

# DIP Homework #1

ID #: B04902028

Department & grade: CSIE 3

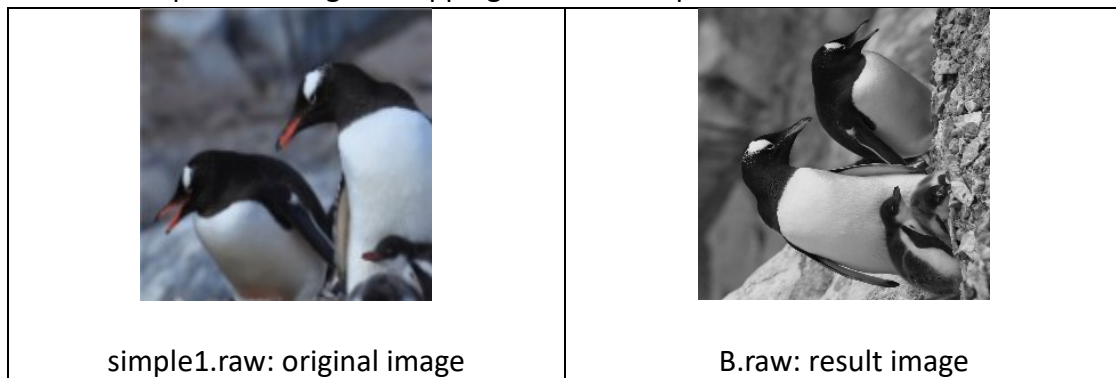
Name: 洪浩翔

Email: [b04902028@ntu.edu.tw](mailto:b04902028@ntu.edu.tw)

Submit time: 03/18/2018

## 1. Warm up: SIMPLE MANIPULATIONS

Please convert the given color image I1 as shown in Fig.1 to a gray-level one. Please also perform diagonal flipping on it and output the result as B.

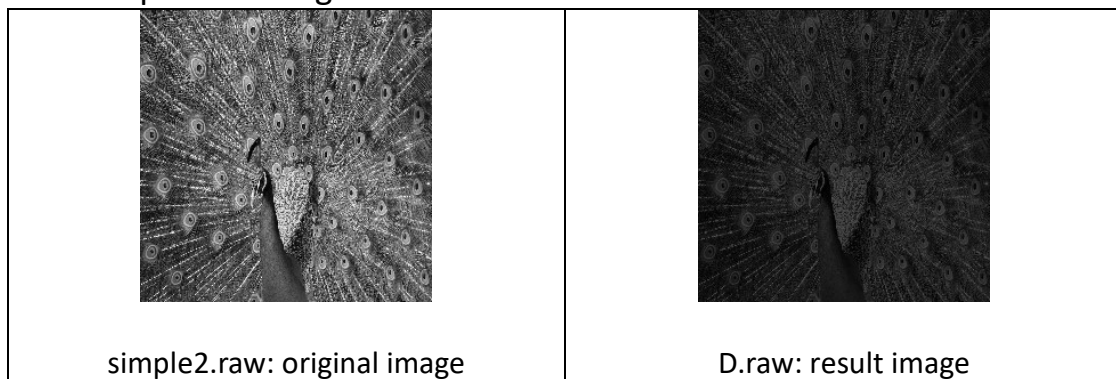


I use the method of “ $\text{grayscale} = \text{round}(0.2989 * R + 0.5870 * G + 0.1140 * B)$ ” to convert RGB image to grayscale image, and run diagonal flipping on it.

## 2. PROBLEM 1: IMAGE ENHANCEMENT

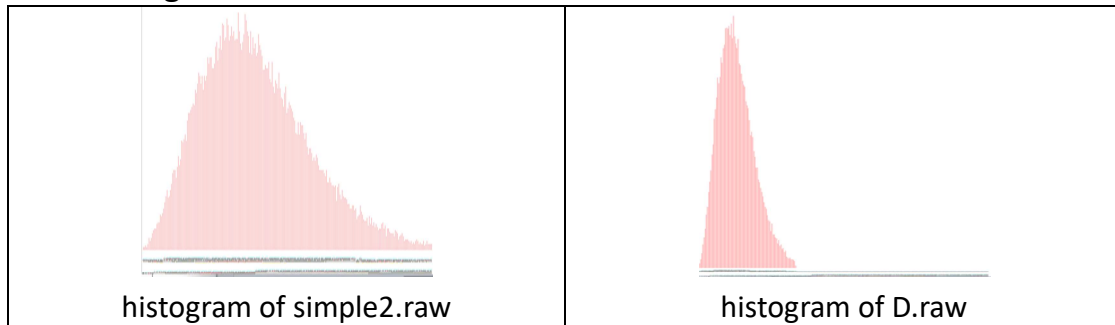
Given an image I2 as shown in Fig. 2. Please follow the instructions below to create several new images.

(a) Decrease the brightness of I2 by dividing the intensity values by 3 and output the image as D.



I just divide all the intensity values by 3 and save them as a new image D. It is obvious that the result image D is darker than the original one.

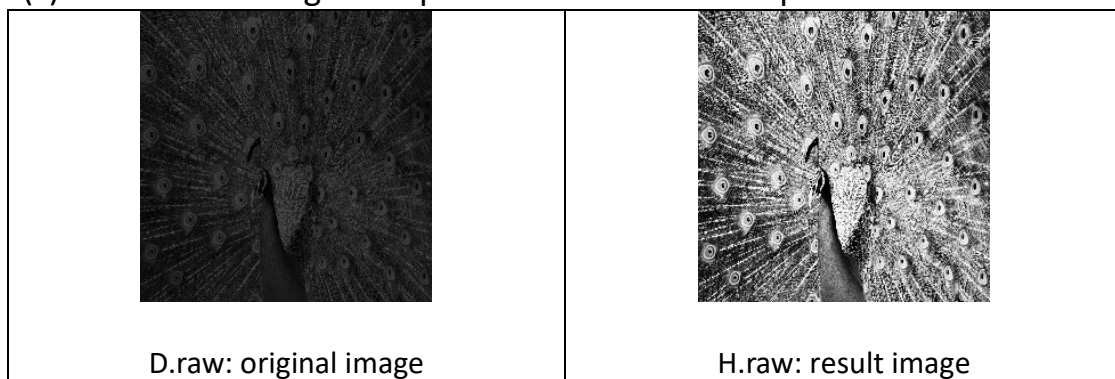
(b) Plot the histograms of I2 and D. What can you observe from these two histograms?



Unlike MATLAB, plot histogram is not easy in cpp. Therefore, I write out the result of histograms as .txt file and snapshot them.

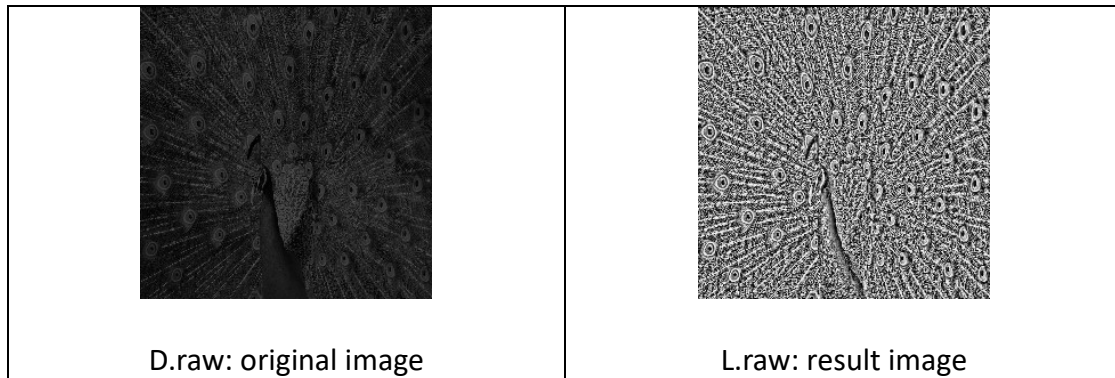
Comparing between these two histograms, it is not hard to find that pixel values in D gather at the dark side, not like that of simple2. That is, the range of pixel value of D is smaller than that of simple2. This causes the image darker because it is hard for human eyes to distinguish the difference between each pixel. In addition, they have a very similar shape, because histogram of D is just doing a scaling on x-axis of the histogram of simple2.

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



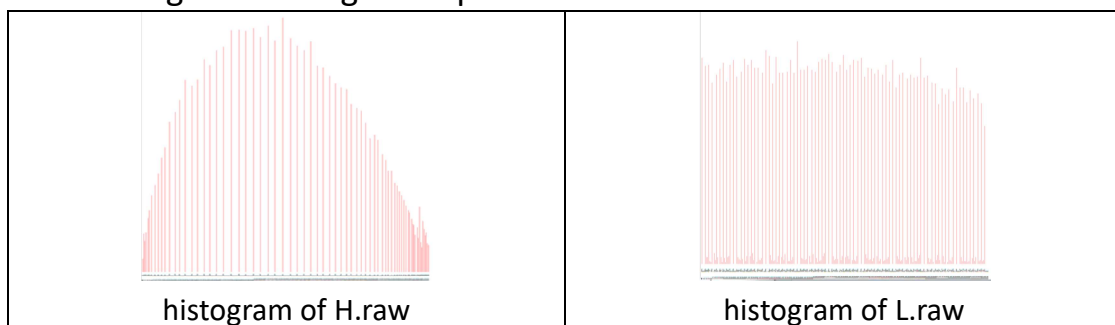
Adjust histogram equalization on D, and it will make the distribution better than the original. This is because histogram equalization will spread out the frequent value, making the range of pixels larger and the distribution better. That is why the result image H is brighter and better.

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







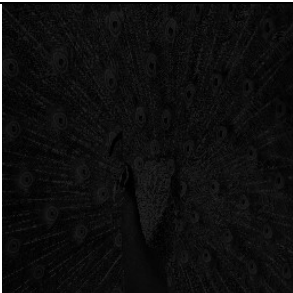
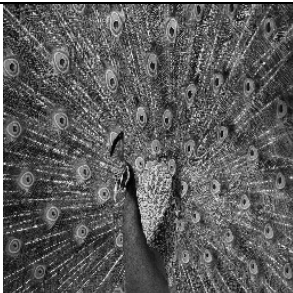

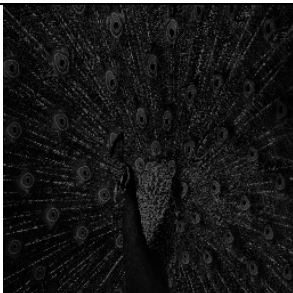
I use a  $9 \times 9$  kernel to adopt local histogram equalization. That is, conduct histogram equalization in every  $9 \times 9$  box, but only update the central value. If comparing with the global one, the local one will emphasize on local part and make local detail much clear. That is why the result image emphasizes on details and makes itself so dazzled.

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



Just like mentioning above, local histogram equalization emphasizes on local details, while the global one emphasizes on the global most frequent value. This condition can be found in the histograms. The histogram of global histogram equalization spread out the highest frequent values, while the local one almost spread out every value to emphasize the details.

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

 <p>D.raw: original image</p>	 <p>log_trans_100.000000.raw: result image with c = 100</p>
 <p>log_trans_886.395020.raw: result image with c = 886.395020</p>	 <p>log_trans_1000.000000.raw: result image with c = 1000</p>
 <p>invlog_100.000000_trans.raw: result image with c = 100</p>	 <p>invlog_644.570190_trans.raw: result image with c = 644.570190</p>
 <p>invlog_1000.000000_trans.raw: result image with c = 1000</p>	 <p>powerlaw_trans_2_1000.000000.raw: result image with c = 1000 and p = 2</p>



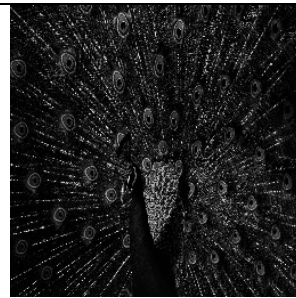
powerlaw\_trans\_2\_2294.999863.raw:  
result image with  $c = 2294.999863$  and  $p = 2$



powerlaw\_trans\_2\_10000.000000.raw:  
result image with  $c = 10000$  and  $p = 2$



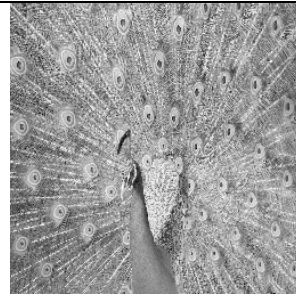
powerlaw\_trans\_3\_1000.000000.raw:  
result image with  $c = 1000$  and  $p = 3$



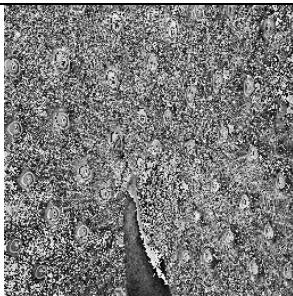
powerlaw\_trans\_3\_6884.999384.raw:  
result image with  $c = 6884.999384$  and  $p = 3$



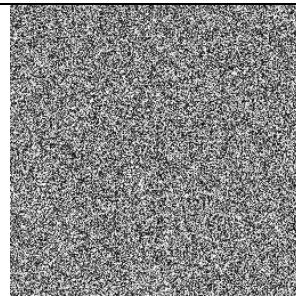
powerlaw\_trans\_3\_10000.000000.raw:  
result image with  $c=10000$  and  $p=3$



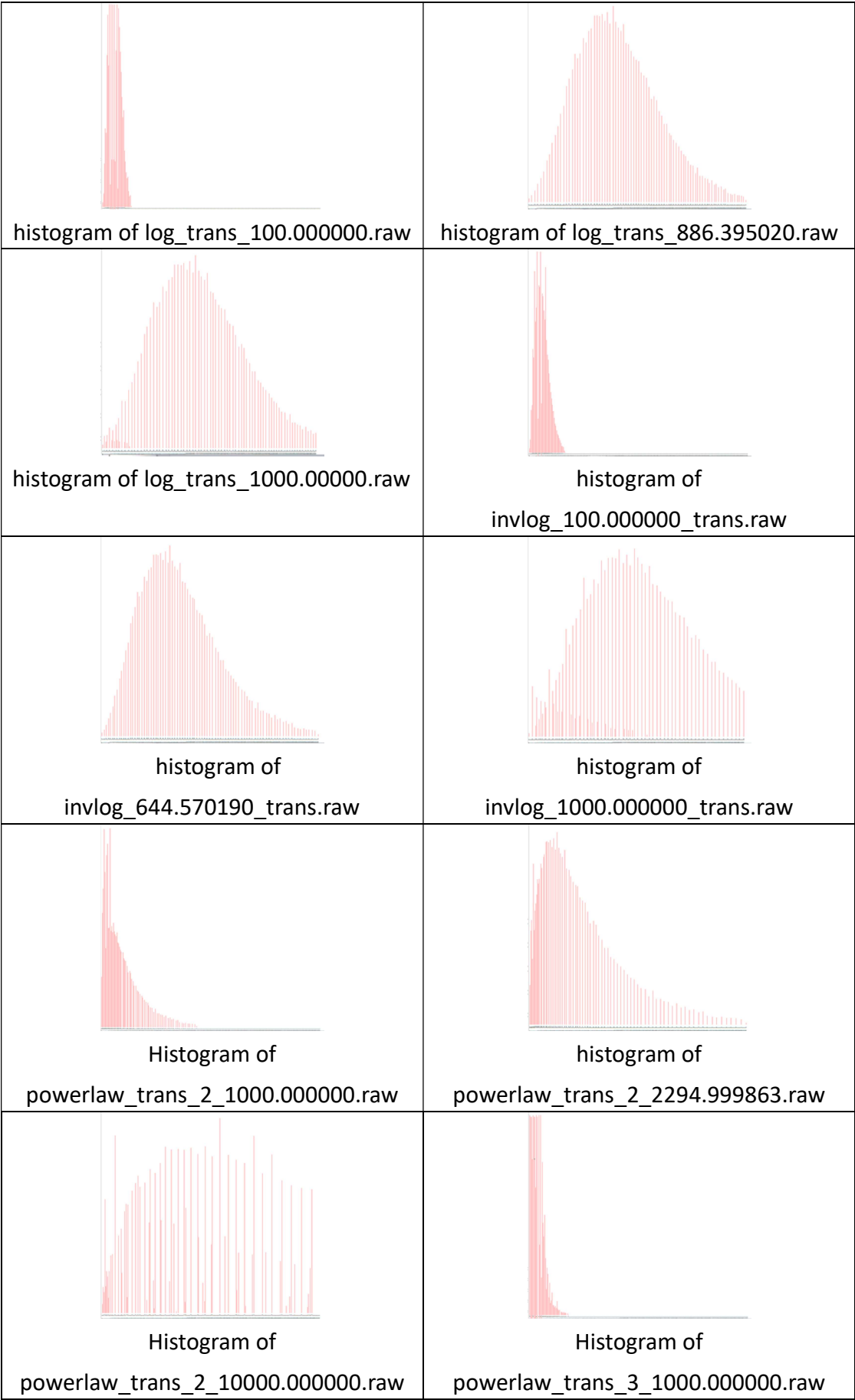
powerlaw\_trans\_0.5\_441.672974.raw:res  
ult image with  $c=441.672974$  and  $p=0.5$

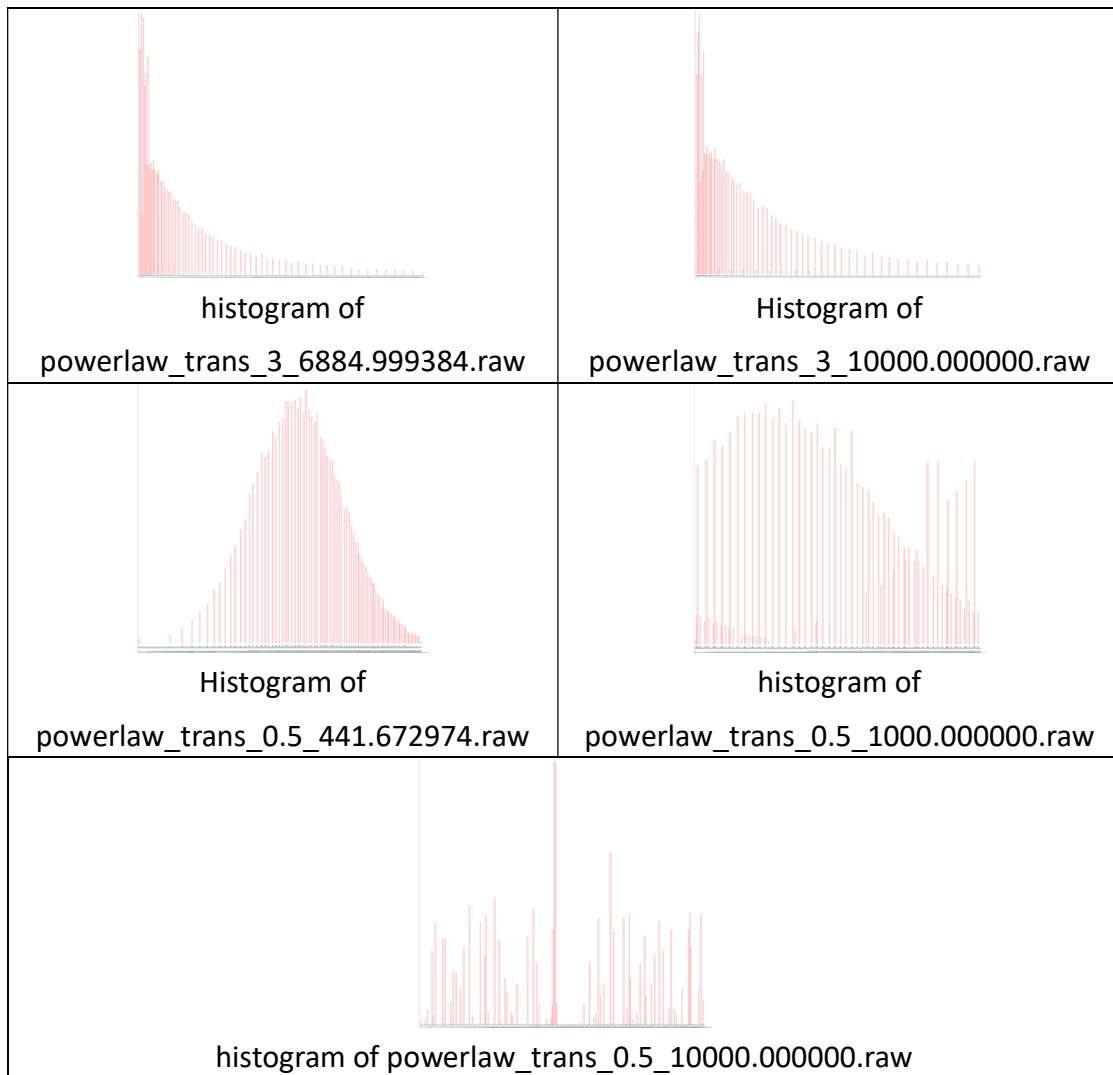


powerlaw\_trans\_0.50000\_1000.000000.  
raw:result image with  $c=1000$  and  $p=0.5$



powerlaw\_trans\_0.50000\_10000.000000.  
raw:result image with  $c=10000$  and  $p=0.5$





### **Log transform:**

There are three images for log transform with  $c = 100$ ,  $1000$ ,  $255/(\log(1 + \max\_value))$ , respectively. Obviously, the image with  $c = 100$  is too dark. The images with  $c = 1000$  and  $c = 255/(\log(1 + \max\_value))$  has just a little different. The reason why choosing  $c = 255/(\log(1 + \max\_value))$  with  $\max\_value$  being the maximum pixel value in the image is that this constant can ensure the maximum magnitude of the image being 255. Therefore, the  $c$  value closing to  $255/(\log(1 + \max\_value))$  can have a better result than those far from the value of the formula.

### **Inverse log transform:**

Similar with log transform, the image with  $c = 100$  is too dark. The image with  $c = 1000$  is too bright, with some weird pixel values in it. The image with  $c = 255/(\exp(\max\_value)-1)$  has a better result. The reason why choose this  $c$  value is the same as what mentioning in log transform.

Comparing the best result with that of log transform, there is almost no

difference between them. It is hard for human eyes to distinguish these two images. The other evidence to prove this condition is that the histograms of these two images are very similar with each other.

### **Power law transform:**

In this part, I try 9 images with  $p = \{0.5, 2, 3\}$  and  $c = \{1000, 10000, 255/(\text{pow}(\text{max\_value}, p))\}$  respectively to find the best result. Obviously,  $p > 1$  seems not a good choice because the images are too dark. Therefore,  $p \leq 1$  is a better choice than  $p > 1$  if compare the results. In addition, just like mentioning above,  $c$  value close to the value of the formula can make a better result. Too large  $c$  results in destroying the image, while too small  $c$  makes the result too dark. I think the reason  $p \leq 1$  making a better result is that the most frequent values in the original image gather at the dark side, and  $p \leq 1$  makes the power law transform similar with log transform or inverse log transform. Therefore, details in dark side will be spread out, making the image better. To conclude, in this case,  $p \leq 1$  and  $c = 255/(\text{pow}(\text{max\_value}, p))$  make a better result.

In my opinion, power law transform with  $c = 255/(\text{pow}(\text{max\_value}, p))$  and  $p = 0.5$  seems to be a best result. The result image is brightest and no weird pixels, letting me comfortable. However, maybe someone else won't think so. The answer of "which one is the best?" may be different, depending on everyone.

## **3. PROBLEM 2: NOISE REMOVAL**

(I) Given an image I3 as shown in Fig. 3(a), please follow the instructions below to create some new images.

(a) Please generate two noisy images G1, and G2 by adding Gaussian noise to I3 with different parameters. What's the main difference between these two images?



G1.raw: result image with mean = 0,  
deviation = 1, amplitude = 10

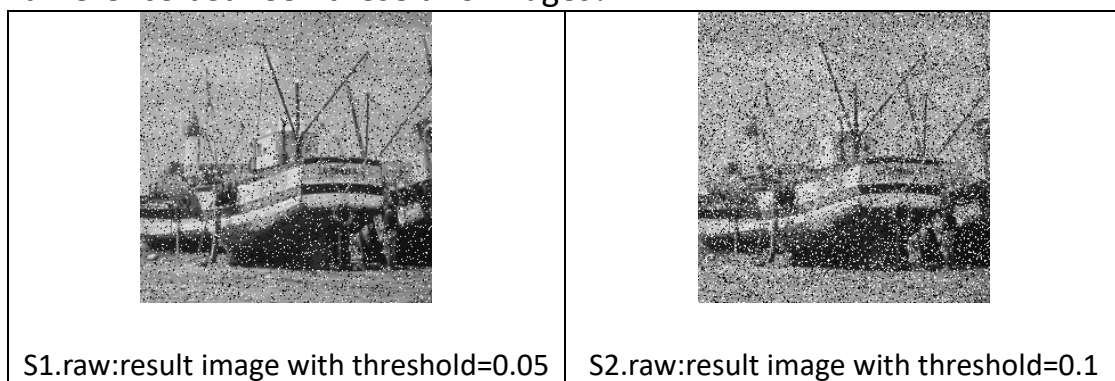


G2.raw: result image with mean = 0,  
deviation = 1, amplitude = 30





Both G1 and G2 are created by mean = 0 and deviation = 1. The only difference is that G1 is created by Gaussian noise with amplitude = 10, while G2 is created by Gaussian noise with amplitude = 30. It is obvious that G2 has much noise than G1. This is because high amplitude results in much noise according to the Gaussian noise formula. Higher amplitude values will scale up the effect of the random numbers generated by the normal distribution, resulting in the noisy images.

(b) Please generate two noisy images S1, and S2 by adding salt-and-pepper noise to I3 with different parameters. What's the main difference between these two images?



S1 is generated by salt-and-pepper noise with threshold = 0.05, while S2 is generated with threshold = 0.1. Similar with the condition of Gaussian noise, it seems that larger threshold of salt-and-pepper noise results in much noisy image according to the formula and the feature of normal distribution. In S2, the objects in the original image are hard to be distinguished because of noise, while S1 still can be distinguished.

(c) Design proper filters to remove noise from G1 and S1, and denote the resultant images as RG and RS, respectively. Please detail the steps of the denoising process and specify corresponding parameters. Provide some discussions about the reason why those filters and parameters are chosen.

 <p>RG.raw: using 3*3 box filter to remove Gaussian noise</p>	 <p>RS.raw: using 3*3 median filter to remove salt-and-pepper noise</p>
--	---

### **G1 -> RG**

For G1->RG, I choose box filter with 3\*3 sizes to remove the noise. The kernel is  $(1/9)*[[1, 1, 1], [1, 1, 1], [1, 1, 1]]$ . I run this kernel on every pixel and store the updated values as the recovered image. The reason why choosing box filter to remove noise is that it is a little faster than median filter, and the result is not bad because Gaussian noise is just adding some uniform noise on the image, and average of these noises and some original pixels is enough to smooth the image. In fact, this is the feature of low-pass filter. In addition, the reason why choosing 3\*3 box size is that in my opinion, too large box size makes image blurred, and the result will be worse than the smaller one. To conclude, that is why I choose box filter with 3\*3 box size as my filter to remove Gaussian noise.

### **S1 -> RS**

For S1->RS, I choose median filter with 3\*3 sizes to remove the noise. The kernel is the same as box filter above. I run this kernel on every pixel, find the median value in these 9 pixel value as the result value, and store the updated pixels as the result image. The reason to choose the median filter is that salt-and-pepper noise is not like Gaussian noise. It is a kind of impulse noise. As a result, the result of box filter in this case may not as good as median filter. The noise in salt-and-pepper noise is actually 0 and 255 these two values, which is two extreme side of pixel value. Therefore, median operation can easily remove these noises by picking up the median value in 3\*3 box. This is because in most of the time, this median value may not be the extreme values 0 and 255. Therefore, median filter is a good choice here. In addition, the reason of choosing 3\*3 box size is the same as mentioning above. Too large box size makes the image blurred. To sun up, that is why I choose median filter with 3\*3 box size as my filter to remove salt-and-pepper noise.

(d) Compute the PSNR values of RG and RS and provide some discussions.

```
PSNR between G1 and RG: 58.22
PSNR between sample3 and RG:
PSNR between S1 and RS: 35.23
```

PSNR results between G1 and RG, sample3 and RG, S1 and RG, sample3 and RS

In the definition of PSNR, the higher value some parts means a better result, because it means that there is a large distortion if the value is small, and vice versa. According to PSNR value in above image, it seems that RS has a better result than RG. In my opinion, I think RS is better as well. However, PSNR is just an objective value. Sometimes it is still depending on human eyes to decide an image is good or not. Additionally, PSNR of S1 and RS is the lowest in these 4 PSNR. It seems that median filter really impact salt-and-pepper noise a lot.

(II) Design your own method to remove the wrinkles on the face of a given image I4 as shown in Fig. 3(b) and make it as pretty as you can. Please describe the steps of your process in detail and provide some discussions as well.



rm\_wrinkle\_box\_filter\_3.raw: use 3\*3  
box filter to remove wrinkles





rm\_wrinkle\_box\_filter\_5.raw: use 5\*5  
box filter to remove wrinkles


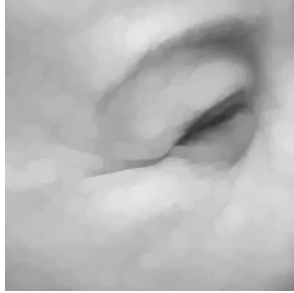




rm\_wrinkle\_median\_filter\_3.raw: use  
3\*3 median filter to remove wrinkles



rm\_wrinkle\_median\_filter\_5.raw: use  
5\*5 median filter to remove wrinkles

 <p>rm_wrinkle_dilation.raw: use dilation operation to remove wrinkles</p>	 <p>rm_wrinkle_opening.raw: use opening operation to remove wrinkles</p>
---	--

 <p>test1.raw: opening first then dilation</p>	 <p>test2.raw: dilation test1.raw</p>
 <p>test3.raw: add test2 and the original image then divide it by 2</p>	 <p>test4.raw: do dilation on test2 4 times and add the result with the original image. Then divide it by 2</p>

I try 4 kinds of main method here to produce 4 images and the other 4 images are produced by the combination of different methods. First, I try box filter with 3\*3 and 5\*5 box size to remove wrinkles. The result is not good. Most of the wrinkles are still on the face. Box filter with 5\*5 box size also makes the image blurred. Second, I try to remove wrinkles by median filter with 3\*3 and 5\*5 box size. The result is a little bit better than box filter, but it is still not good enough. Median filter with 5\*5 box size also makes blurred on the image. After experiencing the failure of box filter and median filter, I try to find other ways to improve the result. Therefore, I choose opening and dilation operation as my method to remove wrinkles. Dilation will

remove lots of wrinkles because the pixel values of the wrinkles are at the dark side, and dilation will replace these values with brighter pixels, but dilation also remove some details. However, opening doesn't do so. Opening operation remove many wrinkles with many details maintained because it is a combination of dilation and erosion. As a result, I try to improve the result with the combination of these two methods. First, I try to do opening first then do dilation. The result seems good. Then, I try to do dilation on the result again. Almost every wrinkle is removed, but some details are removed as well. However, the result is still bearable. To improve this condition, I try to add the result with the original image and divide the result by 2, in order to maintain as many details as it can. This method results in test3.raw, an image combined with many details and less wrinkles. Although not like test2.raw which remove almost every wrinkle, test3.raw looks realer than test2.raw. This is indeed a kind of trade-off between removing wrinkles and maintain details. I still try to improve the result. Therefore, test4.raw is produced. I try to do dilation on test2 4 times and add the result with the original image. Then divide the result by 2. The result is similar with test3.raw, while test4.raw is brighter. I try many other methods after creating test4.raw, but all of them can't be better than test4.raw. Therefore, the best result is either test2.raw or test4.raw.

#### **Dilation:**

Use octagon kernel 5-8-8-8-5 and find the largest pixel value in the kernel. Then use the largest value as the new pixel value of the central pixel.

#### **Erosion:**

Use octagon kernel 5-8-8-8-5 and find the smallest pixel value in the kernel. Then use the smallest value as the new pixel value of the central pixel.

#### **Opening:**

Do erosion first, then, do dilation.

## README

# DIP Homework Assignment #1

# 03/18/2018

# Name: 洪浩翔

# ID: B04902028

# email: b04902028@ntu.edu.tw

# compiled on ubuntu 16.04.3 LTS with g++11

# output name and file is decided, so no need to change

# image path should be "../raw/\*.raw" and output image will be here as well

# histogram file will be the same place as the main code

# type make -f README to compile and run

# below is for makefile

.PHONY: all

CC=g++

LN=g++

All: prob1

prob1 :

    @echo "hw1"

    @echo "compiling and linking the code"

    \$(CC) -std=c++11 -c hw1.cpp

    \$(LN) -std=c++11 -o hw1 hw1.o

    @echo "running the program"

    ./hw1