Image Processing (NYCU CS, Fall 2022)

Programming Assignment #1

1. Introduction / Objectives:

In this assignment, I learned to perform several classic image enhancement tricks from scratch. As the requirement stated, all processing techniques are written as reusable modules. Moreover, contrast adjustment, noise reduction and color correction are implemented on seven different images taken by me, my friends and from the Internet. Implementation details will be discussed in the following sections.

2. A review of the methods you have used (be concise)
   a. Histogram Equalization
      Histogram equalization is an elegant image enhancement technique of all time. I utilized this algorithm on many of the testing images. However, as we will see later, histogram equalization does not always improve the condition of images as we wish.
   b. Gaussian Blur
      Gaussian filter is among the simplest filter. It does not provide strong ability in denoising salt and pepper noise, comparing to median filters.
   c. Median Filter
      Median filter does perform extremely well on salt and pepper noise as the textbook claimed. The experimental results are shown in this report.
   d. Adjusting HSI
      Hue, intensity and saturation are no less important than all those deblurring filters in image processing. Thus, I implemented a module specifically for the adjustment of HSI.
3. Experiments, Interpretation and Explanation
   a. Figure 1. 2.

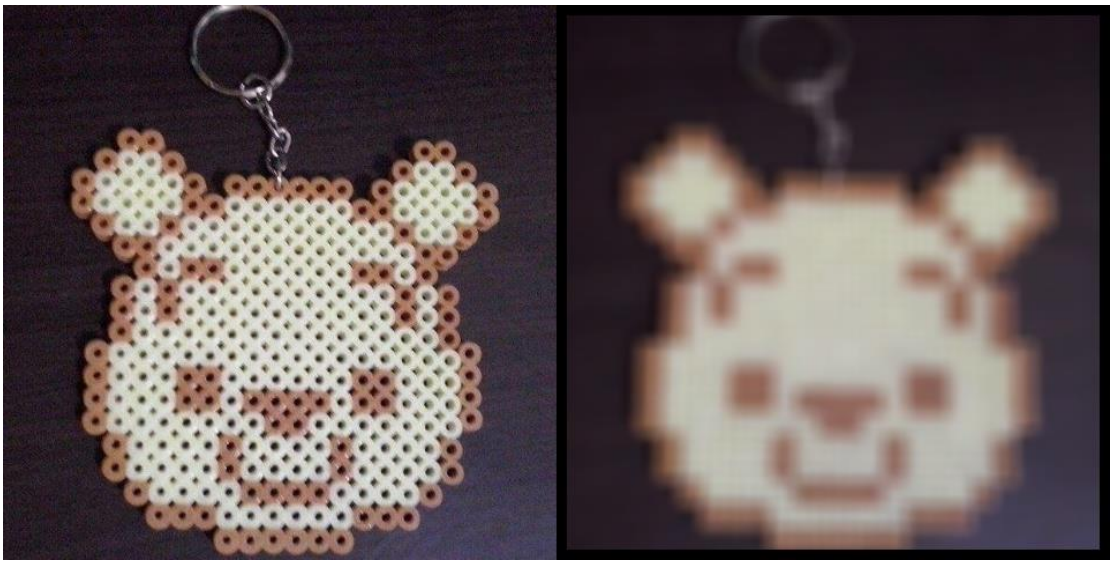
Above is the comparison between original image and the one after 3*3 median filter. As you might figure, there is no visible differences.

However, when we apply histogram equalization first, then apply median filter, the nose and the smile of the black animal is disappeared!!! The white nose and smile can be considered as the unwanted noises. Since the image is a bit too bright, **histogram equalization** helps remapping the intensity, which can help **median filter** to remove noises with only 3*3 filter, **without further damaging the image resolution**.

b. Figure 3.

The above image is enhanced by both histogram equalization and HIS adjustment. We have discussed the benefits of histogram equalization. It can redistribute the intensities; however, it cannot **brighten the dark part** of the image. The image on the right is much preferrable for most application. The adjustment is actually pretty simple. Time 1.5 to the intensities of every pixel and **clip those who exceed 255**. I have tested many other methods to tackle the exceeding problems. Clipping to 255 provide the **smoothest image**. Other methods all lead to visible, strange pixels in the brightest areas.

c. Figure 4. 5.



13*13 Gaussian blur is applied on the above image with zero padding. The little black holes in the center and margin are **almost completely removed**; however, the **image is damaged** to an unbearable extent.

Comparing to gaussian filters, **median filters** are much **more capable of eliminating pepper noise**, even large peppers! The image resolution is only slightly decrease and the black holes are elegantly removed.

d. Figure 6. 7.8.

Figure 6. is simply obtained by timing the **intensity** of every pixel of the image by **5.7**, the image is certainly **brightened up** and one can finally realize what hide beneath the trees beside the sidewalk. What's more, the center part, which is of deeper depth becomes more visible after the adjustment of HIS.

My experiment does not stop here. By further **increasing the saturation** by 1.5, the image releases a **warmer atmosphere**. It is like the one image is taken in winter and the other in summer.

As a **comparison to histogram equalization** shown above, sampling adjusting the intensity of the image leads to a cleaner image. Histogram equalization indeed brighten up the image, but the sky is **full of visible,** unpleasant **black pixels**.

Last but not least, I decided to get my hands dirty and try some interesting combination of vary techniques. The above image is produced by applying **17*17 gaussian blur**, **17*17 median filter** and finally **histogram equalization**. As the filters get larger, the results are developing toward **image segmentation**.

4. Discussions

This programming assignment is very fun. I have taken Introduction to Image Processing before, but the course is all about theories, so I barely get to try and proof those theory works myself.

When I first implement histogram equalization, I apply the method on three color channels separately, which is totally wrong. Never did I realize that histogram equalization is all about intensity, not color. After reading several sources online, I transform the color space representation from BGR to HSI and implement histogram equalization once on the intensity. The result is beautiful, and I get to understand the theory more.

Writing the filters from scratch is also pretty interesting. I can calculate the degree of complexity of each implantation and design my own filters. There are so many variations, including the size of the kernel, the value, and even padding has many different types.

lab1.py

lab1.py > ...

```python
import cv2
import numpy as np
from histogram_equalization import histogram_equalization
from guassian_blur import guassian_blur
from median_filter import median_filter
from adjust_hsv import adjust_HSV


# ------------------------- test1.jpg ---------------------------------
img = cv2.imread('test1.jpg')
h, w, c = img.shape

# Histogram_Equalization
hq_img = histogram_equalization(img)

# Median filter
median_img = median_filter(hq_img, k_size=3)

# Comparison
compare_display = np.concatenate((img, median_img), axis=1)
cv2.imwrite("test1_median_3.jpg", compare_display)
cv2.imshow('test1_hq', compare_display)
cv2.waitKey(0)

# ------------------------- test2.jpg ---------------------------------

img = cv2.imread('test2.jpg')
img = cv2.resize(img, (512, 300))
h, w, c = img.shape
print(img.shape)
```

```
31  # Histogram_Equalization
32  hq_img = histogram_equalization(img)
33
34  # HSI adjustment
35  adjust_img = adjust_HSV(hq_img, h_ratio=1, s_ratio=1, v_ratio=1.5)
36
37  # Comparison
38  compare_display = np.concatenate((img, adjust_img), axis=1)
39  cv2.imwrite("test2_hq_HSI_adjust.jpg", compare_display)
40  cv2.imshow('test1_hq', compare_display)
41  cv2.waitKey(0)
42
43  # ------------------------- test3.jpg ---------------------------------
44
45  img = cv2.imread('test3.jpg')
46  img = cv2.resize(img, (480, 480))
47  h, w, c = img.shape
48  print(img.shape)
49
50  # Guassian blur
51  blur_img = guassian_blur(img, k_size=19)
52  compare_display = np.concatenate((img, blur_img), axis=1)
53  cv2.imwrite("test3_guassian_19.jpg", compare_display)
54  cv2.imshow('test1_hq', compare_display)
55  cv2.waitKey(0)
56
57
```

```
58  # ------------------------- test6.jpg ---------------------------------
59
60  img = cv2.imread('test6.jpg')
61  h, w, c = img.shape
62  img = cv2.resize(img, (480, 360))
63  print(img.shape)
64
65  blur_img = guassian_blur(img, k_size=17)
66  medium_img = median_filter(blur_img, k_size=17)
67
68  # Histogram_Equalization
69  hq_img = histogram_equalization(medium_img)
70  # HSI
71  adjust = adjust_HSV(hq_img, 1, 1.5, 5.7)
72  # Comparison
73  compare_display = np.concatenate((img, adjust), axis=1)
74  cv2.imwrite("test6_hsv.jpg", compare_display)
75  cv2.imshow('test1_hq', compare_display)
76  cv2.waitKey(0)
77
78
79
```

histogram_equalization.py > ...

```python
import cv2
import numpy as np

def histogram_equalization(img): # images with single channel
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    intensity = hsv_img[:, :, 2]
    histogram = np.bincount(intensity.flatten(), minlength=256)
    num_pixel = np.sum(histogram)
    histogram = histogram / num_pixel
    chistogram = np.cumsum(histogram)

    transform_map = np.floor(255 * chistogram).astype(np.uint8)
    old_intensity = list(intensity.flatten())
    new_intensity = [transform_map[i] for i in old_intensity]
    hq_intensity = np.reshape(np.asarray(new_intensity), intensity.shape)
    hsv_img[:, :, 2] = hq_intensity
    hq_img = cv2.cvtColor(hsv_img, cv2.COLOR_HSV2BGR)

    return hq_img

```

guassian_blur.py > guassian_blur

```python
import cv2
import numpy as np

def guassian_blur(img, k_size=3):
    h, w, c = img.shape
    blur_img = img.copy()
    k = int((k_size - 1) / 2)
    for i in range(0, h):
        for j in range(0, w):
            sum = np.zeros(3)
            if i-k >= 0 and i+k < h:
                for u in range(-k, k+1):
                    if j-k >= 0 and j+k < w:
                        for v in range(-k, k+1):
                            sum += img[i+u, j+v]
            blur_img[i, j] = sum/(k_size*k_size)
    return blur_img
```

median_filter.py > median_filter

```python
import cv2
import numpy as np

def median_filter(img, k_size=3):
    if img.ndim == 2:
        h, w = img.shape
    else:
        h, w, c = img.shape
    median_img = img.copy()
    for dim in range(img.ndim):
        k = int((k_size - 1) / 2)
        for i in range(0, h):
            for j in range(0, w):
                receptive_lst = []
                if i-k >= 0 and i+k < h:
                    for u in range(-k, k+1):
                        if j-k >= 0 and j+k < w:
                            for v in range(-k, k+1):
                                receptive_lst.append(img[i+u, j+v, dim])
                receptive_lst.sort()
                m = int(len(receptive_lst)/2)
                if receptive_lst != []:
                    median_img[i, j, dim] = receptive_lst[m]
    return median_img
```

adjust_hsv.py > adjust_HSV

```python
import cv2
import numpy as np

def adjust_HSV(img, h_ratio=1, s_ratio=1, v_ratio=1):
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    print(hsv_img.shape)
    print(hsv_img[0,0,:])
    hsv_img[:,:,0] = hsv_img[:,:,0] * h_ratio
    hsv_img[:,:,1] = np.where(hsv_img[:,:,1] * s_ratio <=255, hsv_img[:,:,1] * s_ratio, 255)
    hsv_img[:,:,2] = np.where(hsv_img[:,:,2] * v_ratio <=255, hsv_img[:,:,2] * v_ratio, 255)
    print(hsv_img[0,0,:])
    hsv_img = cv2.cvtColor(hsv_img, cv2.COLOR_HSV2BGR)
    return hsv_img
```