Digital Image Processing, Spring 2018

Homework 1
DUE DATE: March 28, 2018

Student ID: B03902129 Department: CSIE Name: Peng-Yu Chen

README

To run my program, simply type **README** in the Command Window of MATLAB application, then it'll run all .m files and output the .raw images.

Listing 1: README.m

```
% DIP Homework Assignment #1
% March 28, 2018
% Name: Jay Chen
% ID #: B03902129
% email: b03902129@ntu.edu.tw

%##########################################################################
% Add path first
%##########################################################################

disp('Add path ./warmup, ./prob1, ./prob2 and ./readwriter');
addpath('./warmup');
addpath('./prob1');
addpath('./prob2');

disp('Make a parent folder ./outputs');
mkdir . outputs

%##########################################################################
% WARM—UP: SIMPLE MANIPULATIONS
% Implementation 1: Convert color image(I1) —> gray—level image(gray)
% Implementation 2: Perform diagonal flipping, I1 —> B
% M—files: warmup.m
% Usage: run warmup
% Output: B.raw
%##########################################################################

fprintf('————————————————————————————————————————\n');
fprintf('Running warmup\n————————————————————————————————————————\n');
warmup();

%##########################################################################
% Problem 1: IMAGE ENHANCEMENT
% Implementation 1: Decrease the brightness, I2 —> D
% Implementation 2: Plot the histograms of I2 and D
% Implementation 3: Perform histogram equalization, D —> H
% Implementation 4: Perform local histogram equalization, D —> L
% Implementation 5: Plot the histograms of H and L
% Implementation 6: Perform the log transform, inverse log transform
%                   and power-law transform
% M-files: prob1.m, plotHist.m, histEq.m, localhist.m and trans.m
% Usage: run prob1 to call other .m files
% Outputs: D.raw, H.raw and L.raw
% Parameters:
%       * Local Histogram Equalization: window size = 3, 5, 7, ..., 255
%       * Log Transformation: c = 5, 15, 25, ..., 85
%       * Inverse Log Transformation: c = 1, 2, 3, ..., 9
%       * Power-Law Transformation: p = 0.2, 0.4, 0.6, ..., 3.0
%#########################################################################

fprintf('-----------------------------------------\n');
fprintf('Running prob1\n---------------------------------------\n');
prob1();

%#########################################################################
% Problem 2: NOISE REMOVAL
% Implementation 1: Generate Gaussian noise image, I3 -> G1 / G2
% Implementation 2: Generate salt-and-pepper noise image, I3 -> S1 / S2
% Implementation 3: Design Low Pass Filter, G1 -> RG
% Implementation 4: Design Median Filter, S1 -> RS
% Implementation 5: Computer PSNR
% Implementation 6: Remove the wrinkles
% M-files: prob2.m, addGaussianNoise.m, addSaltAndPepper.m, lowPassFilter.m
%          squareMedianFilter.m, crossMedianFilter.m, PSNR.m and
%          removeWrinkle.m
% Usage: run prob2 to call other .m files
% Output: None
% Parameters:
%       * 3 x 3 Low Pass Filter: b = 1, 2, 3, ..., 30
%       * n x n Square Median Filter: window size = 3, 5, 7, ..., 15
%       * n + n - 1 Cross Median Filter: cross size = 3, 5, 7, ..., 15
%       * Wrinkle remover: threshold = 3
%                          crossMedianFilter with cross size = 15
%#########################################################################

fprintf('-----------------------------------------\n');
fprintf('Running prob2\n---------------------------------------\n');
prob2();
```

WARM-UP: SIMPLE MANIPULATIONS

Please convert the given color image I_1 as shown in Fig.1 to a gray-level one. Please also perform diagonal flipping on it and output the result as B.
To get gray-level image, we add Red, Green and Blue colors, then divide them by 3.
To perform diagonal flipping, we use **permute** function.

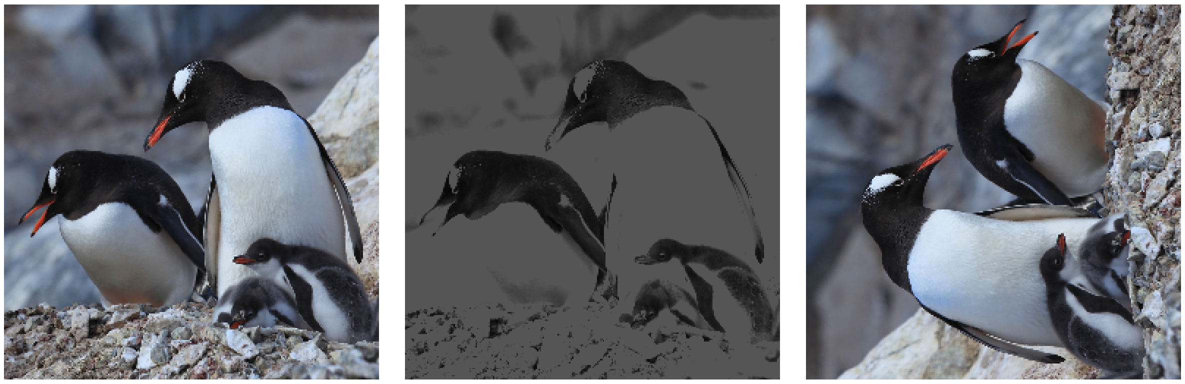

(a) sample1.raw (b) gray-level (c) diagonal flipping

Figure 1: Penguins

PROBLEM 1: IMAGE ENHANCEMENT

Given an image I_1 as shown in Fig. 2. Please follow the instructions below to create several new images.

(a) Decrease the brightness of I_2 by dividing the intensity values by 3 and output the image as D.

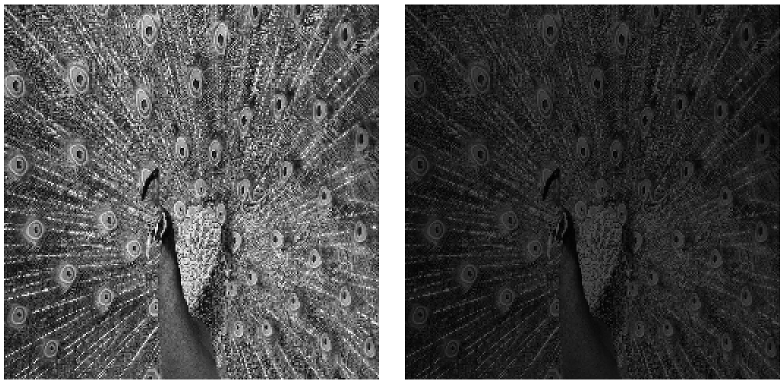
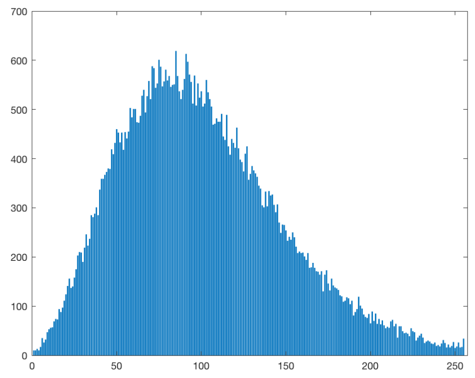

(a) sample2.raw (b) D.raw

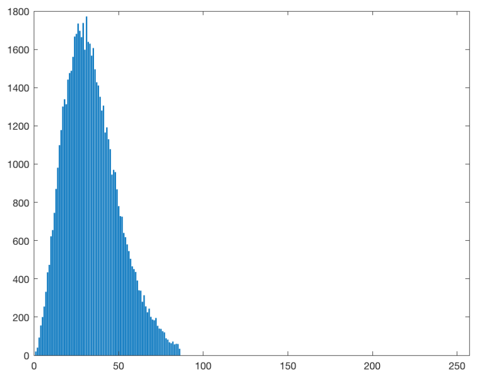
Figure 2: Darker peacock

(b) Plot the histograms of I_2 and D.

Since we get D by dividing I_2 by 3, we can see the histogram of D squeezes to the left dramastically.



(a) histogram of I_2



(b) histogram of D

Figure 3: Histogram comparison after decreasing brightness

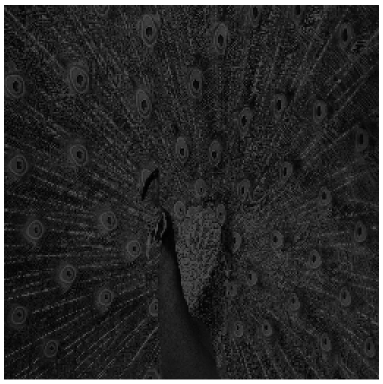
(c) Perform histogram equalization on D and output the result as H.

Implementation: let n_i be the number of occurrences of gray level i, L be the total number of gray levels in the image (here $L = 256$) and $cdf_x(i) = \sum_{j=0}^{i} n_j$.

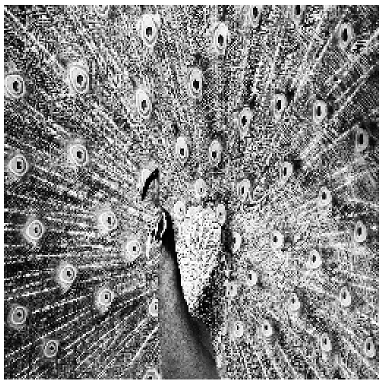
We can get the general histogram equalization formula is:

$$h(i) = \text{round}\Big(\frac{cdf(i) - cdf_{\min}}{M \times N - 1} \times (L - 1)\Big).$$

After performing histogram equalization on D, the global contrast will increase. The intensities can be better distributed on the histogram.



(a) D.raw



(b) H.raw

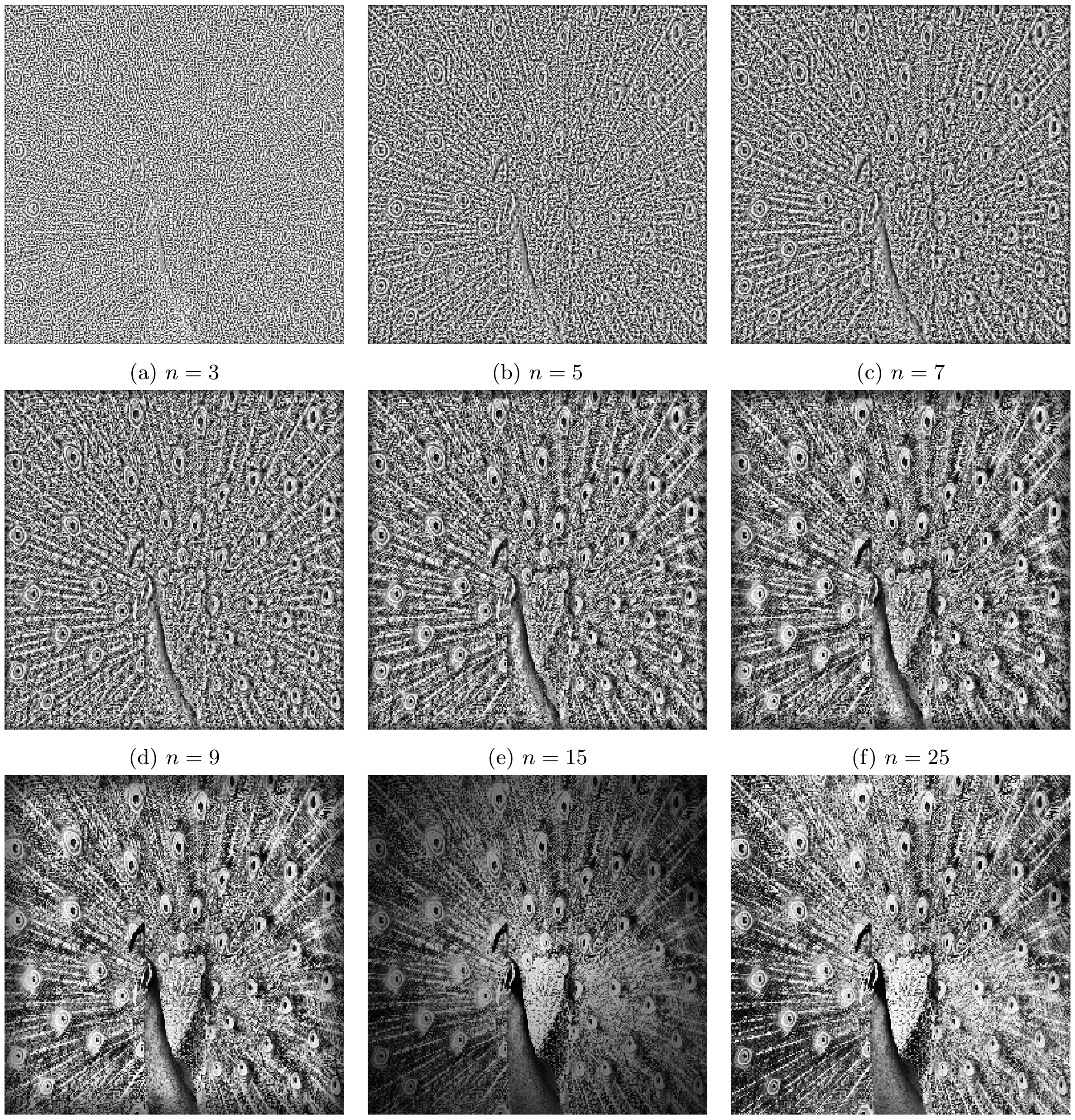
Figure 4: Histogram equalization comparison

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

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

When the $window.size$ is small ($n \leq 7$), the effect is not noticeable.
When the $window.size$ is large ($n > 20$), the contrast become much more readable and it looks similar to H.raw (global histogram equalization).

The main difference is local histogram equalization can distinguish more contrast and can adjust the $window.size$ flexibly.



(a) $n = 3$ (b) $n = 5$ (c) $n = 7$

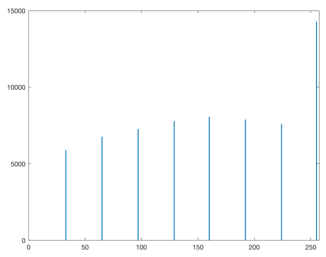
(d) $n = 9$ (e) $n = 15$ (f) $n = 25$

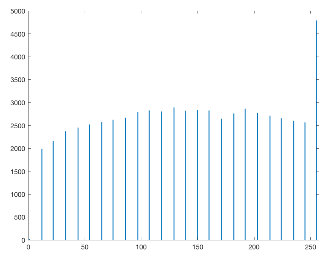
(g) $n = 45$ (h) $n = 255$ (i) H.raw

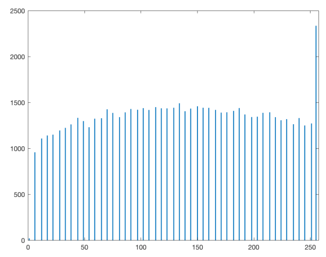
Figure 5: Local Histogram Equalization with different window sizes n

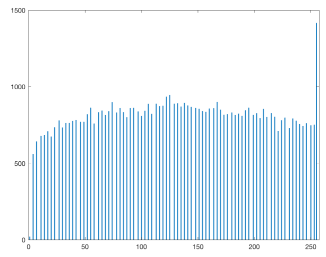
When the $window.size = 45$, the histogram distribution is the most uniform, therefore the picture of which is the most distinguishable, too.
When the $window.size = 255$, the histogram left oblique, it means that the picture is darker in that one.

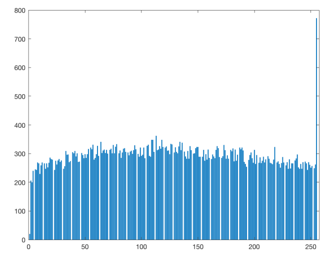
In this graph, it seems that the global histogram equalization is better than any other local histogram equalizations.

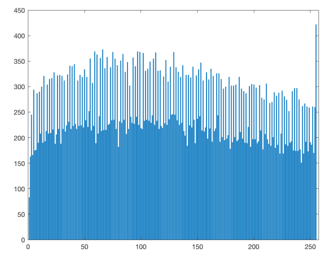

(a) $L, n = 3$

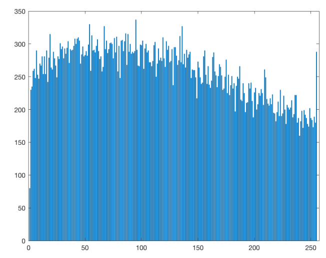

(b) $L, n = 5$

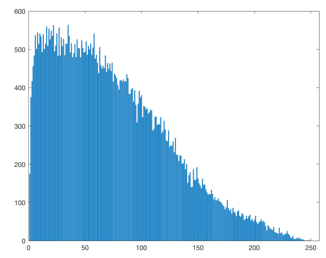

(c) $L, n = 7$


(d) $L, n = 9$


(e) $L, n = 15$


(f) $L, n = 25$


(g) $L, n = 45$


(h) $L, n = 255$

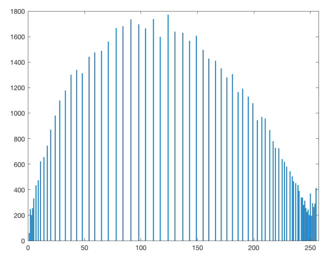

(i) H

Figure 6: Histograms of H and L with different window sizes n

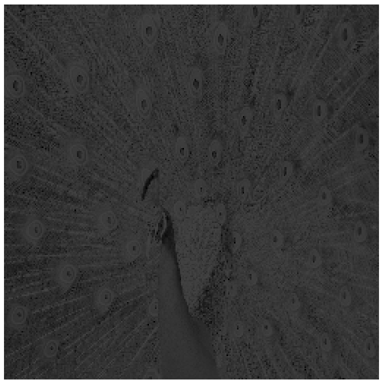
(f) Perform the log transform, inverse log transform and power-law transform to enhance image D. Please adjust the parameters to obtain the results as best as you can. Show the parameters, resultant images and corresponding histograms. Provide some discussions on the results as well.

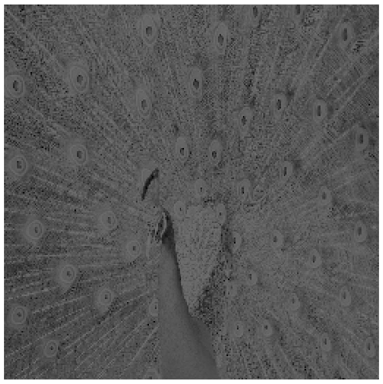
Log Transformation:

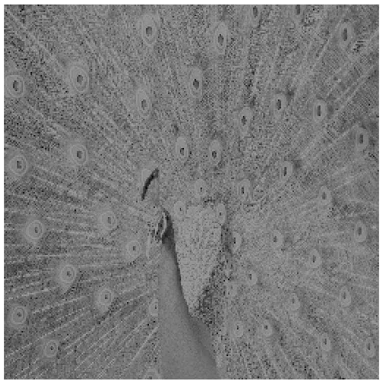
$$LT = c \cdot \log(I + 1).$$

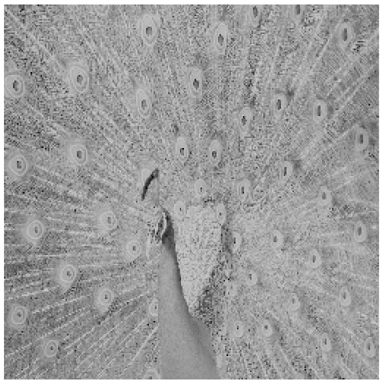
When $c = (L-1)/\log L \approx 46$ $(L = 256)$, the outcome is better.

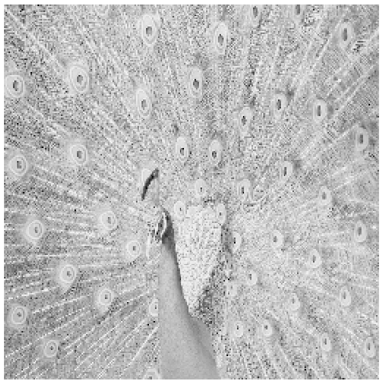
(a) $c = 5$

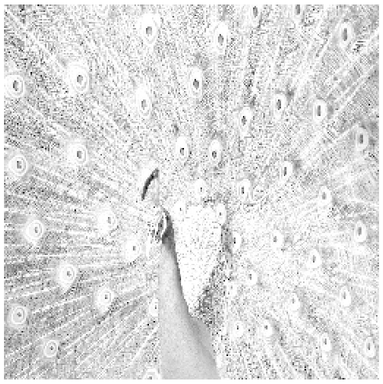

(b) $c = 15$

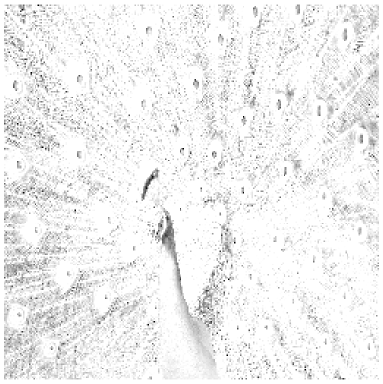

(c) $c = 25$


(d) $c = 35$


(e) $c = 45$


(f) $c = 55$


(g) $c = 65$


(h) $c = 75$

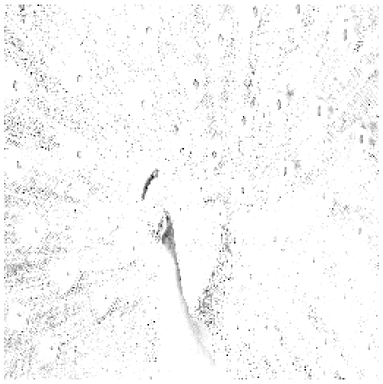
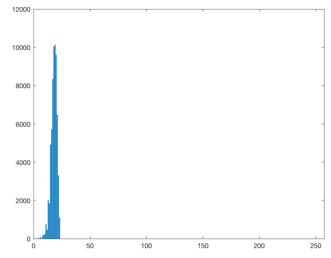
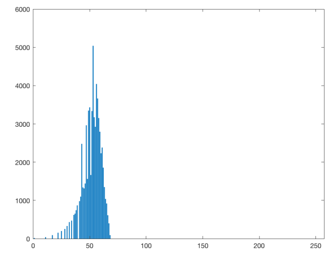
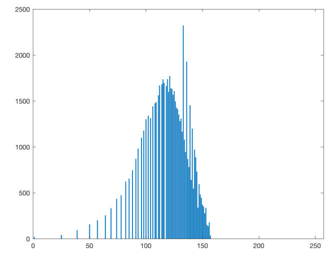

(i) $c = 85$

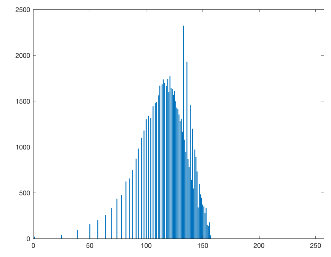
Figure 7: Log Transformation with different parameters c

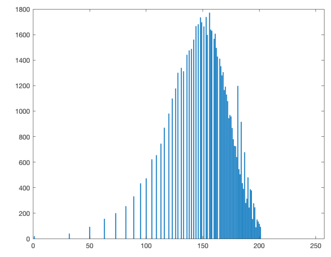
The histogram of log transformation also becomes smoother when c is near $(L-1)/\log L$ $(L = 256)$, i.e., c is between 45 and 55, so I think $c \approx 46$ is the best parameter.

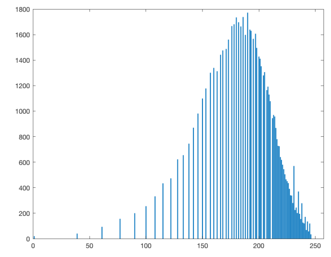

(a) $c = 5$

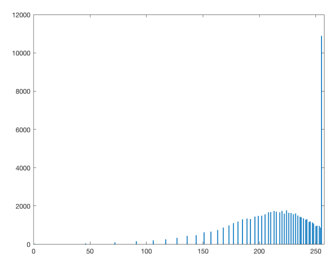

(b) $c = 15$

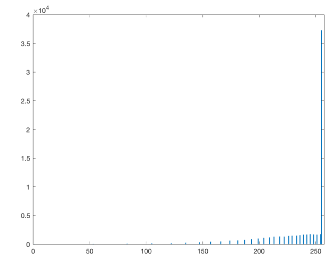

(c) $c = 25$


(d) $c = 35$


(e) $c = 45$


(f) $c = 55$


(g) $c = 65$


(h) $c = 75$

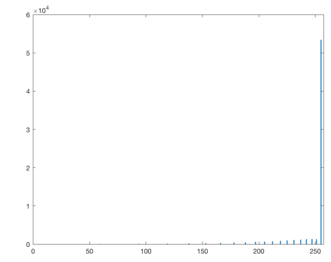
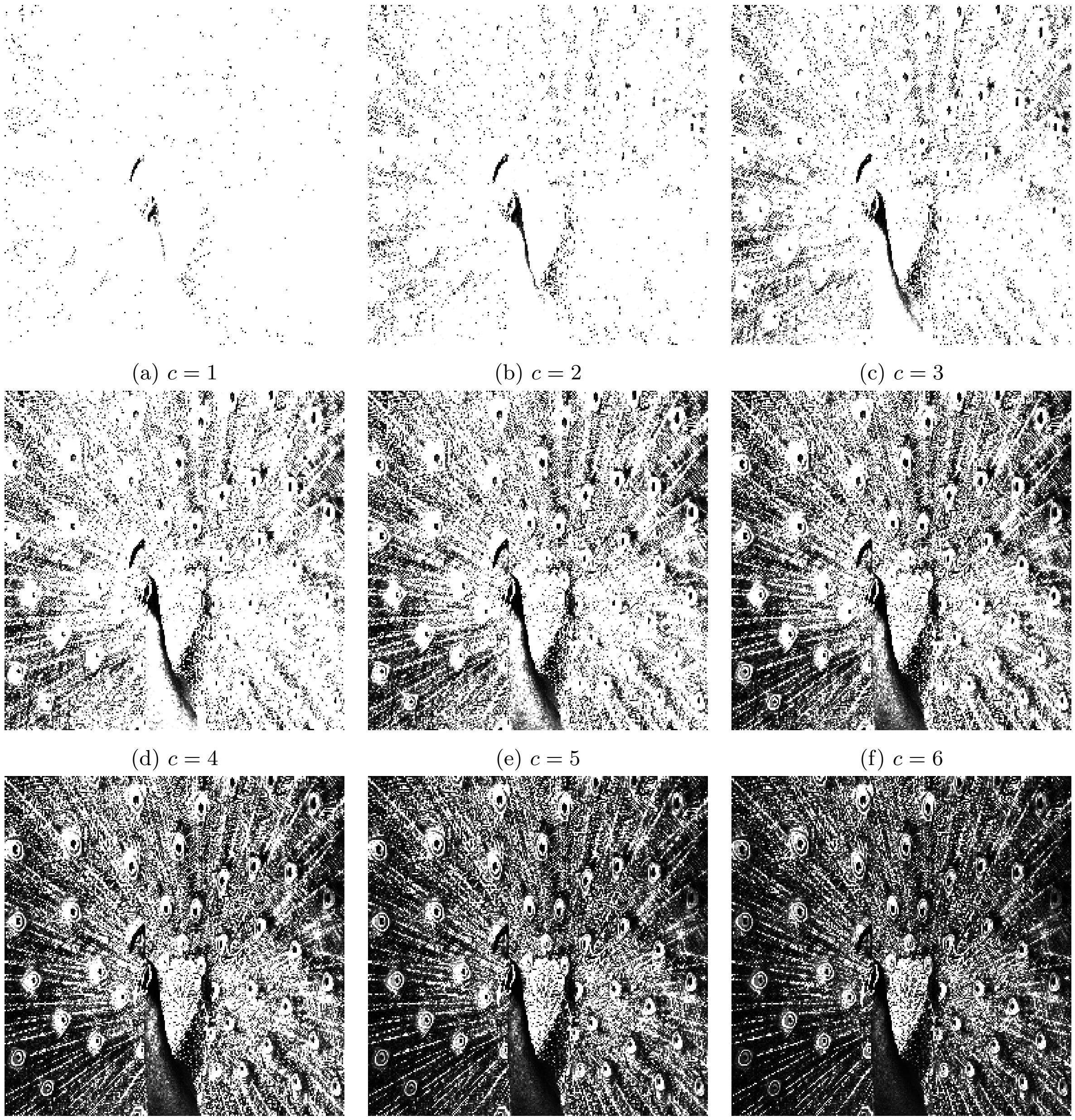

(i) $c = 85$

Figure 8: Histograms of Log Transformation with different parameters c

Inverse Log Transformation:

$$ILT = \left(e^{I}\right)^{(1/c)} - 1.$$



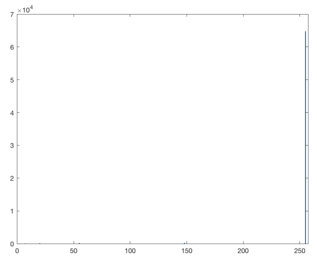
(a) $c = 1$ (b) $c = 2$ (c) $c = 3$

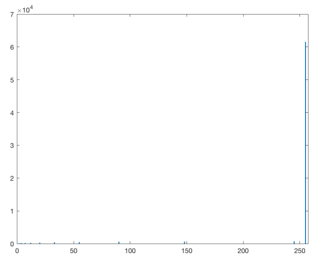
(d) $c = 4$ (e) $c = 5$ (f) $c = 6$

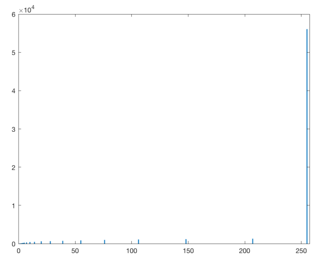
(g) $c = 7$ (h) $c = 8$ (i) $c = 9$

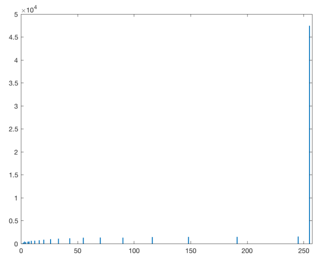
Figure 9: Inverse Log Transformation with different parameters c

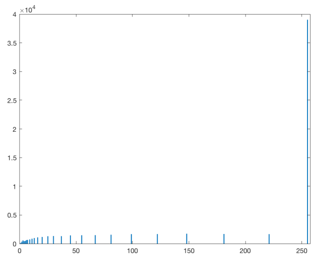
The histogram of inverse log transformation also becomes smoother when c is between 6 and 7, so I think $c \approx 6.5$ is the best parameter.

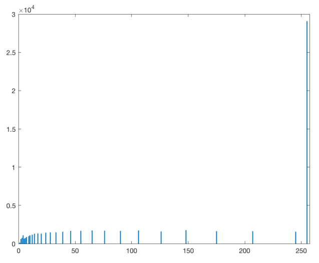

(a) $c = 1$

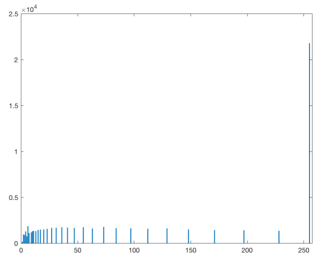

(b) $c = 2$

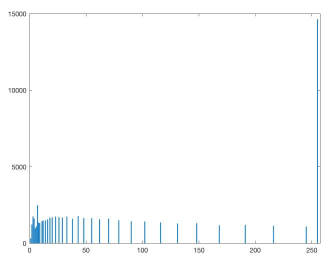

(c) $c = 3$


(d) $c = 4$


(e) $c = 5$


(f) $c = 6$


(g) $c = 7$


(h) $c = 8$

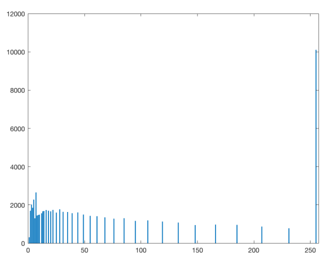

(i) $c = 9$

Figure 10: Histograms of Inverse Log Transformation with different parameters c

Power-Law Transformation

$$PL = I^p$$

When $p < 1$, whiteness becomes detailed.
When $p \geq 1$, darkness becomes detailed.

It seems that when $p \approx 1.5$, the result is best. Other constant p will lead to terrible outcomes.

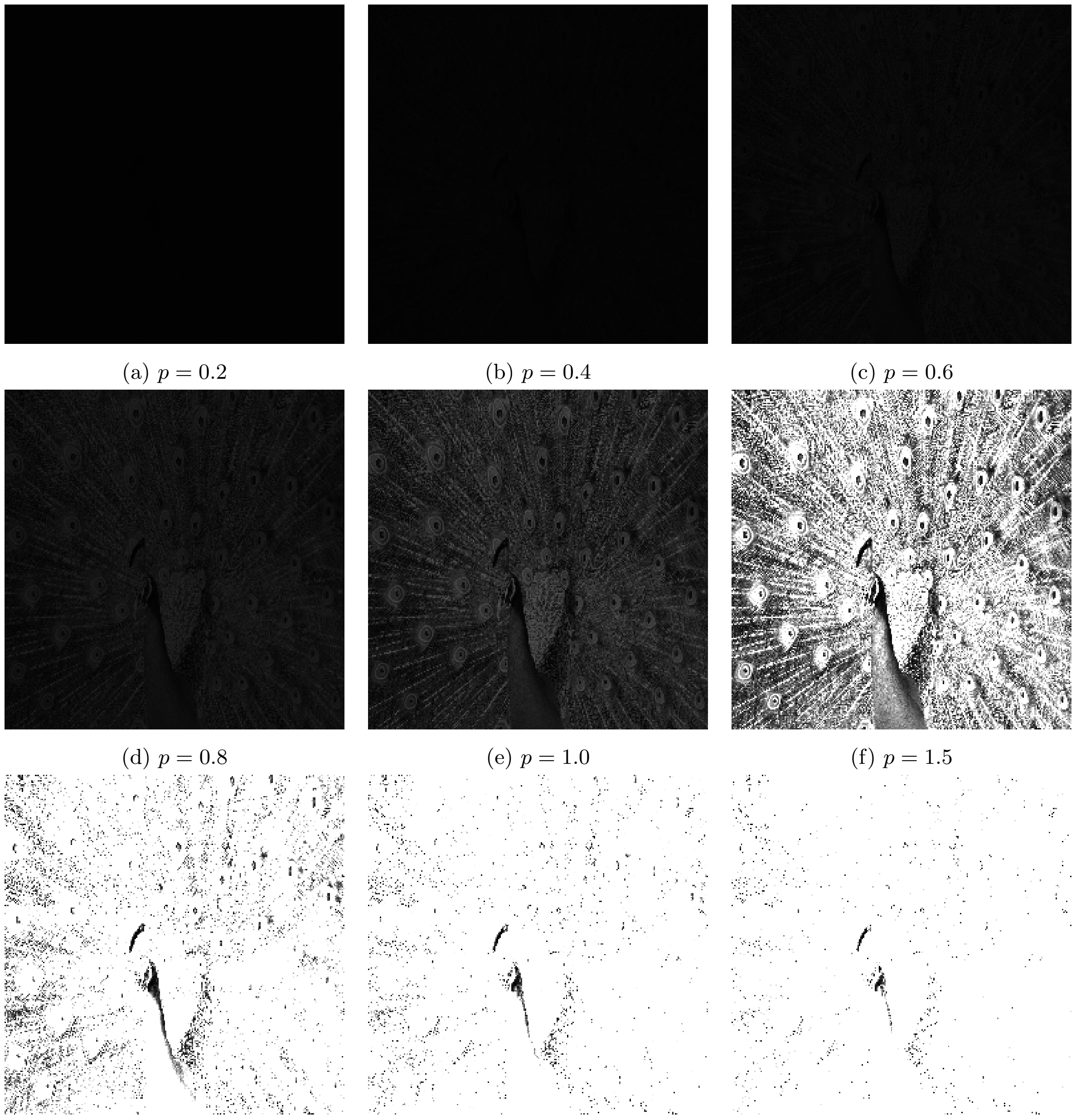
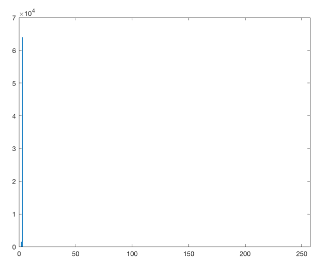
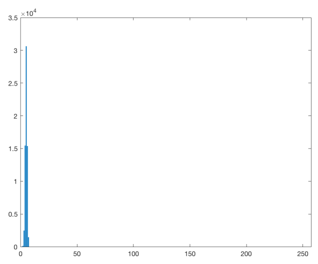
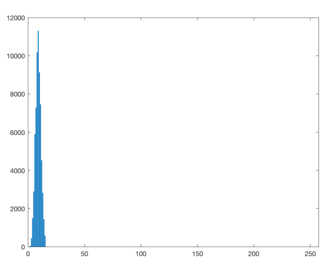

(a) $p = 0.2$ (b) $p = 0.4$ (c) $p = 0.6$
(d) $p = 0.8$ (e) $p = 1.0$ (f) $p = 1.5$
(g) $p = 2.0$ (h) $p = 2.5$ (i) $p = 3.0$

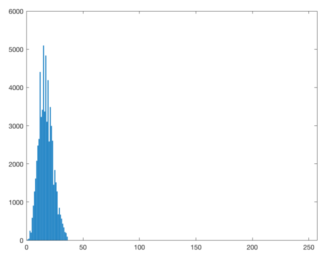
Figure 11: Power-Law Transformation with different parameters p

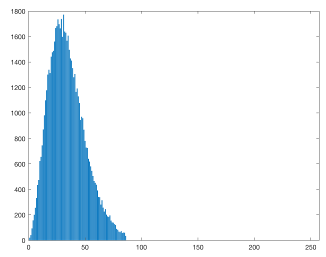
Apparently, when $p = 1.5$, the histogram is smoother than others.

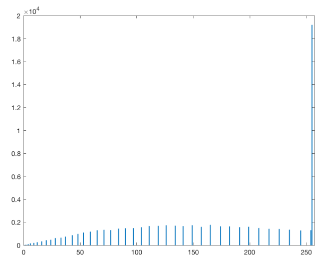

(a) $p = 0.2$

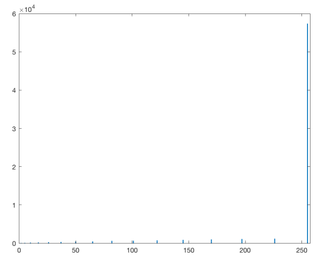

(b) $p = 0.4$

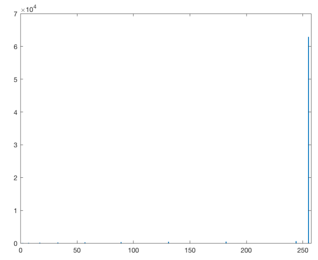

(c) $p = 0.6$


(d) $p = 0.8$


(e) $p = 1.0$


(f) $p = 1.5$


(g) $p = 2.0$


(h) $p = 2.5$

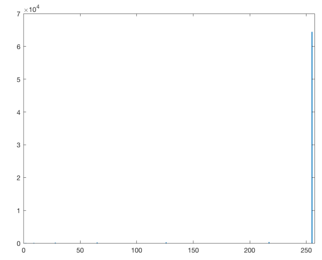

(i) $p = 3.0$

Figure 12: Histograms of Power-Law Transformation with different parameters p

PROBLEM 2: NOISE REMOVAL

(I)

Given an image I_3 as shown in Fig. 3(a), please follow the instructions below to create some new images.

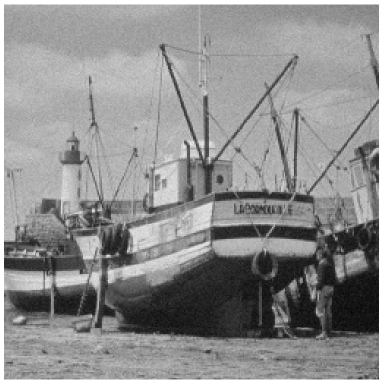
(a) Please generate two noisy images G_1, and G_2 by adding Gaussian noise to I_3 with different parameters. What's the main difference between these two images?

We use

$$G = I + \text{randn}(\text{size}(I)) \cdot \sigma + \mu$$

to generate Gaussian noise. Here we set $\mu = 0$ because it's just a constant value.

With higher σ, the picture becomes more noisy then the lower one.


(a) G_1 with $\sigma = 3$

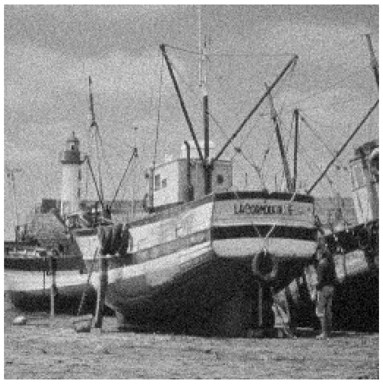

(b) G_2 with $\sigma = 10$

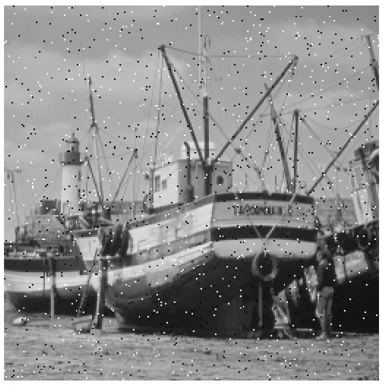
Figure 13: Adding Gaussian noise with different parameters

(b) Please generate two noisy images S_1, and S_2 by adding salt-and-pepper noise to I_3 with different parameters. What's the main difference between these two images?

We use

$$S(i,j) = \begin{cases} 0, & \text{if rand}(0,1) < \text{threshold} \\ 255, & \text{if rand}(0,1) > 1 - \text{threshold} \\ I(i,j), & \text{otherwise.} \end{cases}$$

to generate salt-and-pepper noise, thus higher threshold leads to noisier picture.


(a) S_1 with threshold $= 0.01$

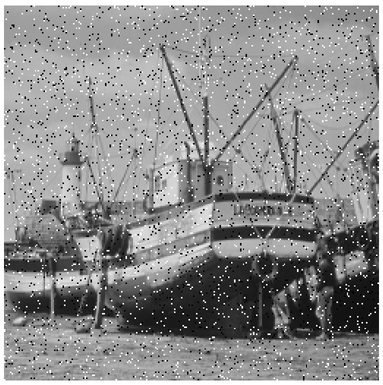

(b) S_2 with threshold $= 0.03$

Figure 14: Adding salt-and-pepper noise with different threshold

(c) Design proper filters to remove noise from G_1 and S_1, and denote the resultant images as R_G and R_S, 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.

Low-Pass Filter

First, we need to pad the image G_1 with replicates on four sides and denotes it as P to deal with the potential out-of-index problem.

```
    P = padarray(I, [1 1], 'replicate', 'both');    % I = G1
```

We use a 3×3 **mask** to perform low-pass filtering. The general form is

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

```
    mask = 1 / (b + 2)^2 * [1, b, 1; b, b^2, b; 1, b, 1];
```

Then, we iterably perform low-pass filtering on each entry. For each entry, we assign **mat** equals to a 3×3 block in order to do Hadamard product with the **mask**.

```
    mat = P(i − 1: i + 1, j − 1: j + 1);
```

Finally, that corresponding $RG(i, j)$ equals to the summation of that 3×3 block after doing Hadamard product.

```
    RG(i − pad, j − pad) = sum(sum(mat .* mask));
```

MATLAB Implementation

```
function RG = lowPass(I, b)
    I = double(I);
    [h, w] = size(I);

    % Initialize resultant image and padding image
    RG = zeros(h, w);                               % initialize the resultant images
    P = padarray(I, [1 1], 'replicate', 'both');    % pad around the image I
    pad = 1;                                        % assume mask size = 3 x 3

    % 3 x 3 mask
    mask = 1 / (b + 2)^2 * [1, b, 1; b, b^2, b; 1, b, 1];

    % Perform low-pass filtering
    for i = 1 + pad: h + pad
        for j = 1 + pad: w + pad
            mat = P(i - pad: i + pad, j - pad: j + pad);
            RG(i - pad, j - pad) = sum(sum(mat .* mask));
        end
    end

    RG = uint8(RG);
end
```

When b is small ($b \leq 5$), the resultant image is terrible, but we can see that when b becomes larger ($b \geq 10$), the resultant image becomes better.

I chose the 3×3 low-pass filter because the Gaussian noise is uniformly distributed noise and a low-pass filter can passes signals with a frequency lower than a certain threshold frequency and eliminate high frequencie.

The parameter b is chosen by experiment.

(a) I_3

(b) G_1

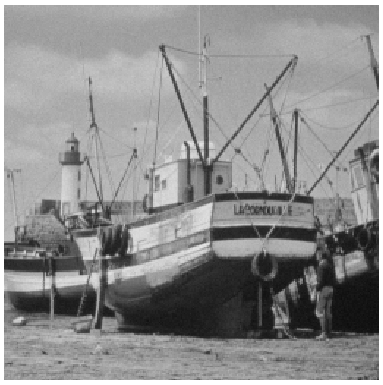
(c) $b = 1$, PSNR $= 41.4883$

(d) $b = 2$, PSNR $= 33.2315$

(e) $b = 5$, PSNR $= 40.1584$

(f) $b = 10$, PSNR $= 38.7482$

(g) $b = 15$, PSNR $= 39.6454$


(h) $b = 20$, PSNR $= 40.0507$

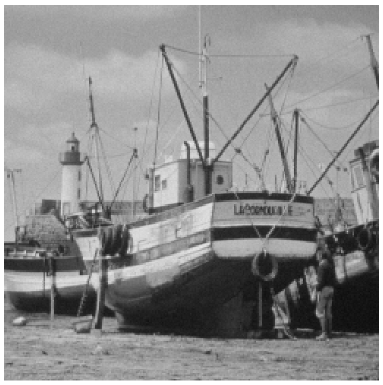

(i) $b = 30$, 40.0775

Figure 15: 3×3 Low Pass Filter with different parameters b

Median Filter

(i) Square Mask: an $n \times n$ mask

(ii) Cross Mask: an $n + n - 1$ cross mask

First, we still need to pad the image S_1 with replicates on four sides and denotes it as P.

```
% square mask
pad = (win - 1) / 2;                              % win: window size of mask
P = padarray(I, [pad pad], 'replicate', 'both');
% cross mask
```

```
    pad = (c − 1) / 2;                                  % c: cross size of mask
    P = padarray(I, [pad pad], 'replicate', 'both');
```

Both masks first append points to the **mat** array.

```
    % square mask
    mat = P(i − pad: i + pad, j − pad: j + pad);
    mat = reshape(mat, [1, win ∗ win]);
    % cross mask
    mat = [P(i − pad: i + pad, j)', P(i, j − pad: j − 1), P(i, j + 1: j + pad)];
    mat = reshape(mat, [1, 2 ∗ c − 1]);
```

Finally, let the corresponding $RS(i, j)$ equals to the median of **mat**.

```
    RS(i − pad, j − pad) = median(mat);
```

MATLAB Implementation

```
function RS = medianFilter(I, win, c, m)
    [h, w] = size(I);
    RS = zeros(h, w);       % initialize the removed salt image

    switch m
        case 1
            % square mask
            pad = (win − 1) / 2;                                % win: window size of mask
            P = padarray(I, [pad pad], 'replicate', 'both');
        case 2
            % cross mask
            pad = (c − 1) / 2;                                  % c: cross size of mask
            P = padarray(I, [pad pad], 'replicate', 'both');
    end

    % For each pixel in P
    for i = 1 + pad: h + pad
        for j = 1 + pad: w + pad
            switch m
                case 1
                    % square mask
                    mat = P(i − pad: i + pad, j − pad: j + pad);
                    mat = reshape(mat, [1, win ∗ win]);
                case 2
                    % cross mask
                    mat = [P(i − pad: i + pad, j)', P(i, j − pad: j − 1), P(i, j + 1: j
                         + pad)];
                    mat = reshape(mat, [1, 2 ∗ c − 1]);
            end
            RS(i − pad, j − pad) = median(mat);
        end
    end
    RS = uint8(RS);
end
```

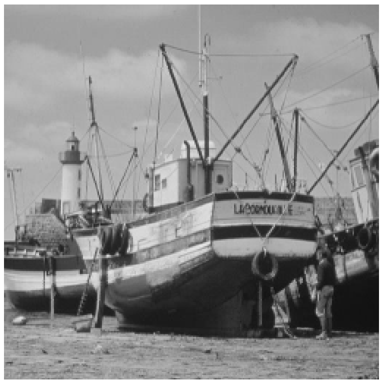
I chose median filter since it has following advantages:

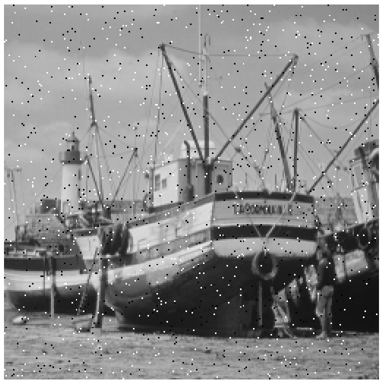
- Preserve sharp edges
- Effective in removing impulse noise
- 1D/2D (directional)

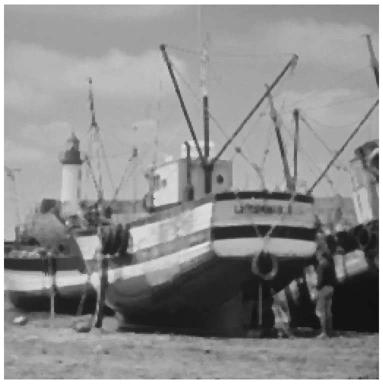
The main idea of the median filter is to run through the signal(image) entry by entry, replacing each entry with the median of neighboring entries. The pattern of neighbors is called the "window", which slides, entry by entry, over the entire image signal(image).

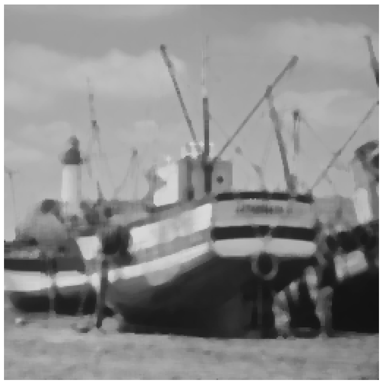
Salt-and-pepper noise is belong to impluse noise, so median filter can work well in this case.

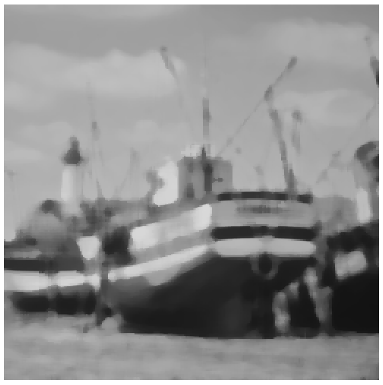
Apparently, when $n = 3$, the effect outstand other parameters. And when n becomes larger, the resultant image becomes more blurred. I think it is due to that if the window size is too large, the median of that window would not be as accurate as you think.

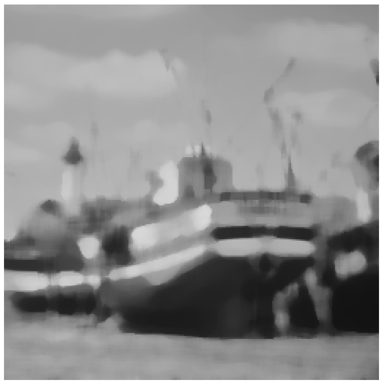

(a) I_3

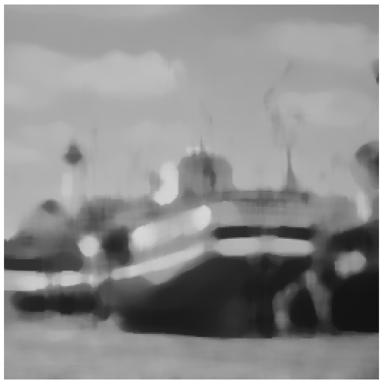

(b) S_1

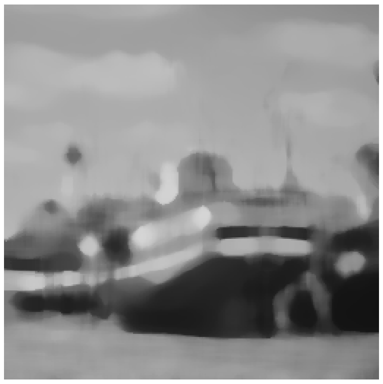

(c) $n = 3$, PSNR = 9.2300


(d) $n = 5$, PSNR = 1.6150


(e) $n = 7$, PSNR = -1.4873


(f) $n = 9$, PSNR = -1.7027


(g) $n = 11$, PSNR = -3.3377


(h) $n = 13$, PSNR = -3.5759

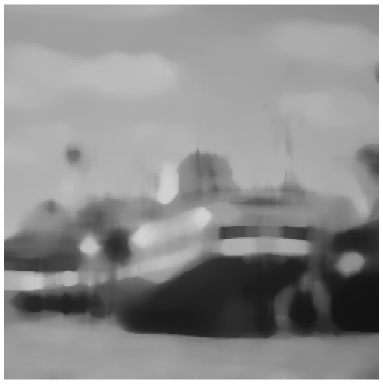
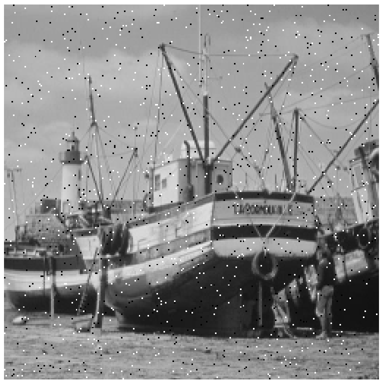

(i) $n = 15$, PSNR = -3.9865

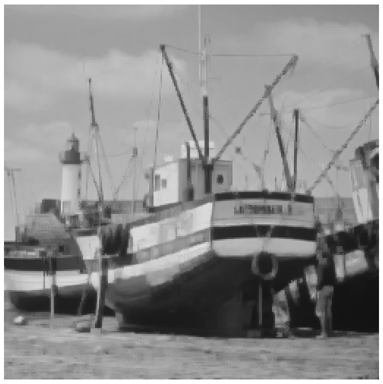
Figure 16: $n \times n$ Square Median Filter with different window sizes n

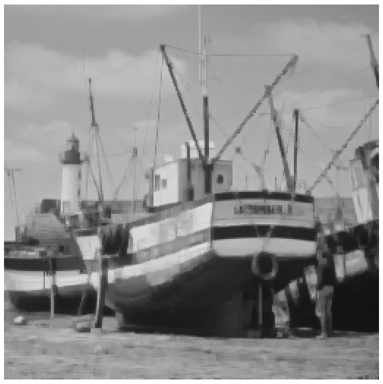
On the other hand, the cross size of the mask does not influence the mask seriously.

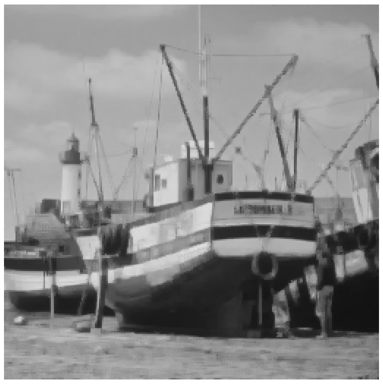
I think this is because, $O(2n) = O(n^2)$

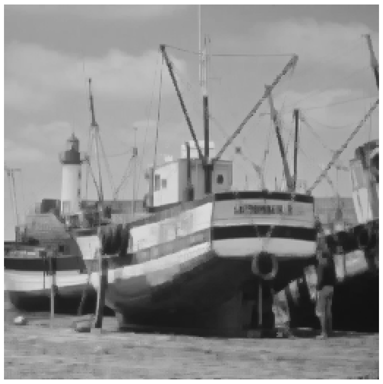
(a) I_3

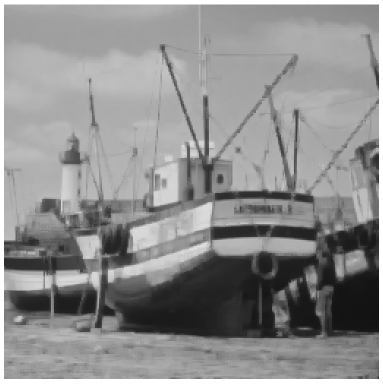

(b) S_1

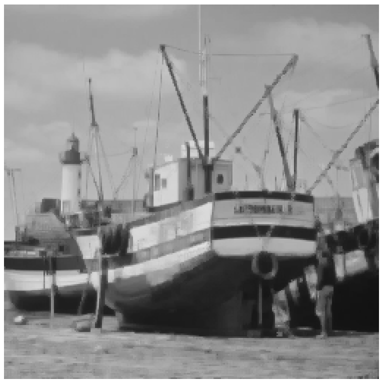

(c) $n = 3$, PSNR = 7.2121


(d) $n = 5$, PSNR = 7.2121


(e) $n = 7$, PSNR = 7.2121


(f) $n = 9$, PSNR = 7.2121


(g) $n = 11$, PSNR = 7.2121


(h) $n = 13$, PSNR = 7.2121

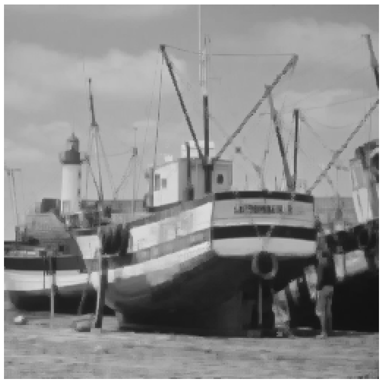

(i) $n = 15$, PSNR = 7.2121

Figure 17: $n + n - 1$ Cross Median Filter with different cross sizes n

(d) Compute the PSNR values of R_G and R_S and provide some discussions.

We've compute the PSNR value of R_G and R_S with different value, and now we plot the PSNR value graph of Low-Pass Filter, Square Median Filter and Cross Median Filter with different parameters.

- In Low-Pass Filter, I found that when $b = 2$, the PSNR value is lower, but all the PSNR values > 30(dB), overall they are good.
- In Square Median Filter, I found that when the n (window size) becomes larger, the PSRN value becomes smaller, and all the PSNR values < 10(dB), overall they are bad.
- In Cross Median Filter, no matter what value n (cross size) is, the PSNR value remains the same. I think this is because that the points of a cross mask is less than the points of a square mask.

The PSNR values of R_G is higher (better) than of R_S. I think this is due to the fact that Gaussian noise is uniform and salt-and-pepper noise is impulsive, so when we do some image processing to the noise pictures, the quality of the Gaussian noise one will become better than the one of salt-and-pepper.

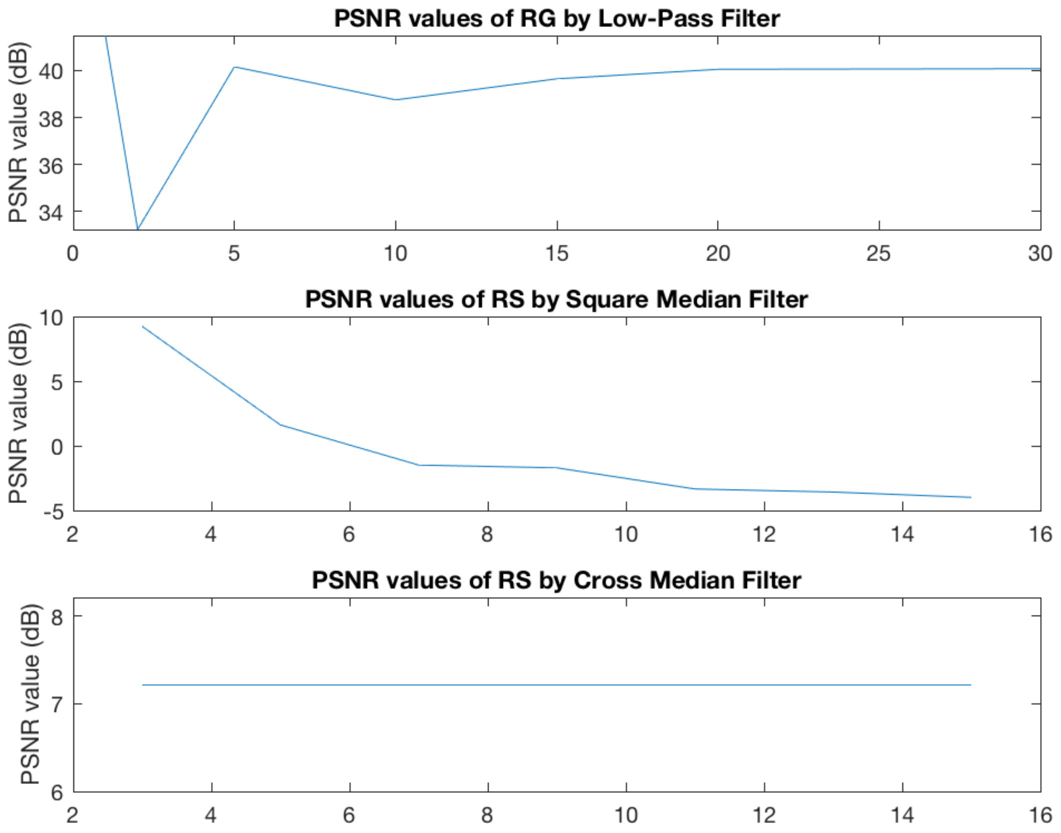


Figure 18: PSNR values of R_G and R_S

(II)

Design your own method to remove the wrinkles on the face of a given image I_4 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.
Assign $face = I$ first.
Convert the I to binary mask.

```
face = I;
binary = I(:, :, 1) < 128;          % Convert image to binary mask
```

Define a disk element, then chop half thickness from that mask and dilate the erode mask.

```
de = strel('disk', 3, 0);           % Define a disk element
halfThick = imerode(binary, de);    % Chop half thickness from mask
dilate = imdilate(halfThick, de);   % Dilate the eroded mask
```

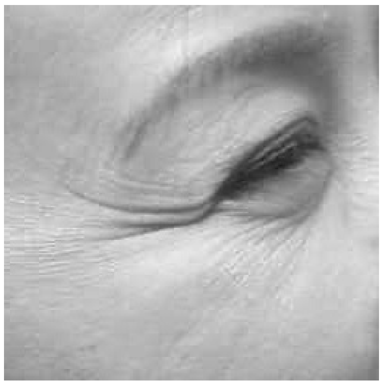
Fro each entry, if the $invalidArea(i, j) = 1$ and $j \leq 140$ (detect the wrinkles by human eyes), we multiply the $I(i, j) + threshold$ by 1.5 to get a *whiter* entry.

```
threshold = 3;
for i = 1: 256
    for j = 1: 256
        if (invalidArea(i, j) == 1) && (j <= 140)
            face(i, j) = 1.5 * (I(i, j) + threshold);
        end
    end
end
```

Finally, we run the cross **medianFilter** with cross size $= 15$ for five times.

```
for i = 1: 5
    face = medianFilter(face, 1, 15, 2);
end
```

The removed wrinkle I_4 is not that pretty, since if I want to remove the wrinkle pefectly, other details will be sacrificed. I think this is because that the median filter will still compromise with some detailed entry.


(a) I_4

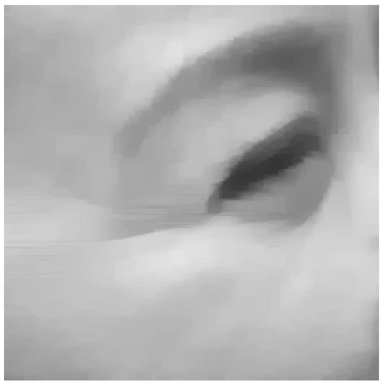

(b) removed wrinkle I_4

Figure 19: Face wrinkle removing