Image Processing - Assignment 1

Image Enhancement

Beryl Li

1 Introduction

In assignment 1, image enhancement techniques are applied to improve the quality of images in bad conditions. Firstly, power-law transformation and histogram equalization can be applied to overexposure and underexposure images so that we can perceive the changes in images better. Secondly, smoothing filters like box, Gaussian, median and bilateral filters can be used to mitigate noise in images so that sharpening filters can later perform a better job on accentuating the edges. Thirdly, laplacian filters are adopted to make edges sharper, which restores some fine details of images.

2 Methods

2.1 Power-law transformation

Power-law transformation is one kind of contrast stretching, which adjusts the range of the gray levels in an image. We can expand the range of gray levels to make some details clearer and reduce the contrast of less interesting parts.

Fig 1 displays the curves for different γ setting. When the slope $(ds/dr) > 1$, contrast is enhanced. Conversely, When the slope $(ds/dr) < 1$ contrast is decreased. When it comes to γ, input image becomes brighter as $\gamma < 1$, and input image becomes darker as $\gamma > 1$.

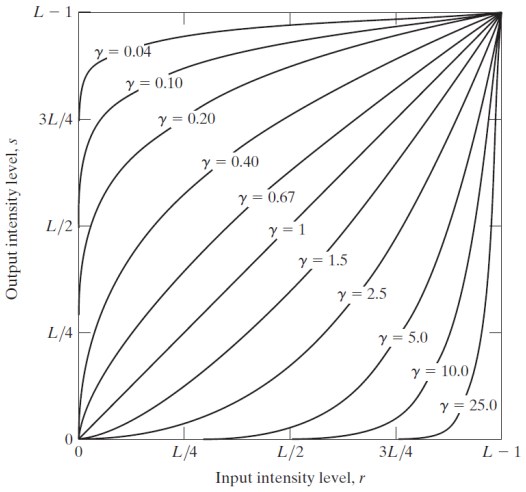


Figure 1: Power-law transformation

2.2 Histogram Equalization

Histogram equalization increases the contrast for images whose intensity values are distributed in a narrow range. After histogram equalization, the intensity value distribution becomes more even. Fig 2 is an example of histograms before and after equalization.

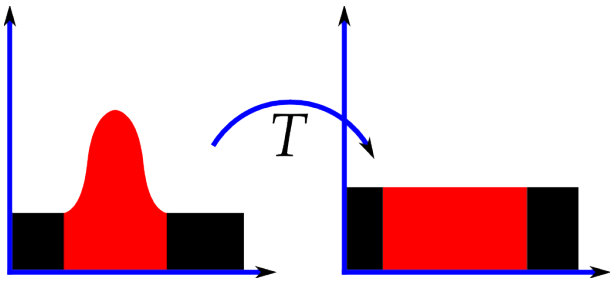


Figure 2: Histograms before and after equalization

2.3 Smoothing Filters

Smoothing filters are applied to images to reduce noise. Box filter and Gaussian filter are implemented by separable kernels.

2.3.1 Box Filter

A box filter weights all samples within a square region of an image equally to make the image smoother and even blur it. It is a separable filter, so it is computationally efficient. Fig 3 shows some box filters with different kernel size.

$$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

(a) Kernel size 3×3

$$\frac{1}{25}\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

(b) Kernel size 5×5

Figure 3: Box filters

2.3.2 Gaussian Filter

A Gaussian filter is a separable filter, which can be the product of two identical one-dimensional Gaussian distributions. It can efficiently reduce noise, but it will blur the edges.

2.3.3 Median Filter

A median filter replaces a pixel with the median of neighboring pixels. It is particularly effective when dealing with salt-and-pepper noise.

2.3.4 Bilateral Filter

A bilateral filter is a smoothing filter which is edge-preserving. It replaces a pixel value with a weighted average of values from neighboring pixels, and its weights are based on Gaussian distribution. Therefore, the closer the neighboring pixel or the more similar the values of neighboring pixel, the greater the weight.

2.4 Laplacian Filters

A laplacian filter is a second derivative method, and it highlights regions of significant intensity change so that is can be used to sharpen the edges of an image. Fig 4 shows three commonly-used laplacian filters, and the rightmost one is separable.

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

−1	2	−1
2	−4	2
−1	2	−1

Figure 4: Laplacian Filters

3 Experiments

3.1 Experiment 1 - Overexposure image

In experiment 1, contrast enhancement is applied to an overexposure image, and Fig 5a is the original image. The correction procedure is as follows. Firstly, use Gaussian filter to get a less noisy image, feed the blurry result to laplacian filter to obtain detected edges, and then add the edges to the original images to get sharpening result. Secondly, bilateral filter with kernel size = 5 and $\sigma = 2$ is adopted. Thirdly, for power law transformation, parameters with constant = [0.8, 1, 1.2] and $\gamma = [1, 1.5, 2.5, 5.0]$ is set to generate results showing in Fig 5c; for histogram equalization, the median filter is applied first before equalizing the histogram, and the result is shown in Fig 5b.

(a) Original image

(b) After histogram equalization

(c) After power-law transformation

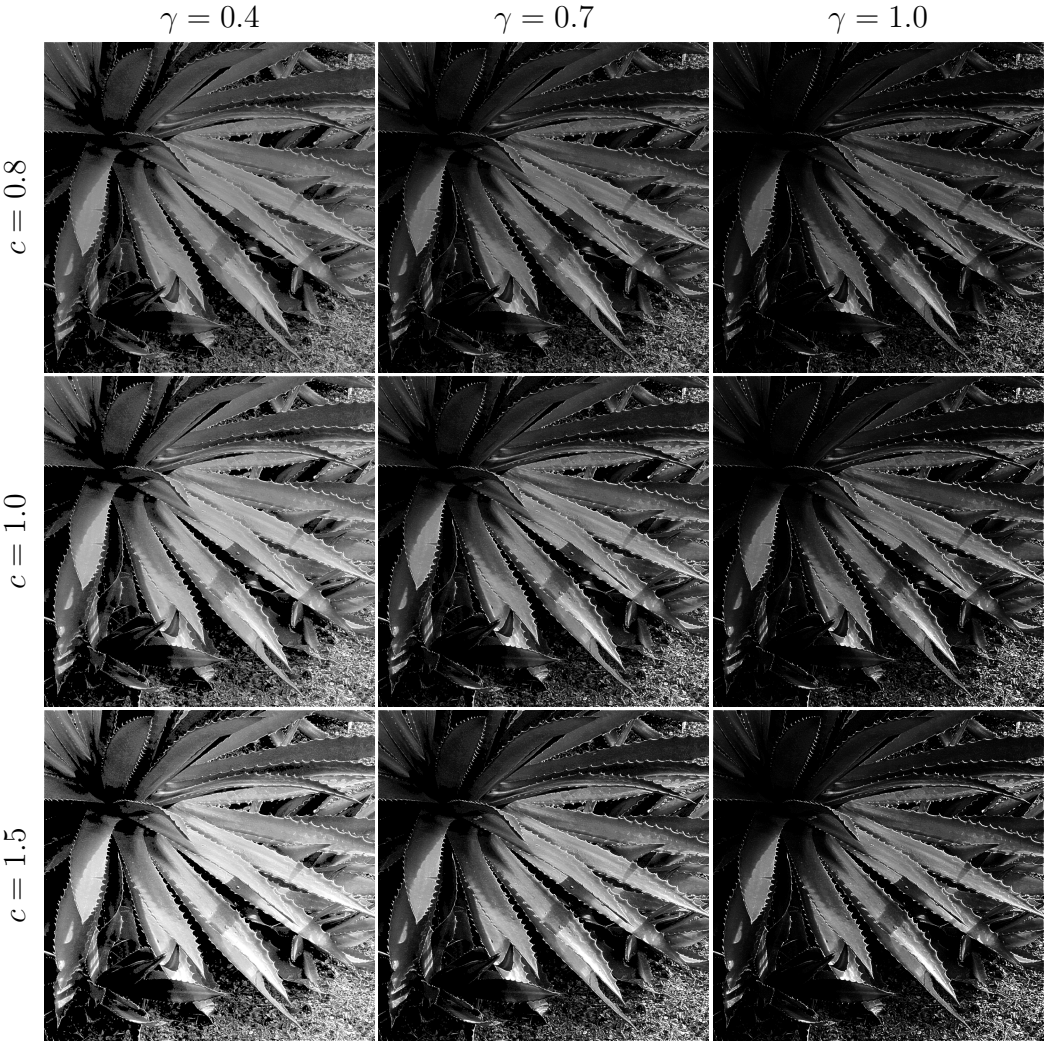
Figure 5: Experiment 1 - overexposure image

3.2 Experiment 2 - Underexposure image

In experiment 2, contrast enhancement is applied to an underexposure image, and Fig 6a is the original image. The correction procedure is almost the same as in experiment 1. There are two differences. One difference is γ of power-law transformation, which is $[1, 1.5, 2.5, 5.0]$ this time. The other one is that the median filter is not applied before equalizing the histogram. Fig 6c shows the results of power law transformation, and Fig 6b shows the result of histogram equalization.

(a) Original image

(b) After histogram equalization



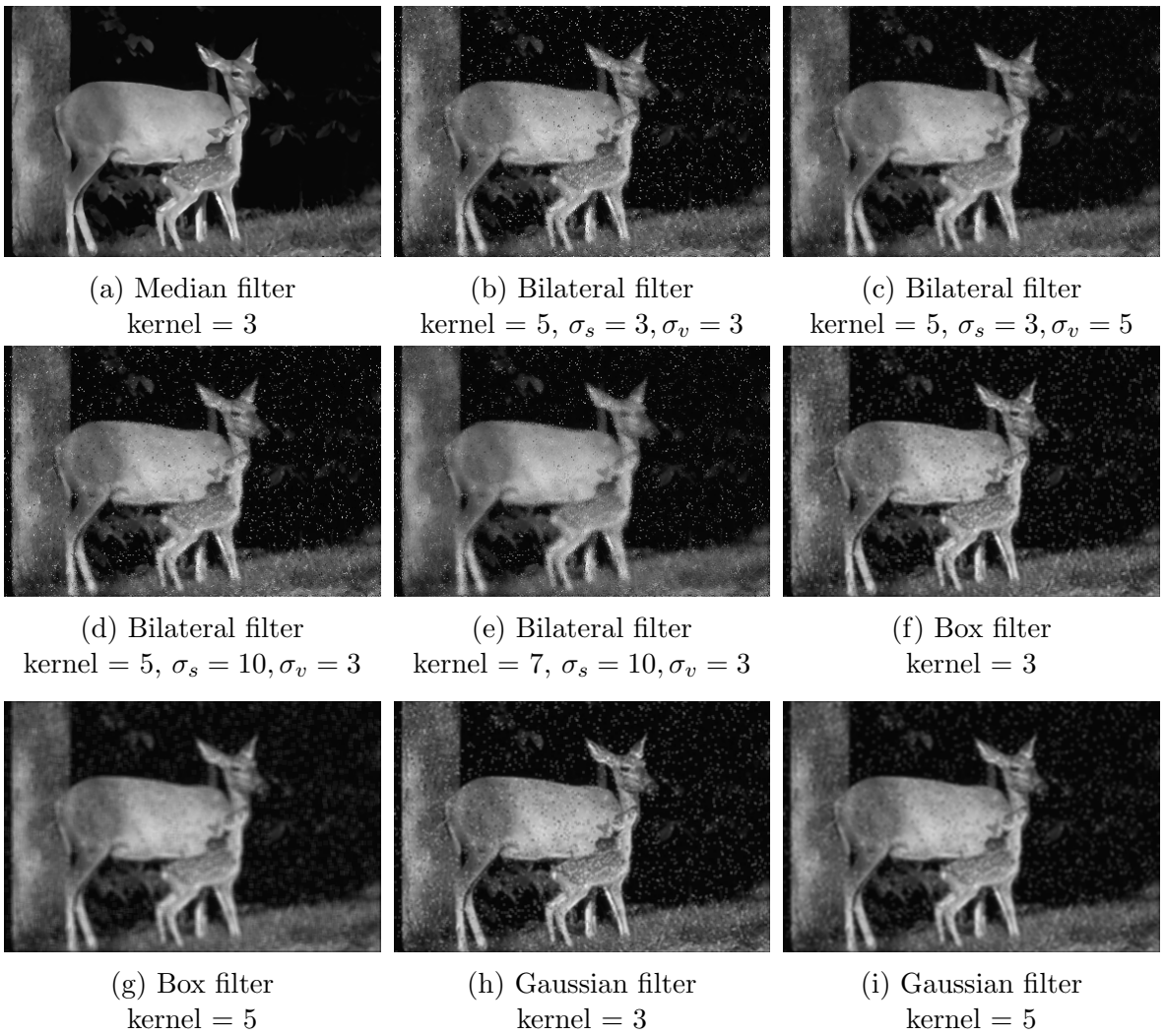
(c) After power-law transformation

Figure 6: Experiment 2 - underexposure image

3.3 Experiment 3 - Salt-and-pepper noise, gray image

In experiment 3, smoothing filters are applied to mitigate salt-and-pepper noise in a gray image, which is shown in Fig 7. Four kinds of smoothing filters are tested, namely median, box, Gaussian, bilateral filters. Fig 8 displays the results of these filters.

Figure 7: Original image



(a) Median filter
kernel = 3

(b) Bilateral filter
kernel = 5, $\sigma_s = 3, \sigma_v = 3$

(c) Bilateral filter
kernel = 5, $\sigma_s = 3, \sigma_v = 5$

(d) Bilateral filter
kernel = 5, $\sigma_s = 10, \sigma_v = 3$

(e) Bilateral filter
kernel = 7, $\sigma_s = 10, \sigma_v = 3$

(f) Box filter
kernel = 3

(g) Box filter
kernel = 5

(h) Gaussian filter
kernel = 3

(i) Gaussian filter
kernel = 5

Figure 8: Experiment 3 - Salt-and-pepper noise, gray image

3.4 Experiment 4 - Salt-and-pepper noise, color image

In experiment 4, smoothing filters are applied to mitigate salt-and-pepper noise in a color image, which is shown in Fig 9. Again, four kinds of smoothing filters are tested, namely median, box, Gaussian, bilateral filters. These filters are directly applied to each of the color channels. Fig 10 displays the results of these filters.

Figure 9: Original image

(a) Median filter
kernel = 5

(b) Bilateral filter
kernel = 5, $\sigma_s = 3, \sigma_v = 3$

(c) Box filter
kernel = 5

(d) Gaussian filter
kernel = 5

Figure 10: Experiment 4 - Salt-and-pepper noise, color image

4 Discussions

In experiment 1 and 2, I do contrast enhancement on overexposure and underexposure images. At first, I tried to directly apply power-law transformation or histogram equalization on the original image, and found that the result was quite noisy, especially for the overexposure image I tested on. As you can see in Fig 11, both resulting images have horizontal stripes, and this effect on the right one is comparatively serious. After some smoothing and sharpening filters are adopted, this problem is solved and better quality images is shown in Fig 5b and Fig 5c.

(a) After histogram equalization (b) After power-law transformation

Figure 11: Experiment 1 - overexposure image without smoothing first

In experiment 1, the result of power-law transform with $c = 1.0$ and $\gamma = 5.0$ seems to be better than the result of histogram equalization to me. I think this is because histogram equalization evenly distributes the intensity values. However, the high intensity values in this image are more meaningful to human eyes. Therefore, the result of power-law transform with $c = 1.0$ and $\gamma = 5.0$ makes me feel better.

In experiment 2, even through the results of histogram equalization and power-law transformation with $c = 1.5$ and $\gamma = 0.4$ are both good, I still prefer the result of power-law transformation. The reason why I prefer the result of power-law transformation is probably because human eyes are more sensitive to changes in dark as compared to bright. Furthermore, I indeed feel that the result of histogram equalization is a little bit too bright for eyes to perceive.

In experiment 3, the median filter performs best among all other kinds of filters with little blur effect when the filter size is small. However, the bilateral filter either causes the blur effect or could not remove noise well. If I increase the kernel size and standard deviation of space, the noise removal is more significant although the blur effect also becomes severe. Comparatively, if I increase standard deviation of value instead, the result seems to be better among all the results of bilateral filter. As for box filter and Gaussian filter, the larger the kernel size the severer the blur effect, and the box filter leads to the most severe blur effect among all the results showing in Fig 8.

In experiment 4 a color image is used. The median filter performs best among all other kinds of filters as well. All the other filters either show severe blur effect or could not remove noise well.

5 Codes

5.1 main.py

```python
def experiment_1():
    parser = argparse.ArgumentParser()
    parser.add_argument('--img_file', type=str, default='img/man.jpg')
    parser.add_argument('--is_gray', type=bool, default=True)
    parser.add_argument('--output_file', type=str, default='output_man_/man')
    args = parser.parse_args()
    img = read_img(args.img_file, args.is_gray)

    ''' smoothing filter '''
    kernel = get_kernel1d(kernel_size=3, type='gaussian')
    img_new = conv_separable(img, kernel)

    ''' laplacian filter '''
    kernel = get_kernel2d(type='laplacian_2')
    img_new = conv_nonseparable(img_new, kernel).astype('int64')
    img = img.astype('int64') + 2 * img_new
    img[img > 255] = 255; img[img < 0] = 0

    ''' bilateral filter '''
    img = bilateral_filter(img, 5, 2, 2)

    ''' power law transformation '''
    output_file = args.output_file+'_{c:02d}_{gamma_id:02d}'
    const, gamma = [0.8, 1, 1.2], [1, 1.5, 2.5, 5.0]
    for c in const:
        for i in range(len(gamma)):
            img_corrected = power_law_trans(img, gamma[i], c)
            save_img(img_corrected, output_file.format(c=int(c*10),
                gamma_id=i))

    ''' median filter '''
    img = median_filter(img, 3)

    ''' histogram equilization '''
    if args.is_gray:
        img = equalize_hist(img)
        save_img(img, args.output_file)
```

Code 1: Experiment 1 - Overexposure image

```python
def experiment_2():
    parser = argparse.ArgumentParser()
    parser.add_argument('--img_file', type=str, default='img/plant.png')
    parser.add_argument('--is_gray', type=bool, default=True)
    parser.add_argument('--output_file', type=str, default='output_plant/plt')
    args = parser.parse_args()
    img = read_img(args.img_file, args.is_gray)

    ''' smoothing filter '''
    kernel = get_kernel1d(kernel_size=3, type='gaussian')
    img_new = conv_separable(img, kernel)

    ''' laplacian filter '''
    kernel = get_kernel2d(type='laplacian_2')
    img_new = conv_nonseparable(img_new, kernel).astype('int64')
    img = img.astype('int64') + 2 * img_new
    img[img > 255] = 255; img[img < 0] = 0

    ''' bilateral filter '''
    img = bilateral_filter(img, 5, 2, 2)

    ''' power law transformation '''
    output_file = args.output_file+'_{c:02d}_{gamma_id:02d}'
    const = [0.8, 1, 1.2]
    gamma = [0.4, 0.7, 1]
    for c in const:
        for i in range(len(gamma)):
            print(output_file.format(c=int(c*10), gamma_id=i))
            print(gamma[i])
            img_corrected = power_law_trans(img, gamma[i], c)
            save_img(img_corrected, output_file.format(c=int(c*10),
                gamma_id=i))

    ''' histogram equilization '''
    if args.is_gray:
        output_file = args.output_file+'_hist'
        img = equalize_hist(img)
        print(args.output_file)
        display_img(img, args.is_gray)
        save_img(img, output_file)
```

Code 2: Experiment 2 - Underexposure image

```python
def experiment_3():
    parser = argparse.ArgumentParser()
    parser.add_argument('--img_file', type=str, default='img/deers.png')
    parser.add_argument('--is_gray', type=bool, default=True)
    parser.add_argument('--output_file', type=str, default='output_deers/drs')
    args = parser.parse_args()
    img = read_img(args.img_file, args.is_gray)

    ''' median filter '''
    img_med = median_filter(img, 3)
    save_img(img_med, args.output_file+'_median')

    ''' bilateral filter '''
    img_bi = bilateral_filter(img, 5, 3, 3)
    save_img(img_bi, args.output_file+'_bi_k5_s3_v3')

    img_bi = bilateral_filter(img, 5, 3, 5)
    save_img(img_bi, args.output_file+'_bi_k5_s3_v5')

    img_bi = bilateral_filter(img, 5, 10, 3)
    save_img(img_bi, args.output_file+'_bi_k5_s10_v3')

    img_bi = bilateral_filter(img, 7, 10, 3)
    save_img(img_bi, args.output_file+'_bi_k7_s10_v3')

    ''' gaussian filter '''
    kernel = get_kernel1d(kernel_size=3, type='gaussian')
    img_gaussian = conv_separable(img, kernel)
    save_img(img_gaussian, args.output_file+'_gaussian_3')

    kernel = get_kernel1d(kernel_size=5, type='gaussian')
    img_gaussian = conv_separable(img, kernel)
    save_img(img_gaussian, args.output_file+'_gaussian_5')

    ''' box filter '''
    kernel = get_kernel1d(kernel_size=3, type='averaging')
    img_box = conv_separable(img, kernel)
    save_img(img_box, args.output_file+'_box_3')

    kernel = get_kernel1d(kernel_size=5, type='averaging')
    img_box = conv_separable(img, kernel)
    save_img(img_box, args.output_file+'_box_5')
```

Code 3: Experiment 3 - Salt-and-pepper noise, gray image

```python
def experiment_4():
    parser = argparse.ArgumentParser()
    parser.add_argument('--img_file', type=str, default='img/rose.jpg')
    parser.add_argument('--is_gray', type=bool, default=False)
    parser.add_argument('--output_file', type=str, default='output_rose/rose')
    args = parser.parse_args()
    img = read_img(args.img_file, args.is_gray)
    img_med = np.empty(img.shape)
    img_bi = np.empty(img.shape)
    img_gaussian = np.empty(img.shape)
    img_box = np.empty(img.shape)

    ''' median filter '''
    img_med[:, :, 0] = median_filter(img[:, :, 0], 5)
    img_med[:, :, 1] = median_filter(img[:, :, 1], 5)
    img_med[:, :, 2] = median_filter(img[:, :, 2], 5)
    img_med = img_med.astype('uint8')
    save_img(img_med, args.output_file+'_median')

    ''' bilateral filter '''
    img_bi[:, :, 0] = bilateral_filter(img[:, :, 0], 5, 3, 3)
    img_bi[:, :, 1] = bilateral_filter(img[:, :, 1], 5, 3, 3)
    img_bi[:, :, 2] = bilateral_filter(img[:, :, 2], 5, 3, 3)
    img_bi = img_bi.astype('uint8')
    save_img(img_bi, args.output_file+'_bi')

    ''' gaussian filter '''
    kernel = get_kernel1d(kernel_size=5, type='gaussian')
    img_gaussian[:, :, 0] = conv_separable(img[:, :, 0], kernel)
    img_gaussian[:, :, 1] = conv_separable(img[:, :, 1], kernel)
    img_gaussian[:, :, 2] = conv_separable(img[:, :, 2], kernel)
    img_gaussian = img_gaussian.astype('uint8')
    save_img(img_gaussian, args.output_file+'_gaussian_5')

    ''' box filter '''
    kernel = get_kernel1d(kernel_size=5, type='averaging')
    img_box[:, :, 0] = conv_separable(img[:, :, 0], kernel)
    img_box[:, :, 1] = conv_separable(img[:, :, 1], kernel)
    img_box[:, :, 2] = conv_separable(img[:, :, 2], kernel)
    img_box = img_box.astype('uint8')
    save_img(img_box, args.output_file+'_box_5')
```

Code 4: Experiment 4 - Salt-and-pepper noise, color image

5.2 img_proc.py

```python
def power_law_trans(img, gamma, c=1.0):
    """ Power-law transformations
    Args:
        img (ndarray): with shape (height, width) for gray / (height, width, channel) for BGR
        gamma (float)
        c (float)
    Returns:
        img_corrected (ndarray): its shape is the same as input's
    """
    img_corrected = np.array(c * 255 * (img / 255) ** gamma)
    img_corrected[img_corrected > 255] = 255
    img_corrected = img_corrected.astype('uint8')
    return img_corrected


def equalize_hist(img):
    """ Histogram equalization
    Args:
        img (ndarray): its shape is (height, width)
    Returns:
        img_corrected (ndarray): its shape is the same as input's
    """
    assert len(img.shape) == 2, 'Input image has more than one channel. '
    count_raw = np.bincount(img.flatten(), minlength=256)
    sum, cnt_average = 0, np.sum(count_raw) / 256
    lookup_table = np.zeros(256)
    for i in range(256):
        sum += count_raw[i]
        lookup_table[i] = int(sum / cnt_average)
    lookup_table[lookup_table > 255] = 255

    (height, width) = img.shape
    img_corrected = np.empty(img.shape, dtype='uint8')
    for h in range(height):
        for w in range(width):
            img_corrected[h][w] = lookup_table[img[h][w]]
    # display_hist(count_raw)
    # count_new = np.bincount(img_corrected.flatten(), minlength=256)
    # display_hist(count_new)
    return img_corrected
```

Code 5: Function: $power\_law\_trans(img, gamma, c)$ and $equalize\_hist(img)$

```python
def get_kernel1d(kernel_size=3, type='averaging'):
    """ Get 1D separable filter
    Args:
        kernel_size (int)
        type (str): type of kernel
    Returns:
        kernel (ndarray): 1D separable kernel
    """
    assert kernel_size % 2 == 1, 'Kernel size is not an odd number. '
    if type == 'averaging':
        return np.ones(kernel_size) / kernel_size
    if type == 'gaussian':
        kernel = np.fromfunction(lambda x: np.exp((-1*(x-(kernel_size-1)/2) **
            2) / 2 ** 2), (kernel_size,))
        return kernel / np.sum(kernel)
    if type == 'laplacian':
        return np.array((-1, 2, -1))
    assert False, 'Input filter type does not exist. '


def conv_separable(img, kernel):
    """ Perform convolution with separable kernel
    Args:
        img (ndarray): its shape is (height, width)
        kernel (ndarray): 1D separable kernel
    Returns:
        img_corrected (ndarray): its shape is the same as input's
    """
    assert len(img.shape) == 2, 'Input image has more than one channel. '
    kernel_size = len(kernel)
    ''' zero padding '''
    (height, width) = img.shape
    img_padding = np.zeros((height+kernel_size-1, width+kernel_size-1))
    origin = int(kernel_size/2)
    img_padding[origin:origin+height, origin:origin+width] = img
    ''' separable convolution '''
    img_temp = np.empty((height+kernel_size-1, width))
    img_corrected = np.empty((height, width))
    for h in range(height+kernel_size-1):
        for w in range(width):
            img_temp[h][w] = np.dot(kernel, img_padding[h, w:w+kernel_size])
    for h in range(height):
        for w in range(width):
            img_corrected[h][w] = np.dot(kernel, img_temp[h:h+kernel_size, w])
    img_corrected[img_corrected > 255] = 255
    img_corrected[img_corrected < 0] = 0
    return img_corrected.astype('uint8')
```

Code 6: Function: $get\_kernel1d(kernel\_size, type)$ and $conv\_separable(img, kernel)$

```python
def get_kernel2d(type='laplacian'):
    """ Get 2D filter
    Args:
        type (str): type of kernel
    Returns:
        kernel (ndarray): 2D kernel
    """
    if type == 'laplacian_1':
        return np.array(((0, -1, 0), (-1, 4, -1), (0, -1, 0)))
    if type == 'laplacian_2':
        return np.array(((-1, -1, -1), (-1, 8, -1), (-1, -1, -1)))
    if type == 'laplacian_sharpen':
        return np.array(((0, -1, 0), (-1, 5, -1), (0, -1, 0)))
    assert False, 'Input filter type does not exist. '


def conv_nonseparable(img, kernel):
    """ Perform convolution with separable kernel
    Args:
        img (ndarray): its shape is (height, width)
        kernel (ndarray): 2D kernel
    Returns:
        img_corrected (ndarray): its shape is the same as input's
    """
    assert len(img.shape) == 2, 'Input image has more than one channel. '
    assert len(kernel.shape) == 2, 'Kernel is not 2D. '
    (kernel_size, _) = kernel.shape
    ''' zero padding '''
    (height, width) = img.shape
    img_padding = np.zeros((height+kernel_size-1, width+kernel_size-1))
    origin = int(kernel_size/2)
    img_padding[origin:origin+height, origin:origin+width] = img
    ''' convolution '''
    img_corrected = np.empty((height, width))
    for h in range(height):
        for w in range(width):
            product = np.multiply(kernel, img_padding[h:h+kernel_size,
                w:w+kernel_size])
            img_corrected[h][w] = np.sum(product)
    img_corrected[img_corrected > 255] = 255
    img_corrected[img_corrected < 0] = 0

    return img_corrected.astype('uint8')
```

Code 7: Function: *get_kernel2d*(*type*) and *conv_nonseparable*(*img*, *kernel*)

```python
def median_filter(img, kernel_size=3):
    """ Perform median filtering
    Args:
        img (ndarray): its shape is (height, width)
        kernel_size (int)
    Returns:
        img_corrected (ndarray): its shape is the same as input's
    """
    assert len(img.shape) == 2, 'Input image has more than one channel. '
    ''' zero padding '''
    (height, width) = img.shape
    idx_median = np.floor(kernel_size * kernel_size / 2).astype('int')
    img_padding = np.zeros((height+kernel_size-1, width+kernel_size-1))
    origin = int(kernel_size/2)
    img_padding[origin:origin+height, origin:origin+width] = img
    img_corrected = np.empty((height, width))
    for h in range(height):
        for w in range(width):
            patch = img_padding[h:h+kernel_size, w:w+kernel_size]
            img_corrected[h][w] = np.sort(patch, axis=None)[idx_median]
    return img_corrected.astype('uint8')
def bilateral_filter(img, kernel_size, sigma_s, sigma_v):
    """ Perform bilateral filtering
    Args:
        img (ndarray)
        sigma_s (float): spatial gaussian std. dev.
        sigma_v (float): value gaussian std. dev.
    Returns:
        img_corrected (ndarray)
    """
    assert len(img.shape) == 2, 'Input image has more than one channel. '
    img = img
    half_size = np.floor(kernel_size/2).astype('int')
    gaussian = lambda r2, sigma: np.exp( -0.5*r2/sigma**2 )
    weight_sum = np.ones(img.shape) * np.finfo(np.float32).eps
    img_corrected  = np.ones(img.shape) * np.finfo(np.float32).eps
    for shft_x in range(-half_size,half_size+1):
        for shft_y in range(-half_size,half_size+1):
            w = gaussian(shft_x**2+shft_y**2, sigma_s)
            off = np.roll(img, [shft_y, shft_x], axis=[0,1])
            tw = w * gaussian((off-img)**2, sigma_v)
            img_corrected += off * tw
            weight_sum += tw
    img_crt = (img_corrected / weight_sum).astype('uint8')
    img_crt[img_crt > 255] = 255; img_crt[img_crt < 0] = 0
    return img_crt
```

Code 8: Function: $median\_filter(img, kernel\_size)$ and $bilateral\_filter(img, kernel\_size, sigma\_s, sigma\_v)$

5.3 img_io.py

```python
import cv2
from pathlib import Path
from matplotlib import pyplot as plt

def read_img(filename, is_gray):
    """
    Args:
        filename (str)
        is_gray (bool)
    Returns:
        img (ndarray): with shape (height, width) for gray / (height, width, channel) for BGR
    """
    img = cv2.imread(filename)
    if is_gray:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img


def save_img(img, output_file):
    """ Output PNG file for the image
    Args:
        img (ndarray)
        output_file (str): without file extension
    """
    Path(output_file).parent.mkdir(parents=True, exist_ok=True)
    cv2.imwrite(output_file+'.png', img)


def display_hist(count):
    """ Given count of each value, display the histogram via bar chart
    Args:
        count (ndarray of ints): length is 256
    """
    plt.bar(range(256), count, width=1.0)
    plt.xlabel("Values")
    plt.ylabel("Frequency")
    plt.title("Histogram")
    plt.show()
    plt.close()
```

Code 9: Function: $read\_img(filename, is\_gray)$, $save\_img(img, output\_file)$ and $display\_hist(count)$