Homework 2

Problem 1:

Algorithm

Output = bitPlane(**String** input_Image, **Integer** cell[?](size up to 8));

//input_Image as name of the input image file

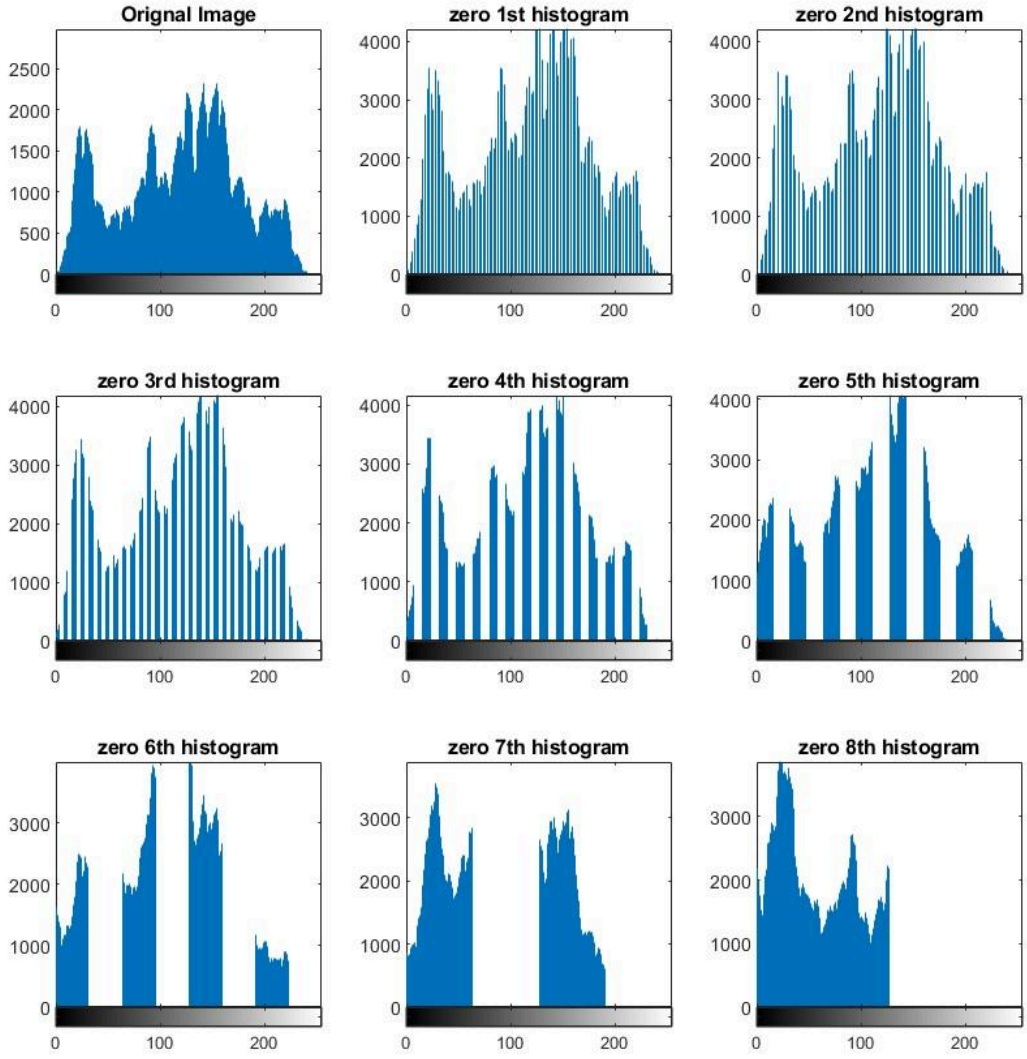
//cell[] contains integer values representing the order bit level. Range will be 1-8.

//if cell[] size == 1, set the corresponding oder bit plane's bit to 0.

//if cell[] size >1, construct the image that combines all corresponding order bit planes in cell[].

- bitPlane(input_Image, cell[]){
    1) Img ← imread(input_Image)
       sizeImg ← size of the image (matrix)
    2) argNums ← size of cell[]
    3) If argNums <= 1{
            Output ← img
       Else
            Output ← zeros matrix with size of sizeImg
                For i=1:argNums{
                    Output ← combine all the cell[i] order bit plane
                } //Selected how many and which order bit plane to combine only
       }
    4) If argNums == 1{
            Set output's cell[1] order bit plane's bit to 0
       } //Set 1 order of bit plane to 0 only
  } //End bitPlane() function

Plot Graph


Orignal Image
zero 1st histogram
zero 2nd histogram
zero 3rd histogram
zero 4th histogram
zero 5th histogram
zero 6th histogram
zero 7th histogram
zero 8th histogram

Result

Observed from above graph, zero the lower-order bit planes of 1 to 4 don't have a big impact on the original image. And the histogram of the original and these four lower bit planes are pretty much kept in the same shape. However, starting from zeroing the level 5 bit plane, the resulting images are getting darker and darker. Also, the histogram produced from zero the high-order bit planes are losing the original shape. By this, I assume the high-order planes are taking more control of the image than the low-order planes.

Below is a set of images constructed by combining only 2 bit planes.

By this, it's clear that using more higher-order bit planes will produce a better image as the original.

For a further test, below is a set of images constructed by using 3 bit planes.

The overall result is the same as the above 2 bit planes test. Using higher-order bit planes produces a better image.

At last, using 4 and 5 of the highest-order bit planes both are producing an image very identical to the original. Below is the comparison of these 2 images and the original.

In my opinion, I can’t distinguish any difference between these 3 images. So, I will say my assumption is correct. And, I guess using the 4 highest order bit planes is actually creating a compressed version of the original image.

Resources that Helped Me

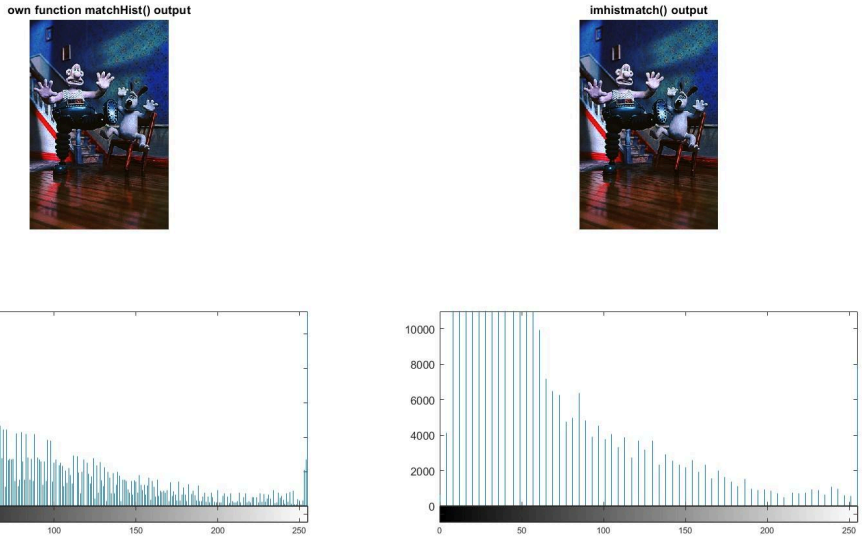
- https://stackoverflow.com/questions/30433172/image-bit-plane-decomposition
- https://www.mathworks.com/help/matlab/ref/nargin.html
- https://www.mathworks.com/help/matlab/ref/varargin.html#bta08tf-1
- https://www.mathworks.com/help/matlab/ref/bitset.html#btig76j-1
- https://www.imageeprocessing.com/2012/11/bit-plane-slicing.html

Problem 2:

Algorithm

1. Mapped, target ← selected an image to be mapped and an image as the source to map.
2. Extract the target image RGB channel
3. Create the histogram for each target's RGB channel
4. Compute to equalize the three channel's histogram
5. Apply the equalizing channel to the mapped image

Plot Graph



Result

By eye observation, the two mapped images created from two different functions are identical. However, the histogram of two mapped images are relatively different. Although the shape of two histograms are similar, the histogram from imhistmatch() function(right side) is having a larger gap between two bin bars. I assumed that it was caused by some compression technique that happened in the imhistmatch() function. Maybe like the above HW Problem1, using only the necessary bit planes to construct the image. Moreover, I had discovered some function for image quality and applied for comparing these two images. Surprisingly, all results are showing these two images are very close to identical, even their histogram don't look like that.

First, the result of imshowpair() function, which will gray the region where two images have the same intensities. Below is the output graph. By

By the definition of the imshowpair() document, “Magenta and green regions show where the intensities are different”, the result above is all gray which proves the two mapped images are very similar.

Second, the function psnr() to get the Peak signal-to-noise ratio of two images. And, the greater the value indicates better quality. The output of psnr() of two mapped images is 46.155. From the resource Intelligent Image and Video Compression Chapter 4 - Digital picture formats and representations, over 40 is a very good value.

Then, using the function ssim(), short for structural similarity index for measuring. A value and an image will output from this function. Values closer to 1 indicated better similarity and the output image will be brighter as well. The Darker area of the output image indicates the

difference between two input images. Below is the output value and image.

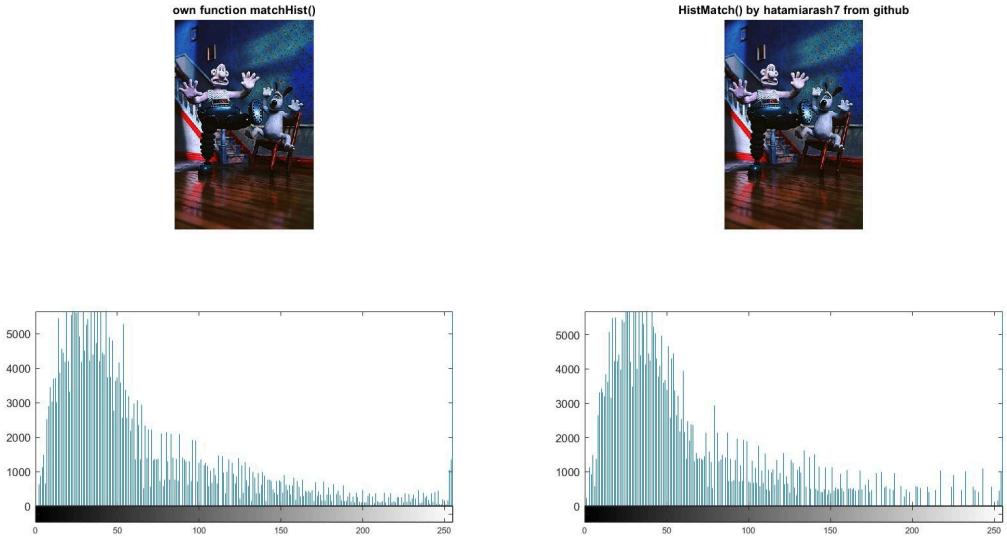
Structural similarity index measuring value: 0.99801

We can see that the image is pretty much all white and the value is also very close to 1.

At last, function immse() to calculate the mean-squared error. A lower value indicates greater similarity between two images. And the result value is 1.5761.

By all these image quality comparisons, we can see these two mapped images are actually very identical to each other. But, my own function matchHist() is using the built-in function histeq() for computing the equalizer. I guess the function imhistmatch() and histeq() are somehow using the same computation technique to get the result. Thus, I think a matically hard code function is worth trying to see if there is any difference between them. For convenience purposes, I used a public resources function histMatch() written by hatamiarash7 in github. And

below is the output showing the result of matchHist() and HistMatch().



In short, they are almost identical.

Resources that Helped Me

https://github.com/hatamiarash7/HistogramMatching/blob/master/HistMatch.m
https://stackoverflow.com/a/6351811
https://stackoverflow.com/questions/23093464/histogram-match-between-an-image-and-one-histogram?rq=3
https://www.mathworks.com/help/images/ref/histeq.html#d126e129188
https://www.youtube.com/watch?v=c2VMpu0Q4UU&list=PLB57s6OrG8LjbxmJ8kU6Hi2GtAngtXaqV&index=4
https://www.mathworks.com/help/images/image-quality.html?s_tid=CRUX_lftnav
https://www.mathworks.com/help/images/ref/immse.html
https://www.mathworks.com/help/images/ref/ssim.html
https://www.mathworks.com/help/images/ref/psnr.html#bt5uhgi-2_1
https://www.sciencedirect.com/topics/computer-science/peak-signal-to-noise-ratio#:~:text=Typical%20values%20for,an%20infinite%20PSNR.
https://www.mathworks.com/help/images/ref/imshowpair.html
https://people.ece.ubc.ca/irenek/techpaps/introip/manual02.html

Problem 3:

Algorithm

1. Image ← read input image
   Filter ← read input filter(matrix)
2. Copy the image to a mirrorFrame which has an extra row and column , and the value set as 0.
3. Iterating each copy pixel in mirrorFrame and perform convolution computation
4. Assign each new compute pixel to the output image's pixel

Graph

Result

The transformation of the image “LENNA.JPG” by applying an average filtering first and following a Laplacian filtering is shown above (The last pic in Graph section). The average mask and laplacian mask were used are:

Average mask:

1	**1**	**1**
1	**1**	**1**
1	**1**	**1**

Laplacian mask(sum 4):

0	**-1**	**0**
-1	**4**	**-1**
0	**-1**	**0**

In my opinion, the result is highlighting the edge well. But, using different laplacian masks might be able to get a better result. Thus, I had tried to apply the mask below to the average image.

Laplacian mask(sum 8):

-1	**-1**	**-1**
-1	**8**	**-1**
-1	**-1**	**-1**

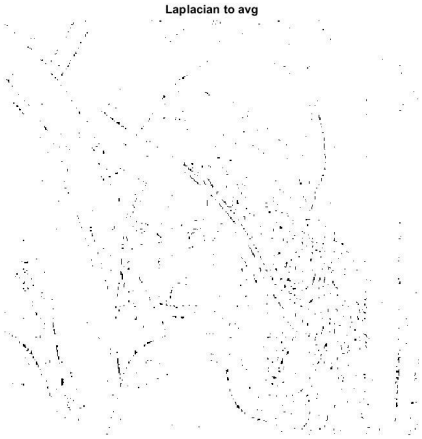
And below is the new result:

It’s hard to tell the difference between them, but I believe the new result is brighter. To verify that, I found out that changing the type of both results to uint8 can show the new result has a more high intensive area. Below is the comparison:

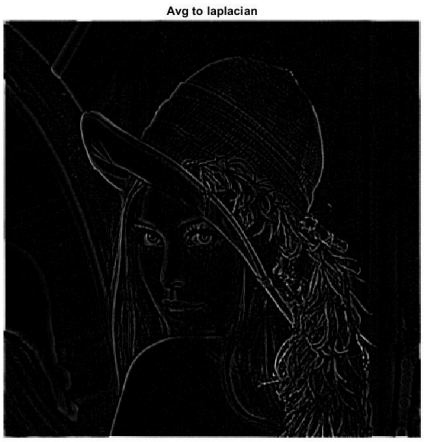
By the edge detection basic principle, which tells us the edge's pixel should have a larger difference compared to its neighbor pixel. The new result is obviously drawing the edge better. So, I decided to use this new laplacian mask(sum 8) for the later experiment, reversing the image transformation process.

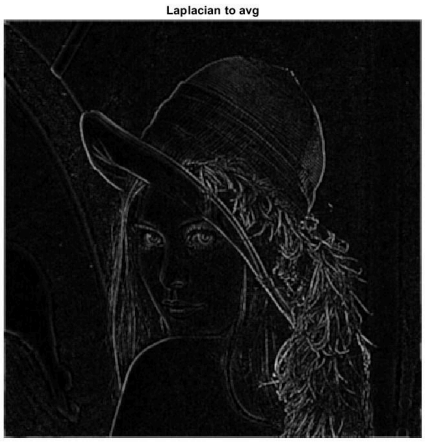
Applying the laplacian filtering first and following the average filtering, I assume the result will be the same either way. Because I think this process is similar to 1+2=2+1, which

yields the same result. However, my assumption is not correct though.



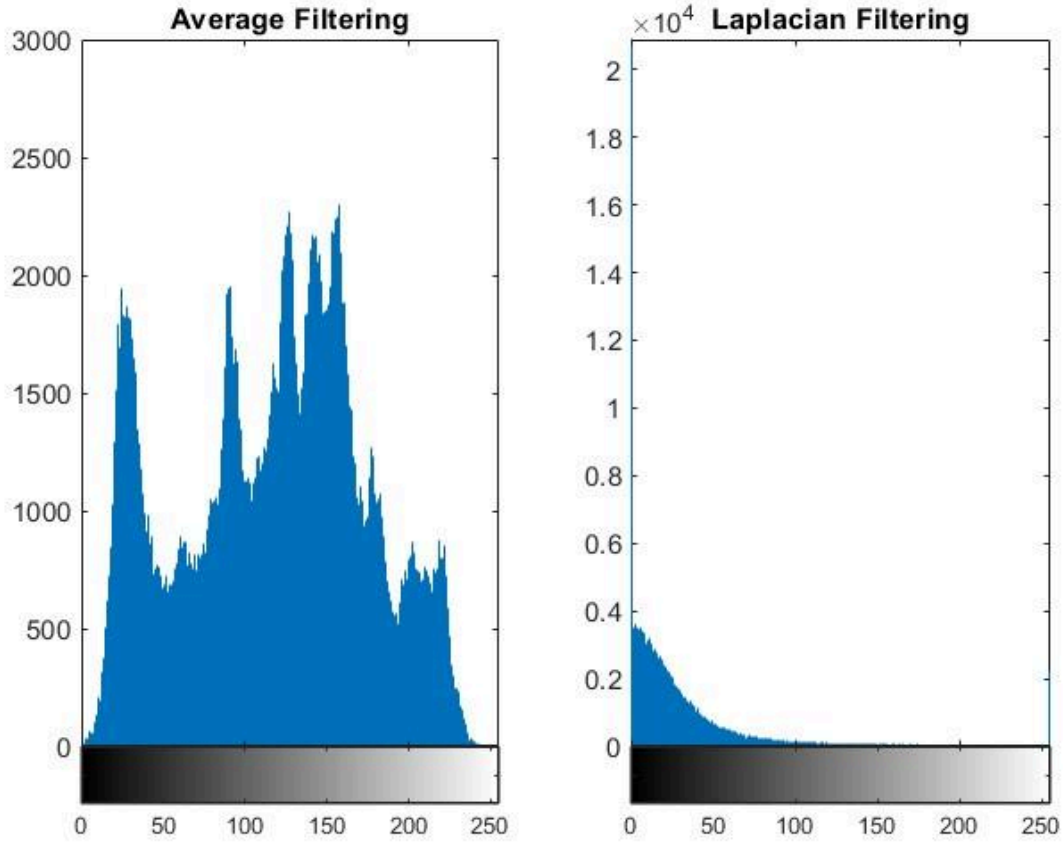
Above is the comparison of the result from two different ways. The result of applying the laplacian first is almost wiping out the image completely. However, by transferring both image types to uint8, the result of applying the laplacian first is looking better(Picture below).





By this, comparing the uint8 type of results doesn't seem to be a good way to see which filtering does a better job. But one thing can be ensured that is the order of applying filters matters on image filtering.

Why the image is being wiped out so much by applying laplacian first. I think we can figure out something by looking at their histogram like the question 2 we did above.



Above picture, the left histogram is the result by applying average filtering only and the right histogram is the result by applying laplacian filtering only. It's clear that the laplacian filtering eliminates a large portion of pixels compared to the average filtering result. So, I guess this is the reason why many areas will be wiped out after that.

Resources that Helped Me

https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html
https://www.mathworks.com/help/matlab/ref/times.html
https://www.mathworks.com/matlabcentral/answers/249558-creating-9x9-average-filter-and-applying-it-to-an-image-with-certain-values