DIP Homework Assignment #1

姓名：鍾毓安
學號：B01902040
系級：資訊四
Email: iamyuanchung@gmail.com
繳交日期：2016.3.13

Source Code

The following script and functions were implemented and their introductions were described as follows. As for more detailed descriptions, please refer to each corresponding function file.

1. README.m: This is the main script that works like the main function. All the required tasks, including Warm-Up, Problem 1 & 2 will be completed one by one when README.m is executed.
2. flipVertical.m: The function flips the given 2D image matrix up-side-down.
3. flipHorizontal.m: The function reverses the given 2D image matrix left-to-right.
4. plotHistogram.m: For a given 2D gray-scale image matrix, plotHistogram returns a 256-length 1D array where the i-th entry stores the number of pixels whose value equals to (i – 1).
5. histEqual.m: The function performs the histogram equalization on the given 2D image matrix. Histogram equalization enhances the given image G by first converting the histogram of G into cumulative distribution function (CDF), then mapping the CDF to a uniformly distributed one so as to make the histogram of the enhanced G more uniformly distributed.
6. localHistEqual.m: The function performs the local histogram equalization on the given 2D image matrix. Similar to histogram equalization, local histogram equalization also tries to enhance the given image using the histogram. However, local histogram equalization introduces an extra window that will go through the image from top to bottom and from left to right. Then, histogram equalization is applied to the region inside the window, and this is how the word “local” comes from. The original histogram equalization is sometimes referred to as the global histogram equalization to make it distinguishable from the local version.
7. logTransform.m: The function performs log transform on the given 2D image matrix. Log transform enhances the low intensity pixels due to its property of concave downward. Especially, for each entry $G(i,j)$ of an given image matrix G, where $G(i,j)$ is already scaled to range $[0,1]$ by dividing 255, the log transform does the following transformation:

   $$G_{new}(i,j) = c \cdot \ln\big(1 + G(i,j)\big),$$

   where $c = 255/\left(\ln\left(1 + \max_{i,j} G(i,j)\right)\right)$ is the scaling constant that ensures the resulting image has a maximum magnitude of 255.

   (Reference: http://homepages.inf.ed.ac.uk/rbf/HIPR2/pixlog.htm)
8. invLogTransform.m: The function performs inverse log transform on the given 2D image matrix. Different from log transform, the concave upward property makes the inverse log transform more useful

in a situation when more details of high intensity pixels are desired. Especially, for each entry $G(i,j)$ of an given image matrix G, where $G(i,j)$ is already scaled to range $[0,1]$ by dividing 255, the inverse log transform does the following transformation:

$$G_{new}(i,j) = c \cdot \left(e^{G(i,j)} - 1\right),$$

where $c = 255/\left(\left(e^{\max_{i,j} G(i,j)} - 1\right)\right)$ is the scaling constant that ensures the resulting image has a maximum magnitude of 255.

9. powerLawTransform.m: The function performs the power-law transform on the given 2D image matrix. For each entry $G(i,j)$ of an given image matrix G, where $G(i,j)$ is already scaled to range $[0,1]$ by dividing 255, the power-law transform does the following transformation:

$$G_{new}(i,j) = c \cdot e^{p},$$

where $c = 255/\left(e^{\max_{i,j} G(i,j)}\right)$ is the scaling constant that ensures the resulting image has a maximum magnitude of 255, and $p > 0$ is a parameter that can be flexibly controlled. When $0 < p < 1$, e^p's property of concave downward makes it a suitable choice for enhancing image with low intensity pixels; when $p > 1$, e^p becomes concave upward, which is a good option for enhancing image with high intensity pixels; and when $p = 1$, e^p simply performs the linear mapping.
(Reference: http://funnotes.net/tofpages/TopicOfFortnight.php?tofTpcFl=topicoffortnight22)

10. addGaussianNoise.m: The function adds the Gaussian (Normal) distributed noise to the given 2D image matrix. The formula is as follows:

$$G_{noise} = G + \delta \cdot N(\mu, \sigma),$$

where δ is the amplitude of the generated Gaussian noise, μ and σ are the mean and variance for Gaussian distribution, respectively.

11. addSaltPepperNoise.m: The function adds the Salt and Pepper noise to the given 2D image matrix, where salt means a totally white (255) pixel and pepper means a totally black (0) pixel. For each entry $G(i,j)$ in the given 2D image matrix G, the salt and pepper noise is generated and added as follows:

$$G_{noise}(i,j) = \begin{cases} 0, if\ U(0,1) < p \\ 255, if\ U(0,1) > 1-p, \\ G(i,j), else \end{cases}$$

where $p \in (0,1)$ is a parameter of probability. When p is small, the outcome image tends to be similar with the clean image; as p grows larger, the outcome becomes noisier.

12. calcPSNR.m: The function calculates the Peak signal-to-noise ratio (PSNR) between two gray-scale 2D image matrix G_1 and G_2 of same shape. The formula for calculating PSNR is as follows:

$$MSE(G_1, G_2) = \frac{1}{m \times n} \sum_{i=1}^{m} \sum_{j=1}^{n} [G_1(i,j) - G_2(i,j)]^2,$$

$$PSNR(G_1, G_2) = 10 \times \log_{10}\left(\frac{255^2}{MSE(G_1, G_2)}\right).$$

The larger $PSNR(G_1, G_2)$ is, the more similar G_1 and G_2 are.

13. lowPassFilter.m: There are usually two types of noise, the first one is uniform noise, including additive uniform noise and Gaussian noise, and the second one is impulse noise, including salt and pepper noise. Low-pass filter is designed to remove the uniform noise. Especially, low-pass filter introduces a mask H, which has the general form as follows (3x3 example):

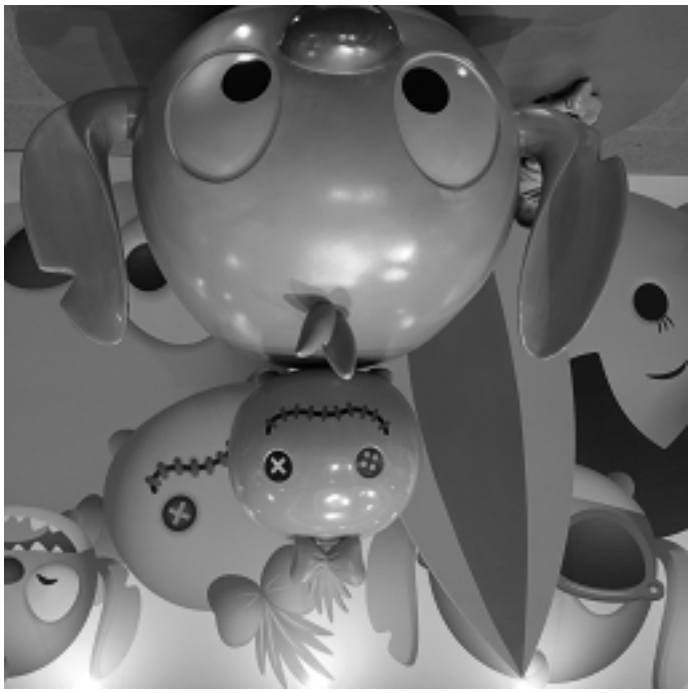
$$H = \frac{1}{(b+2)^2}\begin{bmatrix} 1 & b & 1 \\ b & b^2 & b \\ 1 & b & 1 \end{bmatrix},$$

The noise removal procedure is done by convolving H with the noisy image. During my implementation, I focus on H of size 3x3 and adjust b to achieve the best outcome.

14. outlierDetection.m: For impulse noise, outlier detection and median filter are designed to remove the noise. I implement the function by simply following the course slides. The threshold ε is tuned to locate the best one.
15. myMedianFilter.m: Since there's a built-in function named medianFilter.m in Matlab, I use myMedianFilter.m for my implementation. The square median filter is implemented and the method doesn't need any other controlling parameters.

Warm-Up: Simple Manipulation

Flipped I vertically and horizontally. The resultant images were displayed as follows.

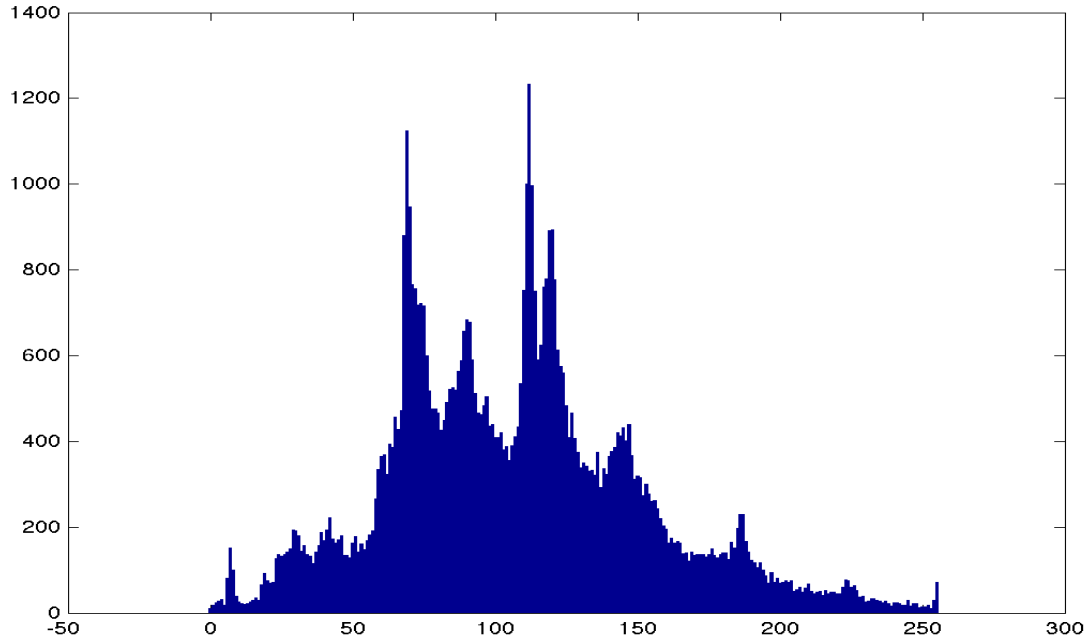


sample1.vertical.png: Flipped I vertically

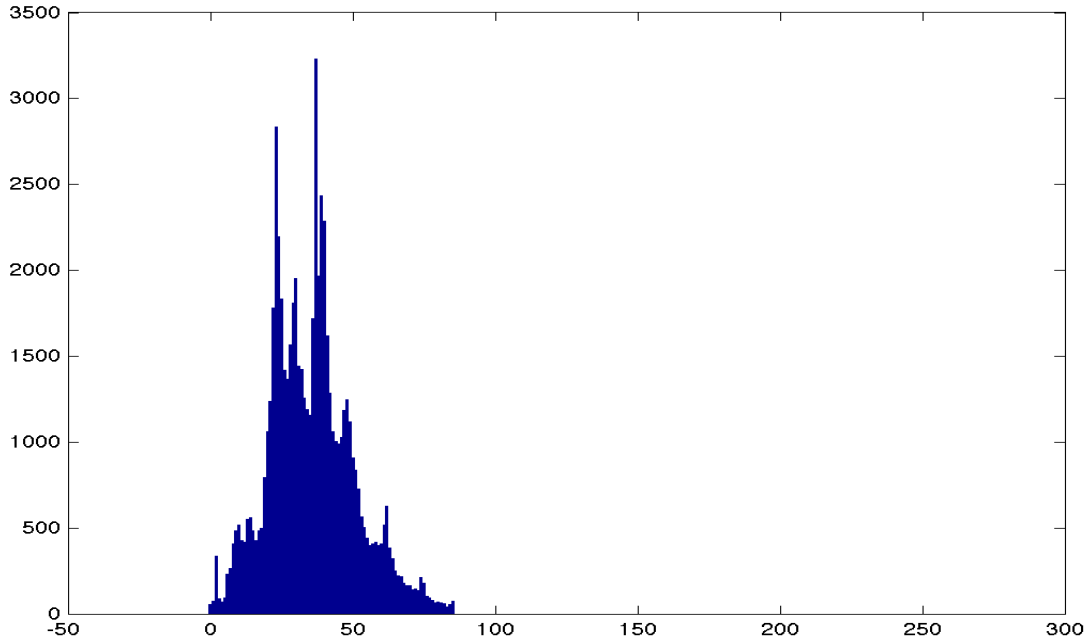
sample1.horizontal.png: Flipped I horizontally

Problem 1: Image Enhancement

(a) Plot the histograms of I and D. What can you observe from these two histograms? What can you do to make D look like I?



sample1.hist.png: The histogram of image I



sample2.hist.png: The histogram of image D

The above two figures, sample1.hist.png and sample2.hist.png, display the histograms of image I and D, respectively. From the figures, we can observe that the histogram of D tends to be centralized at those low intensities regions 0 ~ 100, comparing to the histogram of I, which means that the range between the lowest and the highest pixel values may be too small to be perceived by human eyes, causing image D looks much darker than image I.

To make D look similar to I, we can apply some techniques of image enhancement on D. Image enhancement is the process of adjusting digital images so that the results are more suitable for display or further image analysis. Two approaches, histogram modification and contrast manipulation, have been introduced in class and I will use them for enhancing image D. For histogram modification, histogram equalization and local histogram equalization will be used; and for contrast manipulation, three transfer functions, including log transform, inverse log transform, and power-law will be applied.

(b) Perform histogram equalization on D and output the result as H.

Histogram equalization is usually applied to images with close intensities by effectively spreading out the most frequent intensity values. Through this adjustment, the intensities can be better distributed on the histogram, allowing the regions with local contrast to gain a higher contrast.

sample2.png

sample2.hist.equal.png: The histogram equalized image D, denoted as H

The above two figures displayed D and the resulting image, denoted as H, of performing histogram equalization on image D.

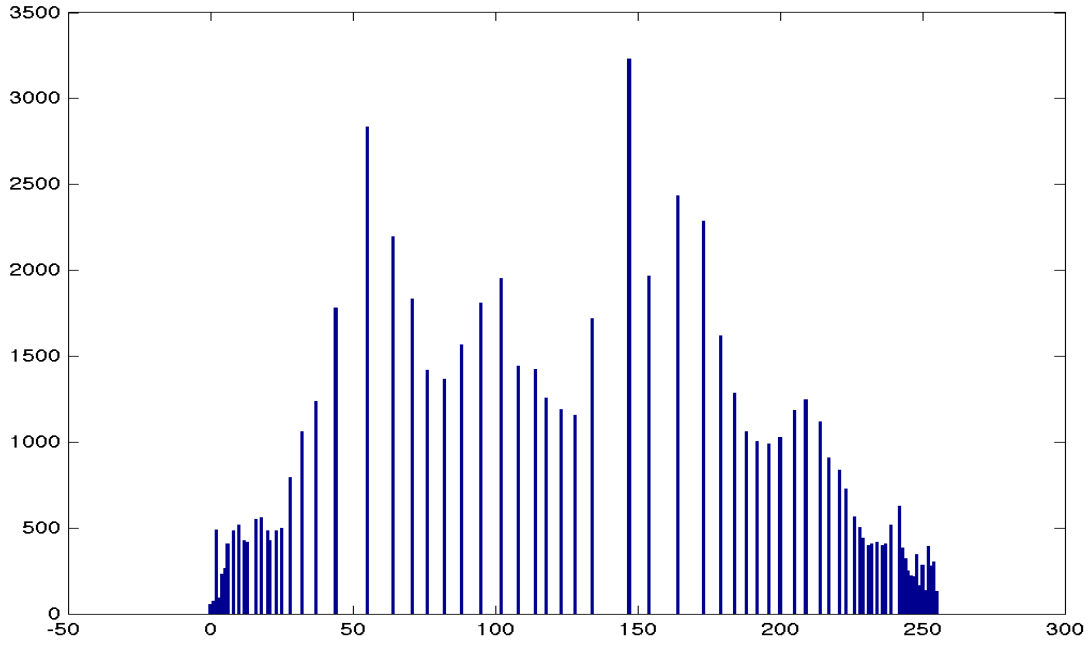
(c) Perform local histogram equalization on image D and output the result as L.

Local histogram equalization is the local version of histogram equalization. The method uses a window that goes through the low contrast image and performs histogram equalization under the window's current position. This makes it emphasizes more on each local gray-level variations than histogram equalization does.

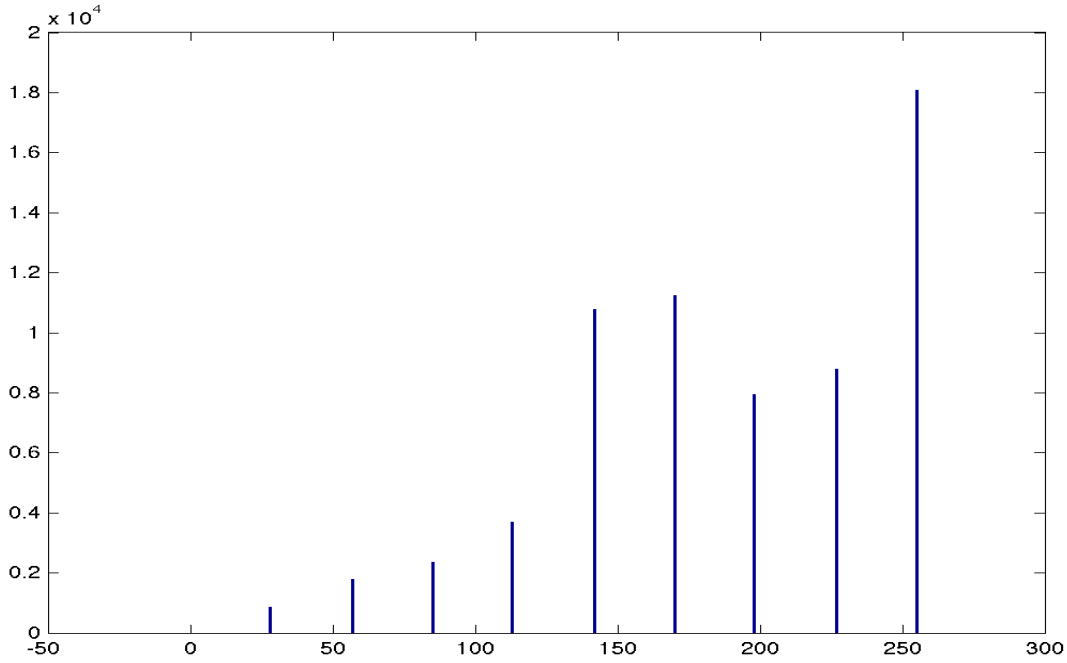
sample2.local.hist.equal.png: The local histogram equalized image D, denoted as L

The above figure displayed the resulting image, denoted as L, of performing local histogram equalization on image D. Here I keep the window size fixed at 3x3.

(d) Plot the histograms of H and L. What's the main difference between local and global histogram equalization?



sample2.hist.equal.hist.png: The histogram of H



sample2.local.hist.equal.hist.png: The histogram of L

The principles of histogram equalization and local histogram equalization have already been discussed in (b) and (c). Comparing the two figures above, we found that local histogram equalization does focus more on each local region since most of the pixels belong to certain intensity values in sample2.local.hist.equal.hist.png.

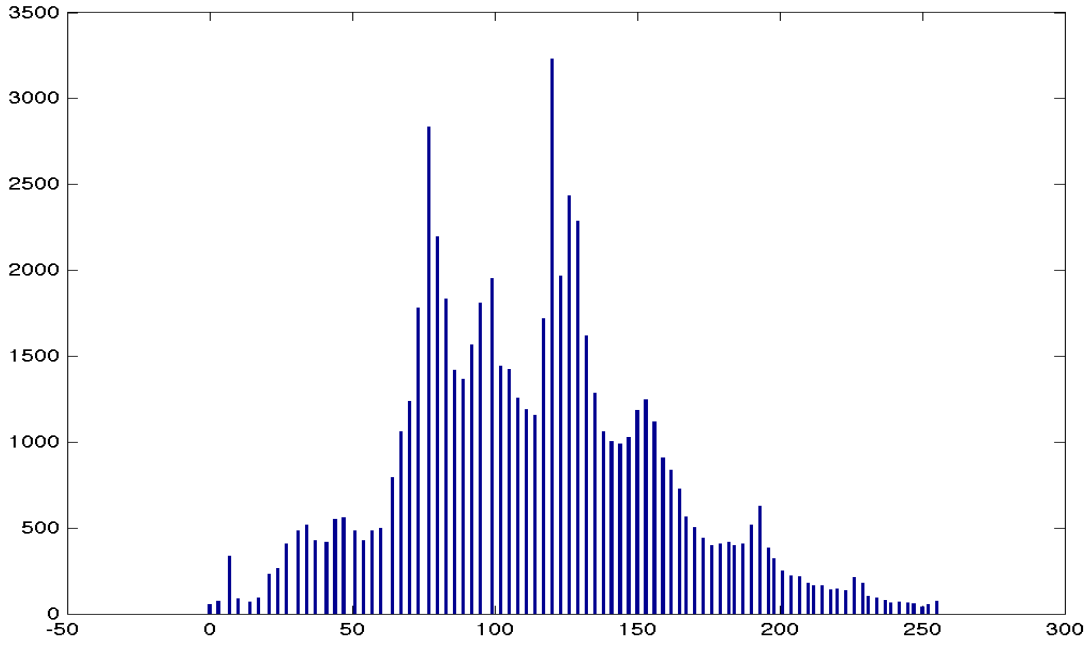
(e) Perform the log transform, inverse log transform and power-law transform to enhance image D. Please adjust the parameters as best as you can. Show the parameters, output images and corresponding histograms. Provide some discussions on the results as well.

All of these three techniques involve using the transfer functions, and their properties have already been discussed in Section: Source Code.

According to my understanding, log transform and inverse log transform do not need extra parameters, while the power-law requires a power p.

Log transform

sample2.log.png: The log transformed image D

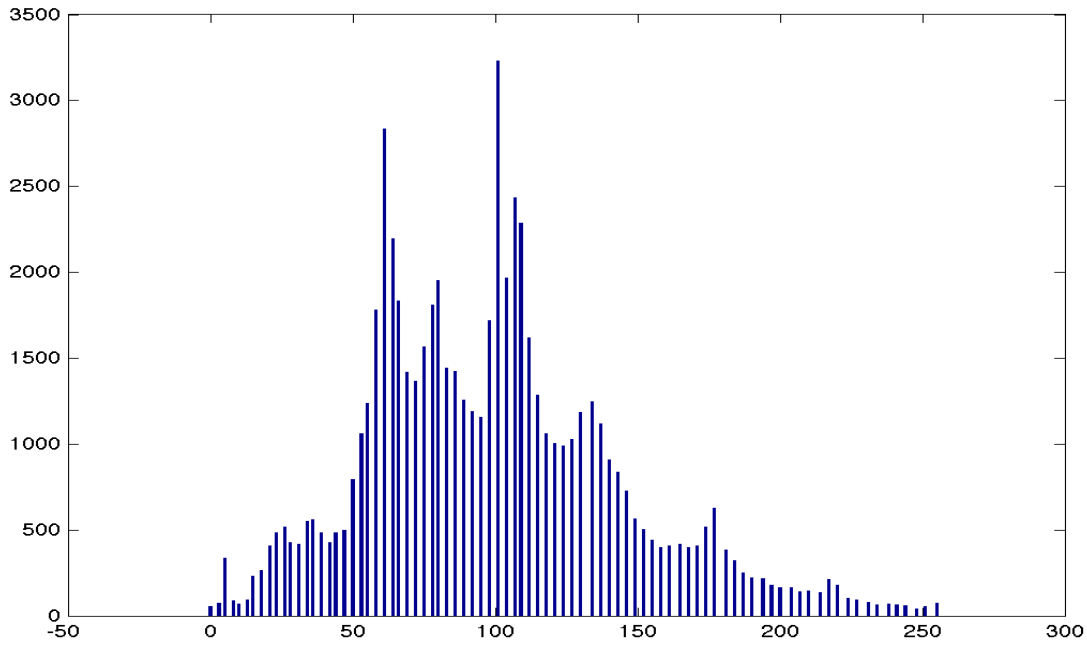


sample2.log.hist.png: The histogram of the log transformed image D

The above two figures displayed the log transformed D and the corresponding histogram.

Inverse log transform

sample2.inv.log.png: The inverse log transformed image D



sample2.inv.log.hist.png: The histogram of the inverse log transformed image D

The above two figures displayed the inverse log transformed D and the corresponding histogram.

Now, let's put I, D, the log transformed D, and the inverse log transformed D together to see their differences:

sample1.png

sample2.png

sample2.log.png: The log transformed image D

sample2.inv.log.png: The inverse log transformed image D

Image I can be viewed as the "pseudo" ground truth (I say "pseudo" since we are trying to enhance image D instead of restoring it). I think both log transform and inverse log transform work quite well as both of them make the details of the dark image D much more perceivable. However, the log transformed D looks somewhat brighter than the inverse log transformed D and is more similar to the original image I, maybe it's because of its property of concave downward.

Power-law transform

Power-law transform requires an extra parameter, that is, the power p. Since our goal is to enhance image D by seeing more details in the dark region, we should choose a p that falls in range $(0, 1)$, this makes the power-law transform has similar efficacy to log transform. To search for the best p, I plot the resulting images when $p \in \{0.25, 0.5, 0.75, 1\}$ below:

sample2.power.law.25.png: The power-law transformed image D with $p = 0.25$

sample2.power.law.50.png: The power-law transformed image D with $p = 0.5$

sample2.power.law.75.png: The power-law transformed image D with $p = 0.75$

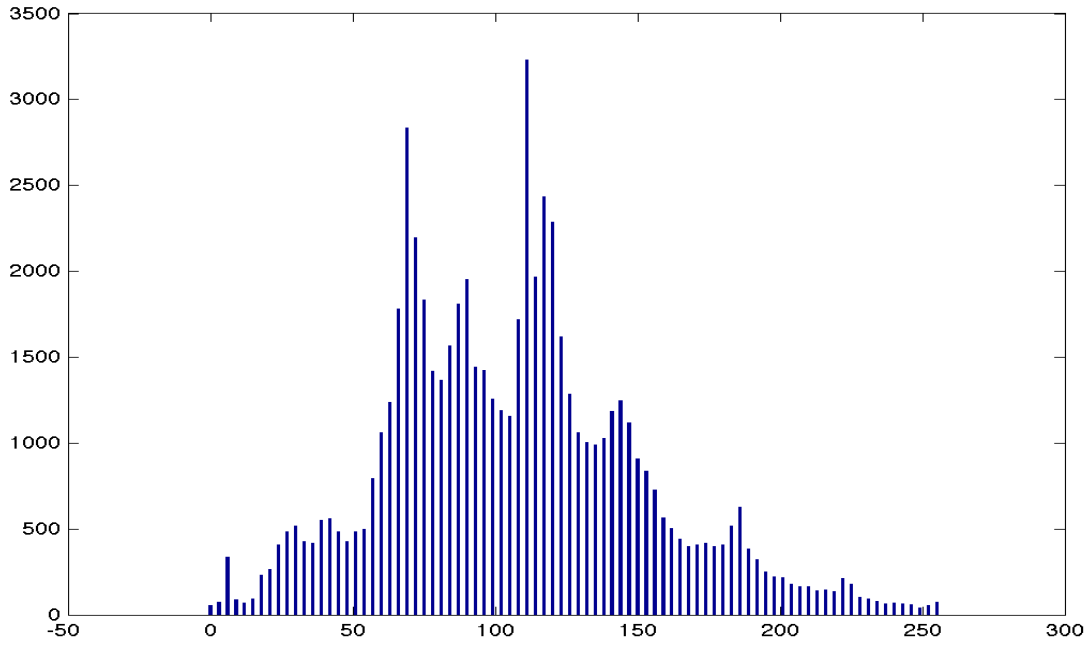
sample2.power.law.100.png: The power-law transformed image D with $p = 1.0$

Among the four figures displayed above, I think $p = 1.0$ resulted in the best outcome. Interestingly, when putting the outcome of $p = 1.0$ with I (sample1.png) together, I found that there are no differences between them!

sample1.png

sample2.power.law.100.png: The power-law transformed image D with $p = 1.0$

However, the histogram of $p = 1.0$ is:



sample2.power.law.hist.100.png: The histogram of the power-law transform D with $p = 1.0$ which is very different with the histogram of image I. However, the differences seem to be unperceivable to my eyes.

Comparing the resulting images and the histograms of log transform, inverse log transform, and power-law with $p = 1.0$, they all look very similar for me. So which one is the best? I think it depends on people. After all, there's no correct answer in image enhancement.

Problem 2: Noise Removal

(a) Add the same kind of noise as in sample3.raw to image I and denote the result as N_1.

Since sample3.raw features sparsely occurring white and black pixels, I infer that sample3.raw is corrupted by Salt and Pepper noise. To add salt and pepper noise to image I (sample1.raw), we first need to specify $p \in (0, 1)$, the larger the p is, the larger extent of corruption of the resulting image will exhibit. Here I set $p = 0.005$, which I think is the closest to sample3.raw. The original image I and the corrupted version, denoted as N_1, are displayed as follows, respectively.

sample1.png

sample1.salt.pepper.png: Image I with salt and pepper noise ($p = 0.005$) added

(b) Add the same kind of noise as in sample4.raw to image I and the output is denoted as N_2.

Since sample4.raw features a uniform corruption, I infer that sample4.raw is corrupted by Gaussian noise. I generated the corrupted image by specifying $\mu = 0, \sigma = 1,$ and the amplitude $\delta = 10$. The original image I and the corrupted version, denoted as N_2 are displayed as follows, respectively.

sample1.png

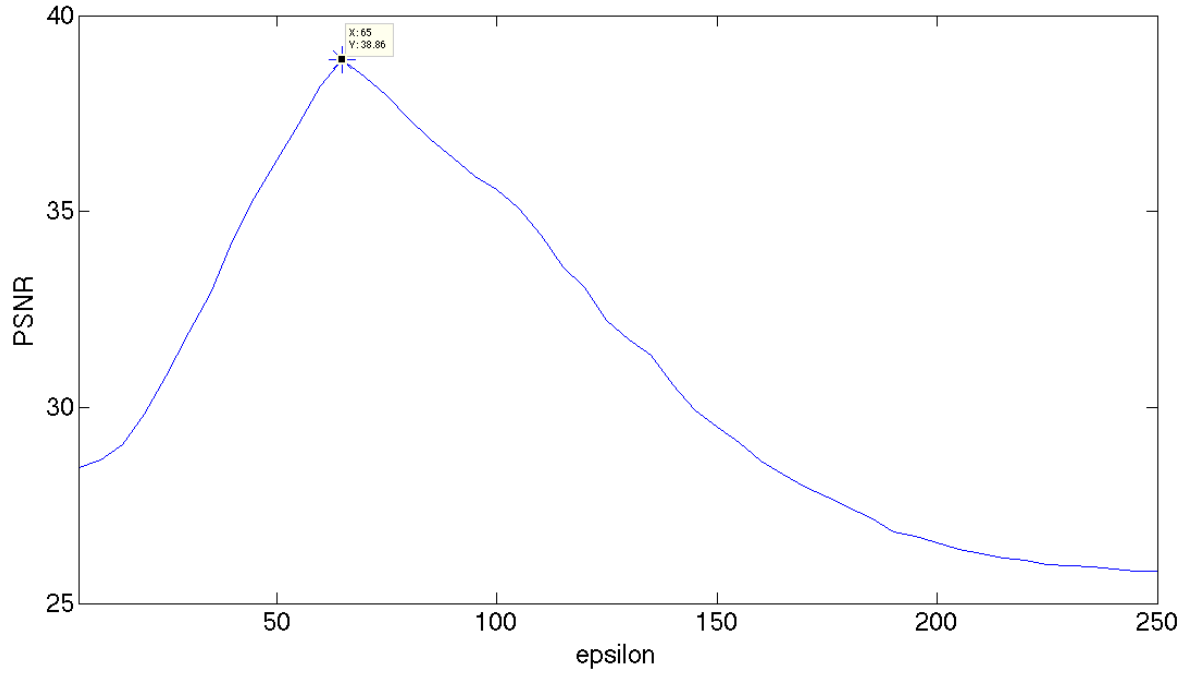
sample1.gaussian.png: Image I with Gaussian noise ($\mu = 0, \sigma = 1, \delta = 10$) added

(c) Choose proper filters and parameters to remove the noise in N_1 and N_2, and denote the resultant images as R_1 and R_2, respectively. Please specify the steps of your de-noise process and provide some discussions about the reason why those filters and parameters are chosen.

$\underline{N_1 \rightarrow R_1}$

Since N_1 is contaminated by salt and pepper noise, which belongs to the impulse noise, median filter and outlier detection are two methods worthy trying. We will discuss them one by one.

For **outlier detection**, there's a threshold ε that decides how large the gap between a certain pixel value with the average of its eight neighbors is that will be considered as an outlier. Unfortunately, such ε is usually hard to choose. Therefore, I decide to plot a curve, where the x-axis is ε and the y-axis is the corresponding PSNR value, and the ε that achieves the highest PSNR value will be selected as the best threshold ε. The curve is displayed as follows, and the range of ε is $[5, 250]$ with sample rate 5.



outlier.psnr.png: The relationship between ε and the achieved PSNR value of outlier detection

According to the above figure, the maximum PSNR achieved is approximate 38.86, and the corresponding ε is 65. The original image I and the resulting image using outlier detection with $\varepsilon = 65$ are displayed as follows, respectively.

sample1.png

sample1.salt.pepper.outlier.65.png: N_1 cleaned by outlier detection with $\varepsilon = 65$

Comparing the above two figures, we can see that the outlier detection approach with $\varepsilon = 65$, which achieves PSNR = 38.86, works quite well, as there were very few white and black points left. I also displayed other figures with different ε to get a feeling of low and high PSNR values.

sample1.salt.pepper.outlier.25.png: N_1
cleaned by outlier detection with $\varepsilon = 25$

sample1.salt.pepper.outlier.50.png: N_1
cleaned by outlier detection with $\varepsilon = 50$

sample1.salt.pepper.outlier.100.png: N_1
cleaned by outlier detection with $\varepsilon = 100$

sample1.salt.pepper.outlier.150.png: N_1
cleaned by outlier detection with $\varepsilon = 150$

sample1.salt.pepper.outlier.200.png: N_1 cleaned by outlier detection with $\varepsilon = 200$

sample1.salt.pepper.outlier.250.png: N_1 cleaned by outlier detection with $\varepsilon = 250$

Surprisingly, the result of outlier detection with $\varepsilon = 50$ also looks pretty good to me, although it achieves lower PSNR value (36.31). This may indicate that sometimes PSNR value does not really reveal the feeling of human perception. As for other ε, the resulting images were obviously worse than $\varepsilon = 50$ and 65, especially for $\varepsilon = 200$ and 250, the outlier detection approach seems not even working!

The second approach I tried is **median filter**. My implementation of median filter does not require any extra parameters, and the resulting image, which achieves PSNR = 33.08, is displayed along with the resulting image of outlier detection with $\varepsilon = 65$, which achieves PSNR = 38.86, as follows, respectively.

sample1.salt.pepper.median.png: N_1 cleaned by median filter

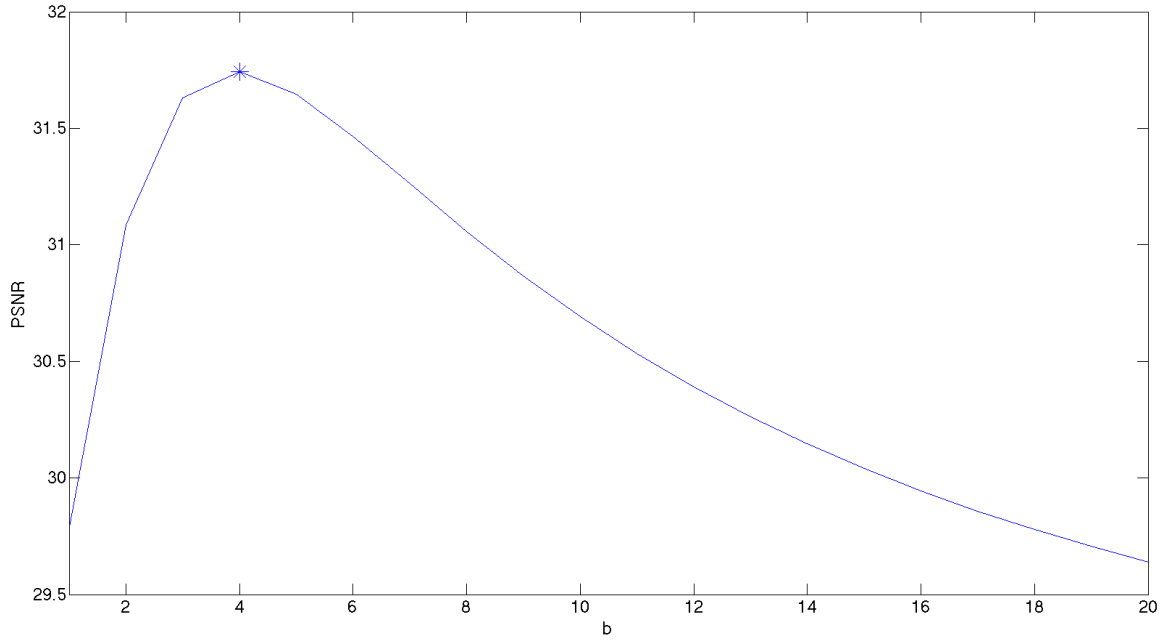
sample1.salt.pepper.outlier.65.png: N_1 cleaned by outlier detection with $\varepsilon = 65$

Interestingly, no white and black points can be perceived in the figure on the left (cleaned by median filter),

this may because that those white and black "outliers" were eliminated by the median operation. However, we can find that such median operation also wiped out the "continuous intensities" between the two neighbors, making the resulting image looks blurrier than the image cleaned by outlier detection approach. Therefore, N_1 cleaned by outlier detection with $\varepsilon = 65$ still looks better than that cleaned by the median filter, so let's denote the former one as R_1.

$N_2 \rightarrow R_2$

For uniform noise like Gaussian noise, low-pass filter is a wise choice. Similar with outlier detection, there's one parameter that can be adjusted in low-pass filter (I fix the window size to 3x3), the b, which determines how the filter looks like. Again, I plot the relationship between $b \in [1, 20]$ and the corresponding PSNR values to decide the best b. The curve is displayed as follows.



low.pass.psnr.png: The relationship between b and the achieved PSNR value of low-pass filter

According to the above curve, the maximum PSNR = 31.74 and is achieved by $b = 4$, that is, the filter looks like this:

$$H = \frac{1}{36}\begin{bmatrix} 1 & 4 & 1 \\ 4 & 16 & 4 \\ 1 & 4 & 1 \end{bmatrix}.$$

We denote the image cleaned by low-pass filter with $b = 4$ as R_2, and the original image I and the cleaned image R_2 are displayed as follows, respectively.

sample1.png

sample1.gaussian.low.pass.png: N_2 cleaned by low-pass filter with $b = 4$

The noisy image N_2 and the cleaned image R_2 are also displayed so that we can observe the improvement.

sample1.gaussian.png: Image I with Gaussian noise ($\mu = 0, \sigma = 1, \delta = 10$) added

sample1.gaussian.low.pass.4.png: N_2 cleaned by low-pass filter with $b = 4$

In my opinion, the improvement is passable and obviously imperfect. To further convince us that $b = 4$ is the best choice, let's take a look at the resulting images cleaned by $b = 1$ (uniform) and 10:

sample1.gaussian.low.pass.1.png: N_2 cleaned by low-pass filter with $b = 1$

sample1.gaussian.low.pass.10.png: N_2 cleaned by low-pass filter with $b = 10$

The PSNR values achieved are 29.78 and 30.69, respectively, both are lower than $b = 4$ (31.74). According to my observation, $b = 1$ looked blurrier than $b = 10$, while $b = 10$ still exhibited significant Gaussian noise, comparing with N_2. $b = 4$ somehow locates between them and can be reluctantly considered the best choice.

(d) Compute the PSNR values of R_1 and R_2 and provide some discussions.

As mentioned in question (c), the outlier detection with $\varepsilon = 65$ is selected as R_1, and low-pass filter with $b = 4$ is selected as R_2. The PSNR values achieved by R_1 and R_2 are 38.86 and 31.74, respectively. The discussion was already presented in question (c).