Homework #1

B05902120 / Yu-Ting, TSENG

Mar 7, 2019

Basic Execution

- Platform: Unix (MacBook)
- Language: C++

Warm-Up: Simple Manipulations

(a) Please perform horizontal flipping on image I_1 as shown in Fig.1 and output the result as B.

The concept of flipping is that we have to swap the pixel value in the correspond position. For instance, we supposed that the size of the image is symbolized by `size` and that `image[i][j]` represent the pixel value of the image at position (i, j). Then we need to swap `image[i][j]` and `image[i][256 - j - 1]`.

```
for (int i = 0; i < size; i ++)
    for (int j = 0; j < size / 2; j ++)
        swap(img[i][j], img[i][size - j - 1]);
```

As a result, we can get the correct images which was displayed as follows.

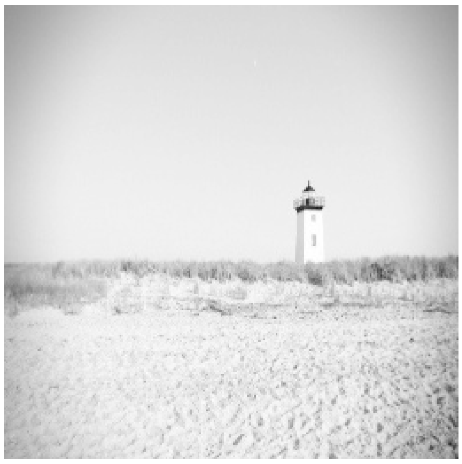
Figure 1: sample1.png

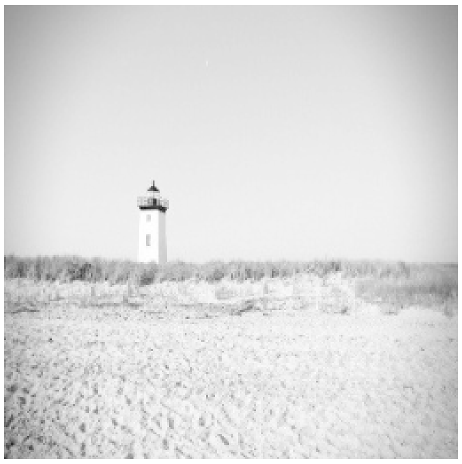
Figure 2: B.png

(b) Please perform a power-law transform to enhance B and adjust the parameters to obtain the results as best as you can. Show the resultant images with corresponding parameters and provide some discussions on the results as well.
A useful function is shown below. We enhance the pixel value by power.

$$G(i,j) = [F(i,j)]^p, \text{ where } 0 \leq F(i,j) \leq 1$$

p is the parameter which I can adjust by myself. Therefore, a section of my code is created as:

```
scanf("%d", &n);     // if n = 4 the result is best
for (int i = 0; i < size; i ++)
    for (int j = 0; j < size; j ++)
       img[i][j] = pow(img[i][j], n) * pow(256, -n + 1);
```

After several times of trial, I find that the image is getting darker and darker to some degree. At first, it is helpful because it enhance the contrast of the image. However, as n increases, it became so dark that some details are hidden. I discover that the best result appear when n is 4. The images below is the result T get when n is 4 and 10 respectively.

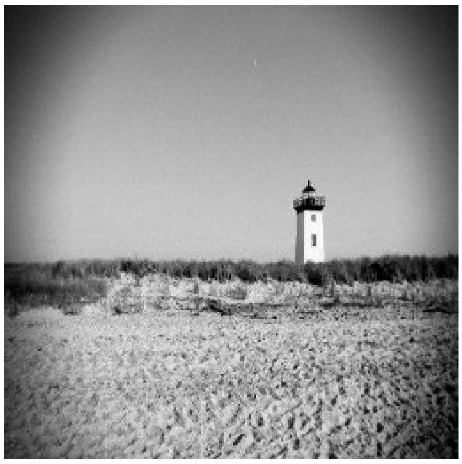
Figure 3: C_4.png

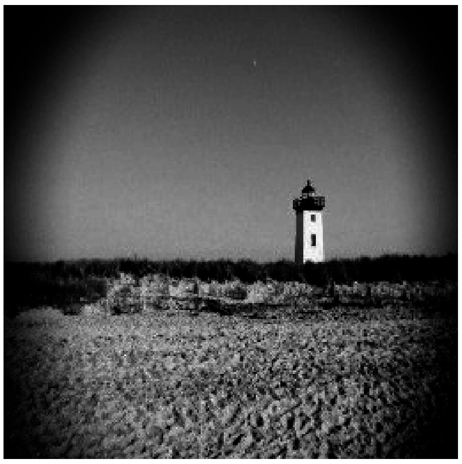
Figure 4: C_10.png

The phenomenon I have found is reasonable. As we have already known, the value would be decrease. The pixel value which is large originally declines much slower than the one which is small. This is the reason why the contrast of the image is being enhanced. On the other hand, with the power getting larger, the pixel value become extremely closed to 0 which is black, leading to the darker image.

Problem1: Image Enhancement

Given a gray-level image I_2 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 2 and output the image as D.

What we have done is simply divide the pixel value by 2 then output the result. For example, we supposed that `image[i][j]` represent the pixel value of the image at position (i, j). Then we need to divide it by 2 and get `image[i][j] / 2`.

```
for (int i = 0; i < size; i ++)
    for (int j = 0; j < size; j ++)
        img[i][j] = img[i][j] / 2;
```

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

The concept is the same. We just change the divisor to 3.

```
for (int i = 0; i < size; i ++)
    for (int j = 0; j < size; j ++)
        img[i][j] = img[i][j] / 3;
```

The pixel value is decreasing and decreasing, hence, we might be able to speculate that the resultant images would be darker and darker. At last, we can get the results of the requirements above as follows.

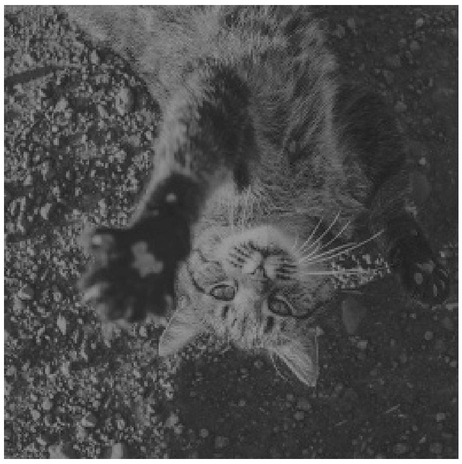
Figure 5: D.png

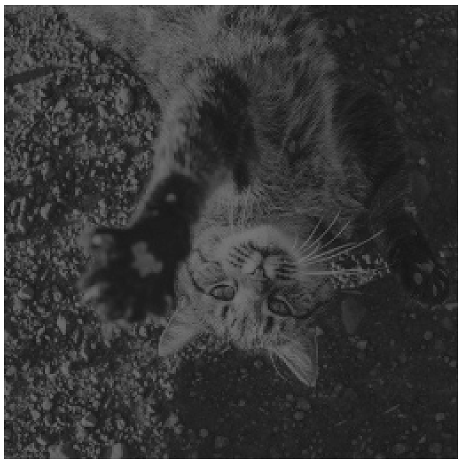
Figure 6: E.png

(c) Plot the histograms of I_2, D and E. What can you observe from these three histograms?

The histograms are what have been displayed. We compute the appearance of each pixel value by code and output as a `.csv` file subsequently. Last, we plot the histograms via the function of excel and get the result.

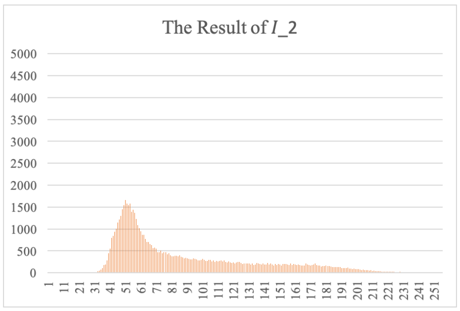

Figure 7: Histogram of I_2

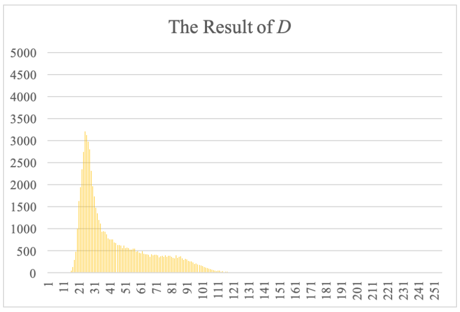

Figure 8: Histogram of D

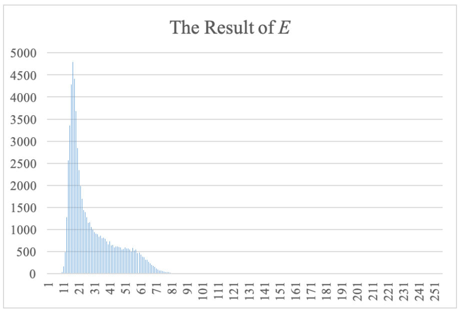

Figure 9: Histogram of E

We might observe the average of the value is being smaller and smaller when the divisor is bigger. It make sense since all values are being divided. Besides, it becomes more concentrated. For instance, there are three numbers, 4, 5, and 6. If we divide them by 2 and take ceiling, we would get 2, 2 and 3. This example illustrates the reason why the plots of the histogram are presented in that way.

(d) Perform global histogram equalization on D and E and output the results as H_D and H_E respectively. Please plot the histograms and provide some discussions on the results.

The concept of histogram equalization is to enhance the given image G by converting the histogram of it into cumulative distribution function (CDF). The next step is to map the CDF to a uniform distribution in order to make the image more uniformly distributed. The process work as follows.

$$p(r_k) = n_k/MN \ (k = 0, 1, ...255)$$
$$s_k = 256 * \textstyle\sum_{i=0}^{k} p(r_i), \text{ where}$$

r_k is the original pixel value,

s_k is the updated pixel value,

n_k is the number of appearance of the original pixel value r_k,

M, N are the width and height of the image respectively.

And without doubt, we should implement it by a section of C++ code.

```
for (int i = 0; i < size; i ++)
    for (int j = 0; j < size; j ++)
        pix[k][img[i][j] - 1] += 1;

for (int n = 1; n < 256; n ++)
    pix[k][n] += pix[k][n - 1];

for (int i = 0; i < size; i ++)
    for (int j = 0; j < size; j ++)
        img[i][j] = pix[k][img[i][j]] / 256;
```

Via the same way already mentioned, we may get the resultant images and the histograms.

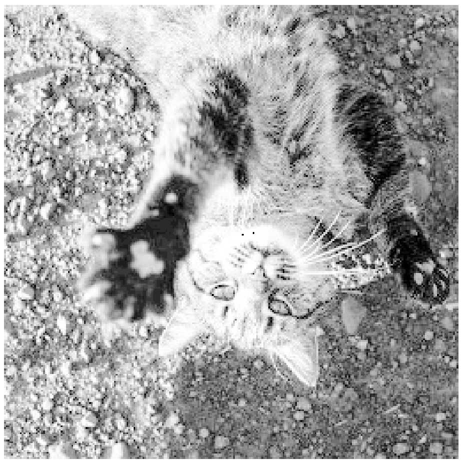

Figure 10: H_D

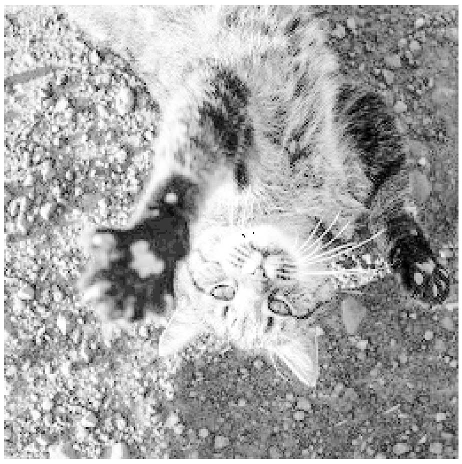

Figure 11: H_E

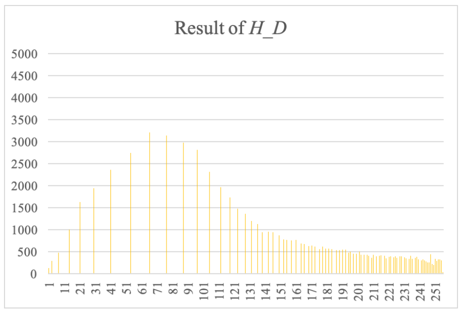

Figure 12: Histogram of H_D

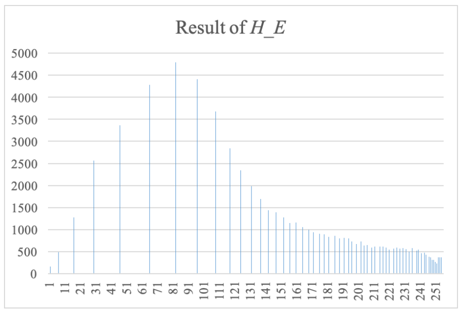

Figure 13: Histogram of H_E

Since the image becomes more uniformly distributed, it indicates that the gap between two pixel values are expanded, and further more represents that the contrasts of the images are enhanced. Observing the original images and the new resultant images, we can find the changes meets what we expected.

(e) Perform local histogram equalization on D and E and output the results as L_D and L_E respectively. Please plot the histograms and provide some discussions on the results.

The basic concepts of the global and the local histogram equalization are the same. Their difference between might be that the latter only consider the pixel in neighbor while the former consider the whole image. We use the window to go through the low contrast image and perform histogram equalization under the window's current position. To implement by the code (fix the window size by 11):

```
int win = 11, hlf = win / 2;
for (int i = 0; i < size; i ++)
    for (int j = 0; j < size; j ++){
        int sum = 0, rnk = 0;
        for (int m = i - hlf; m <= i + hlf; m ++)
            for (int n = j - hlf; n <= j + hlf; n ++){
                if (m < 0 || m > size - 1) continue;
                if (n < 0 || n > size - 1) continue;

                if (img[i][j] > img[m][n]) rnk ++;
                sum ++;
            }
        tmp[i][j] = rnk * 255 / sum;
    }
```

Still, we can get the result by the same approach. According to the information above, we might know local histogram equalization focuses more on each local grey-level variations than global histogram equalization does. Therefore, if we just consider a small area, we may find it more clear because of the contrast. Nevertheless, if we consider the whole picture, we seldom think as the above mentioned. The image is on the next page, including resultant images and the histograms.

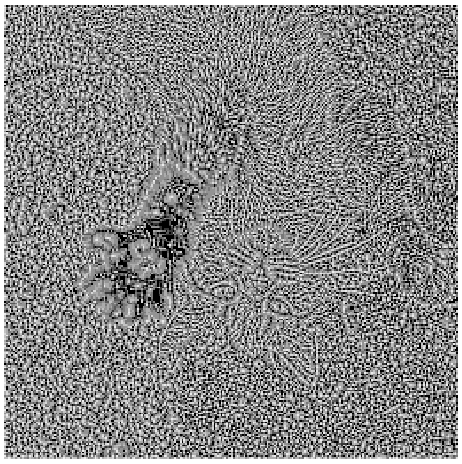
Figure 14: L_D

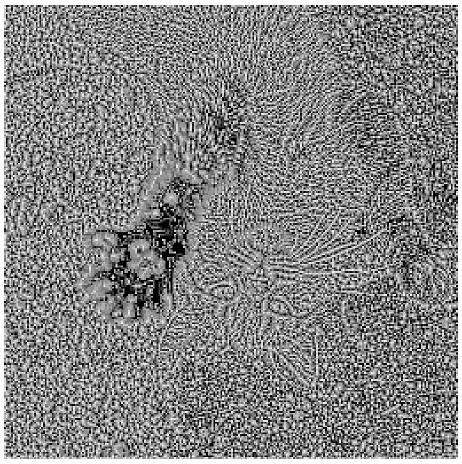
Figure 15: L_E

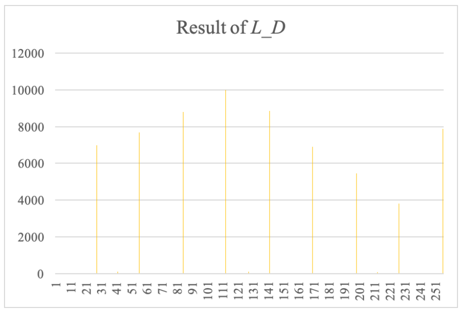

Figure 16: Histogram of L_D

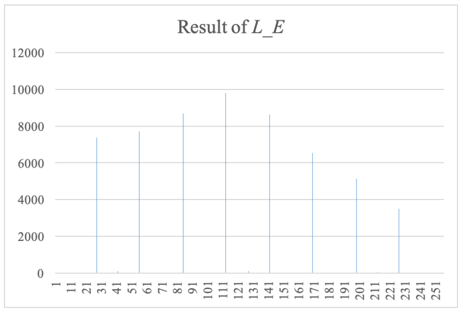

Figure 17: Histogram of L_E

(f) What is the main difference between local and global histogram equalization?

The principle of local and global histogram equalization has been discussed in (e) and (f) respectively. We can find it distribute more uniform, but in my opinion, it is caused by the large window size. The bigger the window size is chosen, the bigger the gap between two common intensity values is. Furthermore, we find that local histogram equalization does emphasize more on each local region since most of the pixels belong to several certain intensity values.

Problem2: Noise Removal

Given an original image as shown in Fig.3(a) and two images corrupted by noise as shown in Fig.3(b) and Fig.3(c), please follow the instructions below to create some new images.

(a) Design proper filters to remove noise from Fig. 3(b) and Fig. 3(c), and output the resultant images as N_1 and N_2 respectively. Please detail the steps of the denoising process and specify all the corresponding parameters. Provide some discussions about the reason why those filters and parameters are chosen.

For the first image, Fig.3(b) is contaminated by uniform noise. Therefore, low-pass filter is a wise choice. The most common filter looks as below:

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

The filter `flt` is the filter and works as follow:

```
int hlf = 1;
for (int b = 1; b < 21; b ++){
    int flt[3][3] = {{0}};
    flt[0][0] = flt[0][2] = flt[2][0] = flt[2][2] = 1;
    flt[0][1] = flt[1][0] = flt[1][2] = flt[2][1] = b;
    flt[1][1] = b * b;

for (int i = 0; i < size; i ++)
    for (int j = 0; j < size; j ++){
        int ara = 0, sum = 0;
        for (int m = i - hlf; m <= i + hlf; m ++)
            for (int n = j - hlf; n <= j + hlf; n ++){
                if (m < 0 || m > size - 1) continue;
                if (n < 0 || n > size - 1) continue;

                sum += img[m][n] * flt[m-i+1][n-j+1];
                ara += flt[m-i+1][n-j+1];
            }

        tmp[i][j] = sum / ara;
    }
}
```

So, what should we choose for b? We plot a graph of PSNR versus different value of b, knowing that the best result (PSNR = 28.662193) happens at $b = 2$.

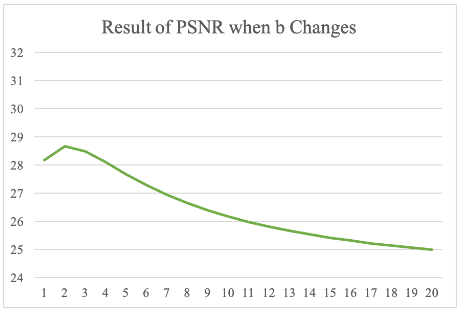

Figure 18: PSNR vs b

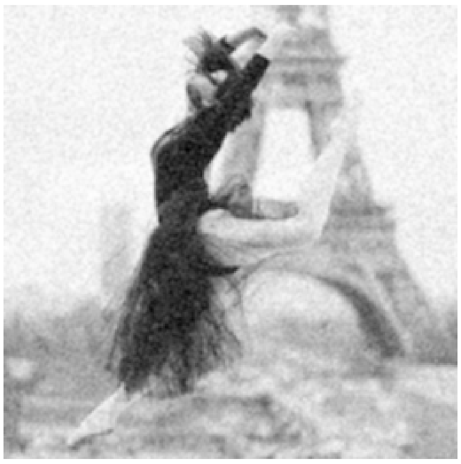

Figure 19: N_1.png

In my opinion, the improvement is apparently not enough. Nevertheless, comparing to other resultant images, it is the one performing the best.

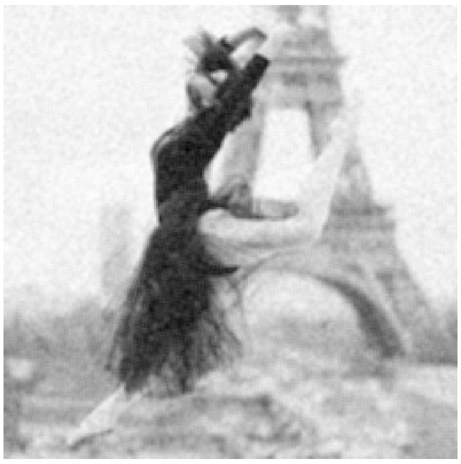

Figure 20: $b = 1$

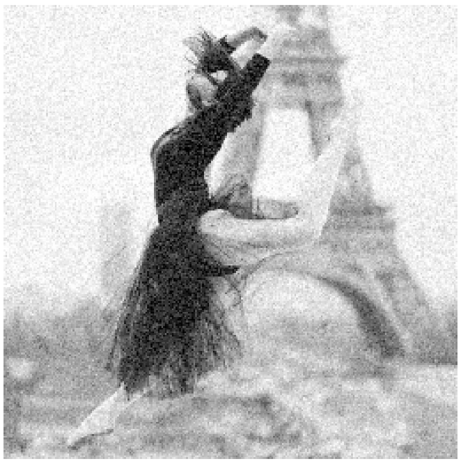

Figure 21: $b = 20$

For the second image, Fig.3(c) is interfered by salt and pepper noise, which is a type of impulse noise. There are two common ways to improve the situation, including Outlier Detection and Median Filter. I have tried these two methods. To begin, I use the Median Filter, we can find that the pixels looking granular are modified, but the outline of the woman become more blurry. On the contrary, if we edit the image via Outlier Detection, we may be able to have a image with clear outline. However, there comes the problem. The granule part still appear in the image. Therefore, I further denoise the image through both method by two distinct order respectively.

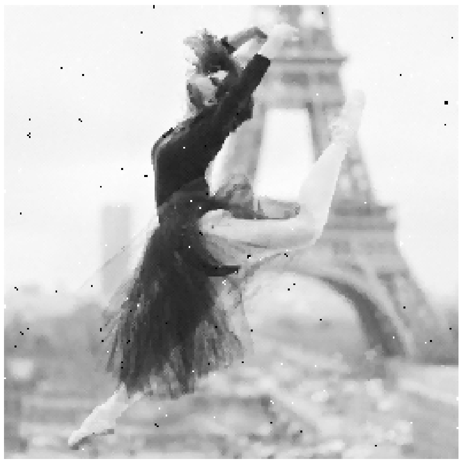
Figure 22: Only Median Filter

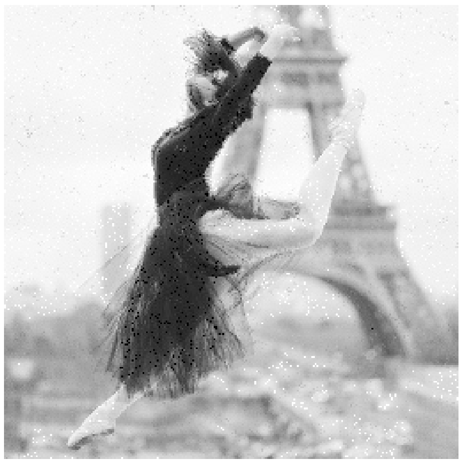
Figure 23: Only Outlier Detection

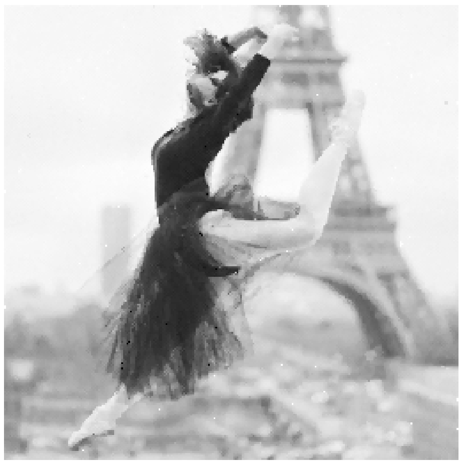
Figure 24: Median before Outlier

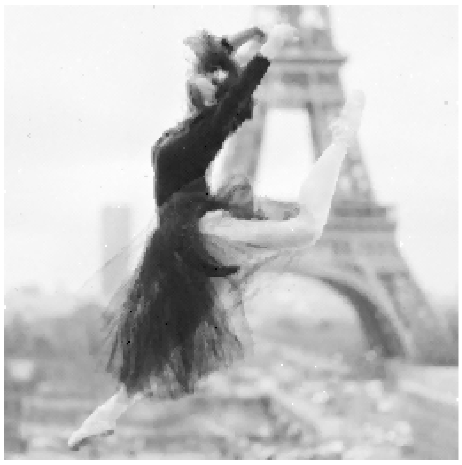
Figure 25: Outlier before Median

It is obviously that Fig.20 does perform well. Besides using my eyes to observe the changing, we may compute the PSNR values to help analyze the results. According the Greedy Algorithm, we find Median Filter (PSNR = 28.374094) does better than Outlier Detection (PSNR = 27.881961) does. The next step is go through the Outlier Detection. We record the different resultant PSNR for variety of ε value chosen.

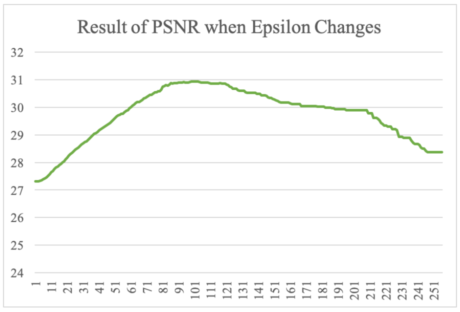

Figure 26: PSNR vs ε

PSNR becomes the largest (PSNR = 30.940705) when ε is 101. We then try to pass the image through Median Filter again subsequently and get the result (PSNR =

31.602384) best ever. Surprisingly, if we do the same thing by the methods totally opposite, the results would not become better and even become much more worse. The image below is the eventual result.

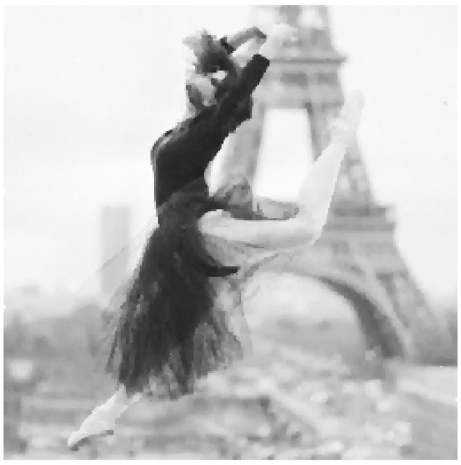
Figure 27: N_2.png

Something worth mention is that some improvement is made to adapt to the image. For instance, the considered adjacent pixels are only up, down, right and left four directions rather than three-by-three grid.

```
int hlf = 1;
for (int i = 0; i < size; i ++)
    for (int j = 0; j < size; j ++){
        int ara = 0;
        vector <int> sum;
        for (int m = i - hlf; m <= i + hlf; m ++)
            for (int n = j - hlf; n <= j + hlf; n ++){
                if (m < 0 || m > size - 1) continue;
                if (n < 0 || n > size - 1) continue;
                if (m == i - hlf && n == j - hlf ||
                   (m == i - hlf && n == j + hlf ||
                   (m == i + hlf && n == j - hlf ||
                   (m == i + hlf && n == j + hlf)
                     continue;

                sum.push_back(img[m][n]);
                ara ++;
            }
        sort(sum.begin(), sum.end());
        tmp[i][j] = sum[ara / 2];
    }
```

Moreover, the size of the window of the Outlier Detection is being changed into ”5” instead of ”3”.

```
int hlf = 2;
    for (int i = 0; i < size; i ++)
        for (int j = 0; j < size; j ++){
            int ara = 0, sum = 0;
            for (int m = i - hlf; m <= i + hlf; m ++)
                for (int n = j - hlf; n <= j + hlf; n
                   ++){
                    if (m < 0 || m > size - 1) continue;
                    if (n < 0 || n > size - 1) continue;
                    if (i == m && j == n) continue;

                    sum += img[m][n];
                    ara ++;
                }

            if (abs(img[i][j] * ara - sum) > e * ara)
               tmp[i][j] = sum / ara;
            else tmp[i][j] = img[i][j];
        }
```

(b) Compute the PSNR values of $N_!$ and N_2 and provide some discussions.
As mentioned above, the PSNR of N_1 and N_2 are 28.662193 and 31.602384 respectively. PSNR, so-called ”Peak Signal-to-Noise Ratio”, is a way to calculate the degree of noise. The bigger the value is, the greater the improvement will be. By calculating the PSNR value, we can know N_2 would be more clear than N_1. By viewing the images, we can also prove the proposal. Other deeply discussion has already present in problem2(a) so I will just skip it.

Appendix

- `Warmup_a.cpp` to horizontally flip the image.
  Compile by `g++ Warmup_a.cpp -o warmup_a`;
  execute by `./warmup_a` and get the output image `B.raw`.

- `Warmup_b.cpp` to adjust the image through power-law function.
  Compile by `g++ Warmup_b.cpp -o warmup_b`;
  execute by `./warmup_b` then input the number n as the power, and get the output image `B.raw`.

- `Problem1_a.cpp` to divide the intensity value by 2.
  Compile by `g++ Problem1_a.cpp -o Problem1_a`;
  execute by `./Problem1_a` and get the output image `D.raw`..

- `Problem1_b.cpp` to divide the intensity value by 3.
  Compile by `g++ Problem1_b.cpp -o Problem1_b`;
  execute by `./Problem1_b` and get the output image `E.raw`.

- `Problem1_c.cpp` to compute the occurrences of specific pixel values.
  Compile by `g++ Problem1_c.cpp -o Problem1_c`;
  execute by `./Problem1_c` and get the output file `Problem1_c.csv`.
  * Plot the histograms by using the function of excel.

- `Problem1_d.cpp` to do global histogram equalization and compute the occurrences of specific pixel values.
  Compile by `g++ Problem1_d.cpp -o Problem1_d`;
  execute by `./Problem1_d` and get the output image `H_D.raw` as well as `H_E.raw` and the output file `Problem1_d.csv`.
  * Plot the histograms by using the function of excel.

- `Problem1_e.cpp` to do local histogram equalization and compute the occurrences of specific pixel values.
  Compile by `g++ Problem1_e.cpp -o Problem1_d`;
  execute by `./Problem1_e` and get the output image `L_D.raw` as well as `L_E.raw` and the output file `Problem1_e.csv`.
  * Plot the histograms by using the function of excel.

- `Problem1_lowfilter.cpp` to do Low Pass Filter for the input image.

Compile by `g++ Problem2_lowfilter.cpp -o Problem2_lowfilter`;
execute by `./Problem2_lowfilter` and input the name of tyhe input image and the name of the output image.
* Plot the histograms by using the function of excel.

- `Problem2_median.cpp` to do Median Filter for the input image.
  Compile by `g++ Problem2_median.cpp -o Problem2_median`;
  execute by `./Problem2_median` and input the name of the input image and the name of the output image.
  * Plot the histograms by using the function of excel.

- `Problem1_outlier.cpp` to do Outlier Detection for the input image.
  Compile by `g++ Problem2_outlier.cpp -o Problem2_outlier`;
  execute by `./Problem2_outlier` and input the name of the input image and the name of the output image.
  * Plot the histograms by using the function of excel.