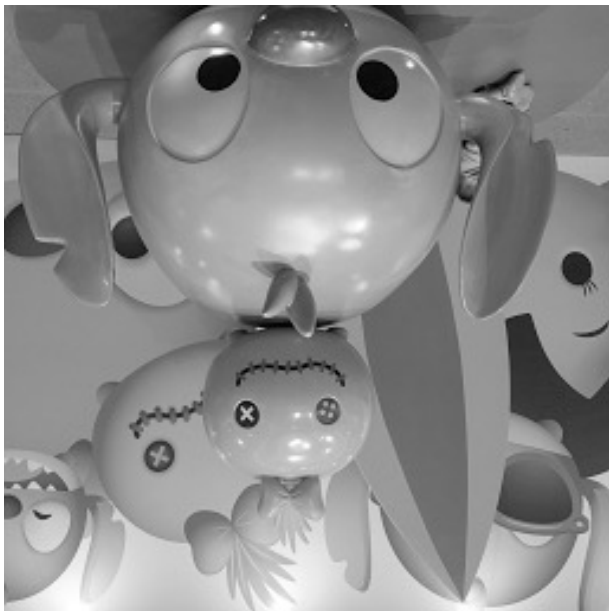
Name: Steven Lin. 林于智
Student ID: R04922170

Warm-up

The flipped images are shown below.

Horizontally Flipped.

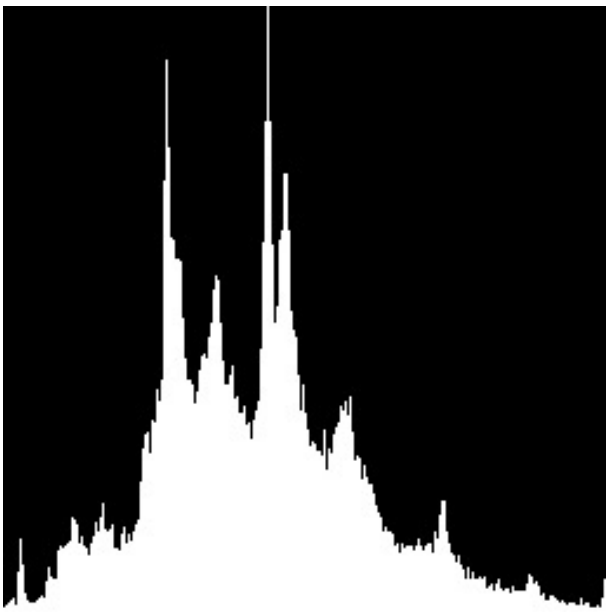


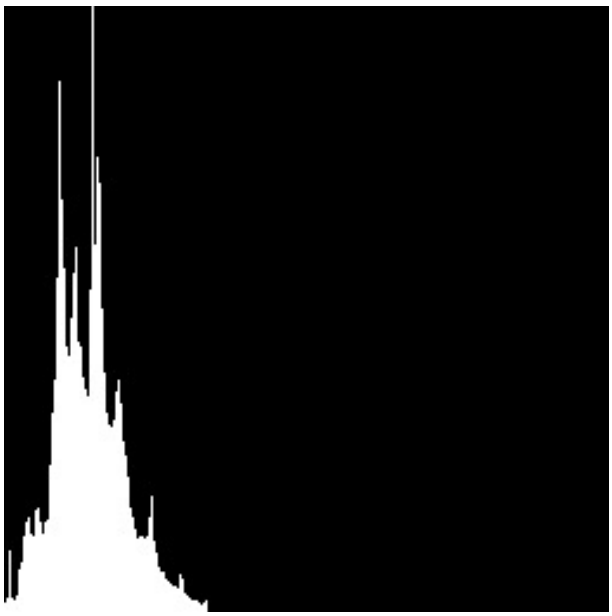
Vertically Flipped.

Problem 1

(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?

The histogram of I and D are shown below. (***left:*** histogram of I; ***right:*** histogram of D.)





The histogram of image I shows that it has a wider range of pixel value (intensity) than image D does. And therefore we may feel lack of contrast when looking at image D.

To make D look like I, we need to do “contrast manipulation”. Such techniques include several transformations; for example, log transform, which we’ll be using later on, is one of the common way to fix this.

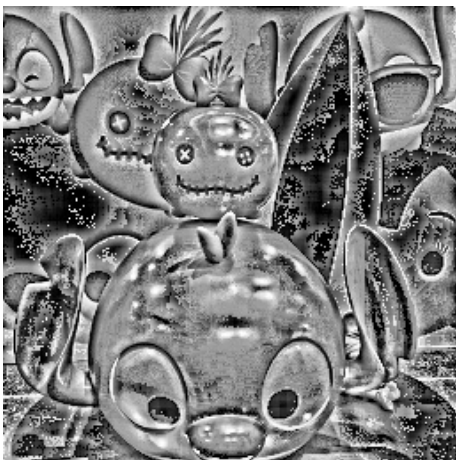
(b) Perform histogram equalization on D and output as H.

The output is attached below:

“H”, the result of performing histogram equalization on image D.

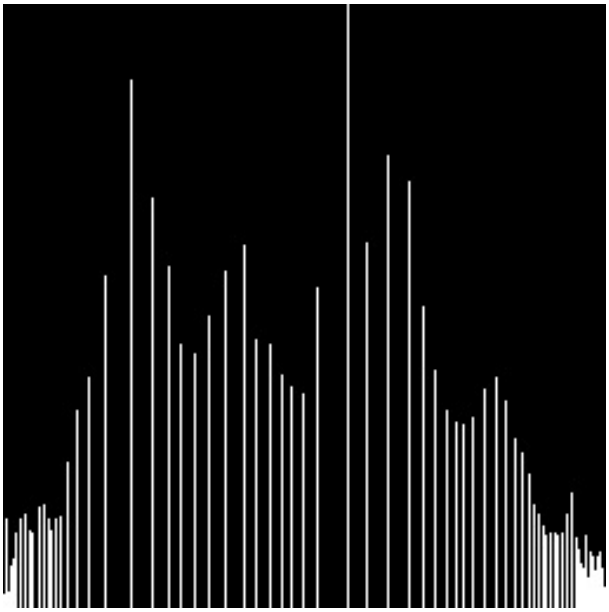
(c) Perform local histogram equalization on D and output as L.

The output is attached below:

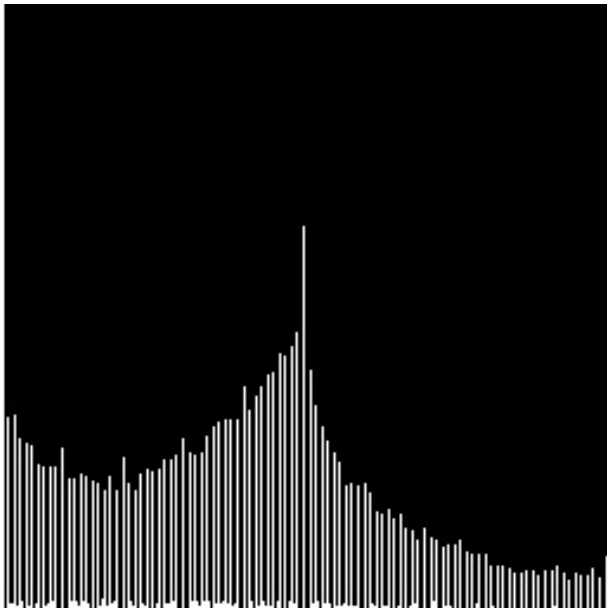


“L”, after performing local histogram equalization on D.
(with window size = 10 x 10, i.e., contextual region.)

(d) Plot the histograms for H and L. What's the main difference?



(histogram equalization)

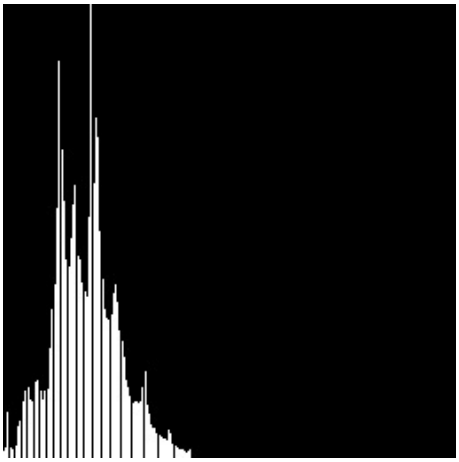


(local histogram equalization)

After histogram equalization, the histogram of the resulted image still has a trend (shape) that resembles the histogram of the original image. It's because histogram equalization is done globally, and pixel values are slightly distributed from high-density area to low-density area. However, on the other hand, local histogram equalization is done only by checking the contextual area (neighboring area) of a pixel, and that's the reason why the resulted image has a much different histogram from the original image.

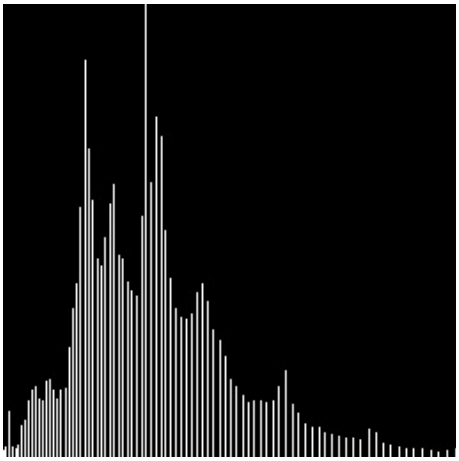
(e) Perform log transform, inverse log transform, and power-law transform to enhance D.

Log Tranform on D.



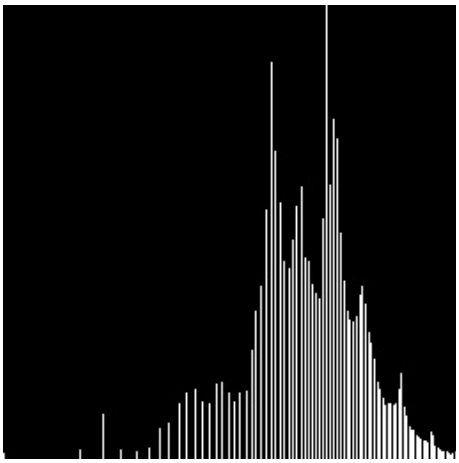
Histogram of Log Transform on D.

Inverse Log Transform on D.



Histogram of Inverse Log Transform on D.

Power-Law Transform on D.



Histogram of Power-Law Transform on D.

Parameters:

Log transform: ***output*** = (log(1 + ***input*** / 255) / log(2)) * 255

Inverse log transform: ***output*** = (exp(***input***/MAX) - 1) / (exp(1) - 1) * 255
(where MAX is the maximum intensity in the original image)

Power-Law transform: ***output*** = (pow(***input***, gamma) - pMIN / pMAX - pMIN) * 255
(where pMIN = pow(MIN, gamma), pMAX = pow(MAX, gamma), and gamma = 2/5)

Basically they all follow the concepts described on the course slides.

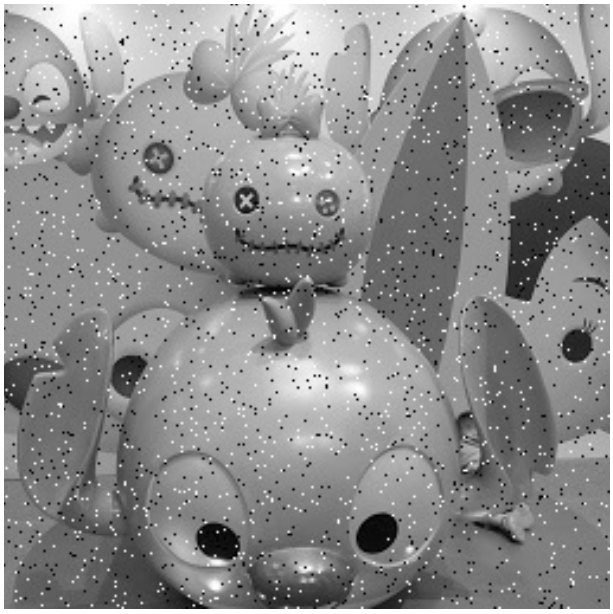
Some discussion:

Log transform makes super-dark areas brighter, but do less to the “already-bright” ares because of the shape of the curve of logarithm. The curve rises more rapidly when input is relatively small, and the increasing tendency slows down when input gets larger. Power-Law transform is apparently brighter than the other 2 transformation because we have ***gamma=2/5***, which makes the distance of

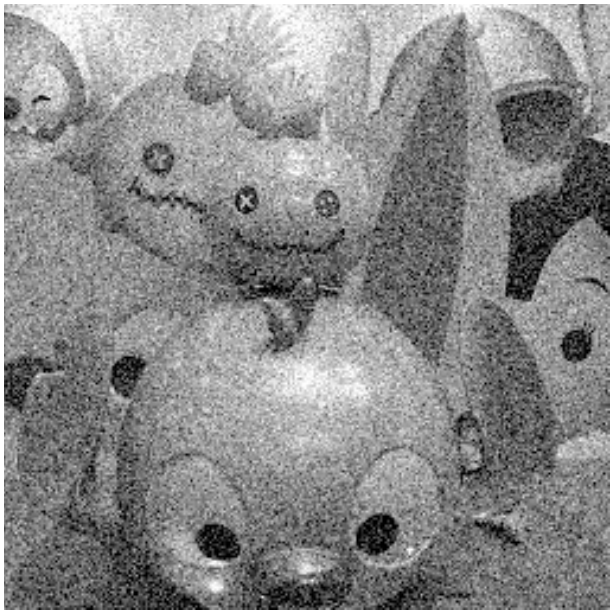
pixel values smaller than they were before. (For example, the two number, ***980*** and ***810*** have a distance of 170, but ***pow(980, 2/5)*** is about 1.15 bigger than ***pow(810, 2/5)*** only.) After the distances are shrunk, we re-map the pixel values to [0, 255] linearly, making “a little bit bright areas” a lot brighter.

Problem 2

(a) The following is N1, the result of adding salt-and-pepper noise to I.



(b) The following is N2, the result of adding uniform noise to I.



(c) De-noise N1 and N2

De-noise N1 with a 3 x 3 median filter:

First, the result is as follows. (Image R1.)

I turn to median filter for de-noising the salt-and-pepper noise. The median filter I used is of size 3 x 3, and the way to handle boundary pixels is the "***Extend***" solution. (more information here: https://en.wikipedia.org/wiki/Kernel_(image_processing))

De-noise N2 with a 3 x 3 low-pass filter:

The result is below. (Image R2.)

To get rid of this kind of uniform noise, low-pass filter is a good choice. The low-pass filter I used here has the form as

$$\begin{matrix} 1 & b & 1 \\ b & b^2 & b \\ 1 & b & 1 \end{matrix}$$

, and I set b as 2. (looks acceptable.)

(d) Compute PSNR values of R1 and R2.

The result of PSNR calculation is:

```
PSNR(I, R1) = 32.6395
PSNR(I, R2) = 25.9544
```

The formula on the course slides is followed when calculating the PSNR values.

PSNR values actually make no sense to human beings because we could not differentiate between PSNR=150 from PSNR=160; however, it is a quantitative indication of the performance of de-noising. As we can tell from the resulted images attached in the above section, we retrieved a better de-noised image using median filter on the noisy image with salt-and-pepper noise than using low-pass filter on the other image with uniform noise. PSNR values also tell the same thing! Therefore, we can still use PSNR values as benchmarks to roughly compare/see the result(s) of de-noising.