Image Process HW 1

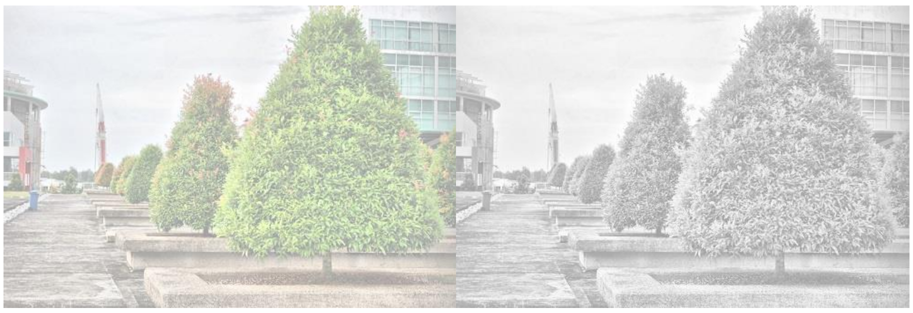
● Experiment Introduce, Result and Observation

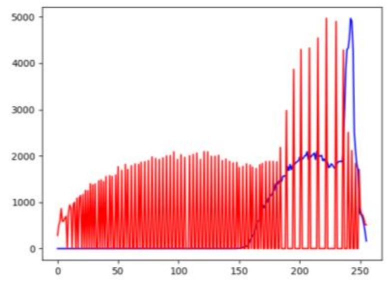
This report consists of 4 parts: HE & AHE, morphology, filter operations, color filter.

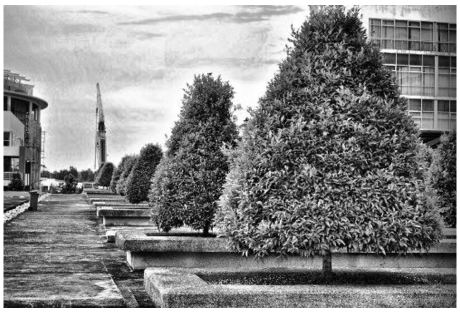
1. HE, AHE

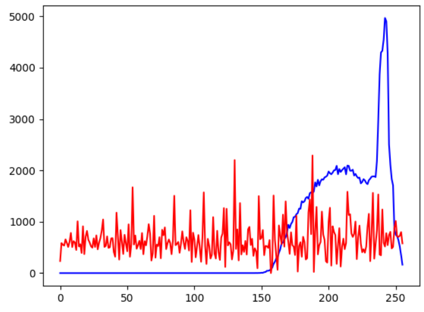
HE and AHE are used to deal with overexposed or underexposed image by making the intensity of image more balanced. The following line chart shows the amount of different intensity of the resource and result image of HE. Blue one means resource image, and the other is result. We can observe that HE rearranges gray level value into [0,255].

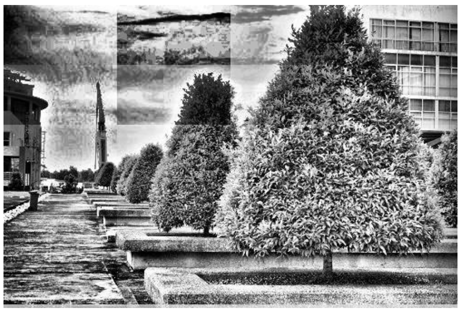
The different between them is implement region: the former does equalization among the whole image and the latter do this per block. Because AHE implement upon image blocks, it sometimes generates significant boundary between each block. The other effect of AHE is showing details in each block clearer.



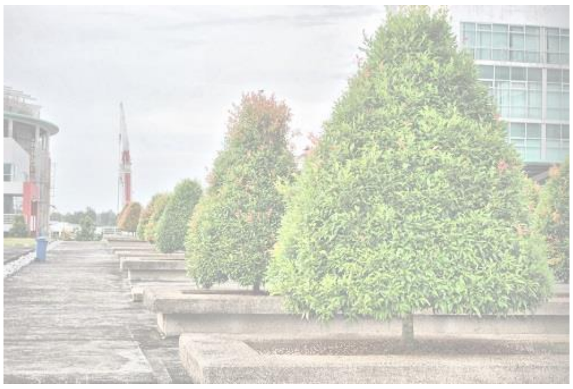


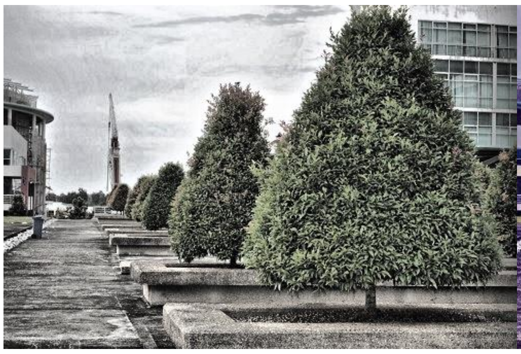
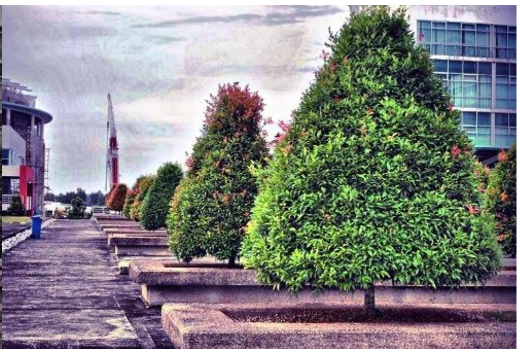


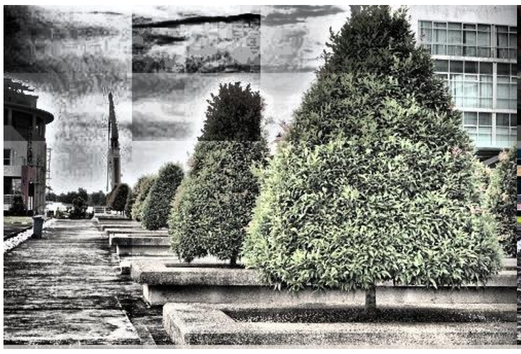
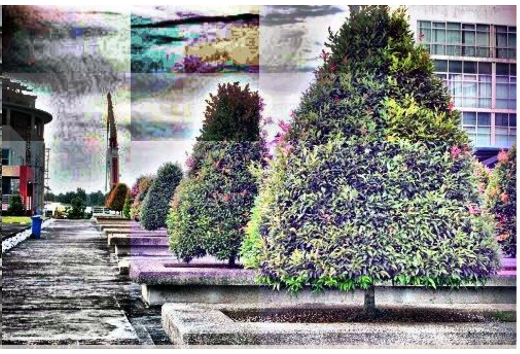




Histogram equalization should be implemented in intensity channel. When the resource image is RGB, it should be transformed into HIS or other color space which has separately intensity channel, instead of doing equalization in RGB channels directly. Doing RGB equalization will generate an image with deviation color.



2. Erosion, Dilation, Opening, Closing

Erosion and Dilation are kinds of mask operation. Erosion is used for noise removing, and the latter is used for making object more visible and filling small hole in image.

Erosion will remove the pixel if its neighborhoods are not totally match the mask, which makes the lines in the image thinner and deletes small blocks that regarded as noises. Dilation will keep or add pixel into the image if one of its neighborhoods match the mask, and it'll make the object bigger than original and fill small hole in the object.

Closing and Opening are the combination of erosion and dilation. Closing is doing dilation first, and the other is doing erosion first.

0	255	0
255	255	255
0	255	0

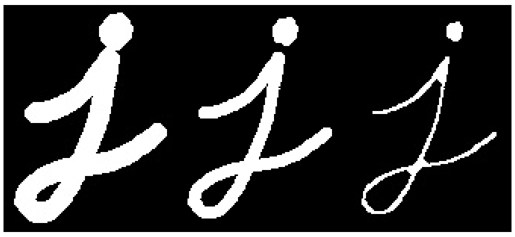


Figure 1 the mask used in dilation and erosion; dilation 3times; original image; erosion 3 times

The main part of the image is white, so adding salt noise only will not generate holes. When using erosion 1 time, it can erase the noise which size is one pixel. Bigger kernel or doing erosion more times can erase bigger noise but also erode majority. After erosion, using dilation can fix the object, the result will almost be same as original.

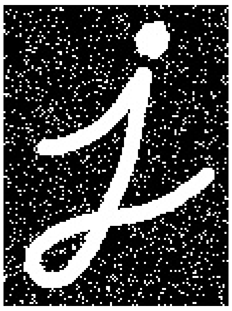


Figure 2 Image adding salt noise

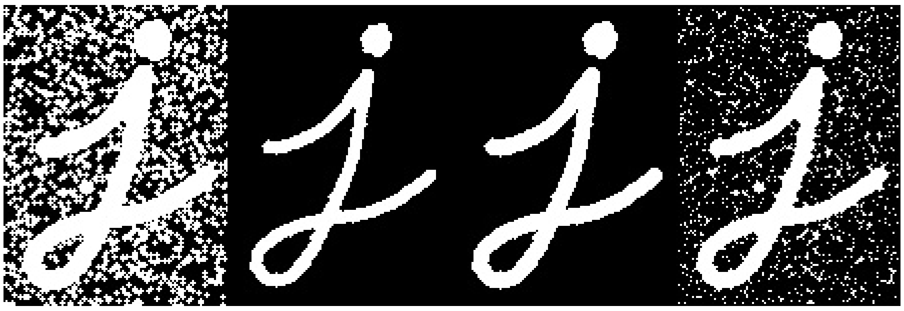


Figure 3 doing 1-time dilation; erosion; opening; closing; opening

If we add salt and pepper noise, there are holes in the “j”. Erosion is not only erasing white noise but also expands these holes. On the contrary, dilation fills holes but also strengthen noises.

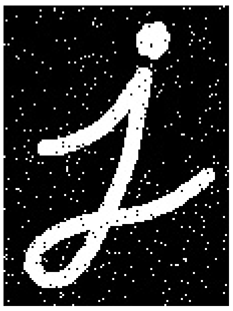


Figure 4 Image adding salt and pepper noise

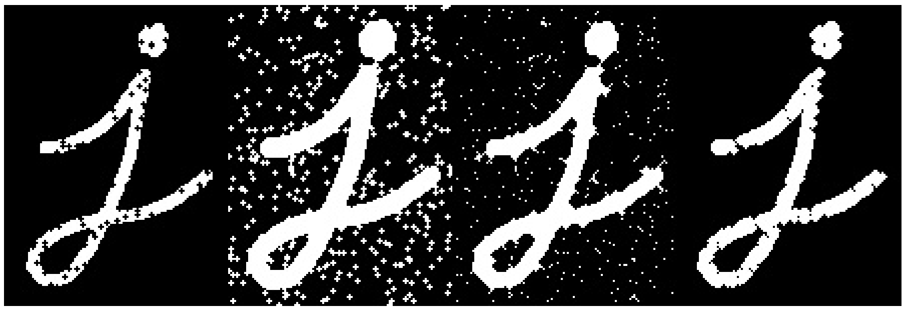


Figure 5 doing 1-time erosion; dilation; closing; opening

3. **Median filter and Average filter**

The object of these two methods is using smoothing to remove noise. The specific way is ranking the neighborhood and get the average or median to replace the target value. Giving different kernel size will bring out different results, bigger kernel has smoother result but also loss more features.

When the noise of image is salt and pepper noise, it’s useful to doing median filter instead of average filter. Average filter makes everything vogue but doesn’t remove noises. Contrastly, median filter cleans the noises perfectly.

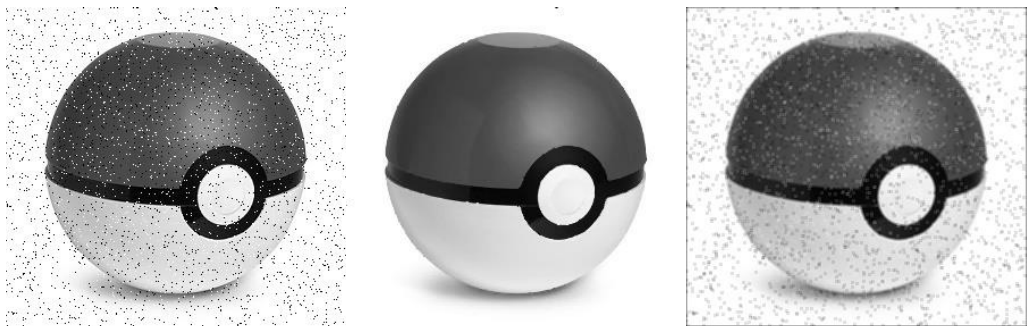


Figure 9 Left is image adding salt and pepper noise;
center is using median filter 3*3; right is using average filter 3*3;

Compare the two ways under giving same kernel size, average filter has less computation and time cost but generate more blurred result. By contrast, median filter needs more time to work and get clearer image as result.

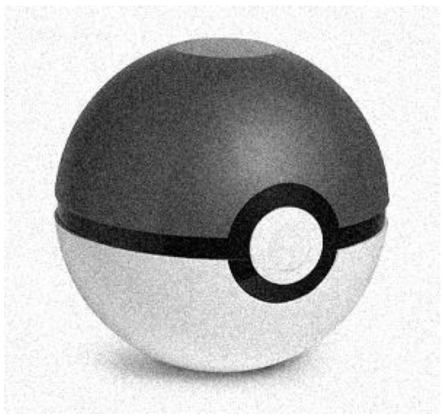


Figure 6 original image with gaussian noise

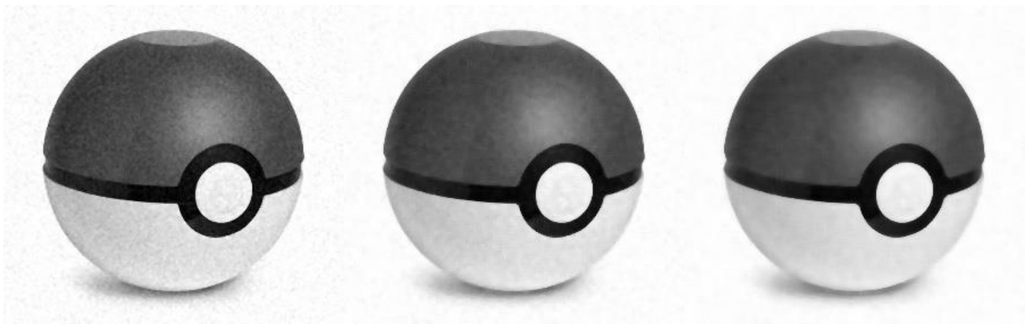


Figure 7 median filter by 3*3, 5*5, 7*7

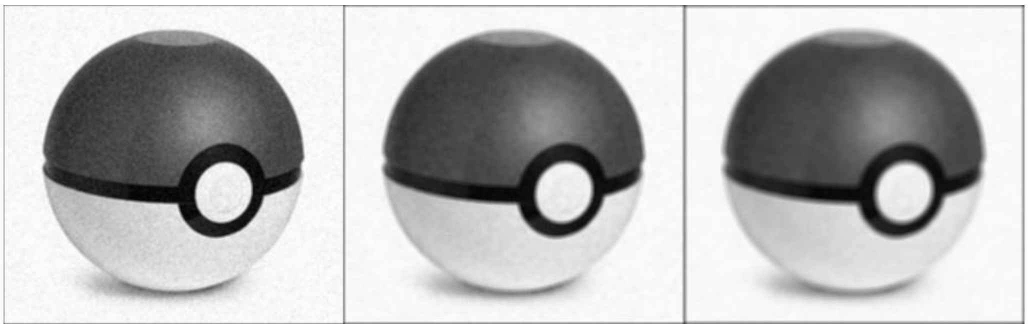


Figure 8 average filter by 3*3, 5*5, 7*7

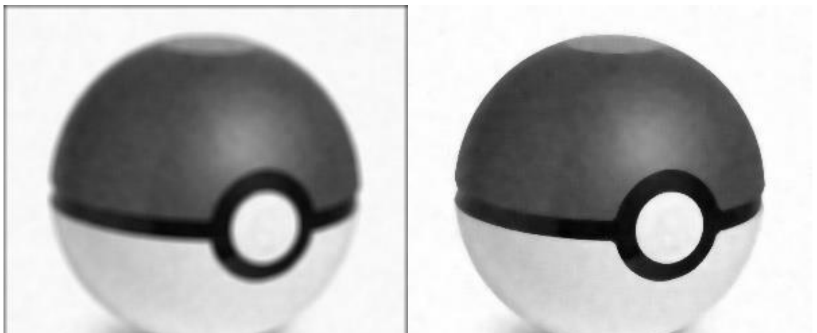


Figure 9 left is average filter by 7*7; right is median filter by 7*7

4. Color filter

This method takes advantage of HSV color space and can pick up the regions of specific color from current image.

First separating H channel into different color ranges (I get these boundary values from Internet), then we can identify whether a pixel is in the range and create a mask which marks all qualified pixels. After putting the mask onto resource image, we'll get an image which only consists the color we wanted.

Compare with doing color filter in RGB, transforming this question into HSV is more convenience, because in the HSV color space, we can directly split the color space into different color region by HSV values instead of doing additional calculate of RGB values to get color proportion.

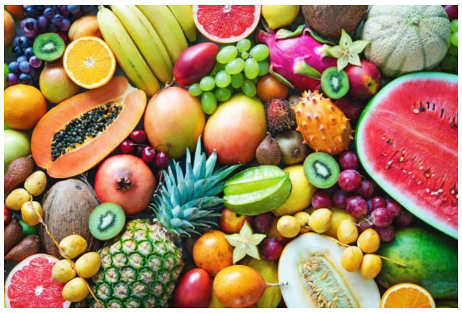


Figure 10 resource image

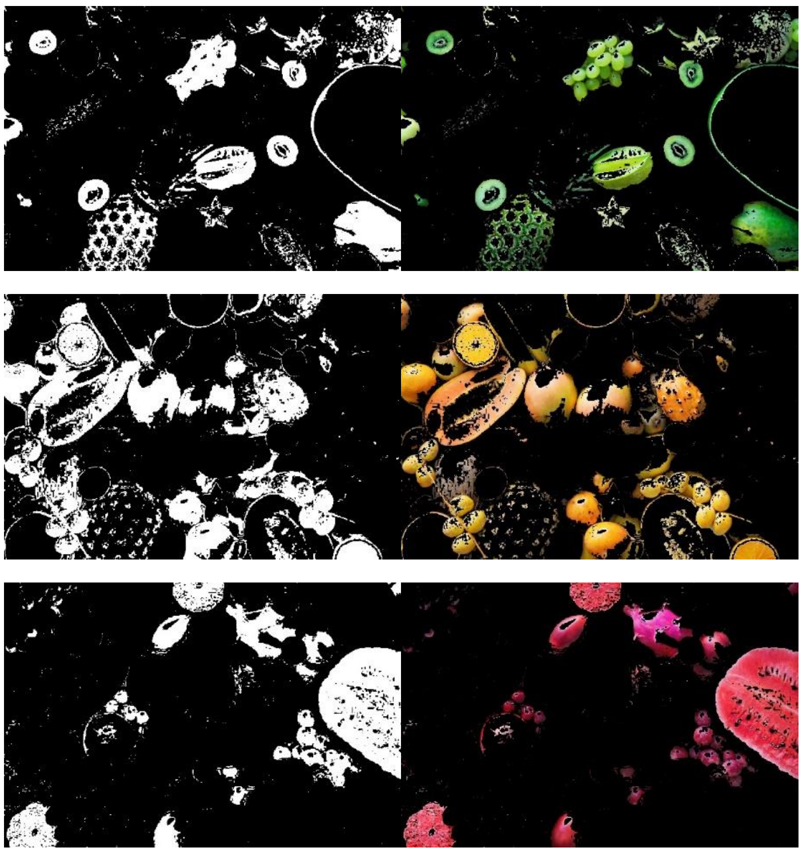


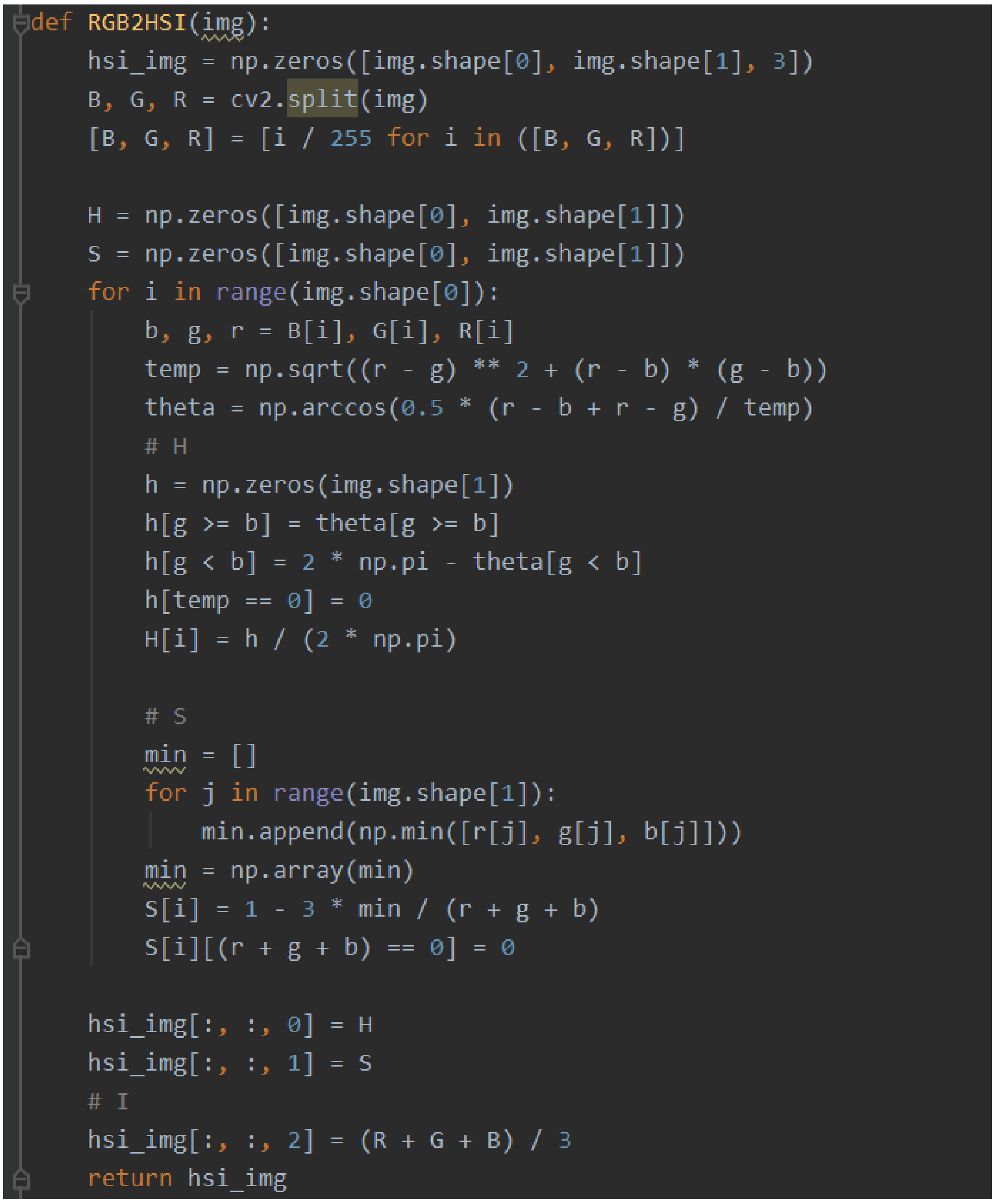
Figure 11 the mask and the result of color filter

● Assumption

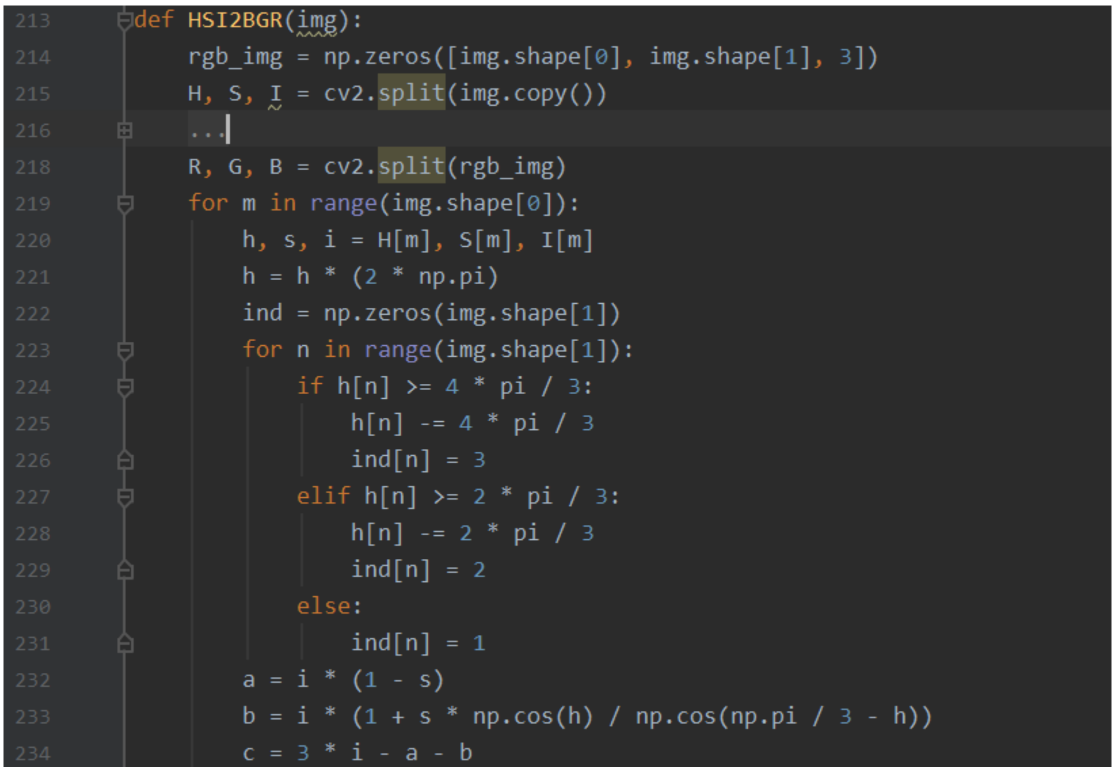
Traditional skills are truly the basic of image process. After doing these experiments I deeply understand how them work. Todays, most of solutions in image related field are using deep learning, but I think these traditional ways are still helpful and easily to implement when dealing with problems, such as doing image preprocess.

- Code

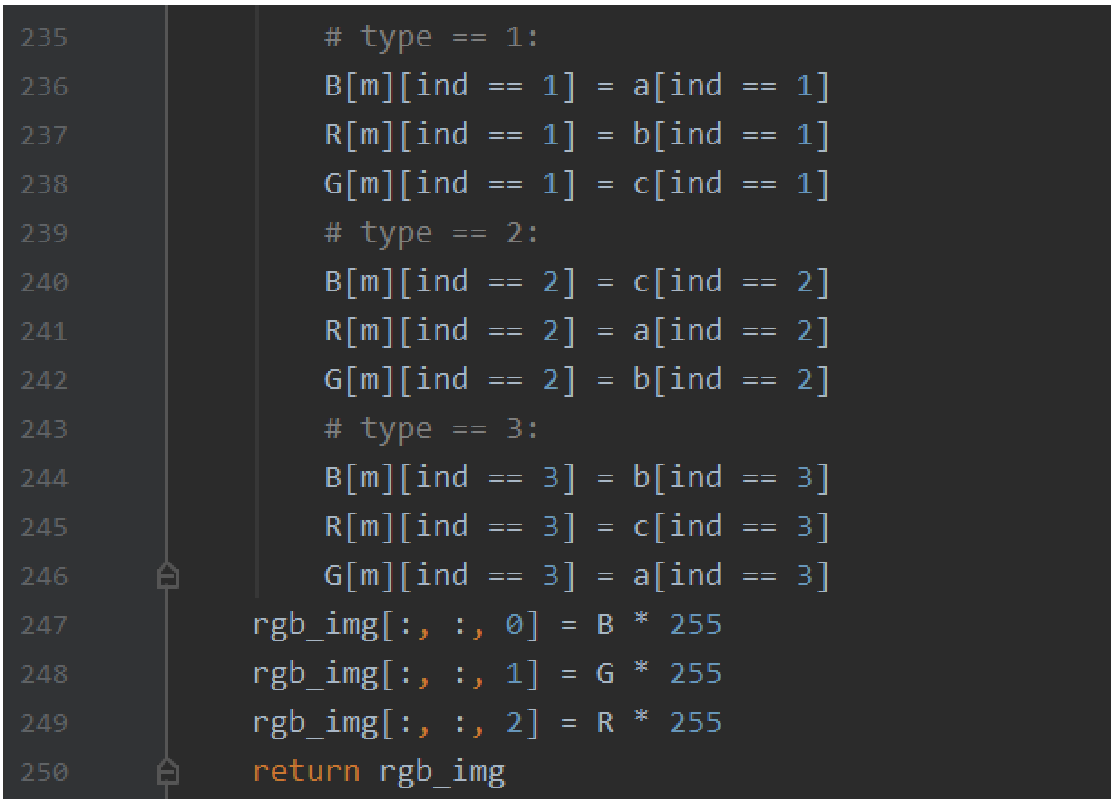
1. RGB and HSI transform



```python
def RGB2HSI(img):
    hsi_img = np.zeros([img.shape[0], img.shape[1], 3])
    B, G, R = cv2.split(img)
    [B, G, R] = [i / 255 for i in ([B, G, R])]

    H = np.zeros([img.shape[0], img.shape[1]])
    S = np.zeros([img.shape[0], img.shape[1]])
    for i in range(img.shape[0]):
        b, g, r = B[i], G[i], R[i]
        temp = np.sqrt((r - g) ** 2 + (r - b) * (g - b))
        theta = np.arccos(0.5 * (r - b + r - g) / temp)
        # H
        h = np.zeros(img.shape[1])
        h[g >= b] = theta[g >= b]
        h[g < b] = 2 * np.pi - theta[g < b]
        h[temp == 0] = 0
        H[i] = h / (2 * np.pi)

        # S
        min = []
        for j in range(img.shape[1]):
            min.append(np.min([r[j], g[j], b[j]]))
        min = np.array(min)
        S[i] = 1 - 3 * min / (r + g + b)
        S[i][(r + g + b) == 0] = 0

    hsi_img[:, :, 0] = H
    hsi_img[:, :, 1] = S
    # I
    hsi_img[:, :, 2] = (R + G + B) / 3
    return hsi_img
```

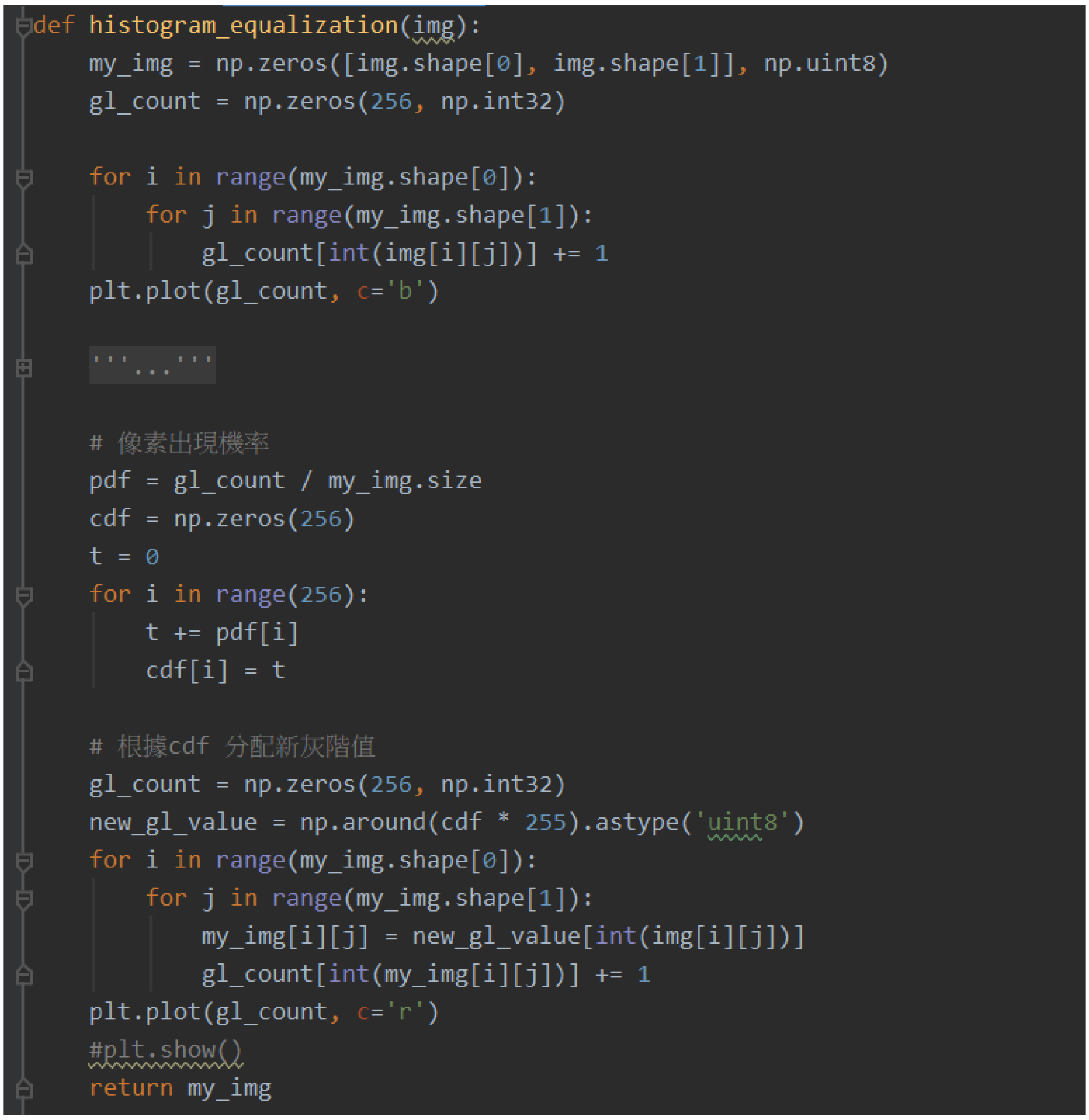


```
def HSI2BGR(img):
    rgb_img = np.zeros([img.shape[0], img.shape[1], 3])
    H, S, I = cv2.split(img.copy())
    ...
    R, G, B = cv2.split(rgb_img)
    for m in range(img.shape[0]):
        h, s, i = H[m], S[m], I[m]
        h = h * (2 * np.pi)
        ind = np.zeros(img.shape[1])
        for n in range(img.shape[1]):
            if h[n] >= 4 * pi / 3:
                h[n] -= 4 * pi / 3
                ind[n] = 3
            elif h[n] >= 2 * pi / 3:
                h[n] -= 2 * pi / 3
                ind[n] = 2
            else:
                ind[n] = 1
        a = i * (1 - s)
        b = i * (1 + s * np.cos(h) / np.cos(np.pi / 3 - h))
        c = 3 * i - a - b
```

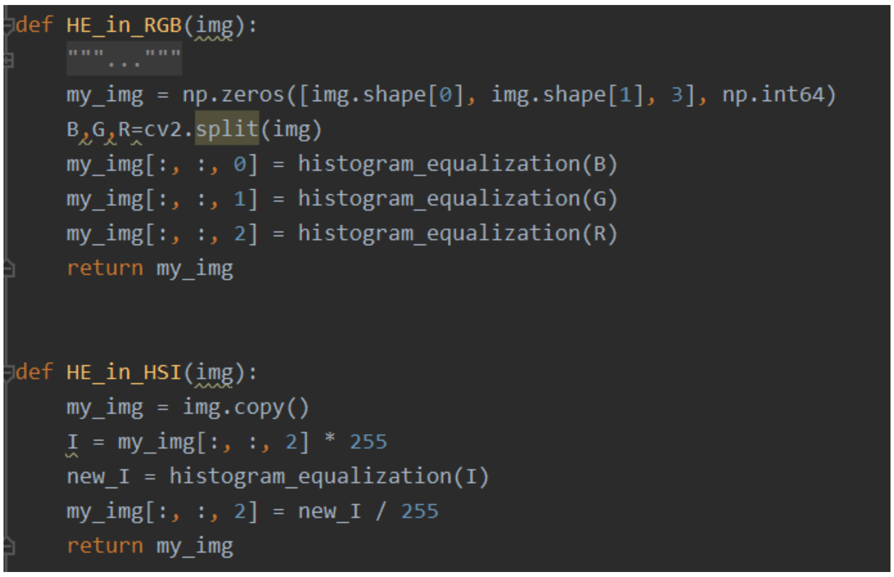


```
        # type == 1:
        B[m][ind == 1] = a[ind == 1]
        R[m][ind == 1] = b[ind == 1]
        G[m][ind == 1] = c[ind == 1]
        # type == 2:
        B[m][ind == 2] = c[ind == 2]
        R[m][ind == 2] = a[ind == 2]
        G[m][ind == 2] = b[ind == 2]
        # type == 3:
        B[m][ind == 3] = b[ind == 3]
        R[m][ind == 3] = c[ind == 3]
        G[m][ind == 3] = a[ind == 3]
    rgb_img[:, :, 0] = B * 255
    rgb_img[:, :, 1] = G * 255
    rgb_img[:, :, 2] = R * 255
    return rgb_img
```

2. HE

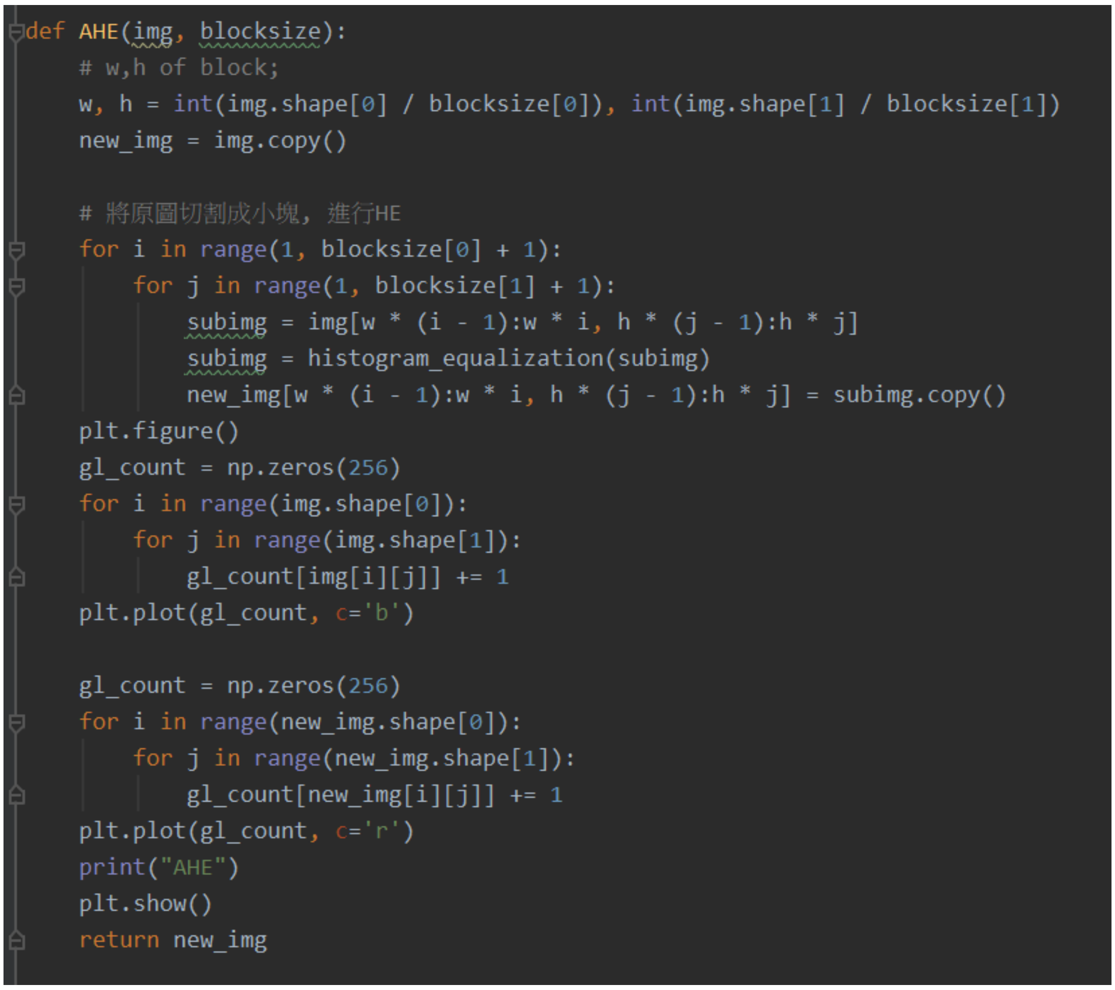


```
def histogram_equalization(img):
    my_img = np.zeros([img.shape[0], img.shape[1]], np.uint8)
    gl_count = np.zeros(256, np.int32)

    for i in range(my_img.shape[0]):
        for j in range(my_img.shape[1]):
            gl_count[int(img[i][j])] += 1
    plt.plot(gl_count, c='b')

    '''...'''

    # 像素出現機率
    pdf = gl_count / my_img.size
    cdf = np.zeros(256)
    t = 0
    for i in range(256):
        t += pdf[i]
        cdf[i] = t

    # 根據cdf 分配新灰階值
    gl_count = np.zeros(256, np.int32)
    new_gl_value = np.around(cdf * 255).astype('uint8')
    for i in range(my_img.shape[0]):
        for j in range(my_img.shape[1]):
            my_img[i][j] = new_gl_value[int(img[i][j])]
            gl_count[int(my_img[i][j])] += 1
    plt.plot(gl_count, c='r')
    #plt.show()
    return my_img
```

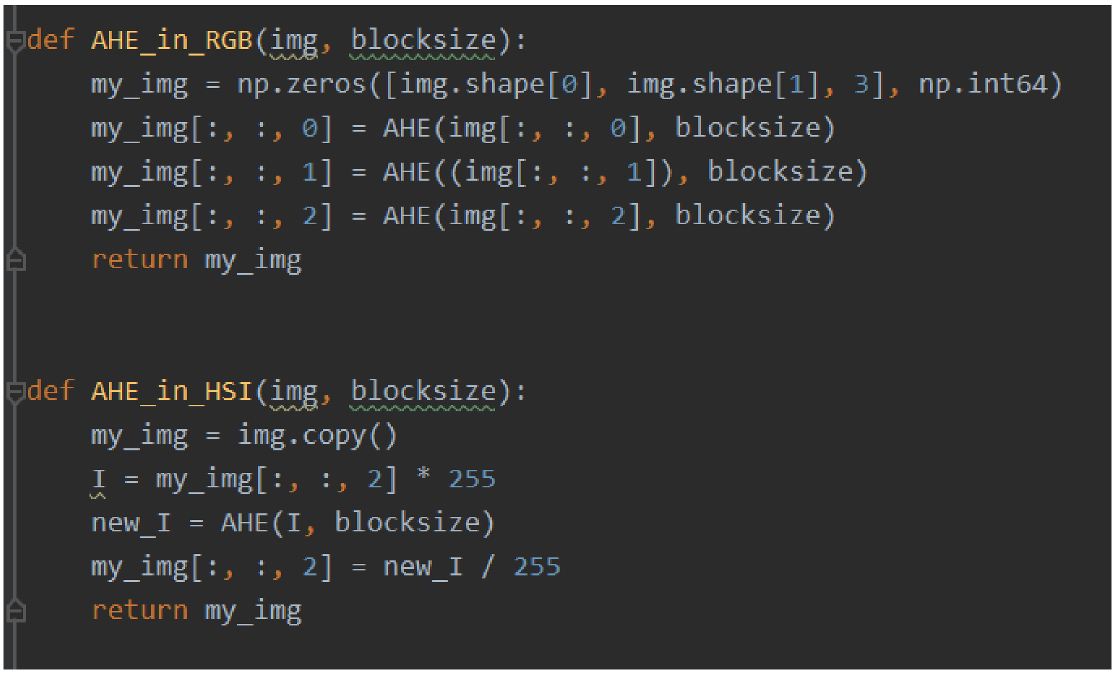


```
def HE_in_RGB(img):
    """..."""
    my_img = np.zeros([img.shape[0], img.shape[1], 3], np.int64)
    B,G,R=cv2.split(img)
    my_img[:, :, 0] = histogram_equalization(B)
    my_img[:, :, 1] = histogram_equalization(G)
    my_img[:, :, 2] = histogram_equalization(R)
    return my_img


def HE_in_HSI(img):
    my_img = img.copy()
    I = my_img[:, :, 2] * 255
    new_I = histogram_equalization(I)
    my_img[:, :, 2] = new_I / 255
    return my_img
```

3. AHE

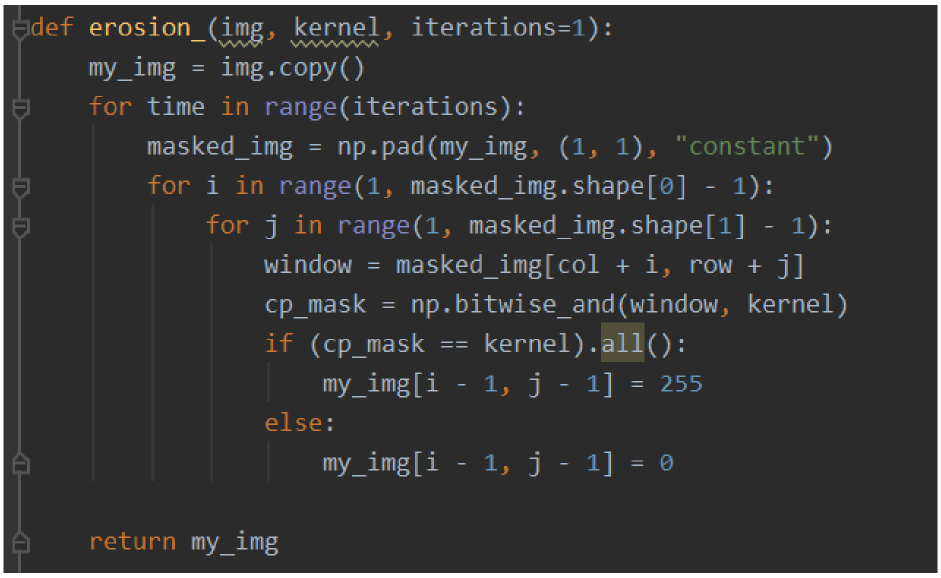


```
def AHE(img, blocksize):
    # w,h of block;
    w, h = int(img.shape[0] / blocksize[0]), int(img.shape[1] / blocksize[1])
    new_img = img.copy()

    # 將原圖切割成小塊, 進行HE
    for i in range(1, blocksize[0] + 1):
        for j in range(1, blocksize[1] + 1):
            subimg = img[w * (i - 1):w * i, h * (j - 1):h * j]
            subimg = histogram_equalization(subimg)
            new_img[w * (i - 1):w * i, h * (j - 1):h * j] = subimg.copy()
    plt.figure()
    gl_count = np.zeros(256)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            gl_count[img[i][j]] += 1
    plt.plot(gl_count, c='b')

    gl_count = np.zeros(256)
    for i in range(new_img.shape[0]):
        for j in range(new_img.shape[1]):
            gl_count[new_img[i][j]] += 1
    plt.plot(gl_count, c='r')
    print("AHE")
    plt.show()
    return new_img
```



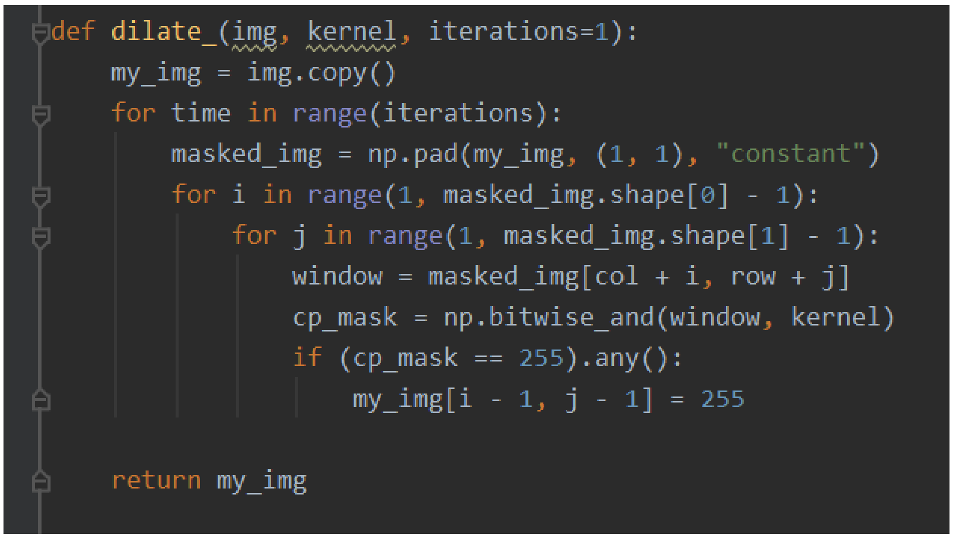
```
def AHE_in_RGB(img, blocksize):
    my_img = np.zeros([img.shape[0], img.shape[1], 3], np.int64)
    my_img[:, :, 0] = AHE(img[:, :, 0], blocksize)
    my_img[:, :, 1] = AHE((img[:, :, 1]), blocksize)
    my_img[:, :, 2] = AHE(img[:, :, 2], blocksize)
    return my_img


def AHE_in_HSI(img, blocksize):
    my_img = img.copy()
    I = my_img[:, :, 2] * 255
    new_I = AHE(I, blocksize)
    my_img[:, :, 2] = new_I / 255
    return my_img
```

4. Erosion



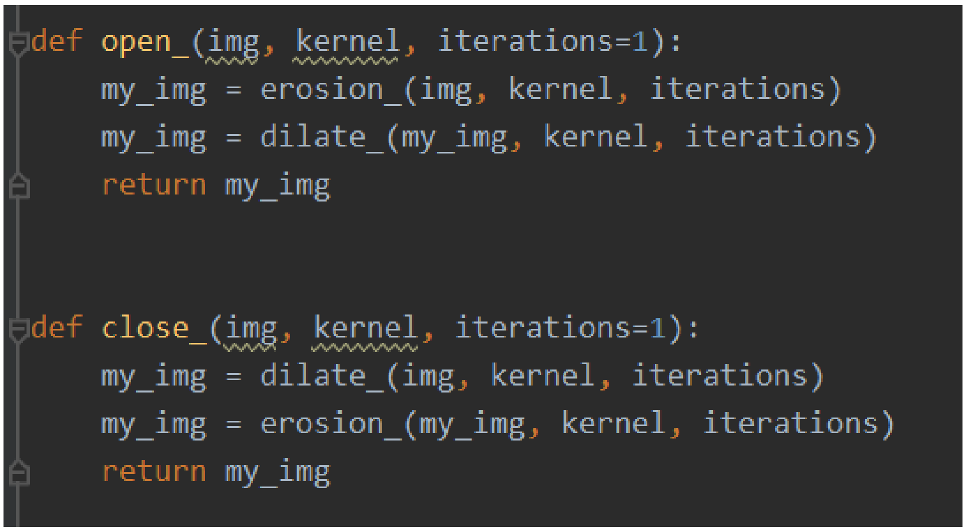
```python
def erosion_(img, kernel, iterations=1):
    my_img = img.copy()
    for time in range(iterations):
        masked_img = np.pad(my_img, (1, 1), "constant")
        for i in range(1, masked_img.shape[0] - 1):
            for j in range(1, masked_img.shape[1] - 1):
                window = masked_img[col + i, row + j]
                cp_mask = np.bitwise_and(window, kernel)
                if (cp_mask == kernel).all():
                    my_img[i - 1, j - 1] = 255
                else:
                    my_img[i - 1, j - 1] = 0

    return my_img
```

5. Dilation



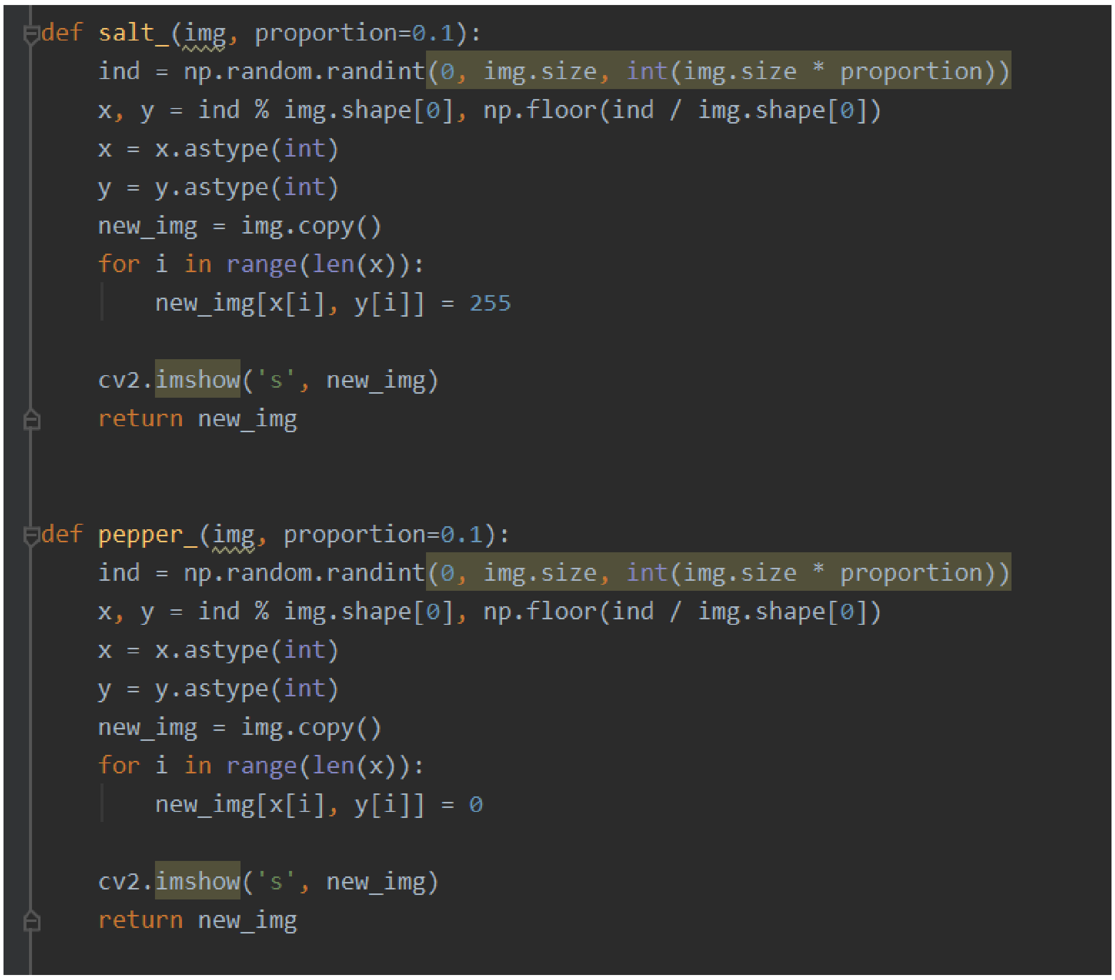
```python
def dilate_(img, kernel, iterations=1):
    my_img = img.copy()
    for time in range(iterations):
        masked_img = np.pad(my_img, (1, 1), "constant")
        for i in range(1, masked_img.shape[0] - 1):
            for j in range(1, masked_img.shape[1] - 1):
                window = masked_img[col + i, row + j]
                cp_mask = np.bitwise_and(window, kernel)
                if (cp_mask == 255).any():
                    my_img[i - 1, j - 1] = 255

    return my_img
```

6. Opening and Closing



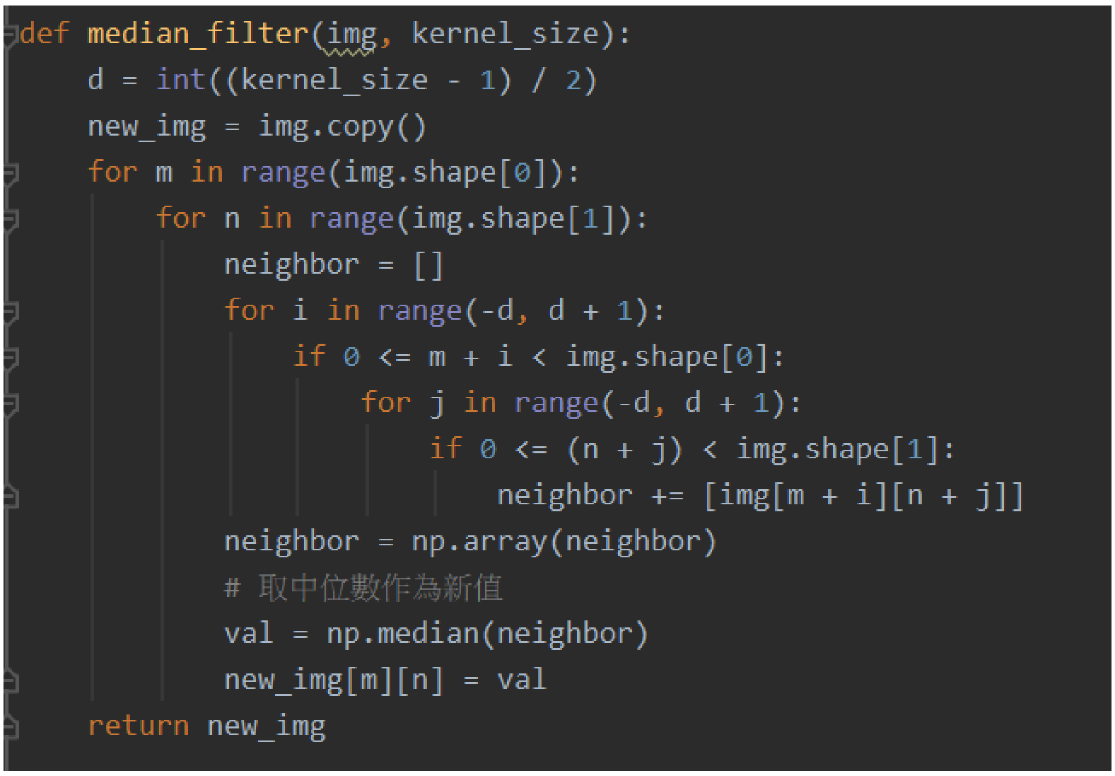
```python
def open_(img, kernel, iterations=1):
    my_img = erosion_(img, kernel, iterations)
    my_img = dilate_(my_img, kernel, iterations)
    return my_img


def close_(img, kernel, iterations=1):
    my_img = dilate_(img, kernel, iterations)
    my_img = erosion_(my_img, kernel, iterations)
    return my_img
```

7. Salt and pepper noise



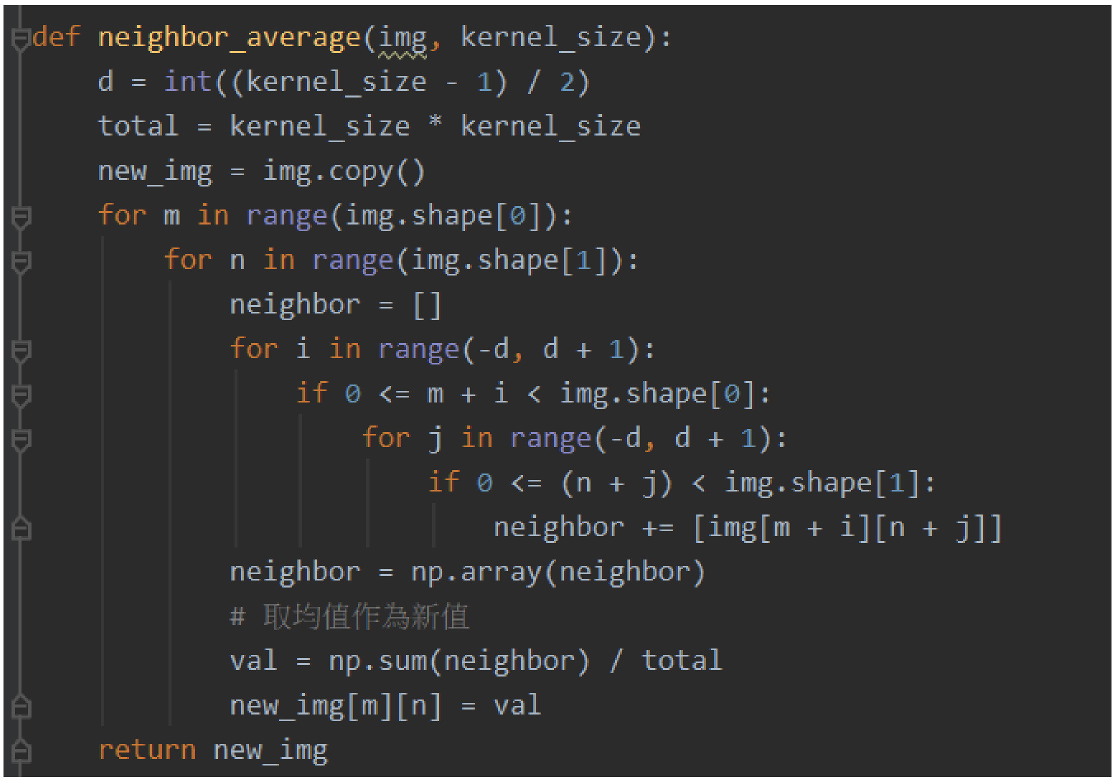
```python
def salt_(img, proportion=0.1):
    ind = np.random.randint(0, img.size, int(img.size * proportion))
    x, y = ind % img.shape[0], np.floor(ind / img.shape[0])
    x = x.astype(int)
    y = y.astype(int)
    new_img = img.copy()
    for i in range(len(x)):
        new_img[x[i], y[i]] = 255

    cv2.imshow('s', new_img)
    return new_img


def pepper_(img, proportion=0.1):
    ind = np.random.randint(0, img.size, int(img.size * proportion))
    x, y = ind % img.shape[0], np.floor(ind / img.shape[0])
    x = x.astype(int)
    y = y.astype(int)
    new_img = img.copy()
    for i in range(len(x)):
        new_img[x[i], y[i]] = 0

    cv2.imshow('s', new_img)
    return new_img
```

8. Median filter



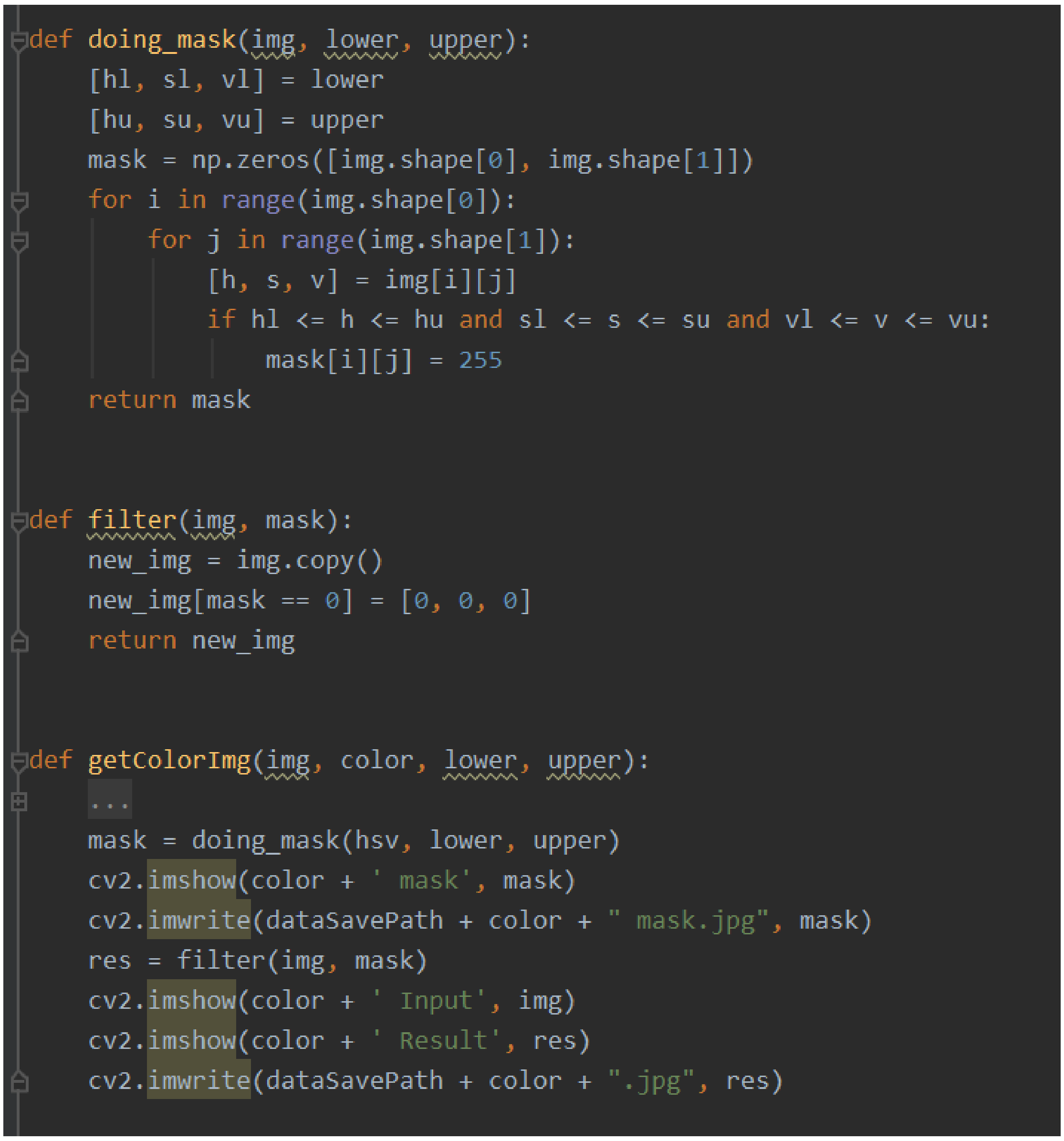
```python
def median_filter(img, kernel_size):
    d = int((kernel_size - 1) / 2)
    new_img = img.copy()
    for m in range(img.shape[0]):
        for n in range(img.shape[1]):
            neighbor = []
            for i in range(-d, d + 1):
                if 0 <= m + i < img.shape[0]:
                    for j in range(-d, d + 1):
                        if 0 <= (n + j) < img.shape[1]:
                            neighbor += [img[m + i][n + j]]
            neighbor = np.array(neighbor)
            # 取中位數作為新值
            val = np.median(neighbor)
            new_img[m][n] = val
    return new_img
```

9. Average filter

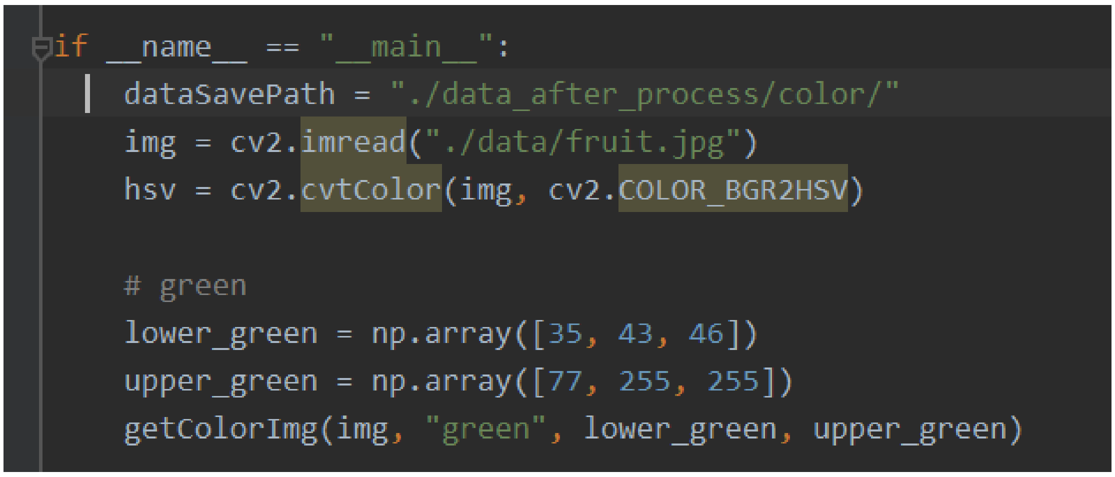


```python
def neighbor_average(img, kernel_size):
    d = int((kernel_size - 1) / 2)
    total = kernel_size * kernel_size
    new_img = img.copy()
    for m in range(img.shape[0]):
        for n in range(img.shape[1]):
            neighbor = []
            for i in range(-d, d + 1):
                if 0 <= m + i < img.shape[0]:
                    for j in range(-d, d + 1):
                        if 0 <= (n + j) < img.shape[1]:
                            neighbor += [img[m + i][n + j]]
            neighbor = np.array(neighbor)
            # 取均值作為新值
            val = np.sum(neighbor) / total
            new_img[m][n] = val
    return new_img
```

10. Color filter



```python
def doing_mask(img, lower, upper):
    [hl, sl, vl] = lower
    [hu, su, vu] = upper
    mask = np.zeros([img.shape[0], img.shape[1]])
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            [h, s, v] = img[i][j]
            if hl <= h <= hu and sl <= s <= su and vl <= v <= vu:
                mask[i][j] = 255
    return mask


def filter(img, mask):
    new_img = img.copy()
    new_img[mask == 0] = [0, 0, 0]
    return new_img


def getColorImg(img, color, lower, upper):
    ...
    mask = doing_mask(hsv, lower, upper)
    cv2.imshow(color + ' mask', mask)
    cv2.imwrite(dataSavePath + color + " mask.jpg", mask)
    res = filter(img, mask)
    cv2.imshow(color + ' Input', img)
    cv2.imshow(color + ' Result', res)
    cv2.imwrite(dataSavePath + color + ".jpg", res)
```



```python
if __name__ == "__main__":
    dataSavePath = "./data_after_process/color/"
    img = cv2.imread("./data/fruit.jpg")
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # green
    lower_green = np.array([35, 43, 46])
    upper_green = np.array([77, 255, 255])
    getColorImg(img, "green", lower_green, upper_green)
```