$$

Where $\sigma$is the standard deviation, this experiment uses four different σ value. k and l represent the location for the current component, i and j represent the center of the filter. Take w(3,3,2,3) as an example. The filter is a 5x5 filter, (3,3) is the center of the matrix. (2,3) is the location of current component. The value for this point is round 0.24197 when the $\sigma = 1$. Finally, the experiment use equation 15 to do the convolutions. Figure 11a, 11b, 11c shows the output image for different σ and different filter size.

c. Bilateral filtering

Bilateral filtering has similar process with gaussian weight filtering, but the filter is not constant. The filter changes according to location of the input image. Generating the filter by using the formula below.

$$w(i,j,k,l) = \; e^{-\frac{(k-i)^2+(l-j)^2}{2\sigma c^2} - \frac{(I(i,j)+I(k,l))\text{^}2}{2\sigma s^2}} \tag{19}$$

Where I represent the input image, i and j is the location of center of current input image, k and l represent the location for the current component of filter. There are two different σ in this filtering. The experiment result shows different image with different σ in this filter. The program uses two for loop to run the different σ. Each different filter use equation 15 to do the

convolution process and generate an output image. Figure 12 shows the best output image for different σ and different filter size. The PSNR value for each image shows on the image as well.

d. Non-local means filtering

This filter similar with Bilateral filter. But filter value depends on larger region, it no longer depends on local window. The filter uses the formula below

$$w(i,j,k,l) = e^{-\frac{(I(N(i,j))+I(N(k,l)))\text{^}2}{h^2}} \quad (20)$$

Where h is the filter parameter, term N is the location for each pixel from larger region, I$(I(N(i,j)) + I(N(k,l)))\text{^}2$ is term use the formula below

$$\left(I\left(N(i,j)\right) + I\left(N(k,l)\right)\right)^2 = \sum_{n1,n2} G_a(n1,n2)(I(i-n1,j-n2) - I(k-n1,l-n2))\text{^}2 \quad (21)$$

Where n1 and n2 are the relative location in the window. Ga use the formula below

$$G_a \quad (n1,n2) = \frac{1}{\sqrt{2\pi}\sigma} \; e^{-\frac{n1^2+n2^2}{2\sigma^2}} \quad (22)$$

Finally, all the number will plot the number into the formula 17.

Where this is the same formulate that using for the gaussian weight filtering. This is the same as equation 18. There is a good open source for this experiment. Figure 13a, 13b and 13c shows different output with different parameters.

e. Block matching and 3-D (BM3D) transform filter

Use open source to write this question is much easier. Block matching and 3-D transform filter is similar to the NLM filter, but more advanced. There are more considerations for the process, and there are more steps for the process. There are two inputs for the open source code. The first one is the input noised image; the second input is the sigma value. The project uses the for loop to run through many possible solutions, Figure 14 shows the best solution for the experiment. In the discussion, I will mention the process and the algorithm for the BM3D.

III. Experimental Results

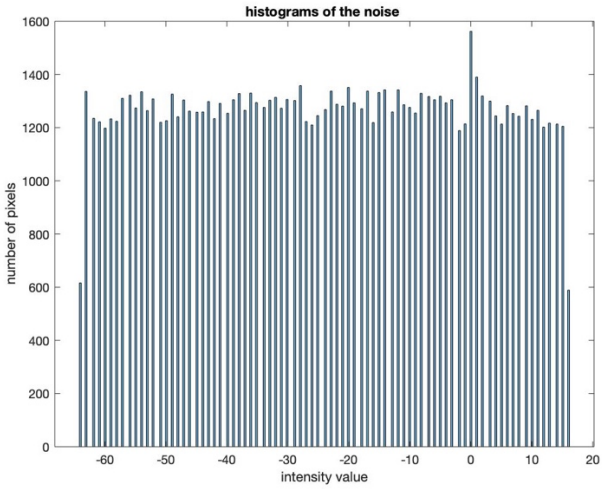


Figure 9. Histogram graph for the noise

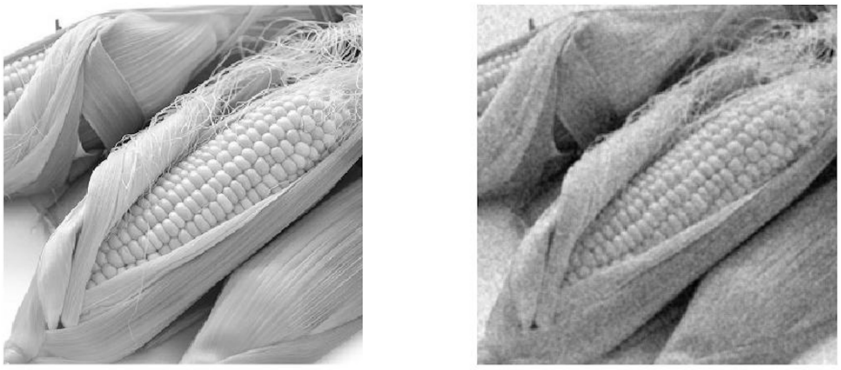


Figure 10a. The original image vs output image with 3x3 filter and the PSNR=19.4340

Figure 10b. The original image vs output image with 5x5 filter and the PSNR=19.2054

Figure 10c. The original image vs output image with 7x7 filter and the PSNR=18.9618

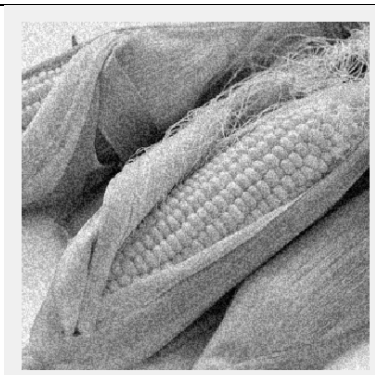
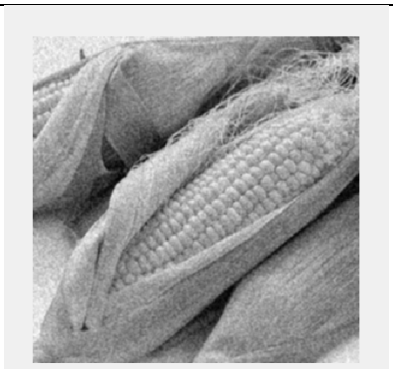
	
σ =0.5, PSNR=19.0304	σ =1, PSNR=19.5266
	
σ =3, PSNR=19.4476	σ =5, PSNR=19.4390

Figure 11a. The output image with 3x3 filter and the σ =0.5, 1, 3,5

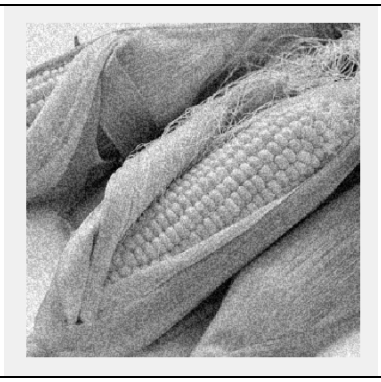
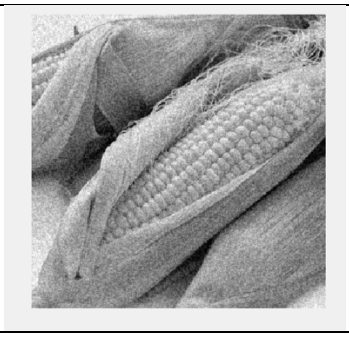
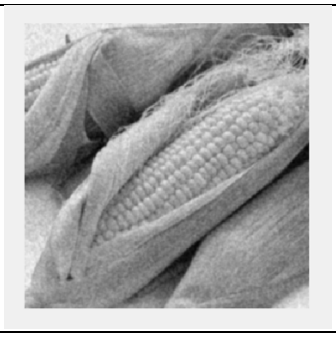
	
σ =0.5, PSNR=19.0323	σ =1, PSNR=19.5296
	
σ =3, PSNR=19.2591	σ =5, PSNR=19.2251

Figure 11b. The output image with 5x5 filter and the σ =0.5, 1, 3, 5

	
σ =0.5, PSNR=19.0323	σ =1, PSNR=19.5267

	
σ =3, PSNR=19.0747	σ =5, PSNR=19.0747

Figure 11b. The output image with 7x7 filter and the σ =0.5, 1, 3, 5

Figure 12a. The best output image with 3x3 filter and the σc =2, 1,σs=67, PSNR=19.6632

Figure 12b. The best output image with 5x5 filter and the σc =1, 1,σs=61, PSNR= 19.6907

Figure 12b. The best output image with 7x7 filter and the σc =1, 1,σs=59, PSNR= 19.6879

Figure 13a. The output image with search window radius=10 and radius of similarity window = 15, h=3 and PSNR= 19.2930

Figure 13b. The output image with search window radius=5 and radius of similarity window = 2, h=10 and PSNR=18.5116

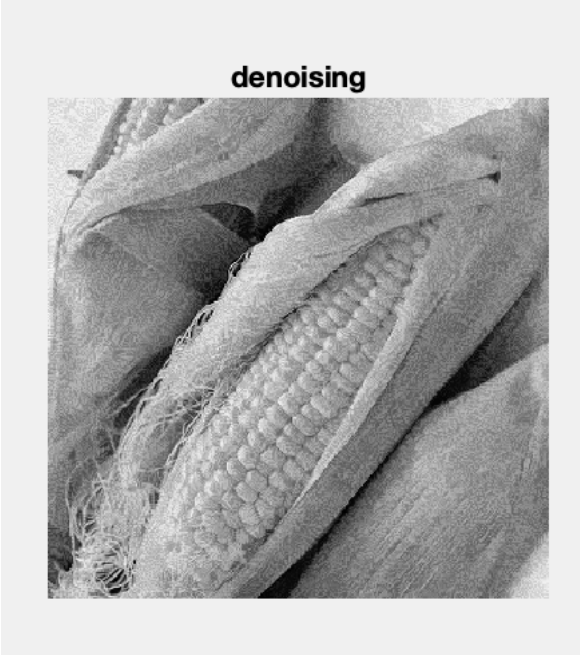


Figure 13c. The output image with search window radius=5 and radius of similarity window = 10, h=15 and PSNR=19.2163

Figure 14. The output image with sigma=0.8 with PSNR=13.4526

IV. Discussion

After the first step process, the embedded noise in Figure 7(b) should be the uniform noise. From the Figure 9, the histogram of the noise seems to be uniform.

When the filter uses uniform weight function, the best filter size 3x3. The PSNR values is 19.4340. The larger PSNR values give us a better result. When the filter size become bigger and bigger, the image just become blurry. From the result, sigma=1 give the best solution for different filter size for the Gaussian weight function. The best filter size for Gaussian weight function is 5x5. When the sigma=1 in 5x5 filter size, the PSNR=19.5296. Where are two ways to compare the compare the uniform weight function and the Gaussian weight function. The first way is observation, I can directly see the image through the eyes, but I cannot tell the different for both pictures. The second way is that comparing the PSNR value for both images. The different between two best PSNR for both images are very small. Therefore, the uniform weight function and the Gaussian weight function will generate similar result, but from PSNR, the Gaussian weight function are slightly better then uniform weight function.

For the bilateral filtering, I use two for loop run the σc from 0.5 to 100 with interval 0.5 and run the σs from 0.5 to 150 with interval 0.5. When the σc =2, σs =67, there is a best PSNR=19.6632 value for 3x3 filter size. When the σc =1, σs =61, there is a best PSNR=19.6907 value for 5x5 filter size. When the σc =1, σs =57, there is a best PSNR=19.6879 value for 7x7 filter size. When the σc close to 1 and σs close to 60, the PSNR will become larger. Comparing this bilateral filtering image with uniform and Gaussian image, the bilateral filtering image is slight better, From the PSNR value, it also shows the same performance. The larger PSNR give us a better picture. The reason for better solution is because the filter is not constant, the filter is changed every time the import image changes. This process makes the image smoother and looks much better. The bilateral filtering is better than uniform and gaussian filter from the PSNR value.

For the Non-local Means filtering process, I tired many different tests, when the h gets bigger, the image become smoother. When the h gets smaller, the image become blurry. From the reading, I learned that then sigma become lager, the window size should make it larger as well. And there is a positive linear relationship between h and sigma. When the sigma become larger, the h will get larger too. Comparing the NLM filter with uniform,

gaussian and bilateral filtering, the image seems smoother, but the PSNR value is not prefer as I think, I think there are still many considerations for NLM. There are many other relationships between each parameter. As we do more tests, the result will get better and better.

For the Block matching and 3-D (BM3D) transform filter, I use a for loop make the sigma from 0.5 to 100. The best sigma value is close to 0.8. The PSNR is close to 13.4526. Comparing the image with uniform, gaussian, bilateral and NLM filter, BM3D is not the best better solution for our image. But the PSNR is not high as I expect. I think maybe the resolution of our image is not big enough. If we have 1000*1000 image, it may get better. For the process of BM3D. There are two steps for BM3D. The first step is finding the similar pixel with current pixel, and use equations to normalize these pixels, this is for reducing the noises for those pixels. They will use similar equation 17. Then those pixels go to the second round to collaborate with wiener filter. Therefore, the image will much better than the NLM filter, because they are two steps for our process.

Extra question for discussion:

1. There are uniform and gaussian noise.
2. Yes, I think we should perform filtering on individual channels separately for those two types of noise. Because the noise will affect every layer (RGB). Therefore, each layer should do a filtering. Finally, we can combine all the layer together.
3. From our experiment result, I think I will use BM3D and NLM filter for this image, because The PSNR for BM3D and NLM are higher than basic and bilateral filter.