Yusufu Feierdawusi
4929-7851-40
Feierdaw@usc.edu

EE569 Homework #1
January 30, 2022

Problem 1: Image Demosaicing and Histogram Manipulation

a. Bilinear Demosaicing

1. Motivation

Along with the improvement of technology in the last few decades, image processing has also become more technologically advanced. We now have image signal processors in almost all digital cameras and the RGB colored images captured are presented with red, green and blue channels. However, when an image is captured, the pixels in each location have only one of the three color values, thus we need to find the other two channels in order to have a colored image. Furthermore, in order to obtain colored images instead of a gray-scale image, we use a demosaicing method, bilinear interpolation, on a Color Filter Array (CFA) with the assumed Bayer Transform pattern (GRBG). This method uses the average of the RGB color channels of the neighboring pixels [1].

2. Approaches and Procedures

For the first part of this problem, we are asked to implement bilinear interpolation, a demosaicing method. We first read the gray-scale image row by row to obtain their intensity values for each of the color channels. Since we know that most images taken in the modern era have three channels for each color, we use a 3x3 convolution filter with the Bayer's transform pattern to approximate the corresponding color channel. These color channels are identified in the pattern based on their coordinates, like (odd, odd) and (even, even) is the green channel, (odd, even) is the red channel and so on. For example, the center pixel in the filter is the target pixel that we want to approximate the color channels for. Based on the color channel of this pixel

in the pattern, we take the average of the other two color channels of the corresponding neighboring pixels. However, this causes issues when we do bilinear interpolation for edge pixels as they have no neighboring pixels on one or both sides. Thus, we use odd boundary extension by reflecting the row or column adjacent to the edge row or column to solve this problem.

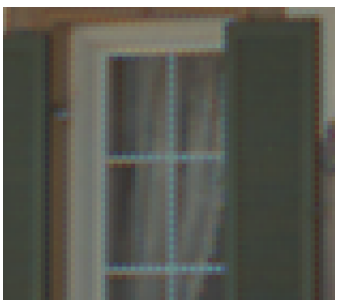
3. Experimental Results

Figure 1: Original House Image

Figure 2: Demosaiced Image

4. Discussions

2). Figure 2 is the resulting image with bilinear interpolation. Compared to figure 1, which is the result of a more advanced demosaicing method, we can see that there are a lot of color artifacts in this figure. When we take a closer look we can conclude that these artifacts are mostly located at the edge of objects bordering other objects in the picture. This is mainly caused by neighboring pixels holding a high average of a certain color channel like Blue. For example, this window in the figure below has artifacts on the muntin bars separating the glass panels. This is because the neighboring pixels who have blue channels according to the Bayer's pattern have a higher average than the other two color channels. One of the ways to improve this demosaic performance is to increase the size of the convolution filter so that the center color channel can



take the average of all three colors in a bigger region. This will help to normalize huge color intensity values by dividing by a bigger number. However, making the size greater on the convolution filter could lead to a more blurred picture, as we will see in problem 2.

b. Histogram Manipulation

1. Motivation

In image processing, we say that an image with a lot of dark or bright regions with pixel intensity values near 0 or 255 mean that the global contrast of the image is low. In order to manipulate this contrast, we first obtained the histogram of the image, which is mainly the count of each pixel according to their pixel values. In order to make the image more visible and pleasant we can perform transfer-function-based or cumulative-probability-based equalization methods on this histogram so that it is more equalized and has a more normal distribution of pixel counts. This means that the pixel values with high count will be spread out to other pixel intensities.

2. Approaches and Procedures

For the transfer-function-based histogram equalization method, we count all the gray-scale pixel values (0 - 255) in the picture. Next, we need to normalize the histogram by dividing every pixel count by the total pixel count to obtain the probability density function. This function gives us a probability of what values to fill in each column. However, we cannot use this function to map out pixel values accordingly. Thus, we find the cumulative-density function by sequentially summing each normalized pixel count while assigning that sum to each pixel value. Finally, for every pixel value scanned from the original image, we obtain the corresponding normalized Y-value and multiply it by 255 (in order to obtain the actual intensity value).

For the cumulative-probability-based histogram equalization method, we first get all the counts per pixel value. We then determine how many “buckets” to have for the histogram based on the number of total pixels. In this problem, the picture size is 256 by 256, which means that

there are 65536 pixels, and we know that there are 256 pixel values ranging from 0 to 255. Thus, we will have 65536 / 256 = 256 pixels per bucket for the histogram. We then assign which pixel values belong to which bucket based on the count of that pixel value. We start with bucket 0, if the corresponding pixel count is more than 256 we store the difference in the next bucket and so on. Furthermore, we save the location of pixels based on their pixel values, then we assign the pixel values to those locations based on which bucket the value ended up in. The main difference between this method and method A is that in this method the histogram becomes linearly equalized, which means that if you graph the histogram, it becomes a line. This also enables each of the pixel intensity values to map to multiple locations.

3. Experimental Results

Figure 1: Original Hat Image

Figure 2: Method A Result

Figure 3: Method B Result

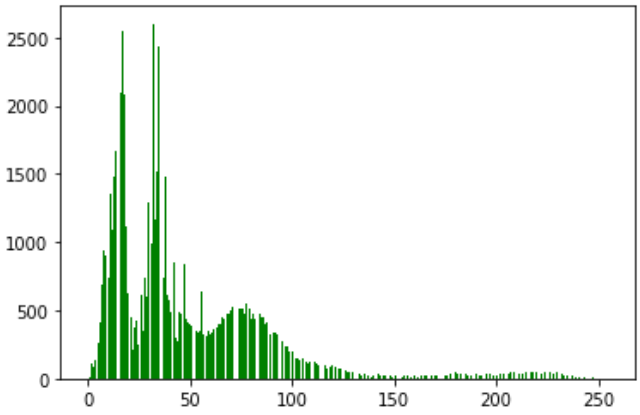


Figure 4: Original Hat Histogram

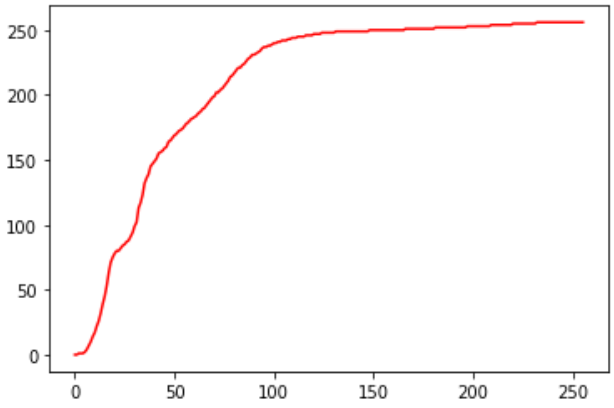


Figure 5: Transfer Function

4. Discussions

4). Figure 2 is the result image of applying the transfer-function-based histogram equalization method. As we can visually observe from this figure that it enhanced the image by increasing the contrast, more specifically it flatten the horizontal dimension (x) in the histogram from figure 4. In this figure, the x axis is the pixel values (0 - 255) and the y-axis is the pixel count. Similarly in figure 5, we can see that the transfer function tries to equalize the histogram

distribution. In a perfect world with the best histogram equalization methods, this transfer function should be perfectly uniform and the resulting histogram would be a uniform distribution. Although I know that my understanding of the algorithm is correct, the result of method B (figure 3) visually has less brightness which implies that my implementation of the algorithm is incorrect. However, the pixel values in each location are different, thus it must be increasing the contrast of the image, although not as strong as method A. We can also observe from figure 3 that there are a few artifacts with pixel values equal to 0 on top of the hat. I believe that these artifacts came from the overflow in my implementation, where there is a 256th bucket with a few pixel locations stored inside. These pixel values would be zero because grey-scale values only range from 0 to 255. I personally believe that method A (figure 2) is a better histogram equalization method because it tends to have a high brightness scene, which makes objects inside the picture more visible. However, it may not be the best method because instead of normalizing the contrast, it now has a higher contrast scene and we can observe some new artifacts in the sky behind the hat.

Problem 2: Image Denoising

a. Basic Denoising Methods

1. Motivation

Almost all images taken today with a modern camera will eventually downgrade in quality, one of the ways that it could deteriorate is to include noise. Noises are mostly caused by a dead sensor or sensor saturation [2]. There are different types of noises, the ones that are common today are additive noise (gaussian and uniform noise), salt and pepper noise and mixed noise. There are also different types of denoising methods, the ones presented in this problem are linear filters using both uniform weight function and gaussian weight function. For the low pass mean filter, we assign weights by averaging each cell of the filter with the total number of cells. Similarly for the gaussian low pass filter, we use gaussian normal distribution to assign each

weight of the cell. With the center weighing the highest and gradually decreasing until the edges of the filter.

2. Approaches and Procedures

For the mean low pass filter using uniform weight function, we first obtain the grey-scale values of each pixel to form a 3-D array. We copy those values into a second array, then perform odd zero-boundary padding to provide temporary values for pixels at the edge of the image. We then create a linear (meaning every element of the filter is 1) convolution filter with size defined by the user. Next, we create a window with the same size as the filter where the center pixel starts at coordinates (0, 0) until the last pixel location in the image. Now we perform element-wise multiplication between these two matrices and average each resulting value by the total number of elements. Additionally, we assign a variable to keep track of the summation of these normalized values and assign that to the center pixel of the filter.

For the gaussian low pass filter using gaussian weight function, most of the steps are identical. The main difference is that instead of assigning 1 to every element of the filter, we use the standard deviation of the Gaussian distribution, sigma. If an element is 1 cell away from the center, it has one standard deviation. As shown in the figure below, we simply apply the weights based on how many standard deviations away a cell is away from the center.

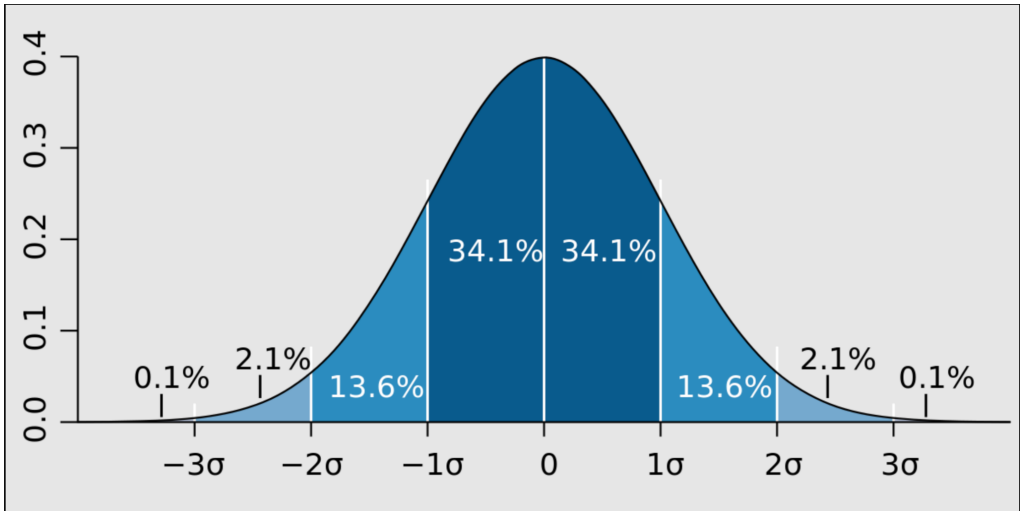


Figure 0: Sigmas of Gaussian Normal Distribution **(Source [3])**

3. Experimental Results

Figure 1: Uniform Weight Function 3x3 Filter Result
PSNR score = 30.2775

Figure 2: Uniform Weight Function 5x5 Filter Result
PSNR score = 26.7636

Figure 3: Gaussian Weight Function 3x3 Filter Result
PSNR score = 31.5987

Figure 4: Gaussian Weight Function 5x5 Filter Result
PSNR score = 29.3802

Figure 5: Original Noisy Image

4. Discussions

1). The type of embedded noise in Figure 6(b) of the question description is additive gaussian noise. When graphing these two images, we can observe from the calculation of these two cumulative probability functions that the additional pixel counts added by the noise creates a gaussian distribution.

2). We can see that the 3x3 mean filter in figure 1 has a higher peak-signal-to-noise-ratio score compared to the 5x5 mean filter in figure 2. Meaning that using a 3x3 filter for uniform weight function will denoise an image better than larger sized filters. We can also visually observe this difference where figure 2 is more blurred out compared to figure 1. Gaussian low pass filter also resulted in a similar fashion where lower sized filter has a higher PSNR value and less blurry compared to a higher sized filter. However, when we compare the mean filter with the gaussian filter we can see that both figure 3 and 4 have a higher PSNR score than figure 1 and 2. Meaning that gaussian filters perform better and are less blurry than mean filters.

b. Bilateral Filtering

1. Motivation

Linear lowpass filters like mean filter and gaussian filter do not have spread parameters to tune, because they mainly depend on the size of the filter and the distance between the center pixel and the neighboring pixels. However, in this case, we ignore how much the pixel intensity value affects the weight calculations. In Bilateral filtering, not only do we have control over how much spatial closeness we want, we can also increase intensity similarity based on the gaussian distance to smooth out the image. One of the main benefits of using bilateral filters is that the algorithm preserves the edges of objects, thus making the images less blurry compared to linear lowpass filters [4].

2. Approaches and Procedures

Similar to the linear filters, we first obtain the pixel values in the image, then we create a boundary extension. Next, we calculate the weights for each element inside the filter using the weight calculation formula. The first half of the equation with the parameter theta c on the

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

denominator represents spatial closeness based on the gaussian distribution. The second half of the equation represents the color intensity closeness based on the gaussian distribution. Following the equation below, we sum the element wise multiplication of this filter and the same window size elements of the original image. Finally we divide this sum by the sum of all weights in the filter window and assign the result to the center pixel.

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

3. Experimental Results

Figure 1: Bilinear 3x3 filter

Theta_c = 80, **Theta_s** = 0.8 **PSNR** = 30.1374

Figure 2: Bilinear 5x5 filter

Theta_c = 80, **Theta_s** = 0.8 **PSNR** = 26.5171

Figure 3: Bilateral 3x3 Filter

Theta_c = 50, **Theta_s** = 0.8 **PSNR** = 29.8057

Figure 4: Bilateral 3x3 Filter

Theta_c = 80, **Theta_s** = 0.2 **PSNR** = 29.9354

4. Discussions

2). From these 4 figures we can see the effects of how different values of parameters theta_c and theta_s affect the image quality. When we compare figure 1 and figure 3, we can see that if we reduce the spatial closeness spreading, the PSNR goes down while the visual difference is not much different. When we compare figure 1 and figure 4, we can see that if we reduce the intensity range spreading, the PSNR goes down. However, if we take a closer look, figure 4 looks a bit more sharper than figure 1 with a higher theta_s value. As we can observe from the zoomed in comparison of the two images below, the flower petal and the small branch

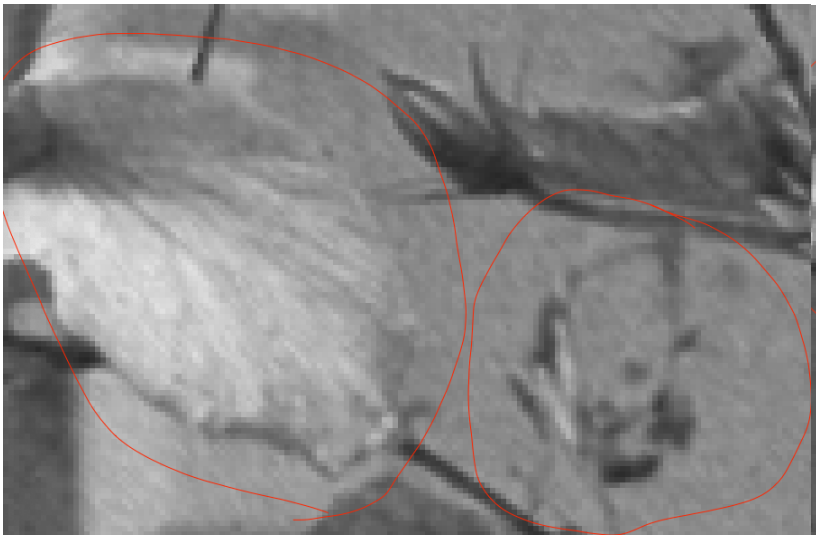


Figure 4 Zoomed in

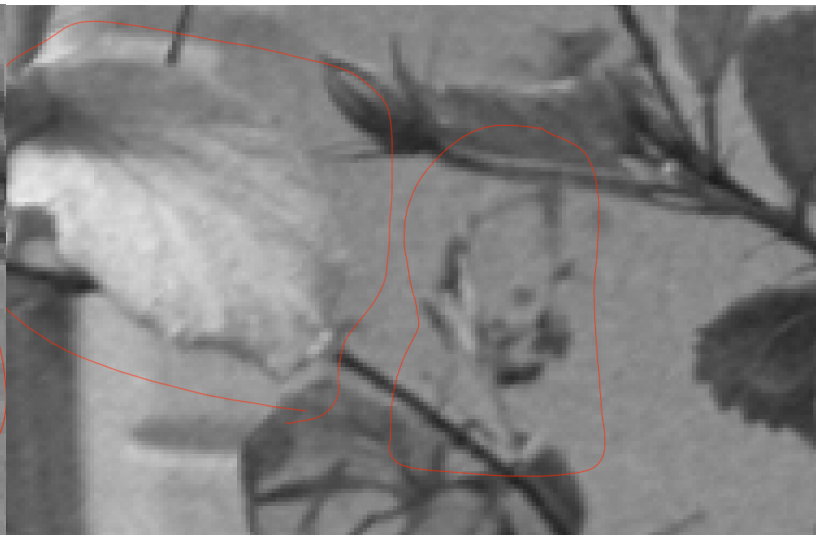


Figure 1 Zoomed in

circled seems to have better edge distinctions with figure 4 zoomed in. The same spots in the zoomed-in figure 1 seem to have a bit more blurred edges. This distinction makes sense when we look at the equation because higher theta_s value will reduce the impact of the intensity range, thus edges will blend and appear more blurred by taking the average of a further range. This means that if we want to smooth out larger edges we would increase this parameter.

3). In the case of this noisy flower image, my Gaussian filter has a better PSNR score than my bilateral filtering method. This is because both filters use gaussian distribution but

bilateral filtering method has an intensity range aspect that could potentially blur a few pixels, thus leading to a small decrease in the score.

Problem 3: Special Effect Image Filters: Creating Frosted Glass Effect

1. Motivation

We have used image filters for demosaicing and denoising, however there are many more uses for convolutional filters. One of these uses is adding special effects on images. In order to create a “frosted glass” effect on an image, we need to randomly assign pixel values based on their neighboring pixels values.

2. Approaches and Procedures

Similar to the other convolution algorithms, we first read in the image, then perform boundary extension. Next, when looping through all the elements in the filter, we assign the values of the input image in a window of the same size as the filter. The center pixel becomes the target to assign the random neighbor’s RGB pixel values. Additionally, the selected neighbor is picked from rand() ranging from 0 to the total size of the filter.

3. Experimental Results

Figure 1: Original Flower Image

Figure 2: Frosted Glass 5x5 Filter

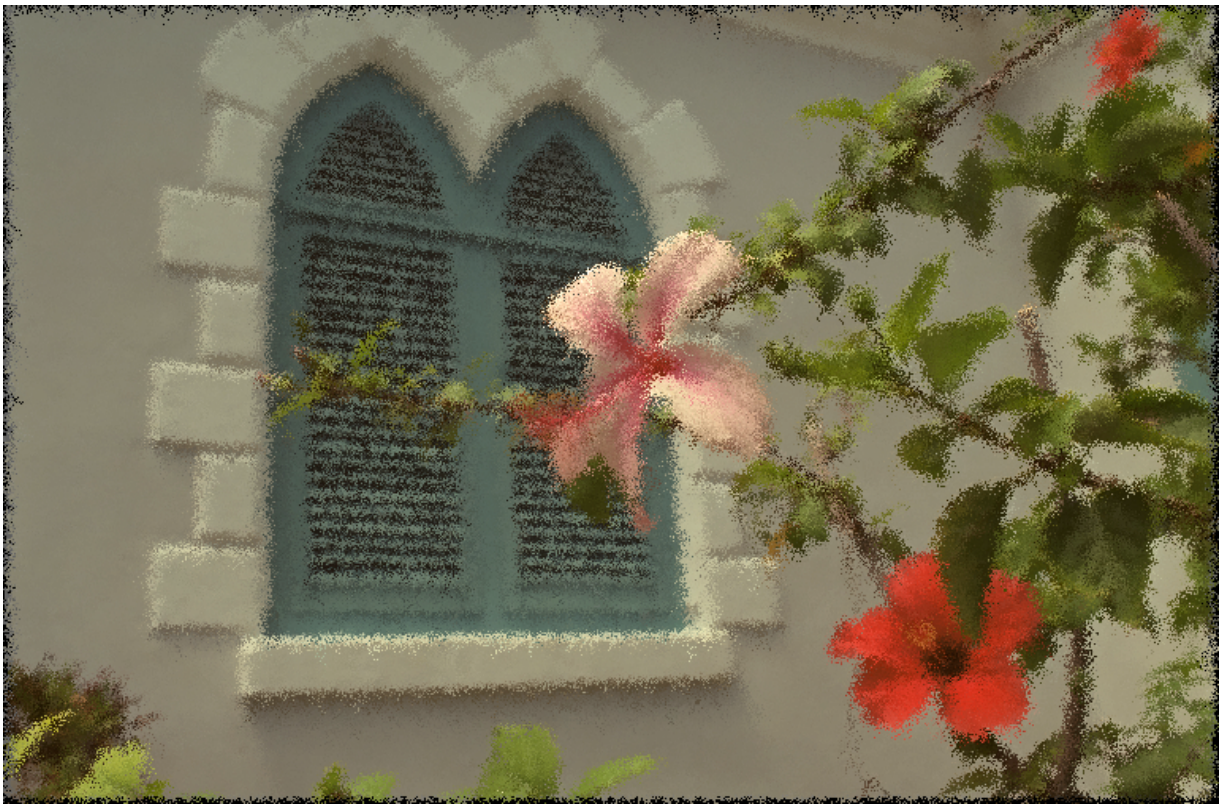


Figure 3: Frosted Glass 7x7 Filter

Figure 4: Original Image with noise

Figure 5: 5x5 Frosted Glass with noise

Figure 6: 7x7 Frosted Glass with noise

4. Discussions

1. Comparing the first two filter results with no noise in figure 2 and 3, we can see that the greater the filter size is, the more distorted the image gets. The edges of objects in the image slowly blend with the background.
2. Comparing the filter results with noise in figure 4 and 5, we can conclude that even with nosied images, the larger the filter size, the more blurred it gets. Furthermore, if we compare the noise in the original noisy image with the results of the filters applied, it seems that the filter produced more random noise all across the image.

Another major artifact that I have noticed in all of these filters applied is that there are some black (intensity = 0) pixels all around the border, this is solely caused by zero-padding boundary extension that I have used. Meaning that during the algorithm, when the center pixel is at one of the boundary locations, it randomly picks a zero-padding pixel and gets assigned that value.

***Note: the answers of written questions from the homework description are numbered in the discussion sections.

Image References

All images are either directly given in the homework description or are the results of my programs. With the exception of source [3]: https://en.wikipedia.org/wiki/Standard_deviation

References

[1]: Demosaicing, From Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/Demosaicing

[2]: EE569 Lecture 2 Image Denoising, Slide 12

[4]: 1998 ICCV Bilateral Filter, C.Tomsi, R.Manduchi, https://courses.uscden.net/d2l/le/content/22123/viewContent/381477/View?ou=22123