EE 569

HOMEWORK 1

By

Morayo Ogunsina

USC ID : 7371213793

ogunsina@usc.edu

Issued 1/20/2021

Due 2/7/2021

Table of Contents

Problem 1 : Image Demosaicing and Histogram Manipulation

a) Bilinear Demosaicing

I. Abstract and Motivation

For image capture sensors such as CMOS sensors, the output image must undergo some transformations to account for missing image values to prevent information loss. One thing about this type of sensors is that they capture one color pers sensor channel. So, R, G and B sensors will capture only their color components respectively. The motivation for bilinear demosaicing is to re-construct the missing color values using bilinear interpolation.

II. Approach and Procedures

For every image capture by the CFA sensor, there is twice as many green color pixels as the red or blue pixels. This forms a special pattern known as the Bayer's pattern (Fig 1, Fig 2). A reason for this uniqueness is due to the sensitivity of the human eye – we are more likely to sense a green color pattern than the other two. With demosaicing, this pattern is present in images captured from a color filter array (CFA). In essence, a demosaiced image has its missing R, G or B values for each pixel.

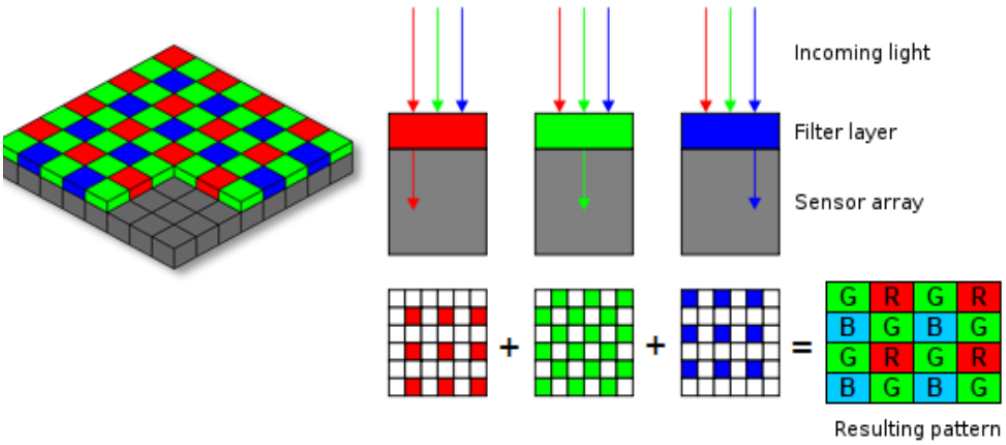
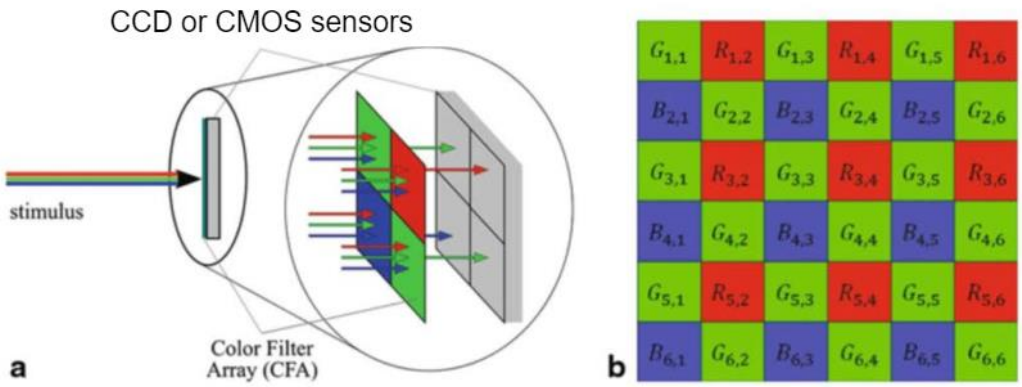


Fig 1. Source : Lecture notes



(a) Single CCD sensor covered by a CFA and (b) Bayer pattern

Fig 2. Source : Lecture notes

To perform bilinear demosaicing, missing color pixel values have to be reconstructed. One way to do so is via bilinear interpolation. For each pixel, the missing color is approximated by taking the average of four or two of its vertical or horizontal pixels adjacent to it.

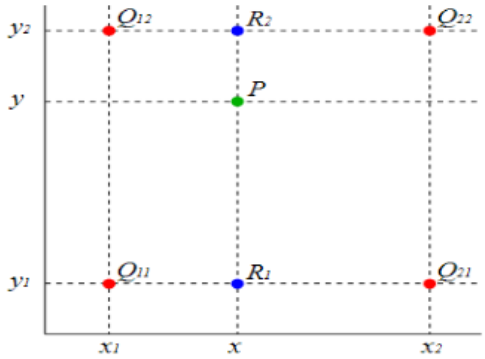
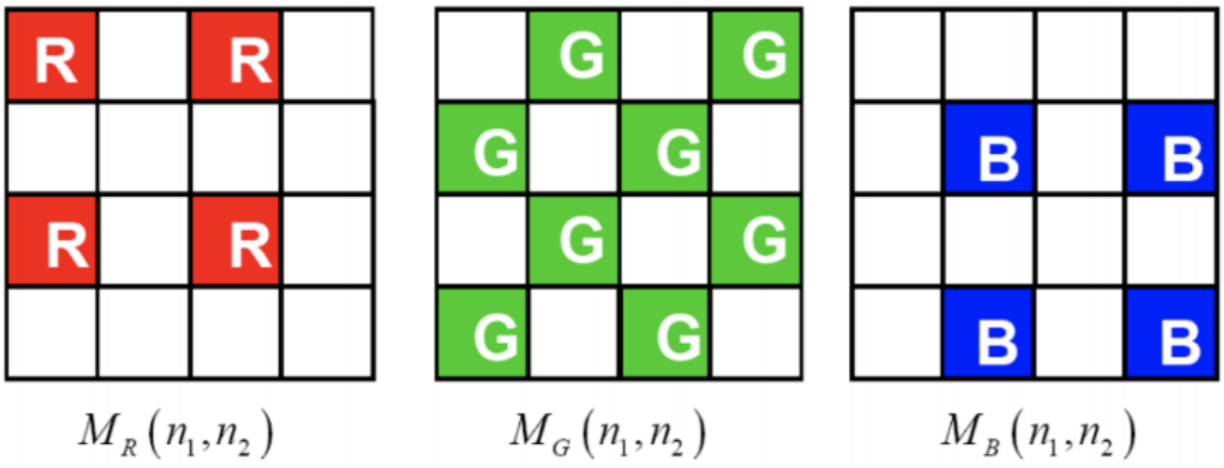


Fig 3. Graphical representation of Bilinear interpolation.

$$\hat{B}_{3,4} = \frac{1}{4}(B_{2,3} + B_{2,5} + B_{4,3} + B_{4,5})$$

$$\hat{G}_{3,4} = \frac{1}{4}(G_{3,3} + G_{2,4} + G_{3,5} + G_{4,4})$$

An estimate of the missing blue and green pixel values for the red pixel at location 3, 4 is given above.

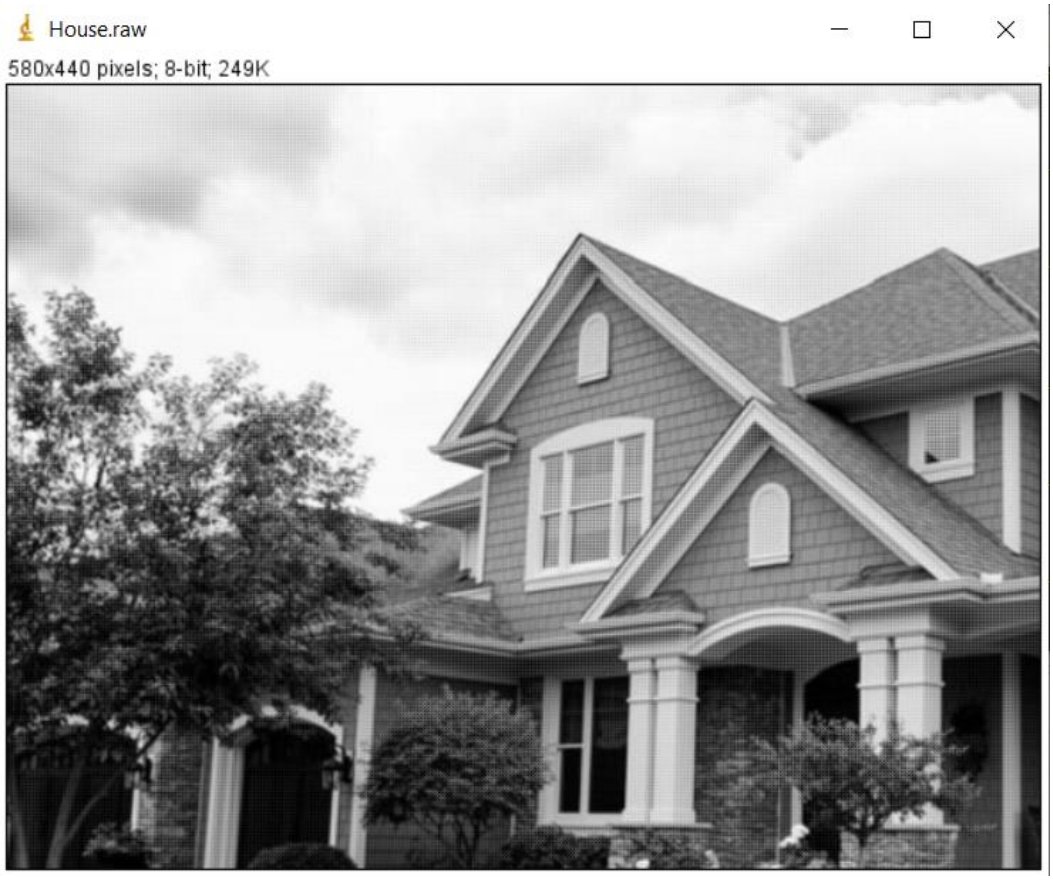


The Bayer Patter followed as a filter for performing the interpolation.

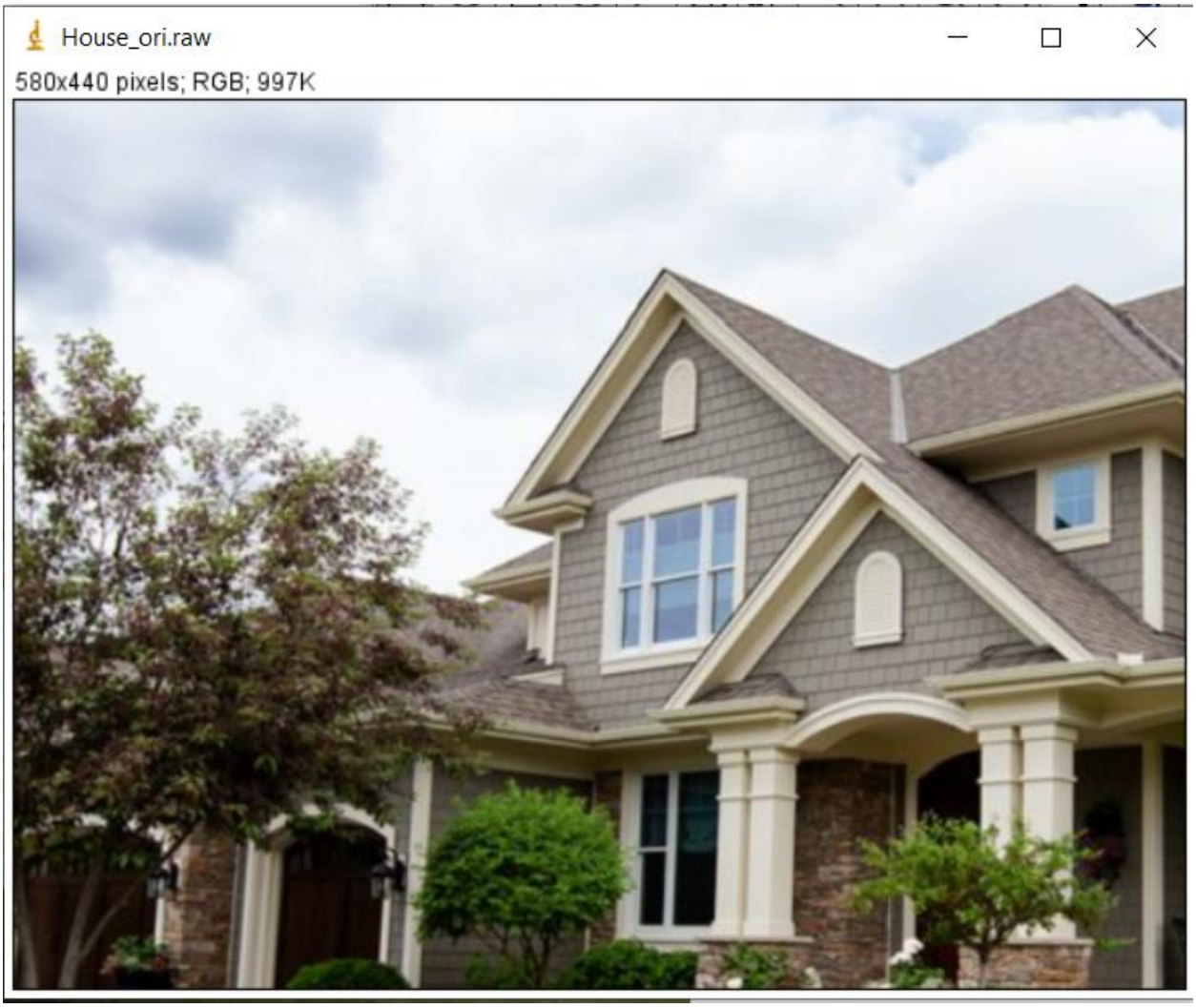
Overall, the steps are summarized to :

- STEP 1 : Run program (check README for instructions).
- STEP 2: Read image file into a 1D array
- STEP 3: For each pixel, do interpolation once each for Red and Blue and twice for Green.
- STEP 4: During interpolation, For RED, get average for horizontal and vertical Green and the average of corner Blues.
- STEP 5: For first pattern of GREEN, get average of horizontal Red and average of vertical Blues. For second pattern of GREEN, get average of vertical Reds and average of horizontal Blues. This is for odd and even locations of pixels.
- STEP 6: For BLUE, get average of horizontal and vertical Greens and average of corner Blues.
- STEP 7 : Write final output in a raw file which is viewed on imageJ.

III. Experimental Results



Original *House* image in 8-bit grayscale , as CFA sensor input



House_ori image in 24-bit color from advanced demosaicing

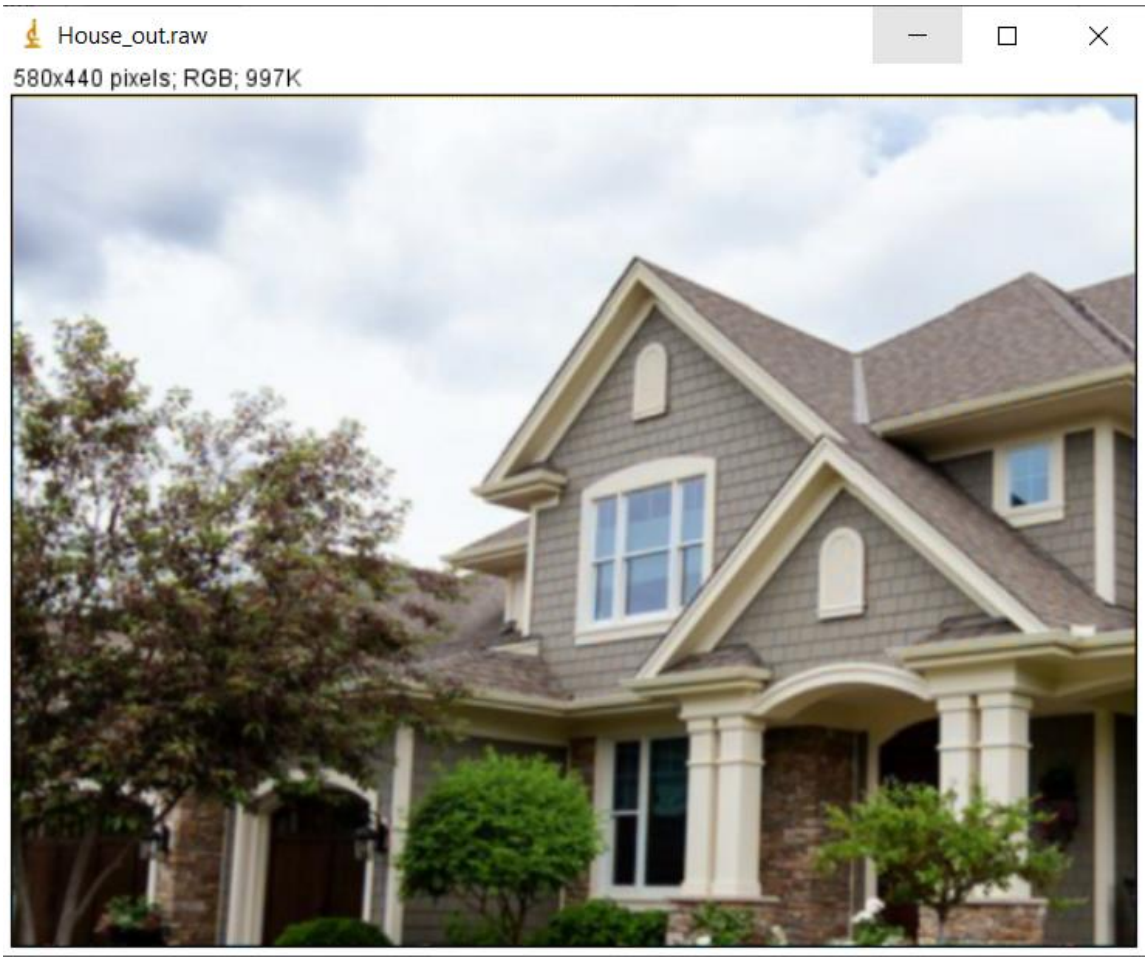


Fig 5. *House* image in 24-bit RGB post bilinear demosaicing.

*IV. **Discussion***

There are artifacts in the image. The Hous_ori image looks more sharper at the edges and has less color loss. The demosaiced image in Figure looks blurry and has some discontinuities of colors at some of the edges and some bit of “unnatural” colors which gives something almost false about the image.

I believe this is because of the approximation errors incurred from doing the interpolation. To improve the demosaicing performance, we can use better algorithm such as the Malvar-He-Cutler (MHC) algorithm. Or maybe we could use a better color array pattern.

b) Histogram Manipulation

I. Abstract and Motivation

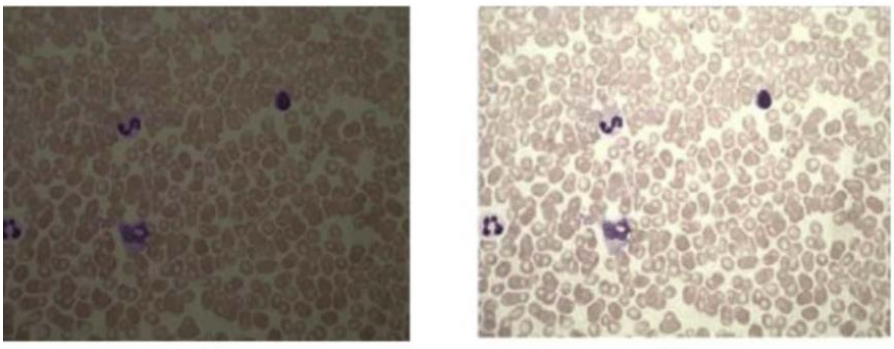
What happens when images we take are not bright enough or too dark to see important details? We get an image that needs to be adjusted through some means to enhance them for better detail representation. One thing that can cause images to have this problem is in the exposure, aperture size, sensitivity and shutter speed, something that most advanced digital cameras now tackle.

Histogram equalization is a way that digital cameras solve this problem of enhancing the image to get better detailed results, specifically in brightness. The idea is to evenly distribute the pixel values that have more occurrence and hence make the difference in each pixel value (in relation to one another) higher. Two methods for this equalization are the transfer-function-based and the cumulative-probability-based equalization.

II. Approach and Procedures

First, the idea of a histogram for this problem is to get the number of times each pixel value (gray in this case) appears in an image. So, the x-axis will be the range of values for the gray color (0 - 255) and y-axis will represent the frequency of each of the values. A too bright image will have its values concentrated on the high end of the pixel range while a too dark image will have its values at the low end. Note that each pixel value represents the intensity of the image at that location.

8-bit Gray-Scale Images
Gray-scales: 0, 1, …, 255
0 -> black (darkest), 255 -> white (brightest)



(a) (b)

(1) Transfer Function Based Histogram Equalization

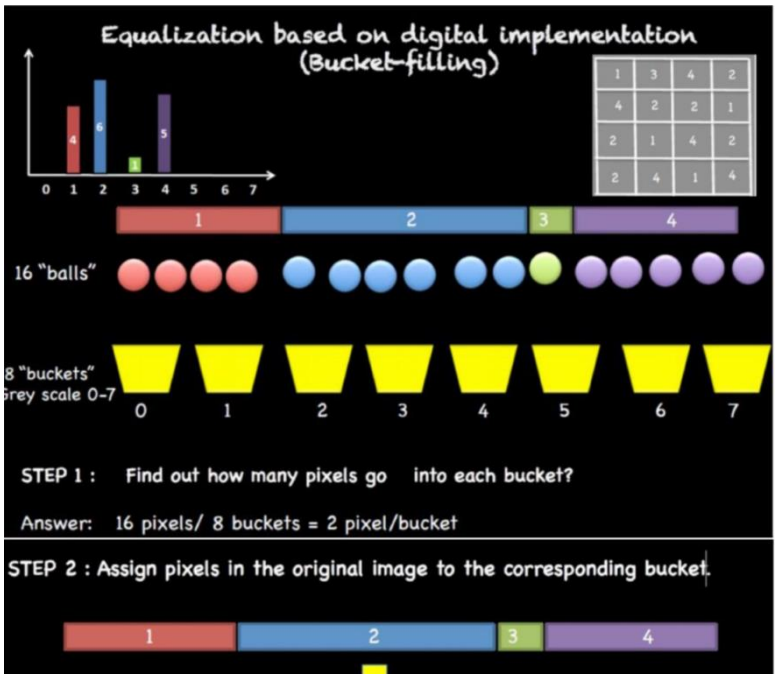
The transfer function method simply maps a pixel value of the input image to a new, enhanced pixel value based on the result from a function, basically like a mapping table.

i.e., y = F(x) and x can take values from 0 to 255. For this method, we integration of the input image, so we use the cumulative distribution function (CDF) to get started.

The steps are summarized to :

- STEP 1 : Run program (check README for instructions).
- STEP 2: Read image file into a 1D array
- STEP 3: Get the frequency or count the number of pixels for a particular pixel intensity value ranging from 0 - 255.
- STEP 4: Then normalize the pixels, which will be needed to calculate the probability of each pixel intensity.
- STEP 5: Then get the cumulative probability and round down the values.
- STEP 6: Map the pixel values to the new values from the histogram equalization which means they are replaced with the new enhanced intensities.
- STEP 7 : Write final output in a raw file which is viewed on imageJ.

(2) Cumulative-Probability-Based Histogram Equalization

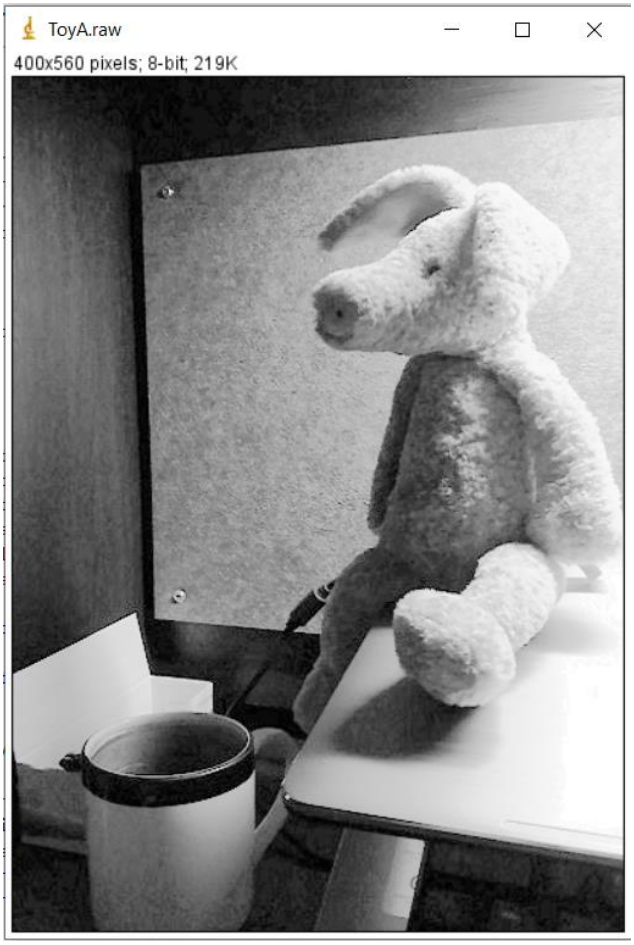


The steps are summarized to :

- STEP 1 : Run program (check README for instructions).
- STEP 2: Read image file into a 1D array
- STEP 3: Get the frequency or count the number of pixels for a particular pixel intensity value ranging from 0 - 255.
- STEP 4: Then track the row and column index for each of the pixel.
- STEP 5: modify the pixel values at the tracked locations based on the bucket size in such a way that the bucket will be uniform in number of pixels.
- STEP 6 : update the pixel values for the image with the new values gotten from the bucket filling method. The rows and column index tracked in STEP 4 are used to properly place the pixels back.
- STEP 7 : Then get the cumulative density of the resulting image as the new pixels.
- STEP 7 : Write final output in a raw file which is viewed on imageJ.

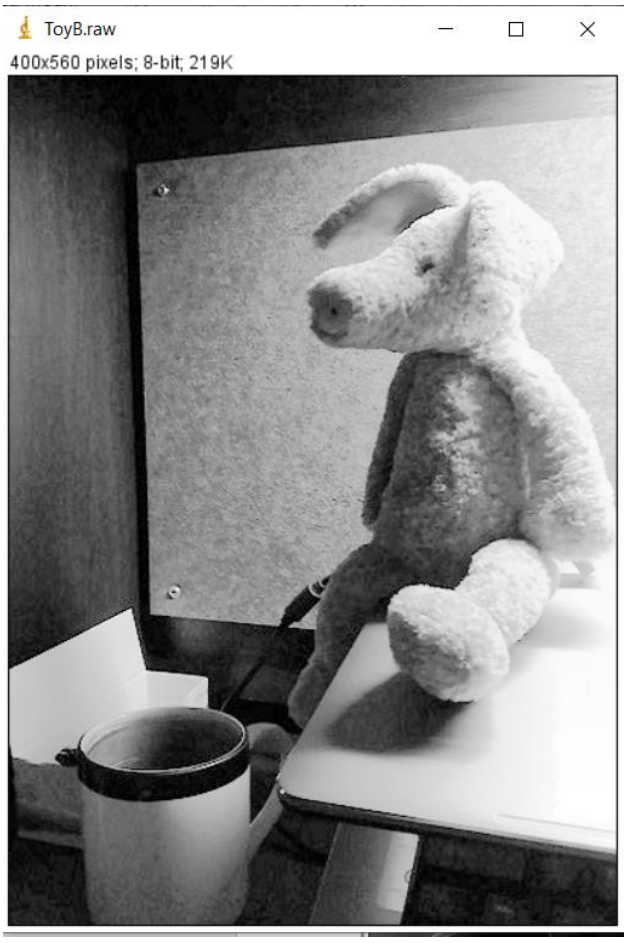
I. ***Experimental Results***

Original Image

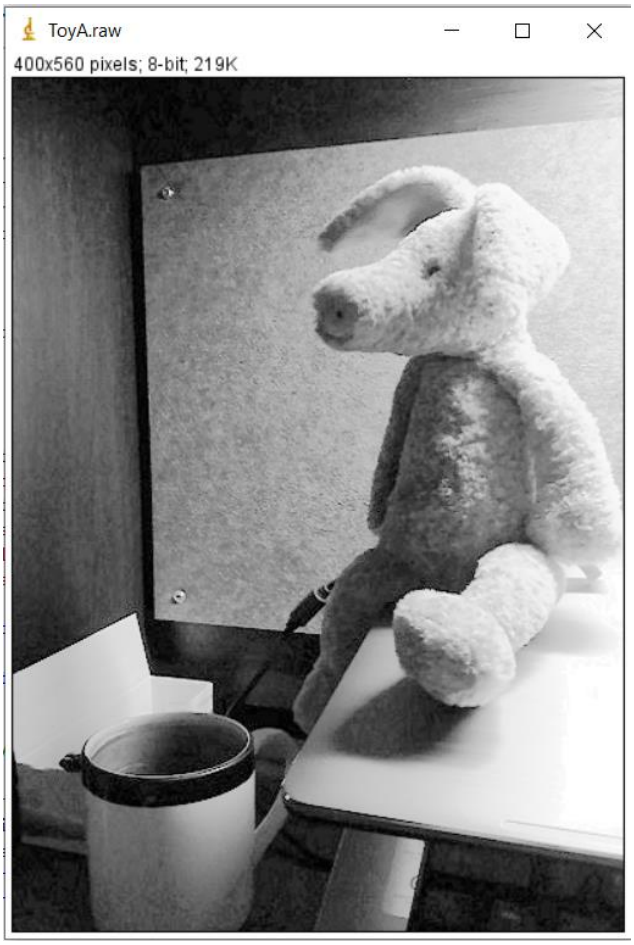


Enhanced image using transfer function

Original Image

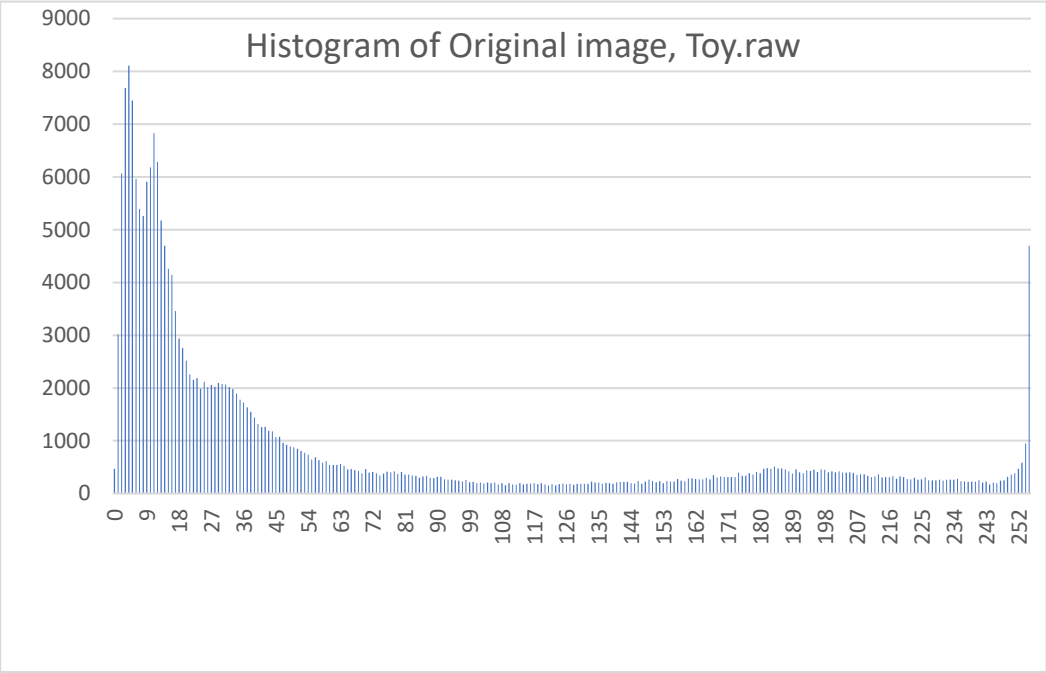


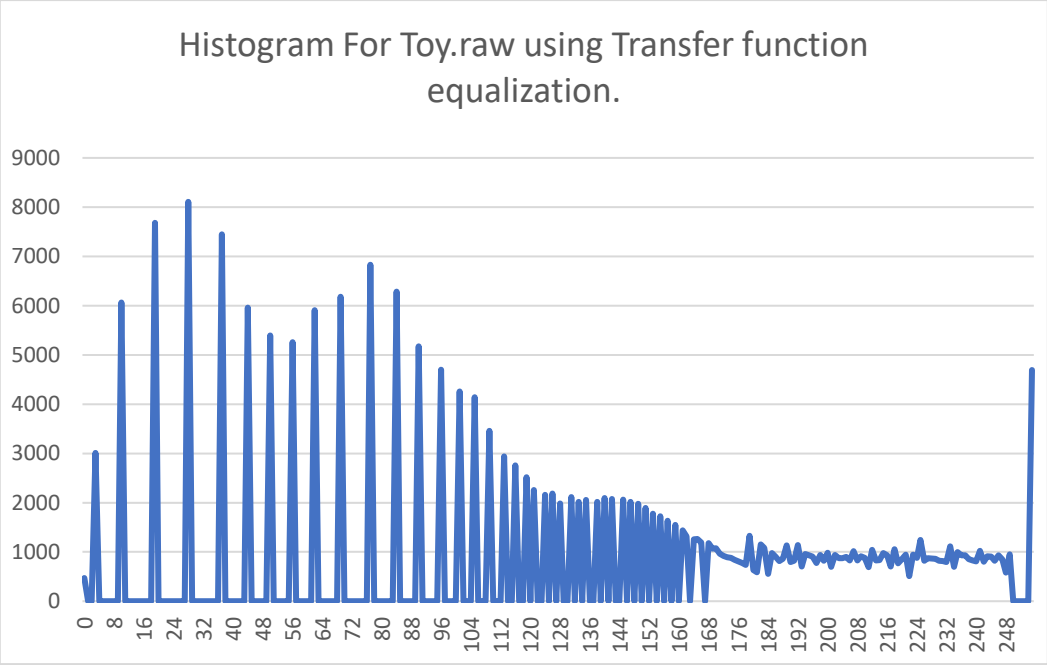
Enhanced image, Cumulative Probability Method

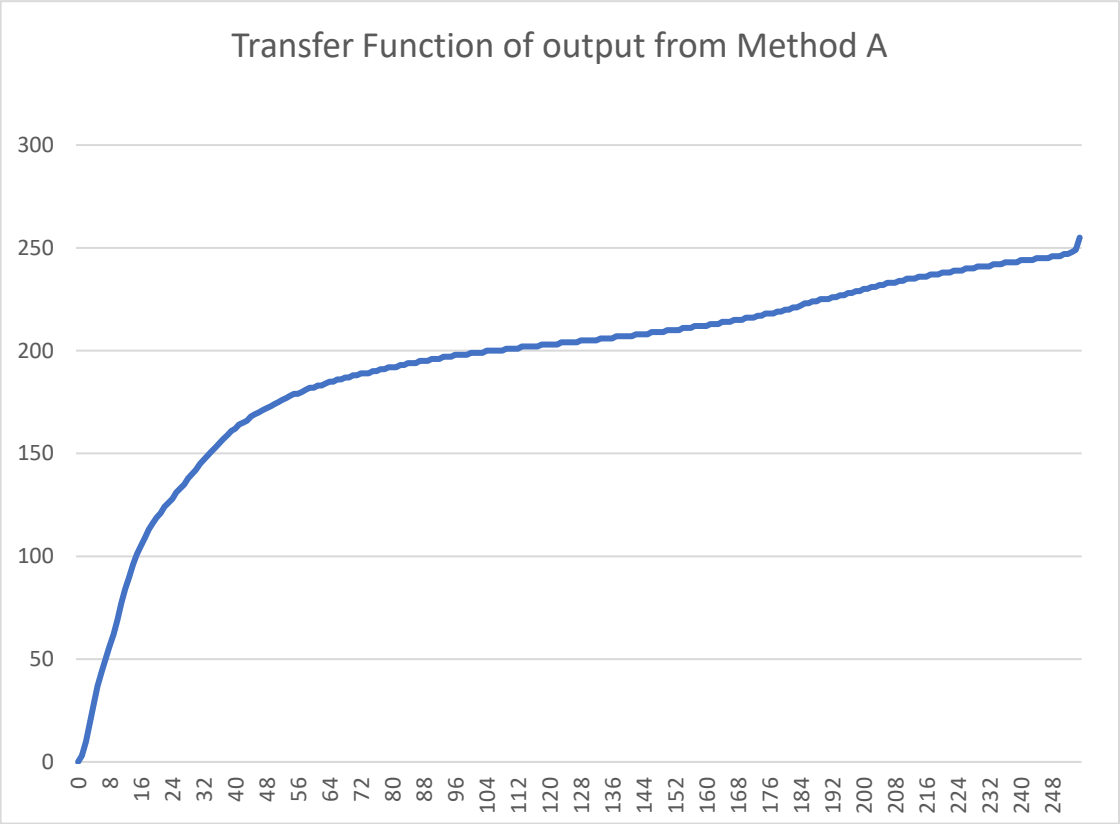


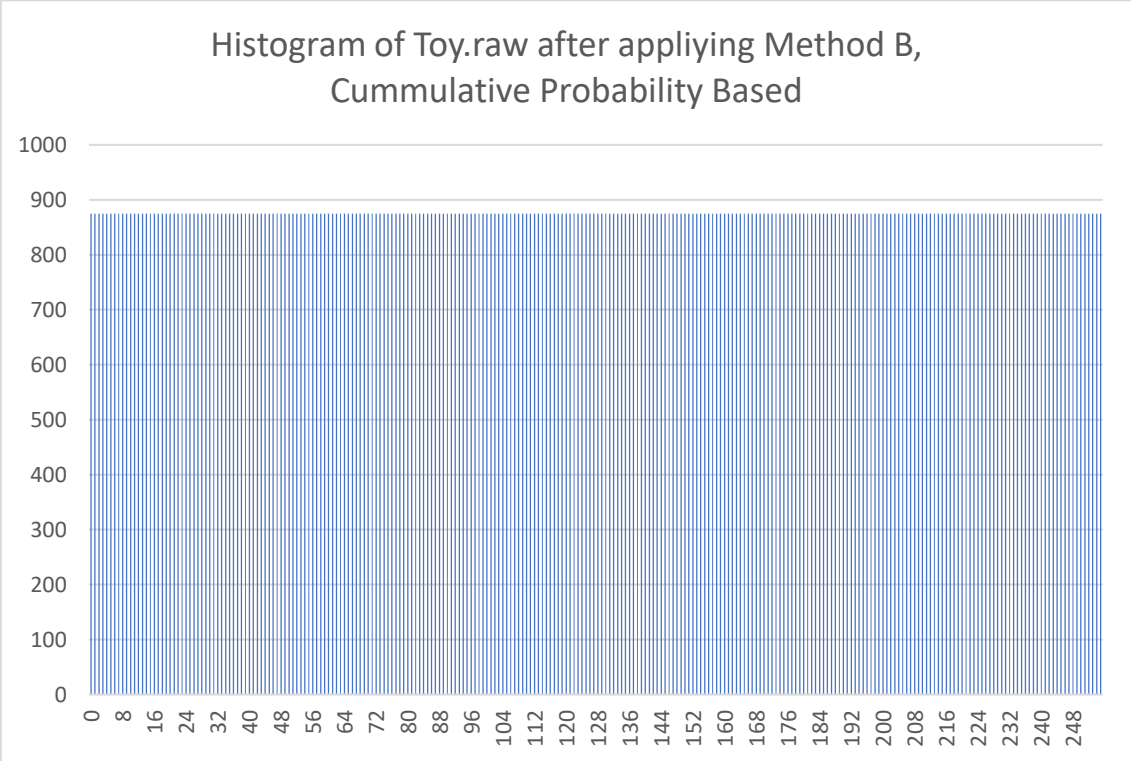
Transfer Function Method

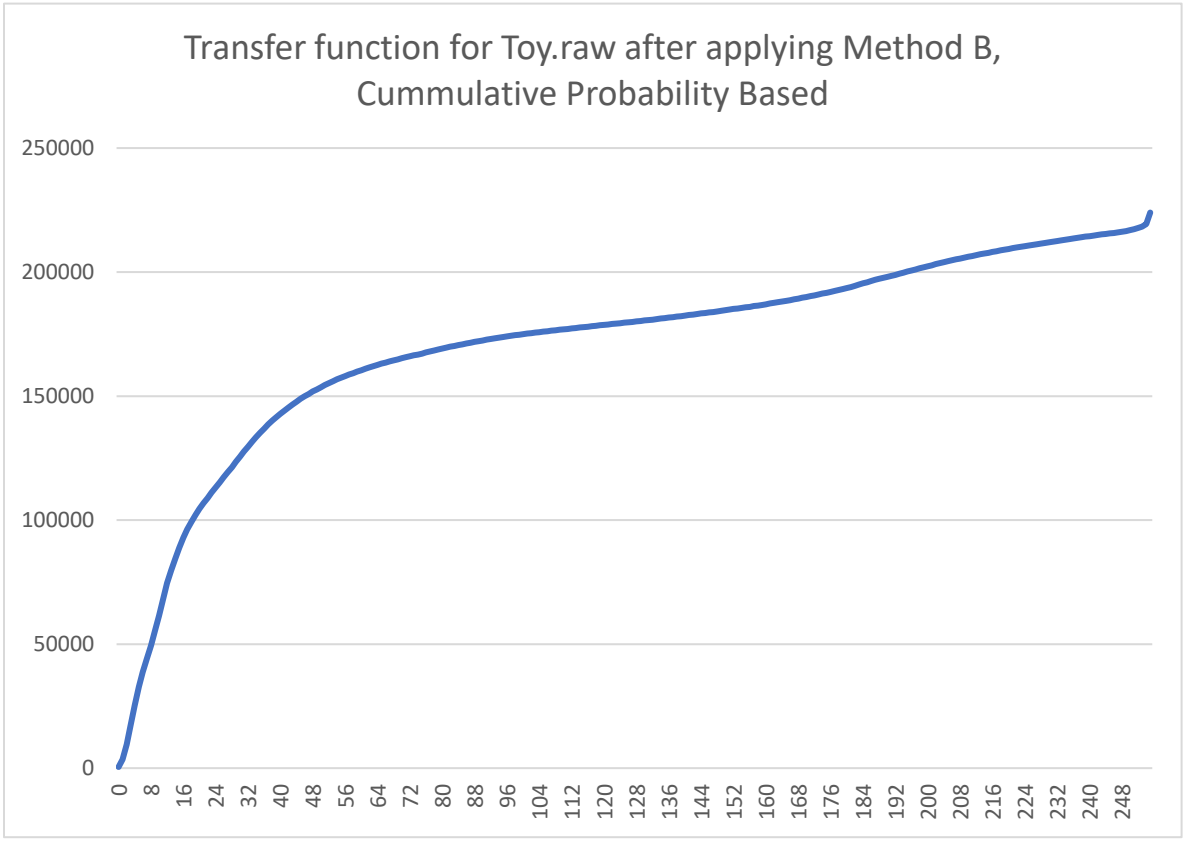
Cumulative Probability Method

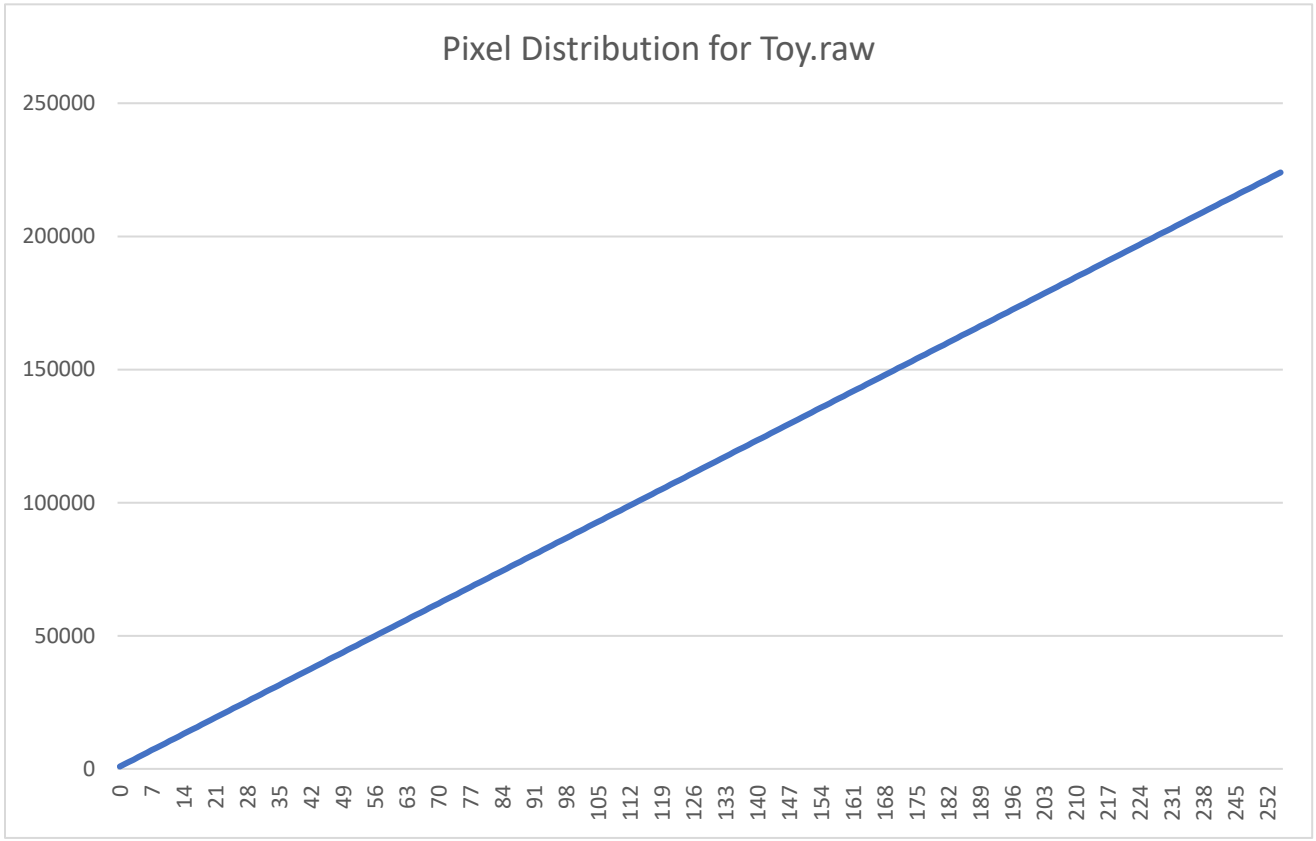

Histogram of Original image, Toy.raw
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0 9 18 27 36 45 54 63 72 81 90 99 108 117 126 135 144 153 162 171 180 189 198 207 216 225 234 243 252


Histogram For Toy.raw using Transfer function equalization.
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168 176 184 192 200 208 216 224 232 240 248


Transfer Function of output from Method A
300
250
200
150
100
50
0
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168 176 184 192 200 208 216 224 232 240 248


Histogram of Toy.raw after appliying Method B,
Cummulative Probability Based
1000
900
800
700
600
500
400
300
200
100
0
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168 176 184 192 200 208 216 224 232 240 248


Transfer function for Toy.raw after applying Method B, Cummulative Probability Based
250000
200000
150000
100000
50000
0
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168 176 184 192 200 208 216 224 232 240 248


Pixel Distribution for Toy.raw
250000
200000
150000
100000
50000
0
0 7 14 21 28 35 42 49 56 63 70 77 84 91 98 105 112 119 126 133 140 147 154 161 168 175 182 189 196 203 210 217 224 231 238 245 252

II. Discussion

Visually, the two enhanced images look similar but on closer inspection, I noticed that the resulting image using method A seem to give a better result. Both methods were able to reveal the darkened objects at the far left and bottom left to a greater detail. And they both intensify the bright areas on the top right, middle right, and bottom right areas of the Toy image. I believe for this image; they have given very similar image brightening results.

Also, their transfer function plot looks very much alike and it is difficult to tell which one of them gives a better result. Method B gets us a well brightened image as seen from the histogram plot which makes sense as it places an equal number of pixels in the bucket. In terms of steps to compute and how simple or complex each method is, I would say that Method A is much simpler to compute and implement than method B. For B, each pixel index must be tracked in relation to its intensity value which means that all the intensity color range is used and hence, more computation. The presence of this means that it explores all the intensity values and therefore should give better results. From the histogram of the original image, the pixels are more concentrated on the low end, but there is a sharp number of intensities at the high (255).

I will conclude that I prefer Method A as it gives a slightly better version of the image output from Method B (in terms of the brightening). Additionally, as mentioned above, it is computationally efficient and gives a linear based mapping of the pixels. But overall, I think that the type of histogram equalization to use depends on the type of image to be enhanced and the computation resources at hand.

Problem 2 : Image Denoising

a. Basic Denoising Methods Using Linear Filters

I. Abstract and Motivation

An important aspect of image processing is removing noise and one such way is by using filters. In the real world, data can become corrupted or have some layers of impurities, either from bad signal reception or transmission, strong interference along the way or transmitting via a faulty medium.

Noise can be additive in that at different levels of transmission, a layer of noise is added which leads to mixed noise data. Noise elimination is used to generate a cleaner version of the original data, without a large data loss. It also means that the errors in original digital image are corrected on the receiving side of the transmission.

1) Uniform Weight Function

A uniform weight filter is a type of linear filter used to eliminate noise. More specifically, it is a mean filter that works by getting the mean of the pixels in a vicinity (for each pixel). This eliminates pixels with high-frequency, hence it is a low pass filter.

A low-pass filter kernel or sliding window is used and convolved with each pixel. It uses a uniform weight function below to generate its kernel of size N x N (usually odd) given below :

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

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

Fig 5

2) Gaussian Weight Function

The gaussian-weight function-based filter is an extension of the uniform weight filter. Instead of a uniform distribution of kernel weights, it has a gaussian-type weight distribution, so the center of the kernel has higher weights and pixels farther away from the center have smaller weights. The weight function is given by the formula below:

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

Fig 6

One metric to compare the performance of denoised data for different filters or methods, is the peak-signal-to-noise ratio (PSNR) given below :

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

$$\text{where MSE} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(Y(i,j) - X(i,j)\right)^2$$

X : Original Noise-free Image of size $N \times M$

Y : Filterd Image of size $N \times M$

Max: Maximum possible pixel intensity = 255

Fig 7

A larger value for PSNR of a noisy image implies a good performance of the denoising algorithm. I will be using this metric to analyze my results.

3) Bilateral Filtering

The Bilateral filter is a non-linear filter which is an extension of the gaussian filter. It uses the concept of checking the difference between the values of the pixels in its vicinity and at the center to assert a sharp change in the pixel value for large differences, and to assert similarity in pixel values for small differences. If the former occurs, it reduces the weight through some calculations as it is a region or pixel that is an edge. The bilateral filter has better image preserving quality than either gaussian or uniform weight filters, so basically, it has a better edge preservation quality. A convolution kernel is used to process each pixel value at each pixel location.

A bilateral filter denoises and preserves the quality of each pixel by updating its kernel weights given by the formula below:

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

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

SigmaC and SigmaS are two weights that are varied to tune the quality of the filtered image output.

II. Approach and Procedures

The steps for **Uniform** filer are summarized to :

- STEP 1 : Run program (check README for instructions).
- STEP 2: Read image file into a 2D array since it is grayscale
- STEP 3: Generate a filter of size N , uniform weights. Start with 1
- STEP 4: Normalize the filter / weights and then convolve this sliding window with each pixel
- STEP 5: For each pixel, get the distance between neighboring pixels and the pixels with noise.
- STEP 6 : update the pixel values for the image with the new values gotten from the convolution.
- STEP 7 : Calculate the PSNR (dB).
- STEP 7 : Write final output in a raw file which is viewed on imageJ.

The steps for **Gaussian** are summarized to :

- STEP 1 : Run program (check README for instructions).
- STEP 2: Read image file into a 2D array since it is grayscale
- STEP 3: Generate a filter of size N , using the gaussian weight equation in Fig 6
- STEP 4: Normalize the filter / weights and then convolve this sliding window with each pixel
- STEP 5: For each pixel, get the distance between neighboring pixels and the pixels with noise.
- STEP 6 : update the pixel values for the image with the new values gotten from the convolution.

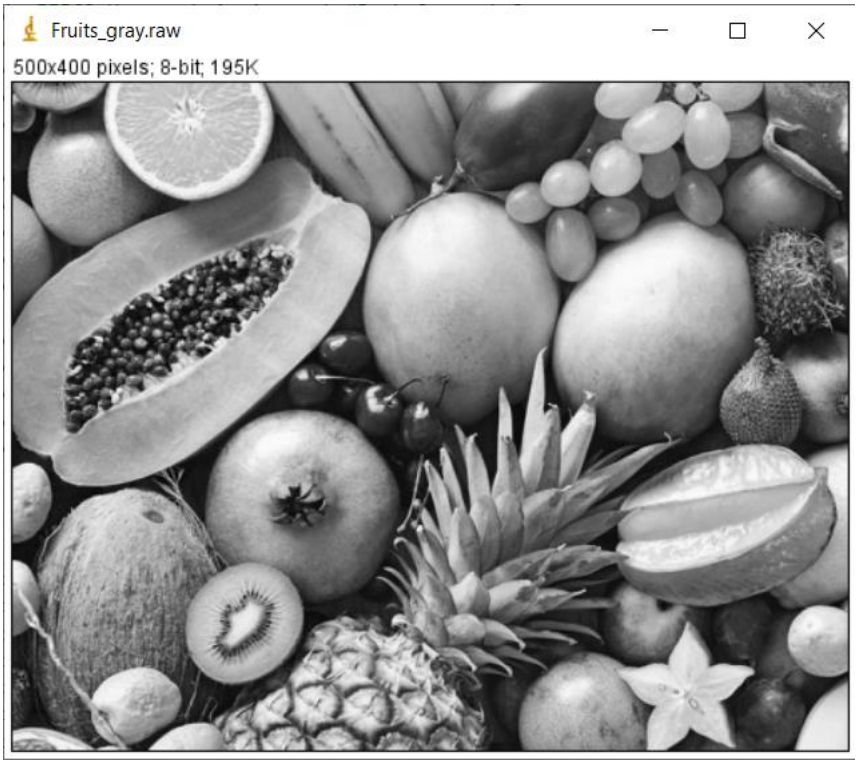
- STEP 7 : Calculate the PSNR (dB).
- STEP 7 : Write final output in a raw file which is viewed on imageJ.

One important aspect of the bilateral filter is that normalizes its weights and uses the two important parameters Sigmac and SigmaS to determine the size of pixels in its vicinity and the smallest amplitude value for an edge respectively. So, it removes or reduces the weights of pixels with intensity values less than the amplitude and keeps those that are not.

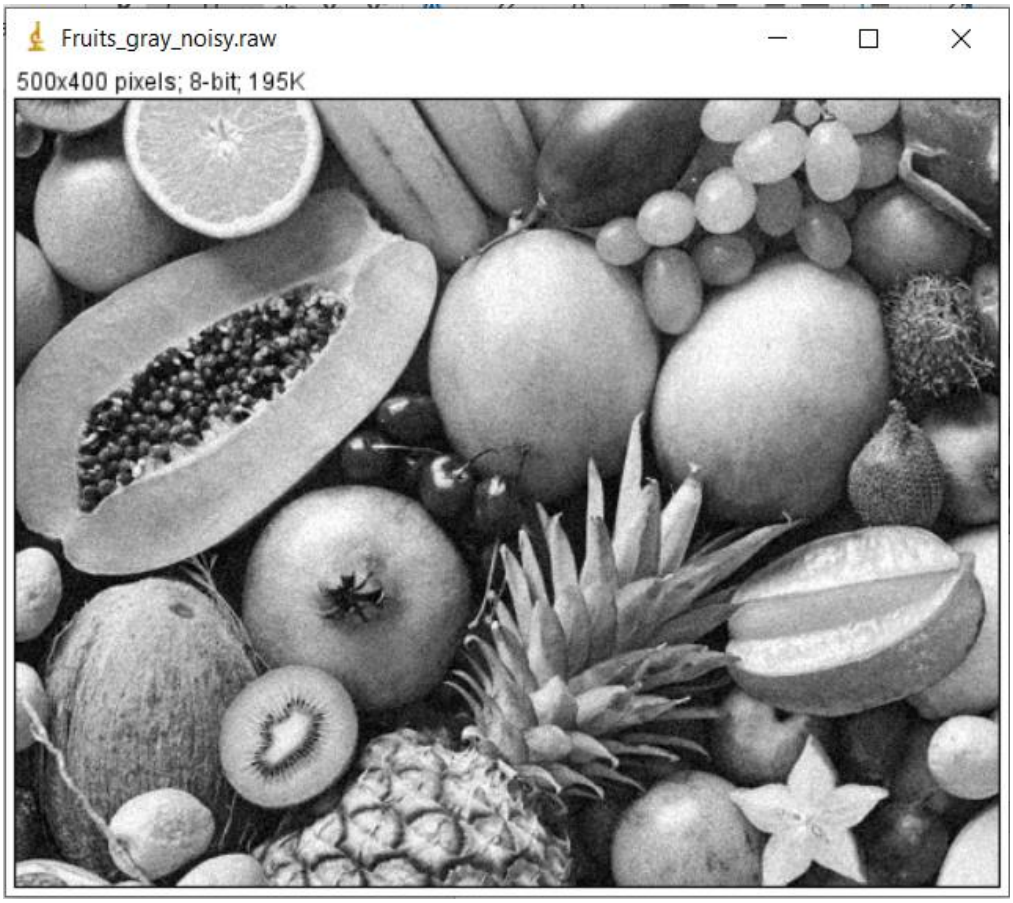
The steps for **Bilateral Filtering** are summarized to:

- STEP 1 : Run program (check README for instructions).
- STEP 2: Read image file into a 2D array since it is grayscale
- STEP 3: For each pixel, get the distance between neighboring pixels and the pixels with noise. This becomes the weights.
- STEP 4: Normalize the filter / weights and then convolve this sliding window with each pixel
- STEP 6 : update the pixel values for the image with the new values gotten from the convolution.
- STEP 7 : Calculate the PSNR (dB).
- STEP 7 : Write final output in a raw file which is viewed on imageJ.

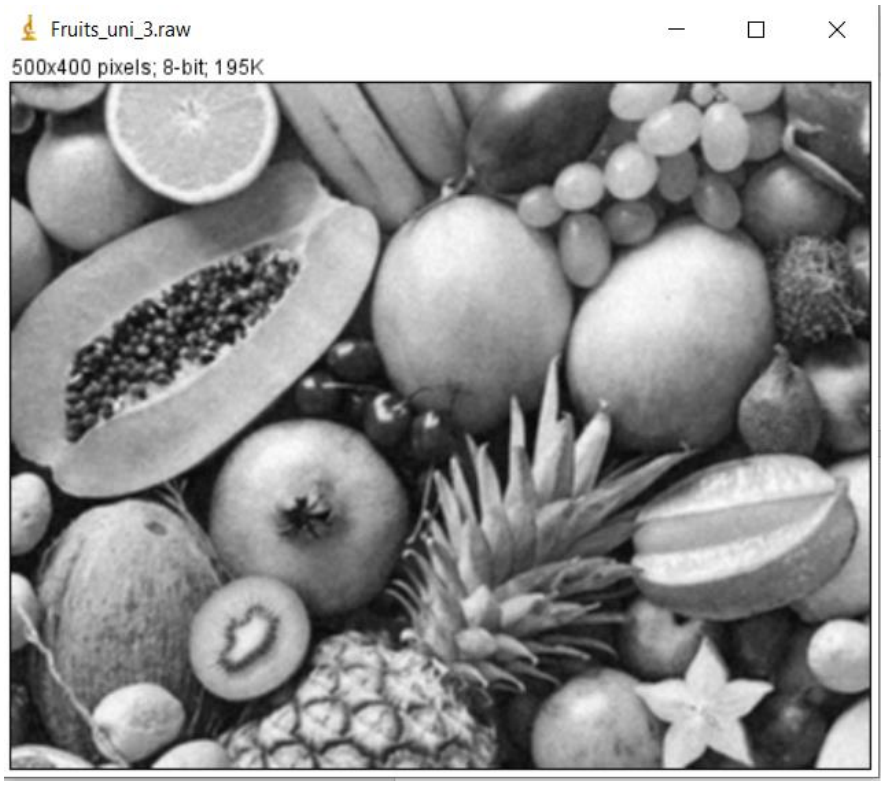
III. Experimental Results



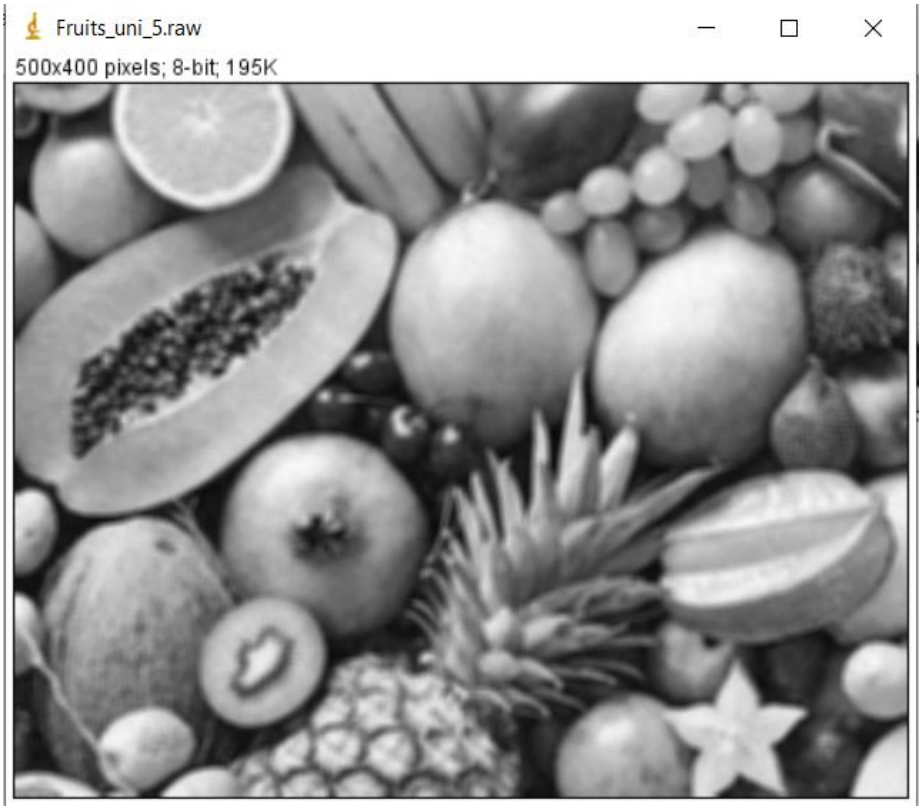
Original gray image



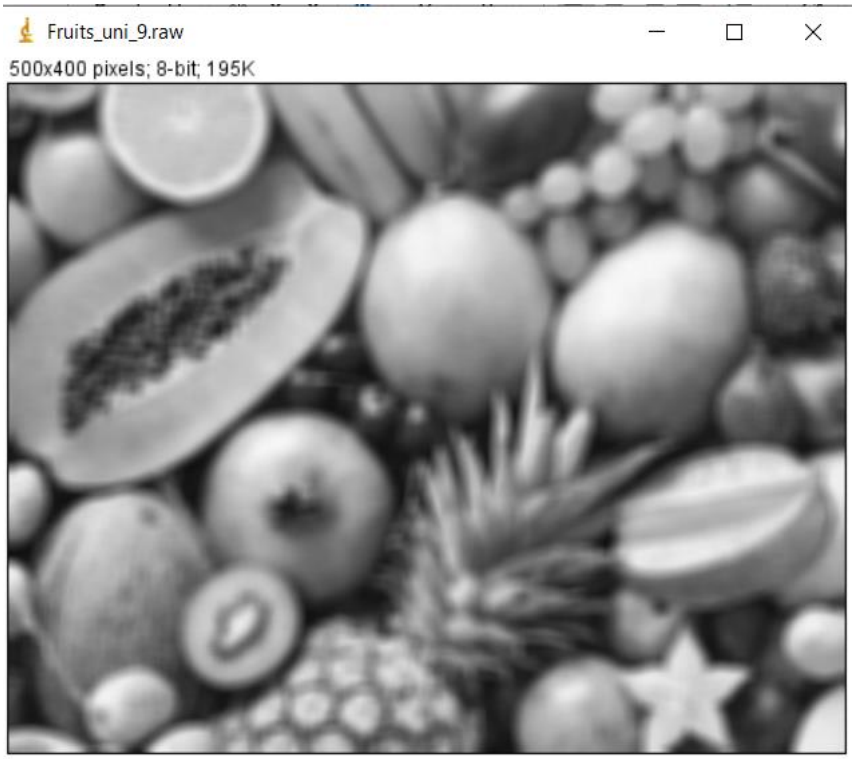
Original noisy gray image, PSNR for Noise Image =28.18dB



Uniform filter denoised, N = 3 PSNR for Filtered Image =27.004dB

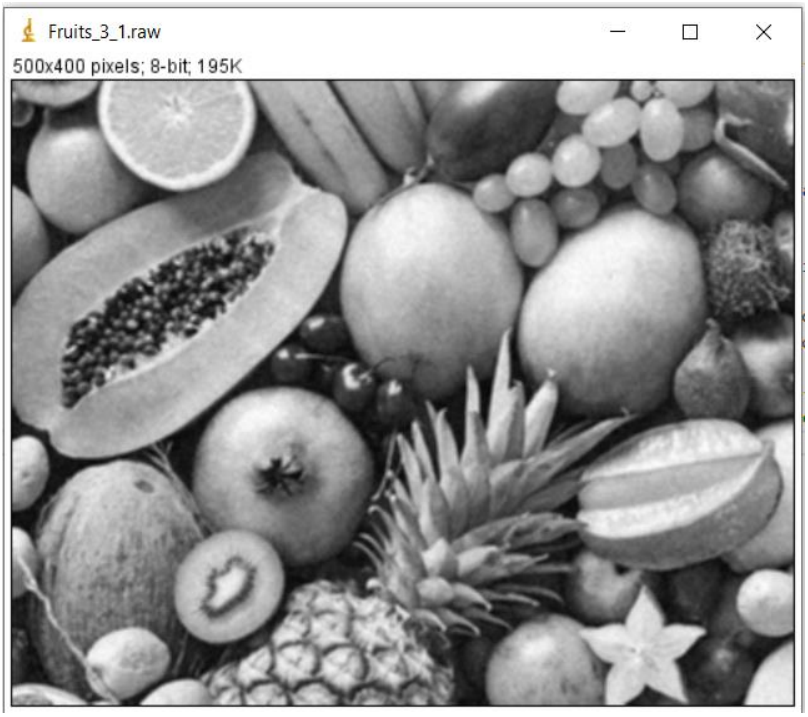


Uniform filter denoised, N = 5 PSNR for Filtered Image =23.744dB

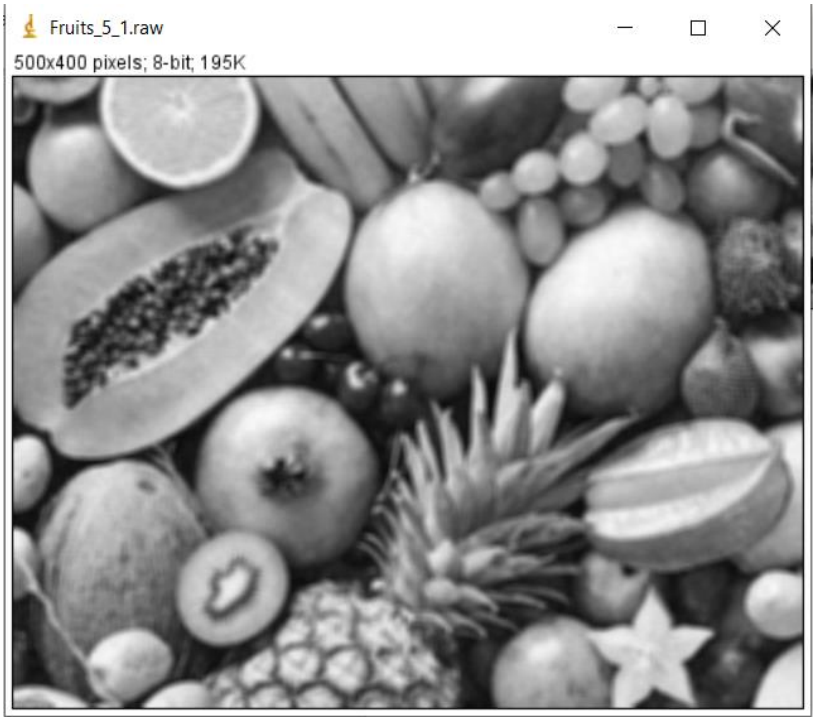


Uniform filter denoised, N = 9 PSNR for Filtered Image =20.777dB

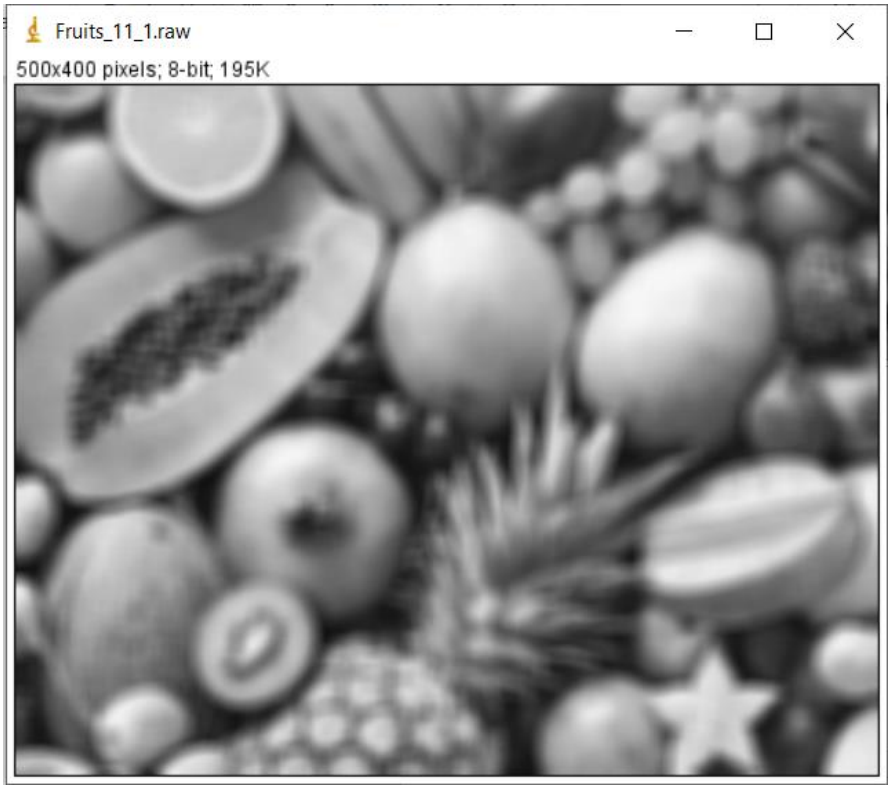
Uniform filter denoised, N = 13 PSNR for Filtered Image =19.173dB



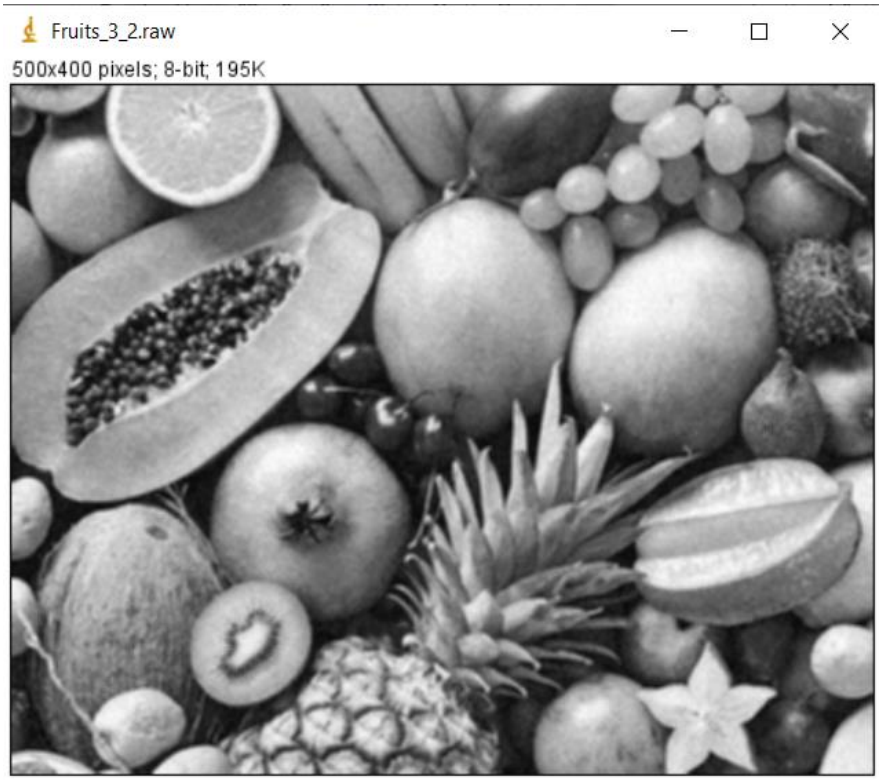
Gaussian filter denoised, N = 3, Sigma = 1.0 , PSNR for Filtered Image =27.004dB



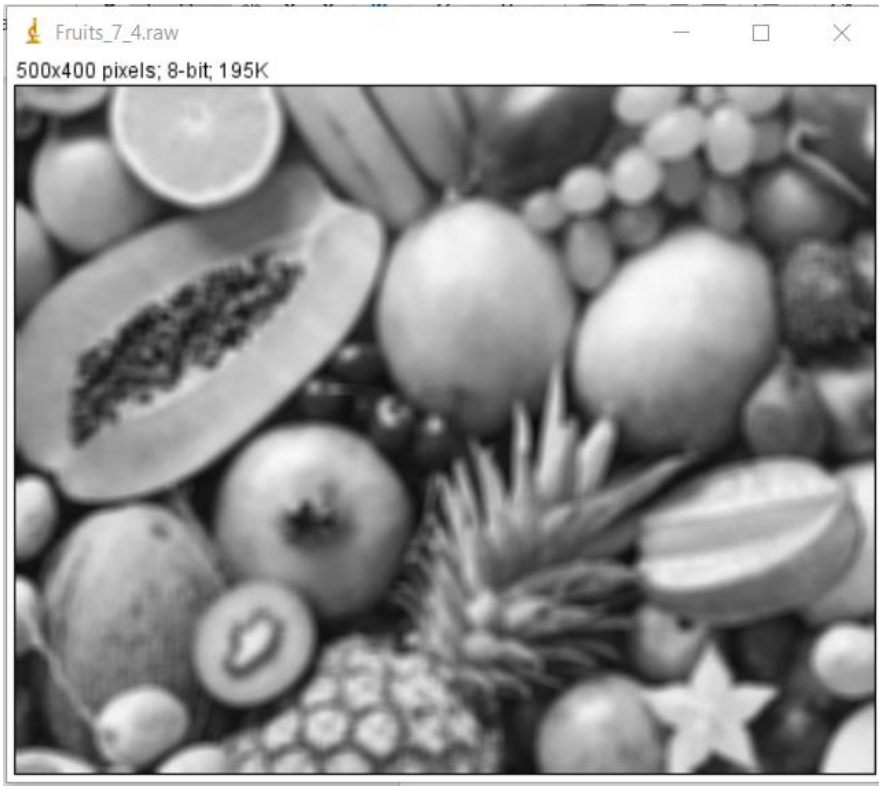
Gaussian filter denoised, N = 5, Sigma = 1.0, PSNR for Filtered Image =23.744dB



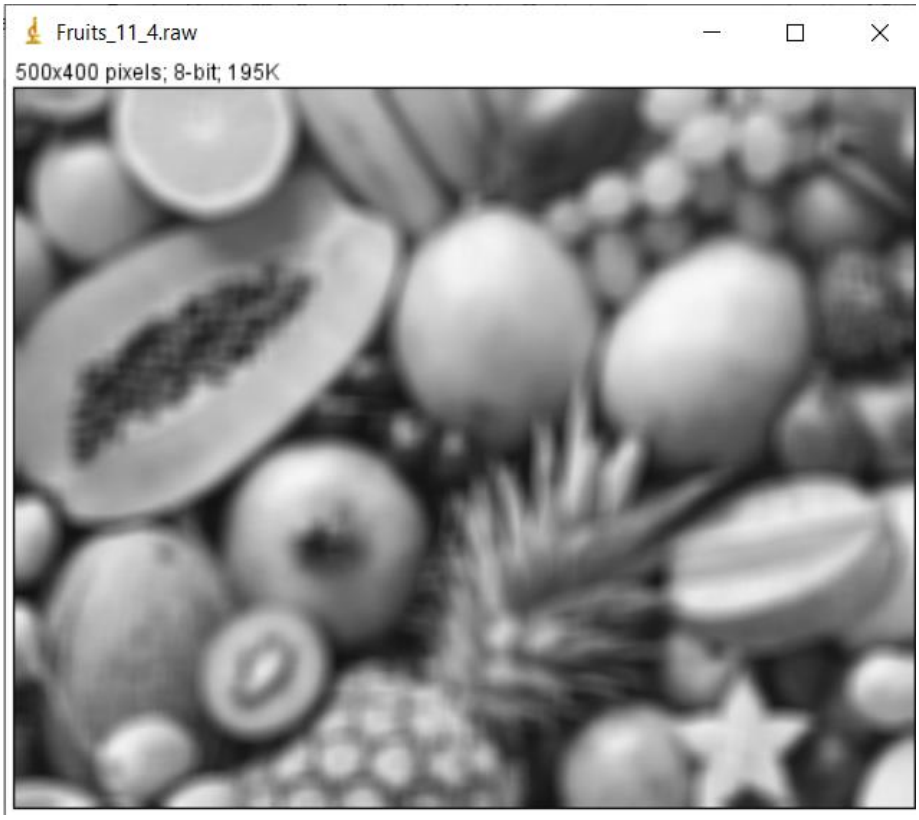
Gaussian filter denoised, N = 11, Sigma = 1.0, PSNR for Filtered Image =19.887dB



Gaussian filter denoised, N = 3, Sigma = 2.0, PSNR for Filtered Image =27.004dB

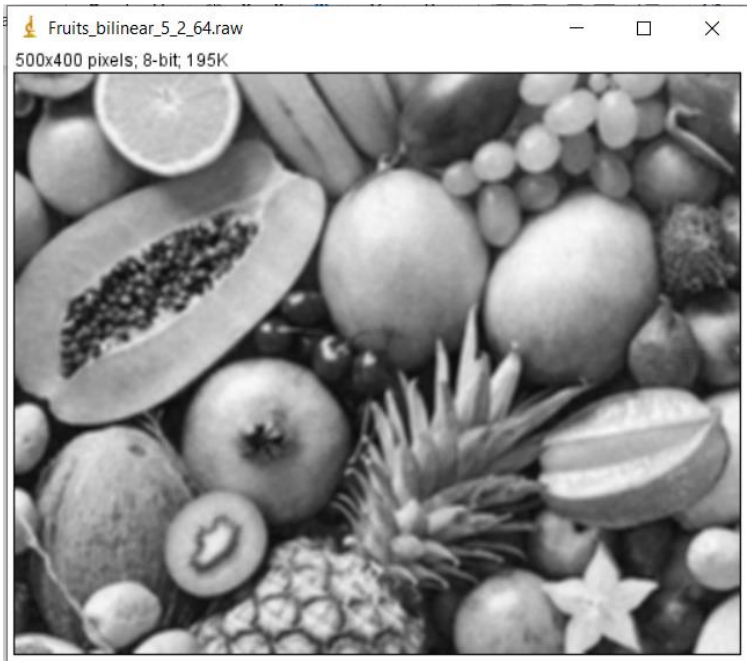


Gaussian filter denoised, N = 7, Sigma = 4.0, PSNR for Filtered Image =21.97dB

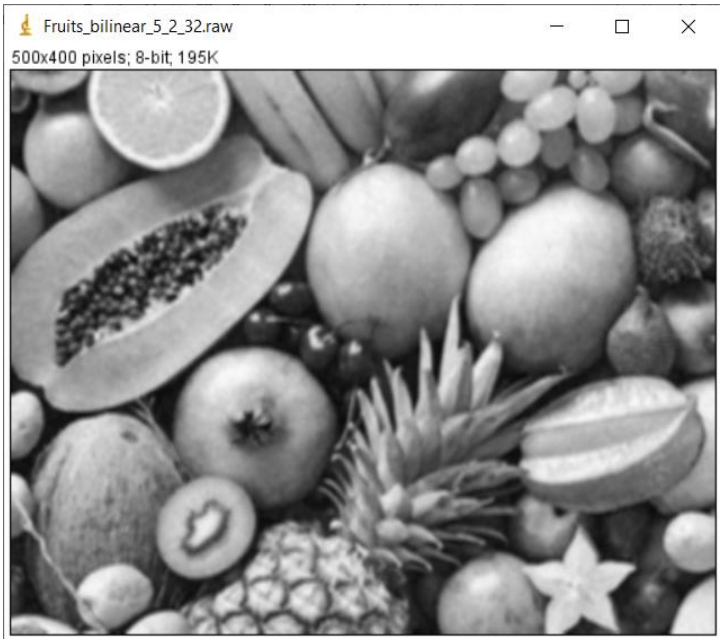


Gaussian filter denoised, N = 11, Sigma = 4.0, PSNR for Filtered Image =19.887dB

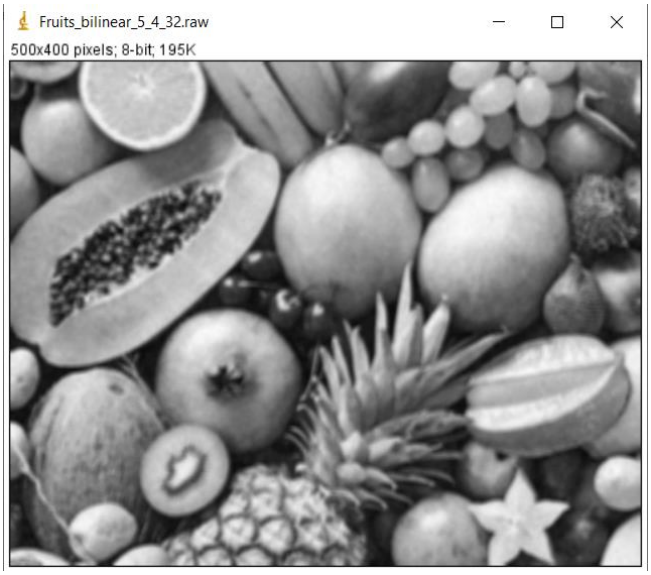
Bilateral Filter results :



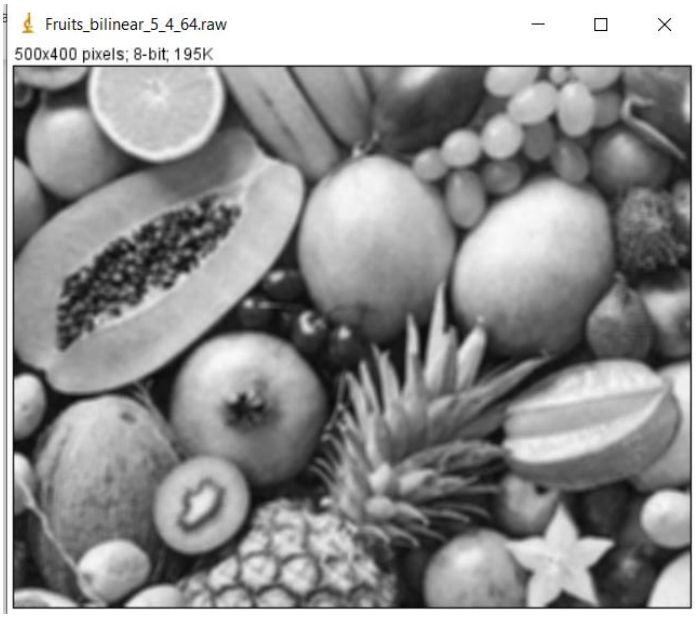

```
N = 5, SigmaC = 2.0, SigmaS = 64
PSNR for Noise Image =15.364dB
PSNR for filtered Image =15.348dB
```



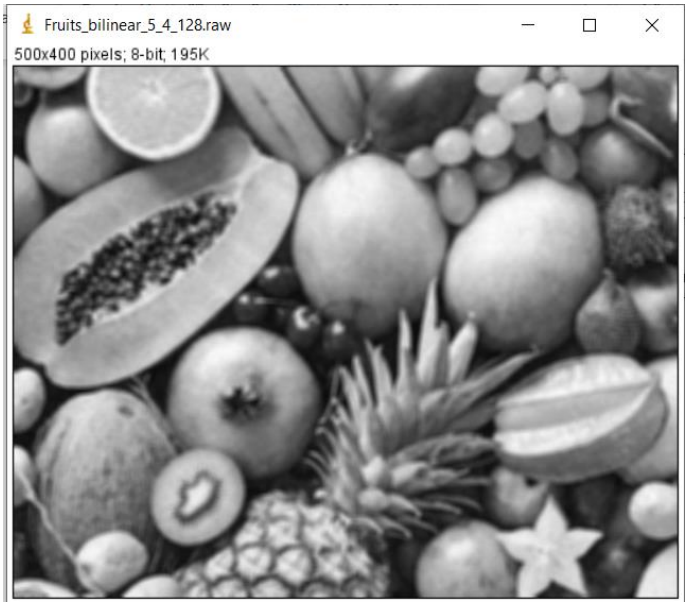

```
N = 5, SigmaC = 2.0, SigmaS = 32
PSNR for Noise Image =15.345dB
PSNR for filtered Image =15.335dB
```



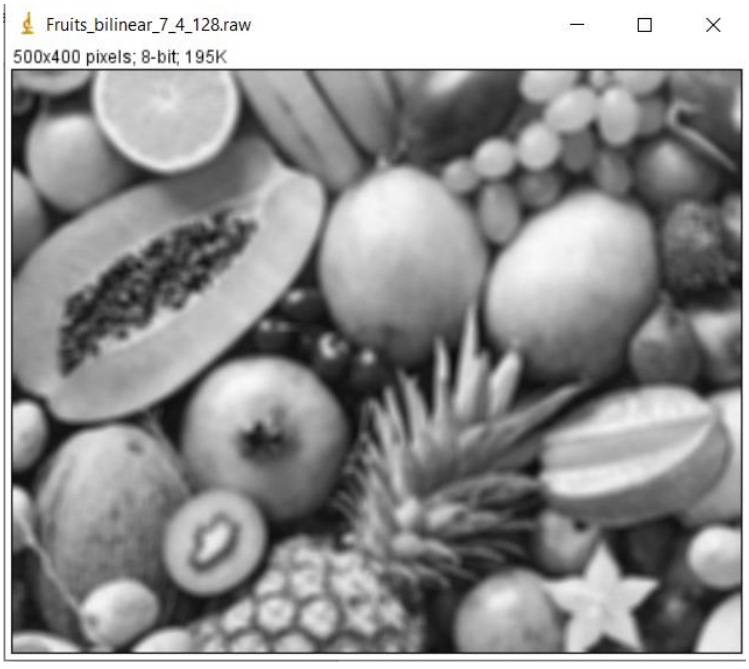
N = 5, SigmaC = 4.0, SigmaS = 32
PSNR for Noise Image =15.375dB
PSNR for filtered Image =15.312dB



N = 5, SigmaC = 4.0, SigmaS = 64
PSNR for Noise Image =15.394dB
PSNR for filtered Image =15.327dB



N = 5, SigmaC = 4.0, SigmaS = 128
PSNR for Noise Image = 15.4dB
PSNR for filtered Image =15.325dB



N = 7, SigmaC = 4.0, SigmaS = 128
PSNR for Noise Image =15.549dB
PSNR for filtered Image =15.337dB

IV. Discussion

For the linear filters, the value of the window size influences the performance of the denoised images. The window size or filter or filtering mask size takes odd values from 3. As N gets larger, say from 7, there is a noticeable blur in the images. This applies to the uniform weight filter and the gaussian weight filter. A good size for N is 5.

Another metric to use is the PSNR values, to evaluate the performance of the linear filters. From visual inspection and the PSNR values, there seems to be an inverse relationship. Larger values of N give a decreasing PSNR value. The Gaussian filter performs better than the mean filter at the same sizes and I believe this is possible due to the presence of the extra parameter, SigmaC which allows for a much better tunning for denoising images.

For the Bilateral filter is an extension of the gaussian filter and uses gaussian weights distribution to perform its noise removal. From the observations in the results, for the same value of N, I observed that the result is less blurry than that obtained from the gaussian and uniform weights, and the edges are not overly distorted. Essentially, the images are more defined. the parametes SigmaC and SigmaS allow for more tuning of the gaussian weights distribution. SigmaC and SigmaS are parameters that control how “smooth” the final image is as I observe from the experimental results. Increasing SigmaC allows more features to be smoothened.

Problem 3 : Special Effects Image Filters :

Creating Oil Painting Effect

I. Abstract and Motivation

The basic idea of color quantization is to have a reduction of the number of colors palettes that exist in an image. And this can find its use in creating filters or processors for devices that have a limited color range representation. One important aspect of image processing is image filtering.

Depending on the method and parameter tunning, a different number of filter effects can be achieved. One such is the oil painting effect – an effect which visually, seem to reduce the number of color distribution in an image. The oil painting effect can be applied as a filter for photos as seen in popular photo processing applications in mobile phone technology. The idea of creating the oil painting effect is to do the quantization of colors and apply that to an image.

II. Approach and Procedures

The image files are read into an array for computation processing. The first step is to extract each of the color channel, and this is done with the ***separateChannels()*** function. The result of this function ensures that the image file data is separated into the 3 color channels R,G, and B.

For quantization, the idea is to find, for each color plane, the color with the highest frequency in a window size (in the N x N vicinity) and then use the bucket filling method to divide the color levels into separate bins with the same number of pixels. The weighted average of each bin is used to replace the pixels back into each bin. Next is the special effect function which is a filter that takes each quantized color channel data, finds the color with the most frequency and replaces it with that color – so the image is quantized and has a reduced color space.

Overall, the steps are summarized to :

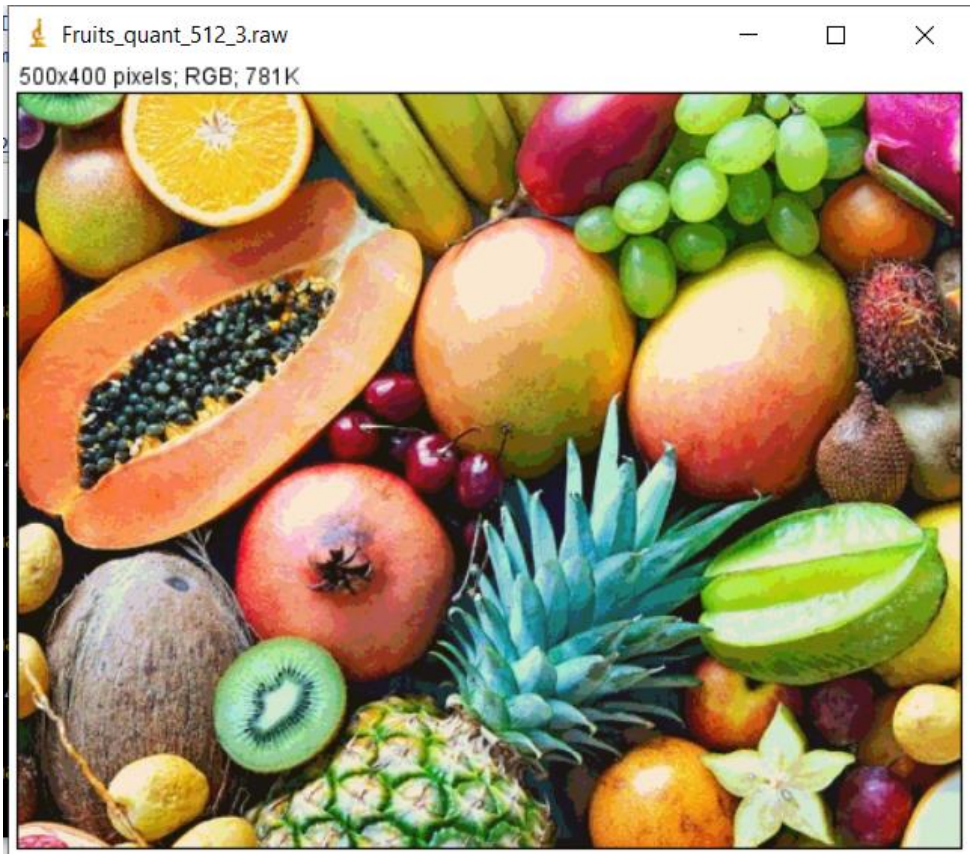
- STEP 1 : Run program (check README for instructions).
- STEP 2: Read image file into a 1D array
- STEP 3 : Transform it into a 3D array for easy access and manipulation.
- STEP 4: Extract each color place of Red, Green and Blue.
- STEP 5 : Quantize the image data of each plane.
    - Access each pixel and get the histogram of each plane.

    - Get the mean of each bin and apply weighted mean at each pixel location and quantize.
- STEP 6: Apply the filter for the oil effect
- Combine all the quantized color planes back as a 1D array. This is saved in a .raw file which is viewed on imageJ. Do the same for filtered color planes.

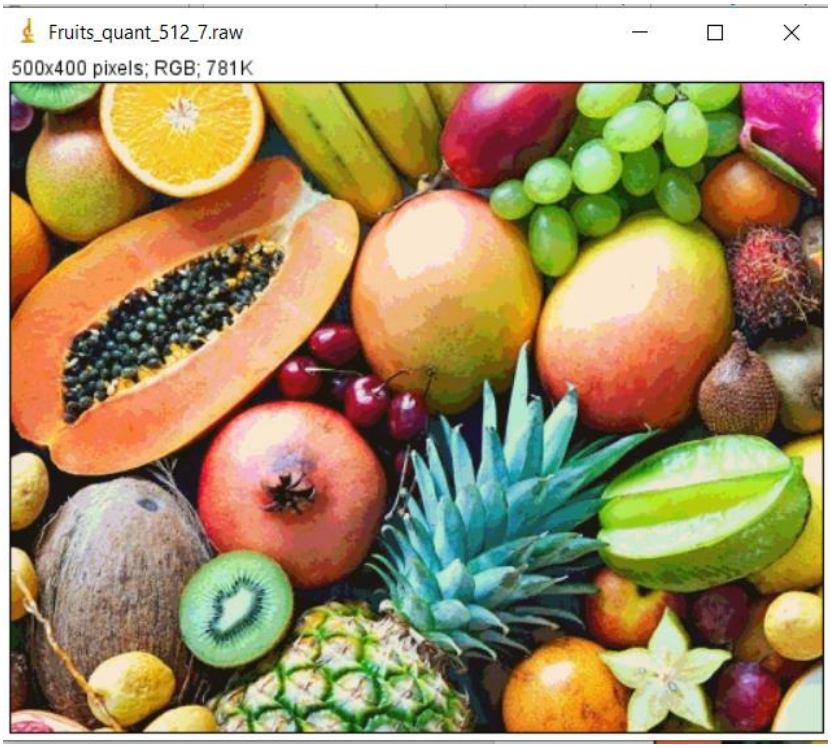
III. Experimental Results

Values for **N**, the window size was varied starting from 3 through 11, only for odd values. Values for quantization level were also varied for 64 and 512. Results are shown below.

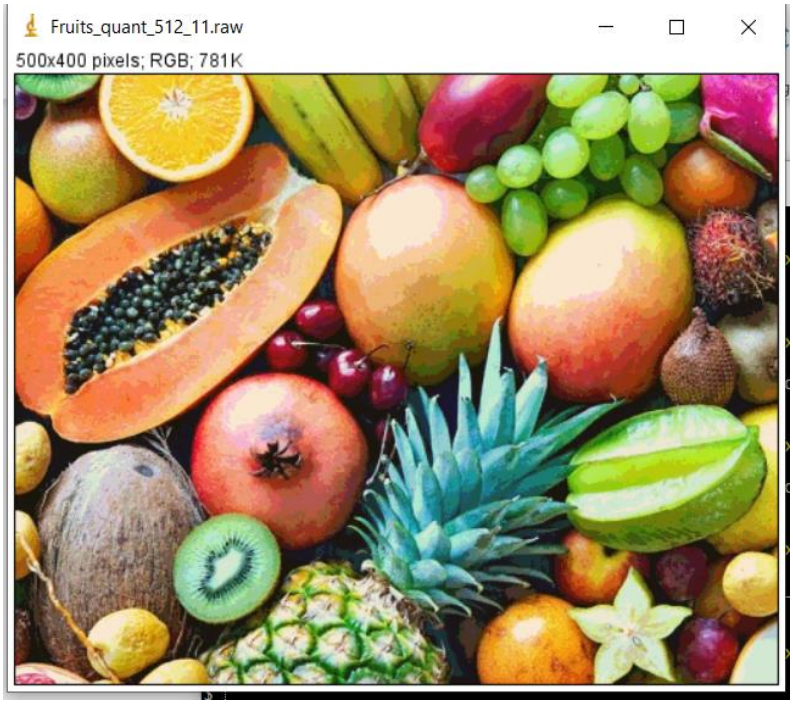
Quantized, 512 colors, N = 3



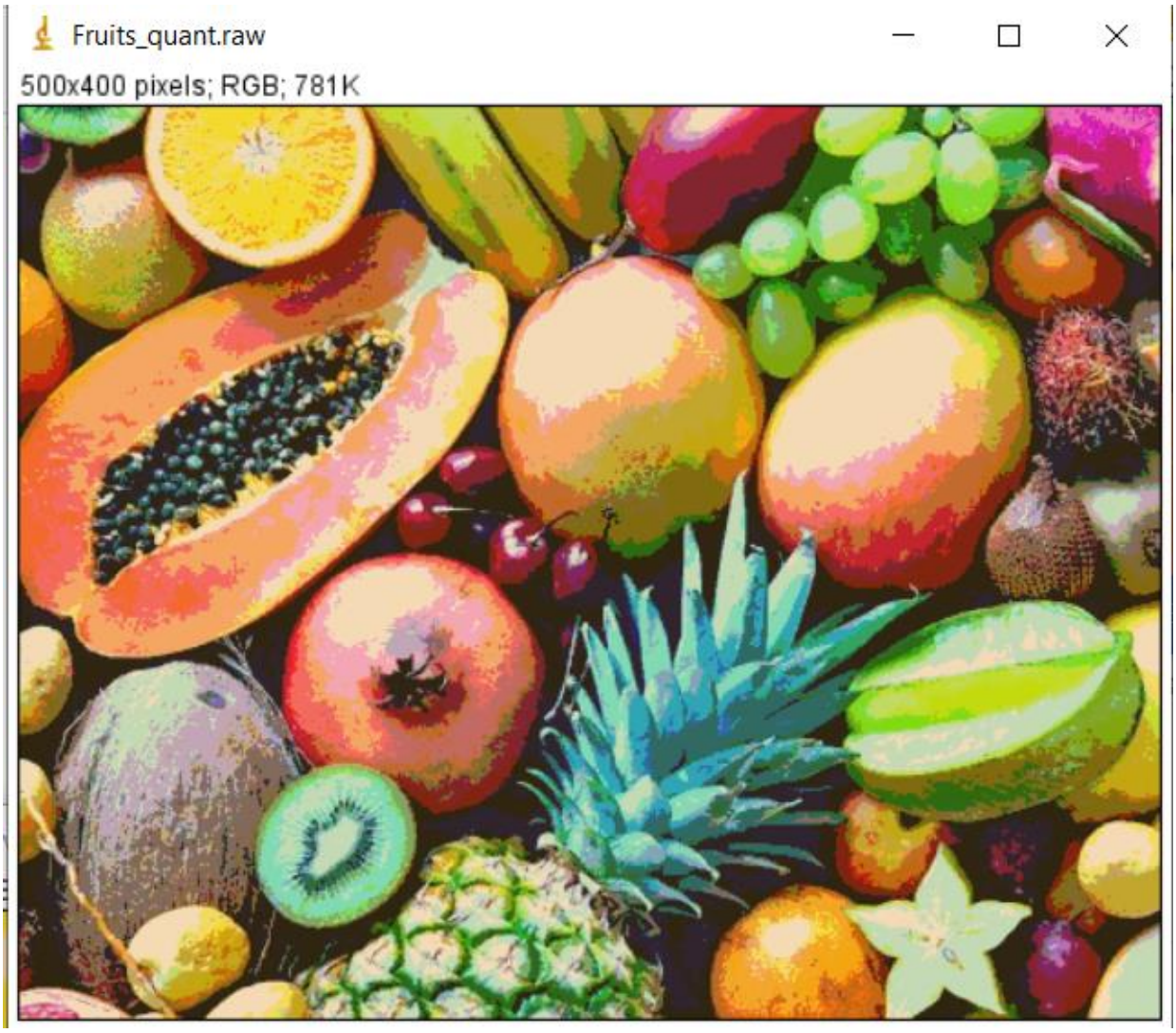
Quantized, 512 colors, N = 7



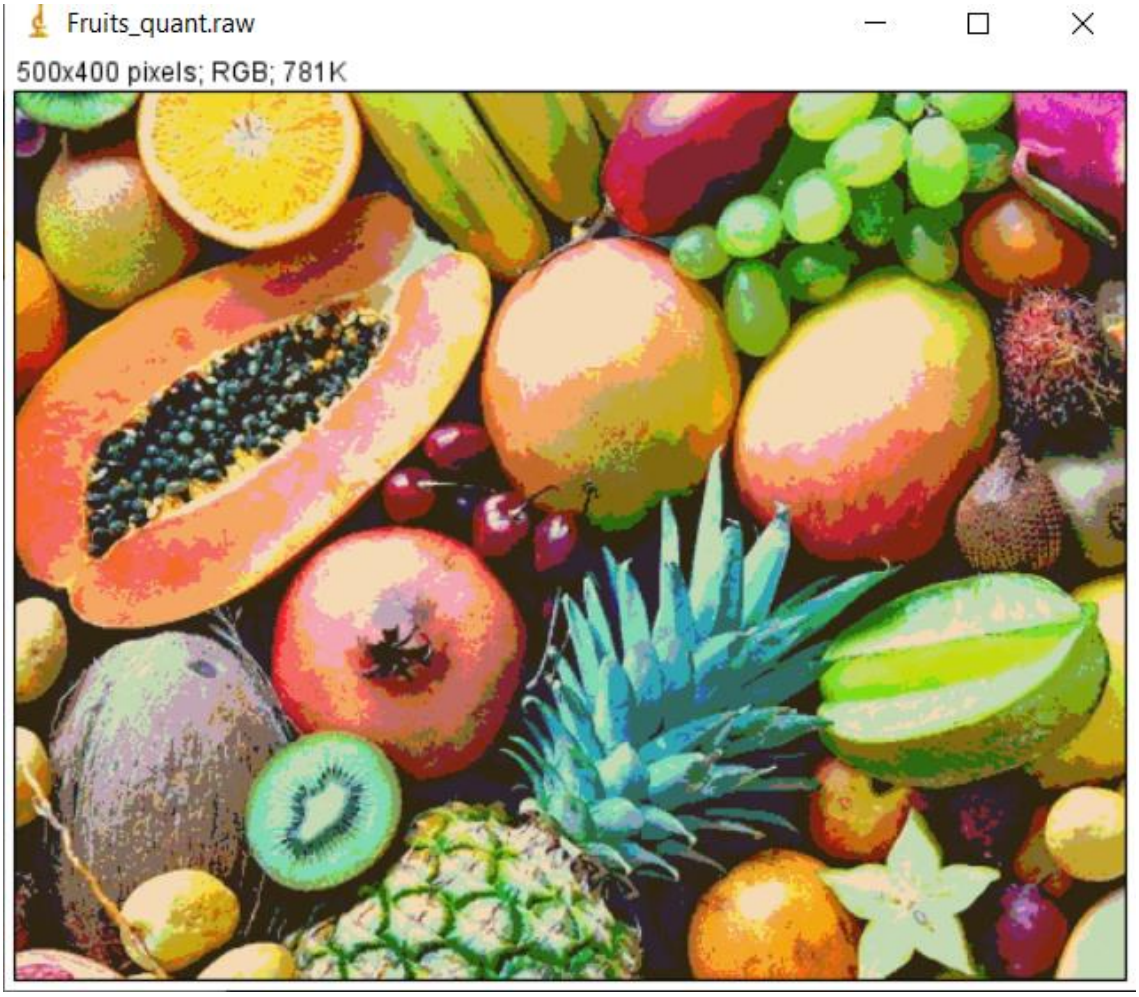
Quantized, 512 colors, N = 11



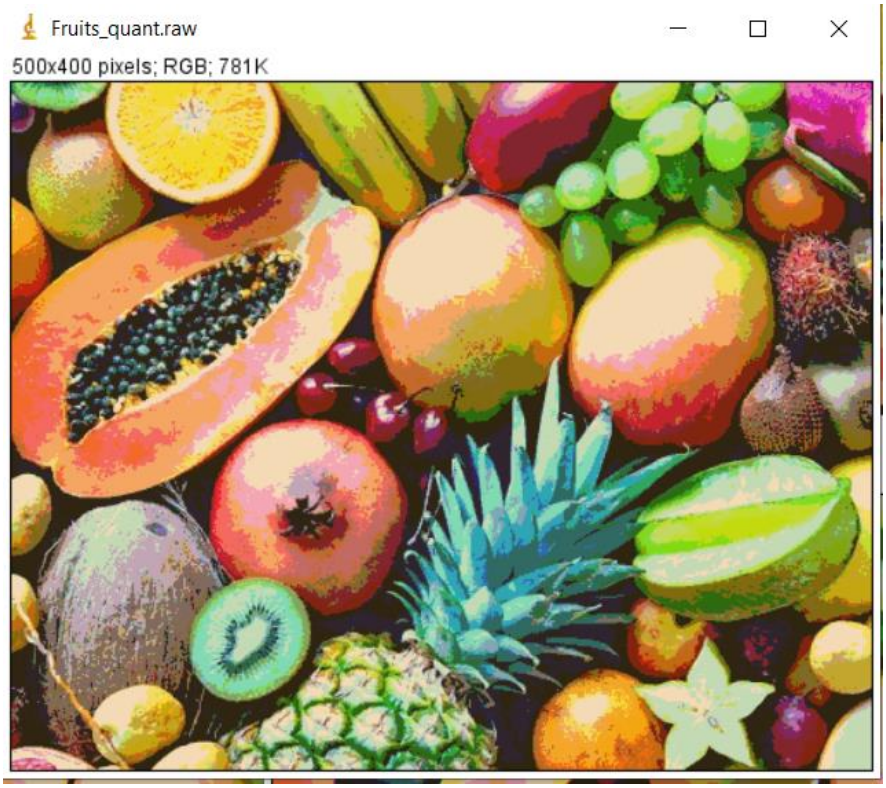
Quantized, 64 colors, N = 5



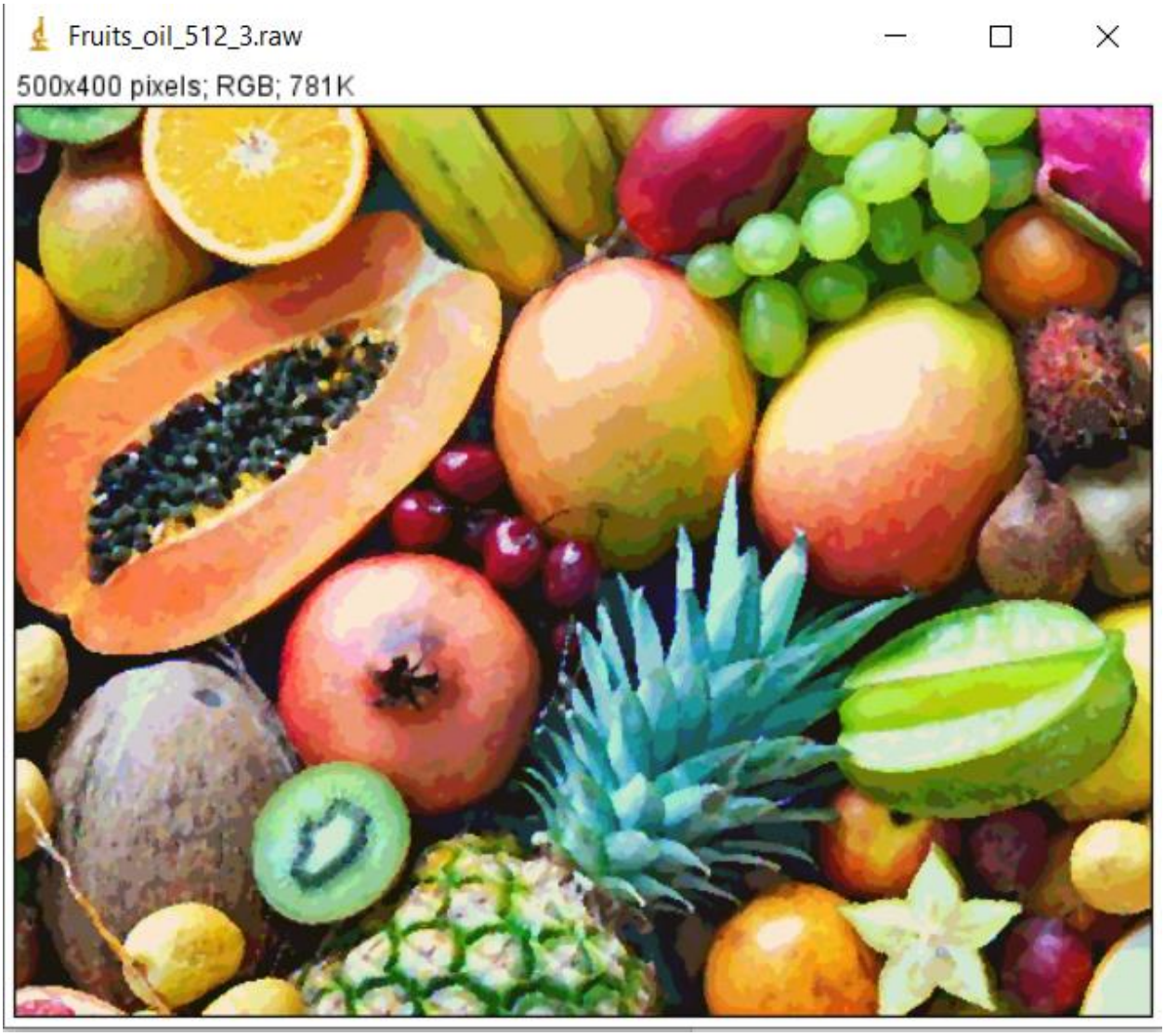
Quantized, 64 colors, N = 7



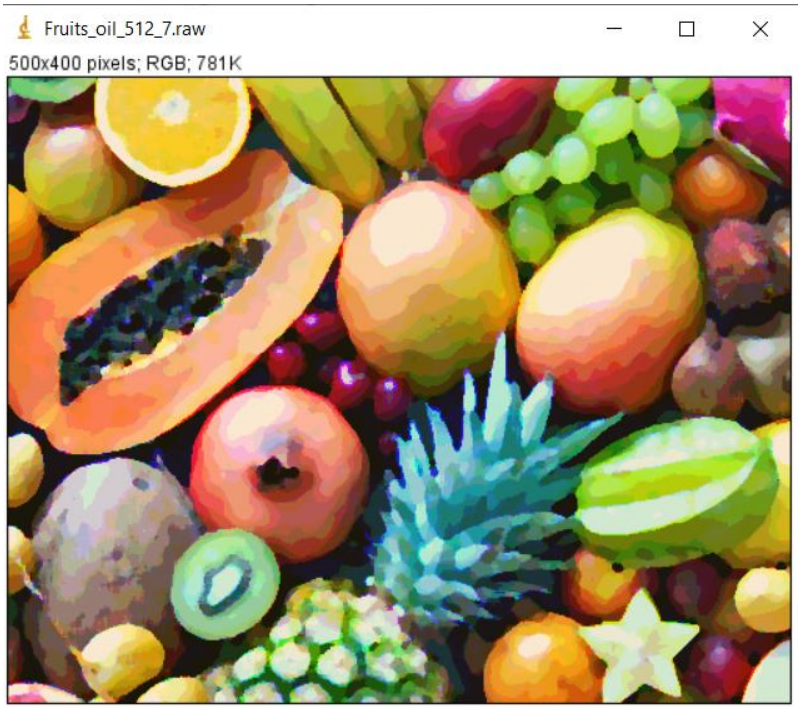
Quantized, 64 colors, N = 11



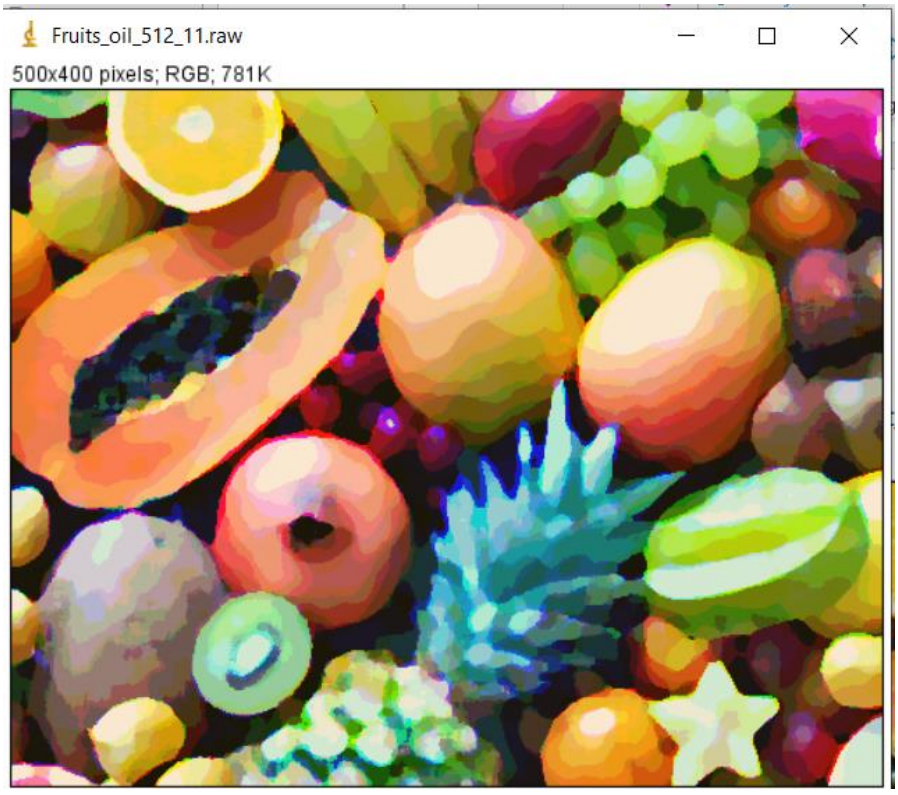
Oil Effect, 512 colors, N = 3



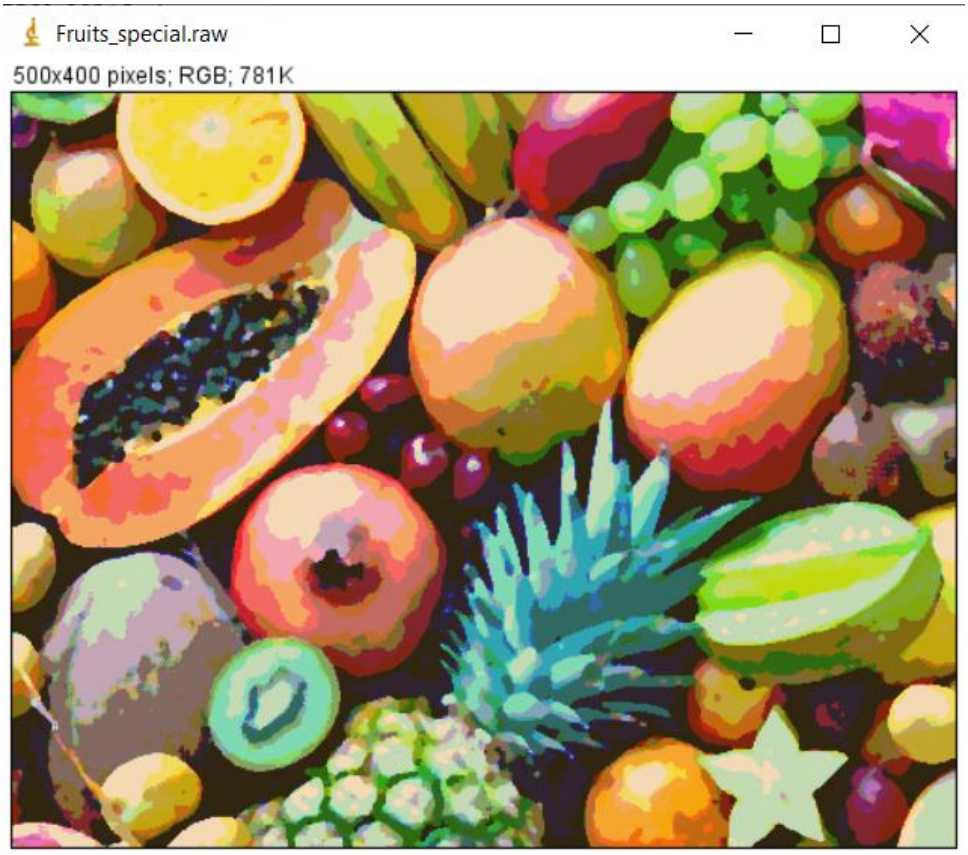
Oil Effect, 512 colors, N = 7



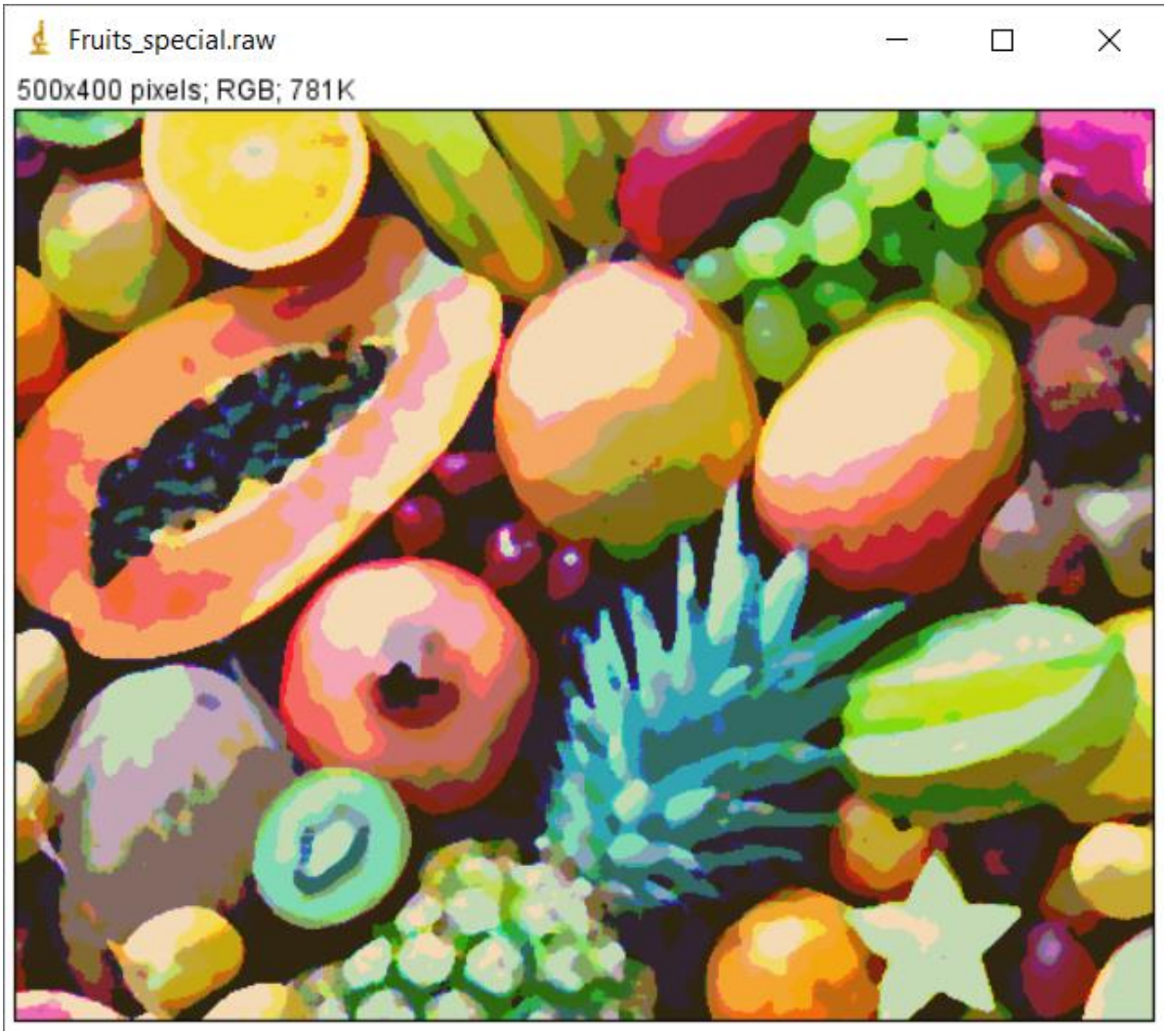
Oil Effect, 512 colors, N = 11



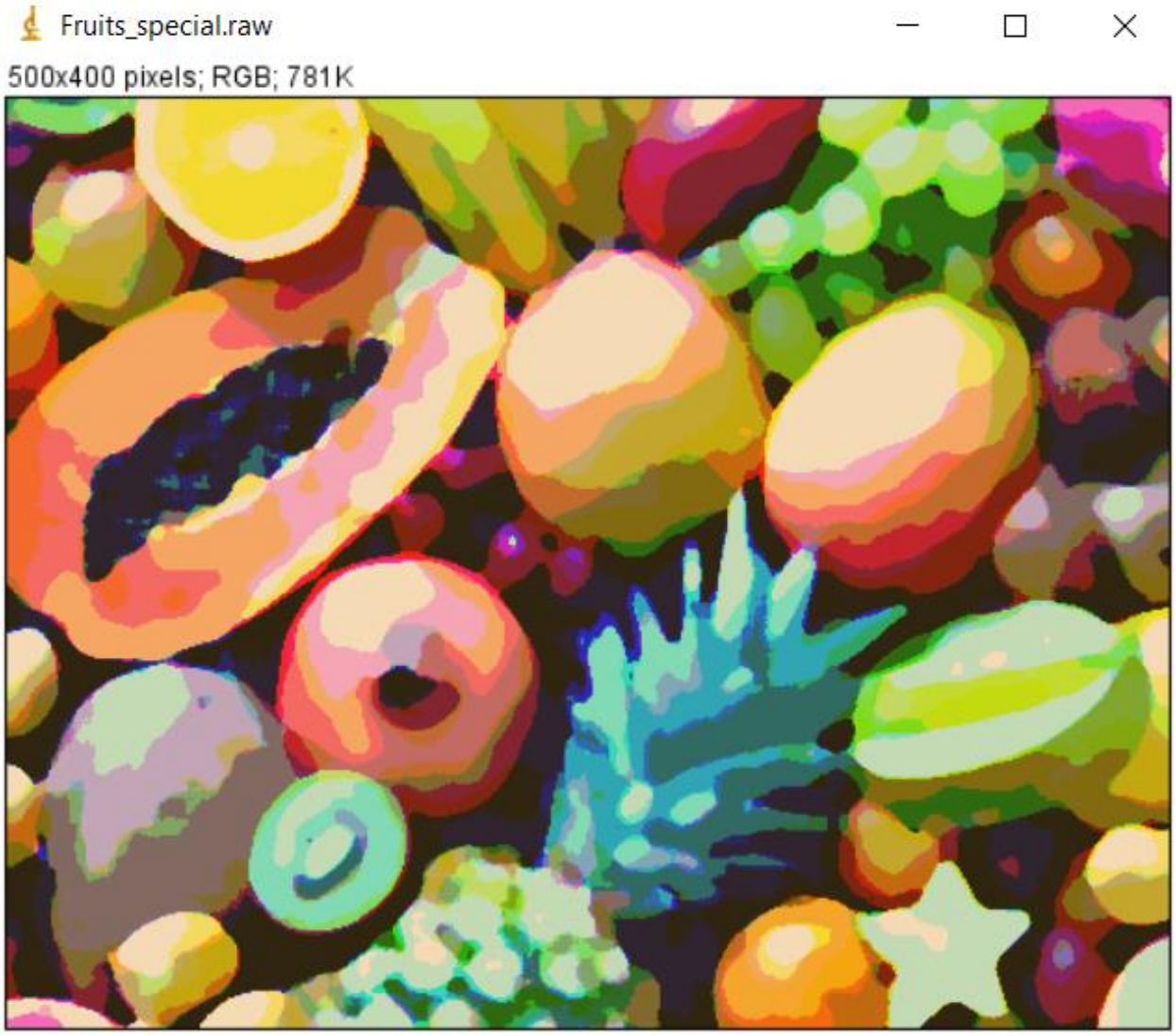
Oil Effect, 64 colors, N = 5



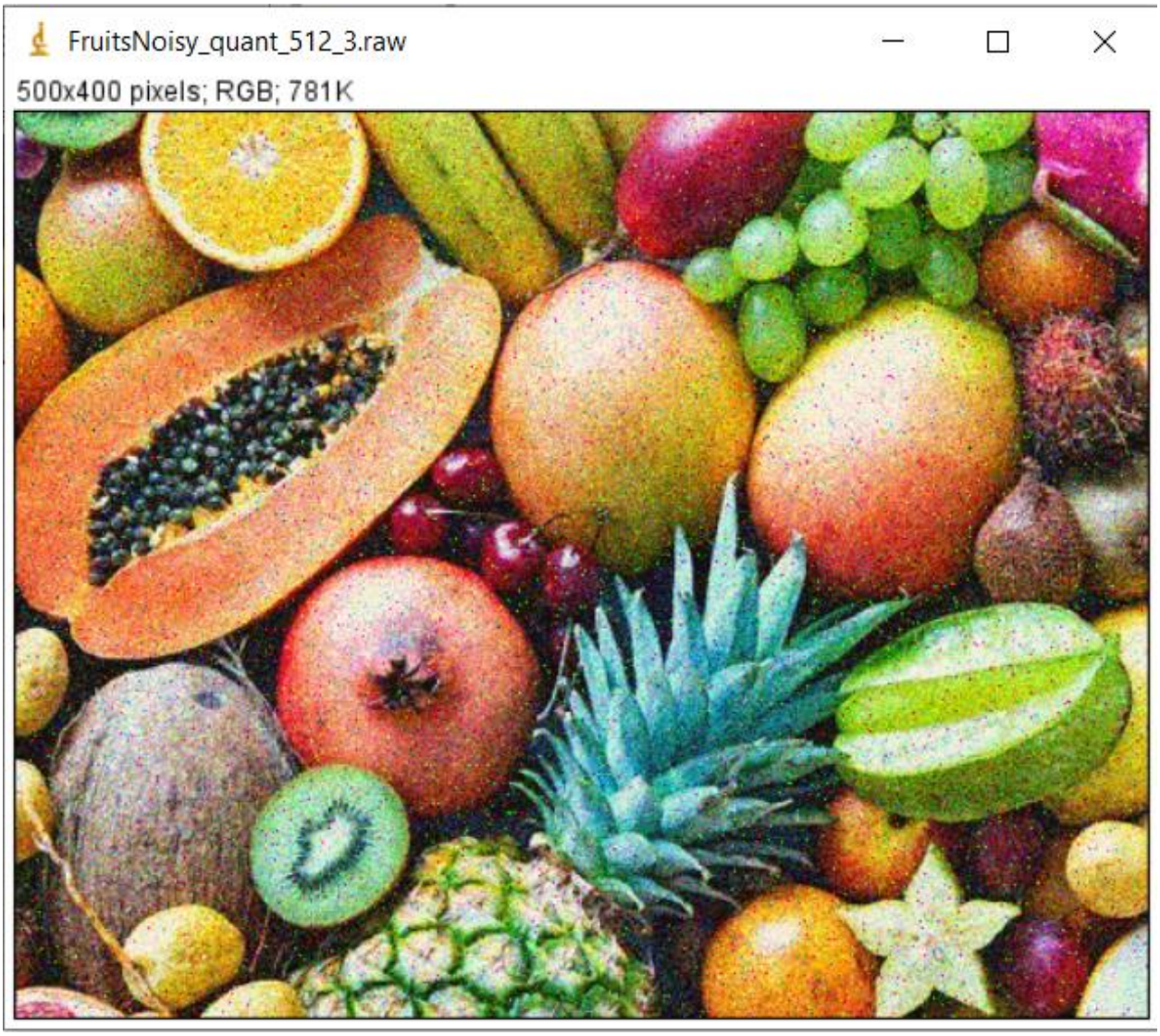
Oil Effect, 64 colors, N = 7



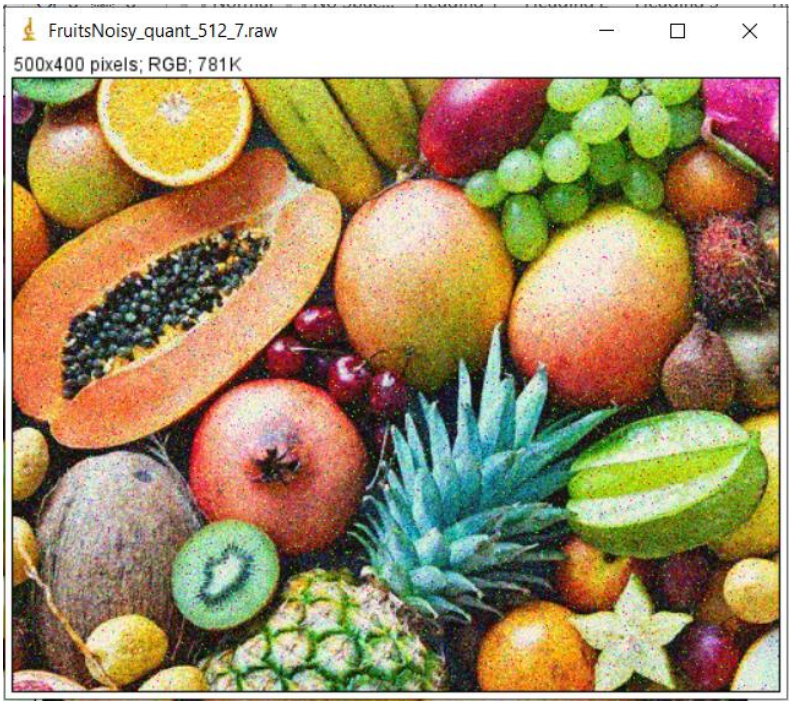
Oil Effect, 64 colors, N = 11



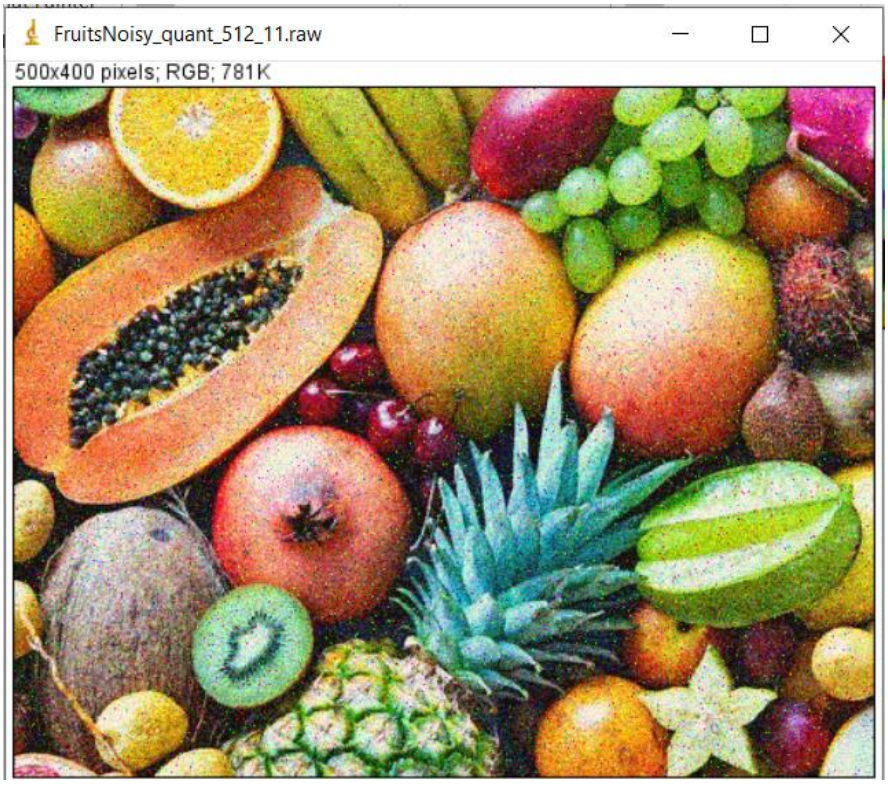
Quantized, 512 colors, N = 3 on Noisy image



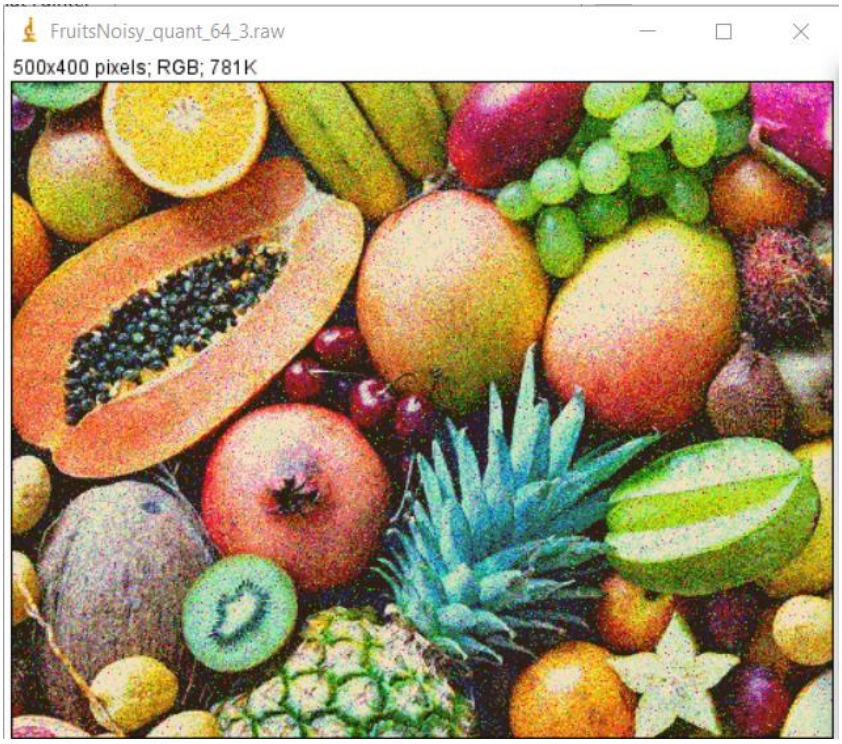
Quantized, 512 colors, N = 7 on Noisy image



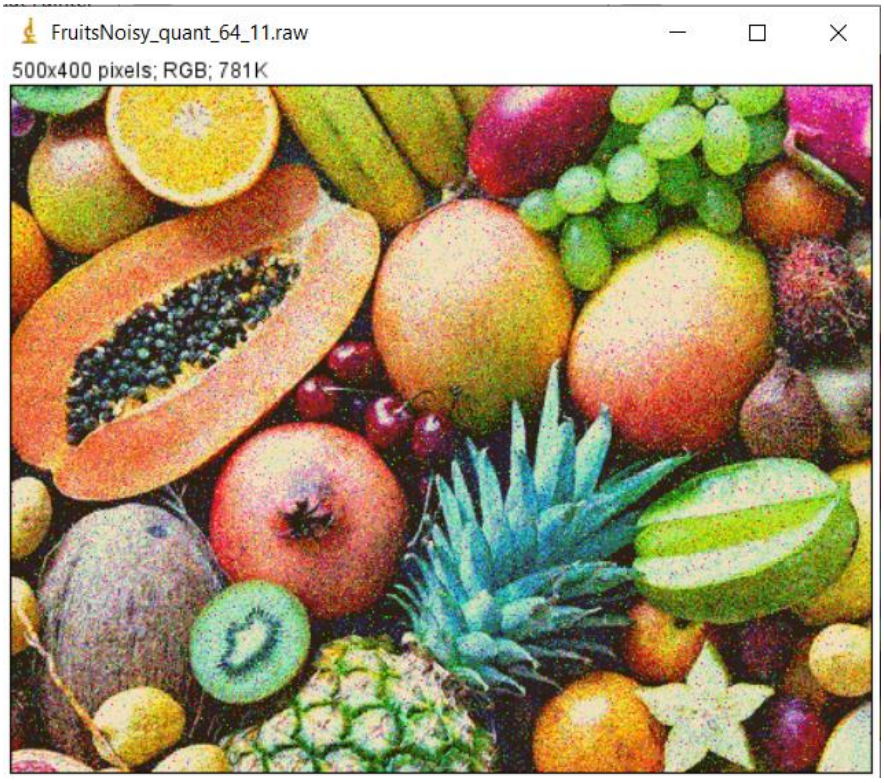
Quantized, 512 colors, N = 11 on Noisy image



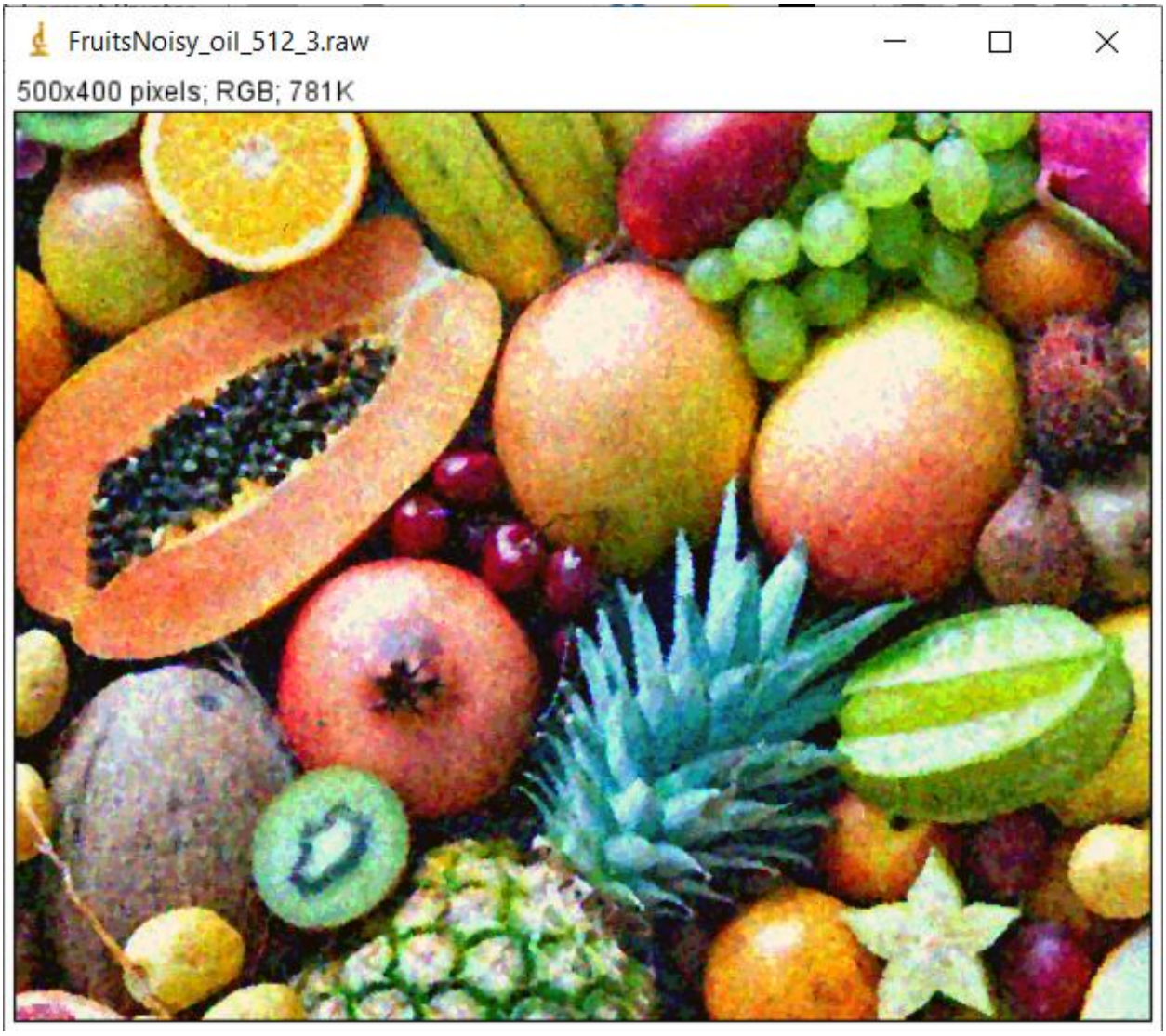
Quantized, 64 colors, N = 3 on Noisy image



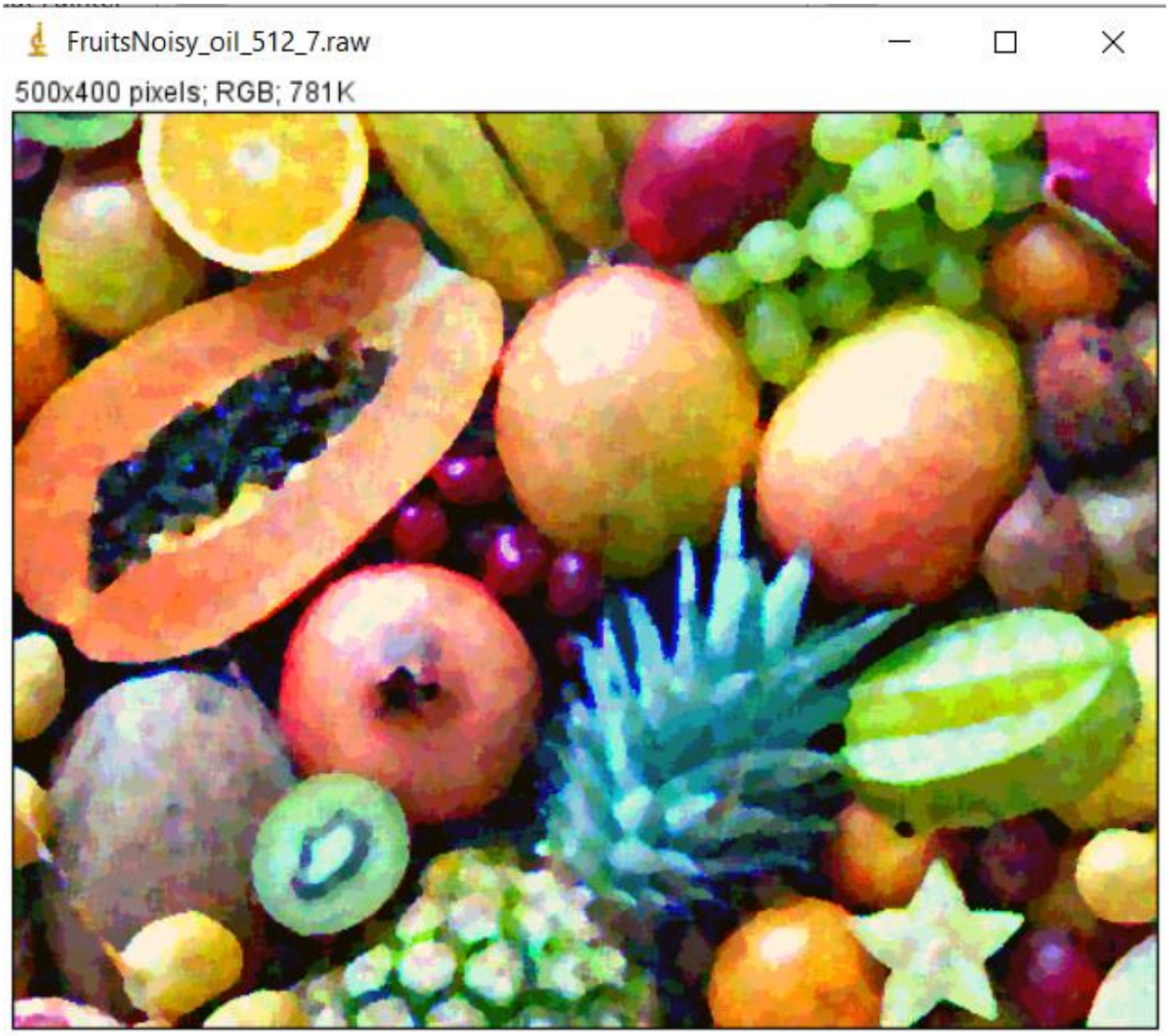
Quantized, 64 colors, N = 11 on Noisy image



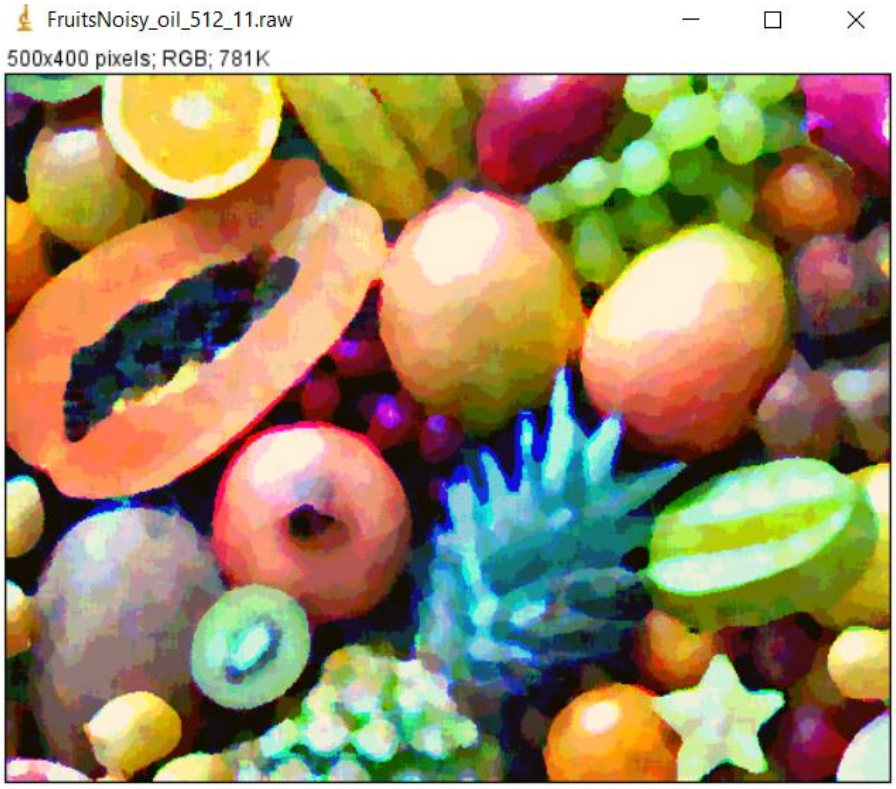
Oil Effect, 512 colors, N = 3 on Noisy image



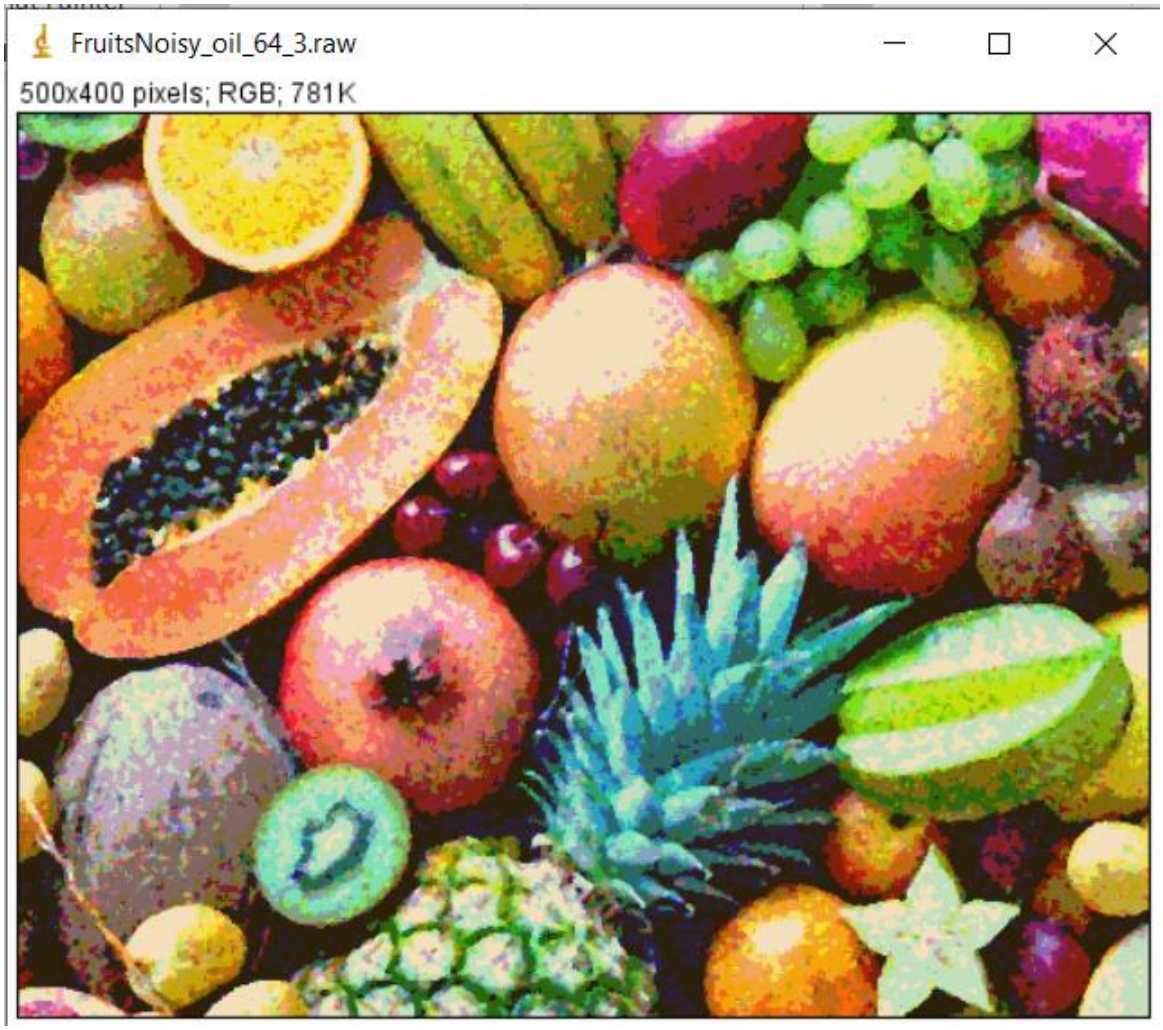
Oil Effect, 512 colors, N = 7 on Noisy image



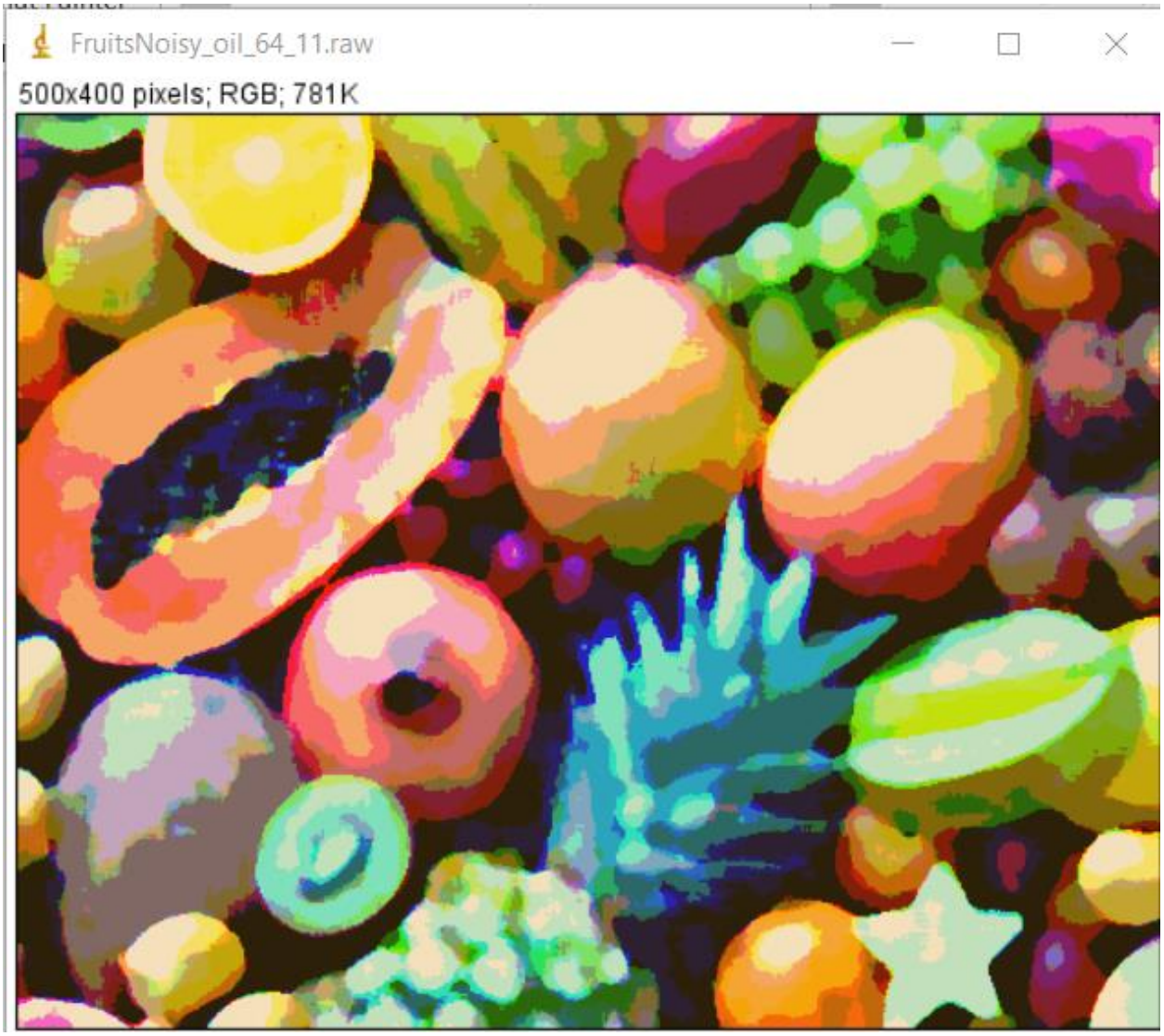
Oil Effect, 512 colors, N = 11 on Noisy image



Oil Effect, 64 colors, N = 3 on Noisy image

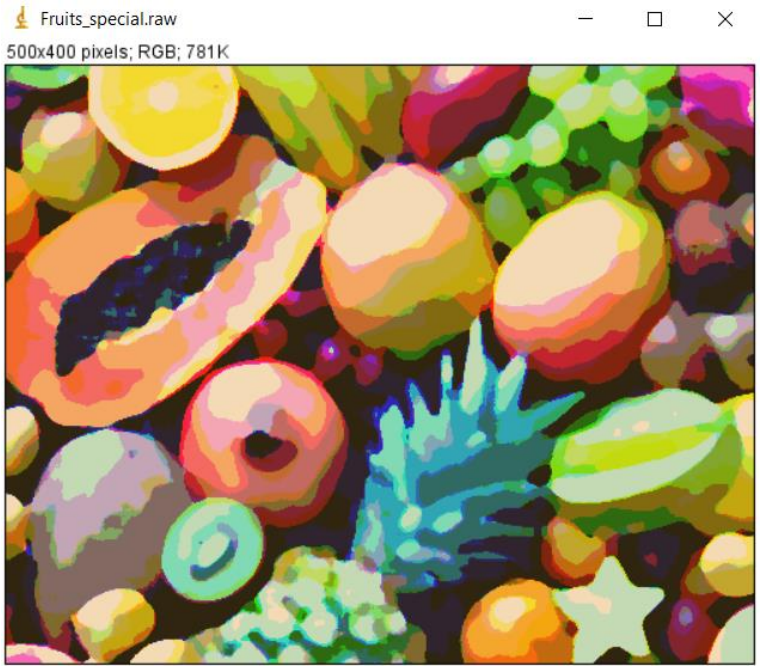


Oil Effect, 64 colors, N = 11 on Noisy image

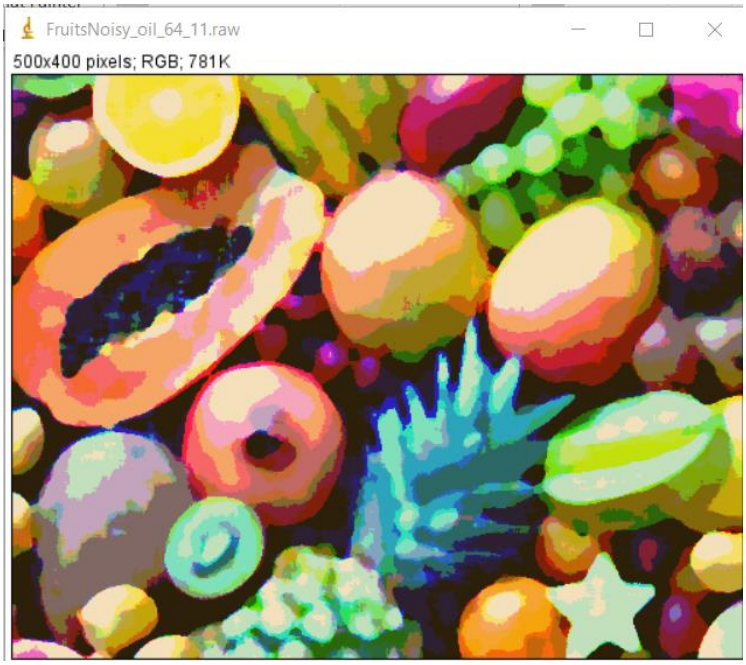


To effect on noisy image, side by side comparison.

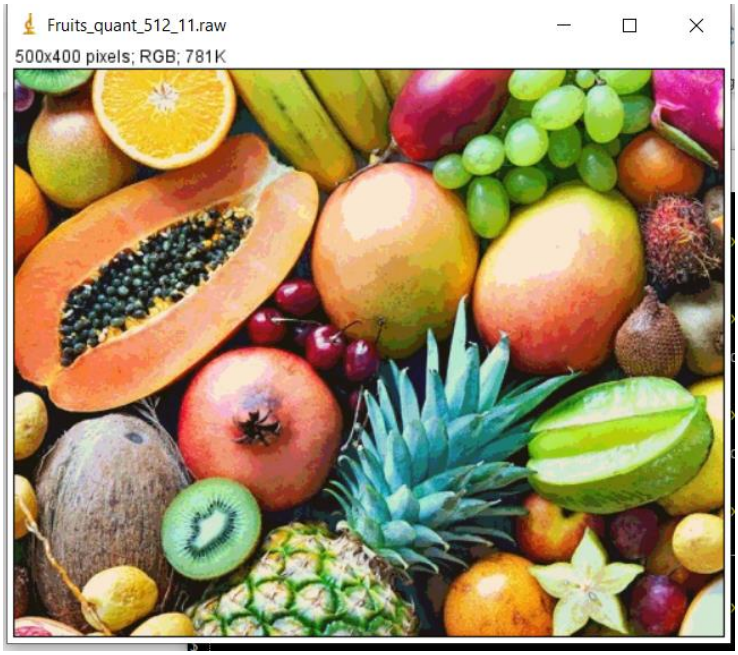
(a) 64 colors, N = 11, without noise



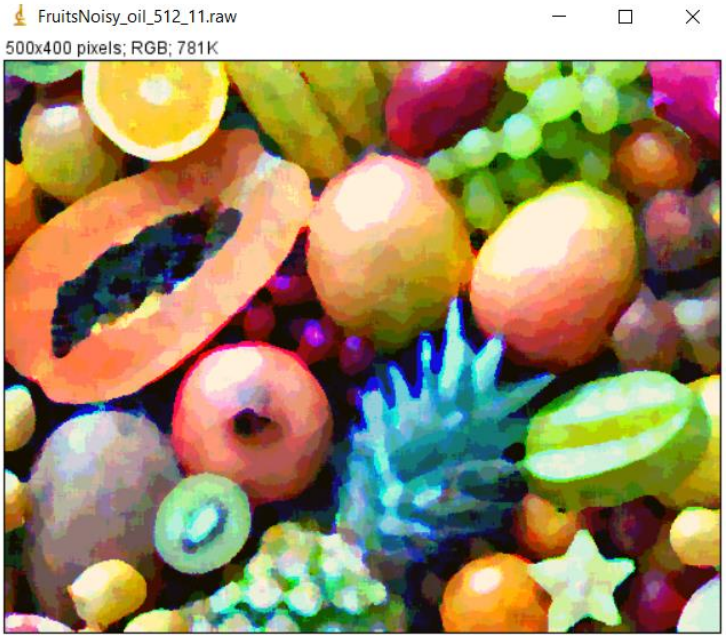
(b) 64 colors, N = 11, noisy



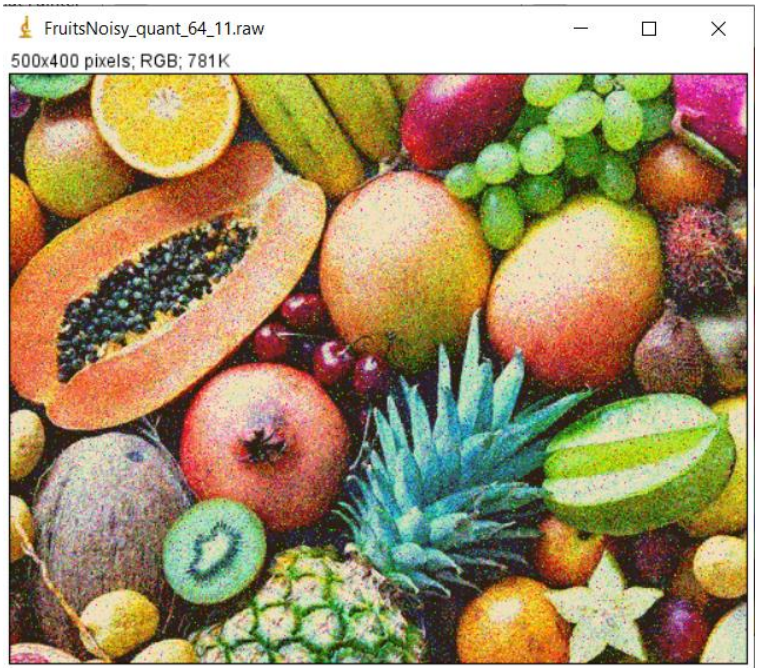
(a) 512 colors, N = 11, without noise



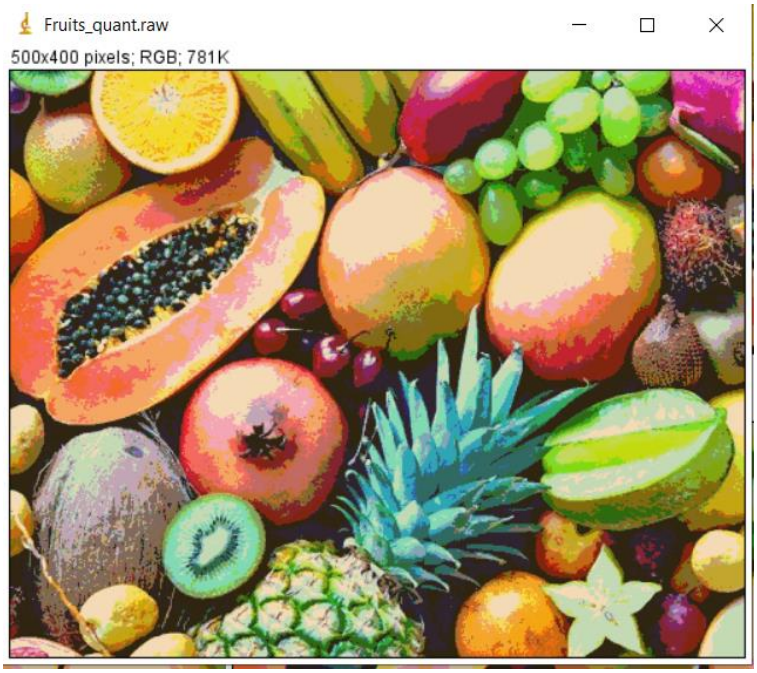
(b) 512 colors, N = 11, noisy



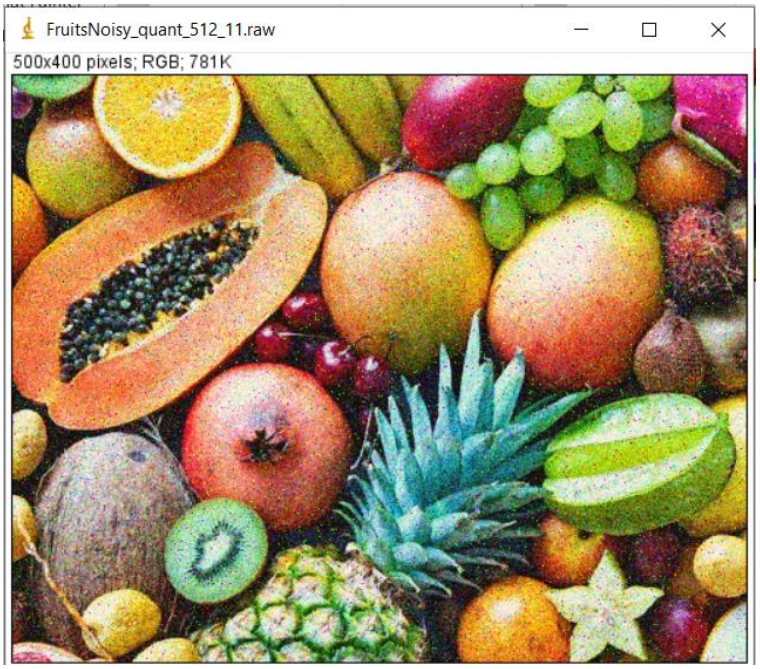
(c) 64 colors, N = 11, noisy



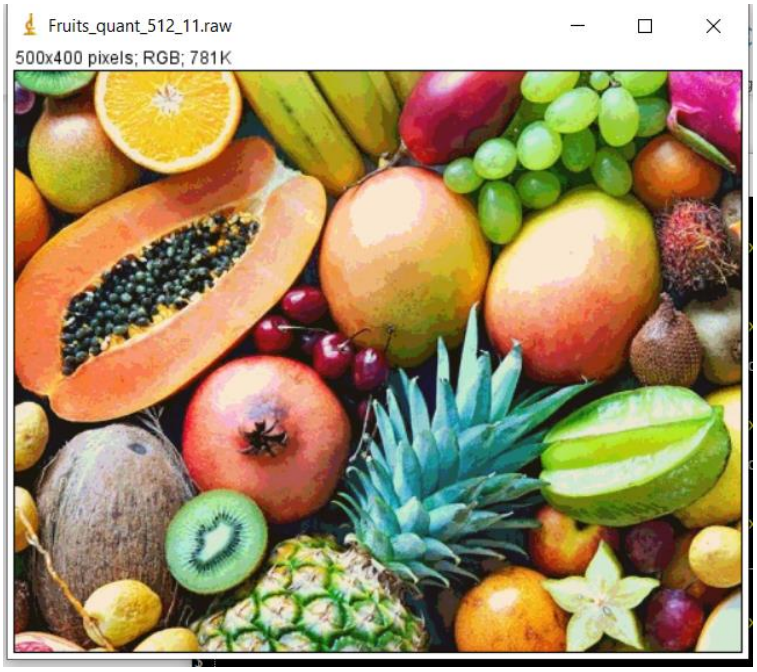
(d) 64 colors, N = 11, without noise



(e) 512 colors, N = 11, noisy



(f) 512 colors, N = 11, without noise



IV. Discussions

Experimental results revealed that higher values of N produced a more effective and visually pleasing oil effect. The images get blurry or in this case, “oilier” and colors overlap. Also, the edged become less defined and disappear with most of the image detail being lost. I think this makes sense as the idea of quantization is to reduce the amount of unique color representations in an image. For the value of N = 11, at 512 and 64 colors, the image Fruits gives better results.

At 512 colors, the image looks more cleaner and less blurry. This is because there are more colors to choose from and most details of the image is preserved.

For the noisy image, taking 3 results of equal settings, i.e., **N = 11 and 64 colors**, gives different visual results for the **oil painting effect**. I can observe that the edges are less smooth around areas of sharp color transition in the Noisy image in (b). For the original image, (a), the edges are blurrier. Although they both appear to look the same.

Comparing results of **Quantized** image with the setting at **N = 11 and 64 colors**, and for **512 colors** , the image with more colors (f) gives a better contrast transitioning than that in (d). The presence of noise does have some effect in that the colors seem to be more distributed in the image. I believe the noise “prevents” the proper quantization of the image so more colors are preserved.

REFERENCES

[1] Lecture Notes, Discussion Notes, Piazza posts and Homework handout.

[2] Mean filter, or average filter — Librow — Digital LCD dashboards for cars and boats

[3] ImageFiltering_2016.pdf (nyu.edu)

[4] https://stackoverflow.com/questions/3902648/c-representing-a-3d-array-in-a-1d-array

[5] Wikipedia