Image processing HW1

311605011 黃品振

1. Introduction

這次的作業是透過不同對圖片的處理來觀察結果，我這次所安排的實驗如下所示:

1. 對一個灰階圖隨機加入噪聲之後，分別透過三種降噪方式來觀察結果(median filter、Gaussian filter、mean filter)。
2. 對二彩色圖做 histogram equalization，比較不同圖片之間的差異。
3. 將一圖片在 RGB 以及 gray scale 分別做 histogram equalization，比較在彩色以及灰階上操作的結果。
4. 對兩張不同圖片套用 Laplacian filter，比較結果。

2. Review of methods I used

1. Gaussian filter:

高斯模糊，一種將圖像平滑化的技巧，也可用於降噪。原理是透過權重 mask 乘上對應像素點的值來改變中間像素值的大小。而權重的分配與像素之間的遠近有關，越遠的權重越低，代表他和這個像素的關聯性較低。

2. Median filter:

中值濾波器的作法是將 Mask 遮到的數據先做由小到大的排序(sort)接著直接對此數列取中值（Median）再取代中間數據。

3. Mean filter:

均值濾波的做法是取得 mask 之像素矩陣後加總除以總數(計算平均值)。

mask 如下為例:

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

4. Histogram Equalization:
這是一種增加對比度的方式，目的是希望可以透過一個函數，將原圖的 histogram 可以近似於 uniform distribution，均衡各種像素值的數量，來讓圖片的品質更好。

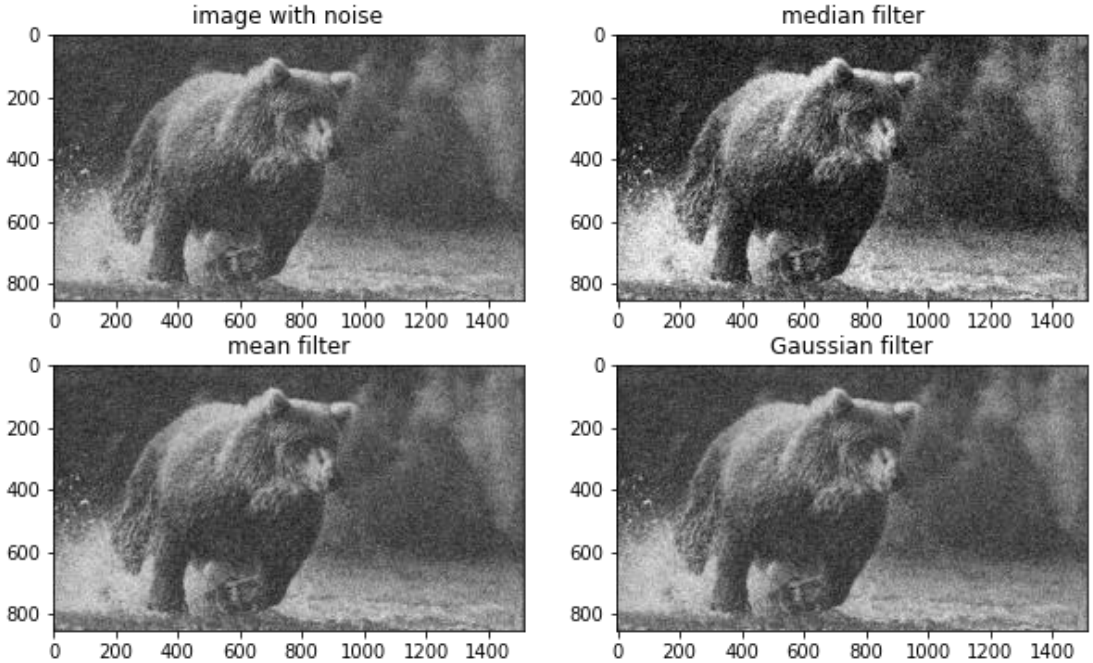
3. Experiments and results

1. 實驗一

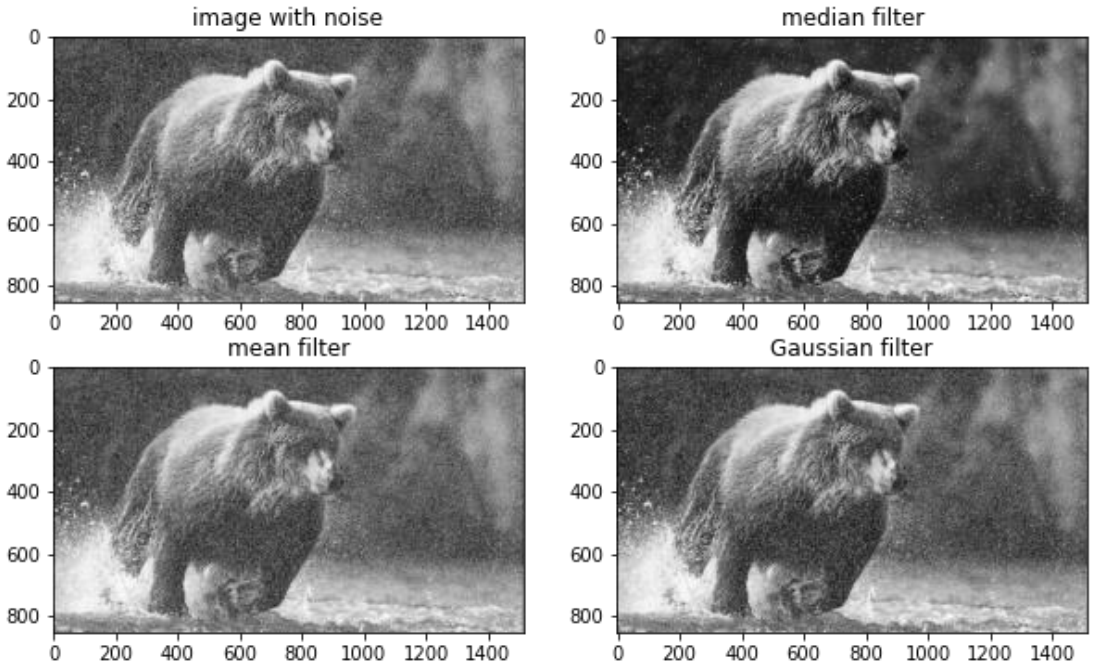
我在第一個實驗是對一個灰階圖隨機加入高斯噪聲以及隨機白噪之後，分別透過三種降噪濾波的方式來觀察結果(median filter、Gaussian filter、mean filter)。

原圖:

Gaussian noise:



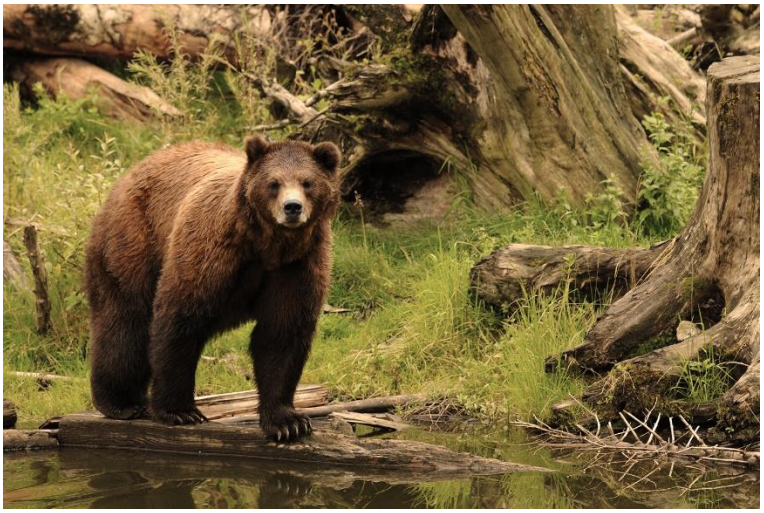
Random white noise:



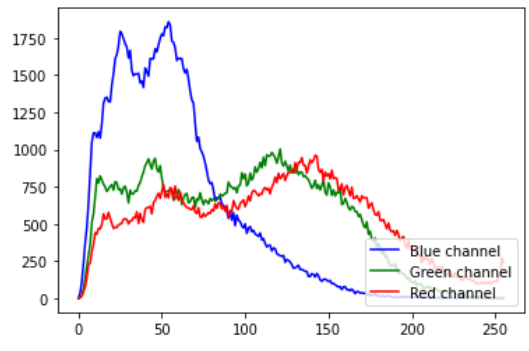
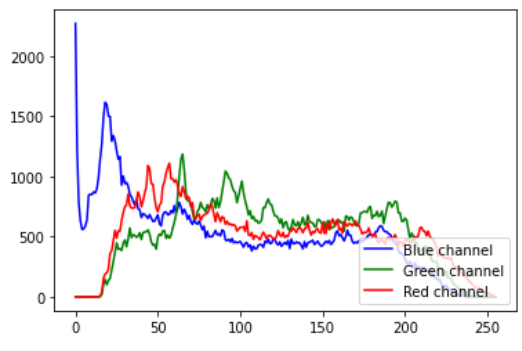
2. 實驗二

對二彩色圖做 histogram equalization，比較不同圖片之間的差異。

Origin images:

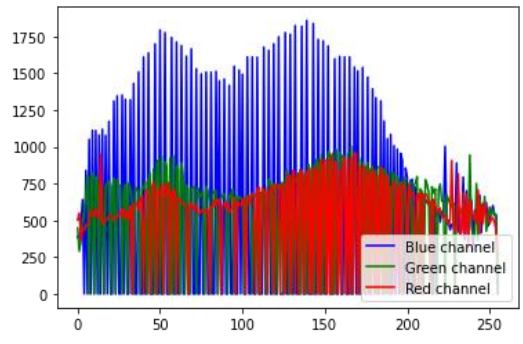
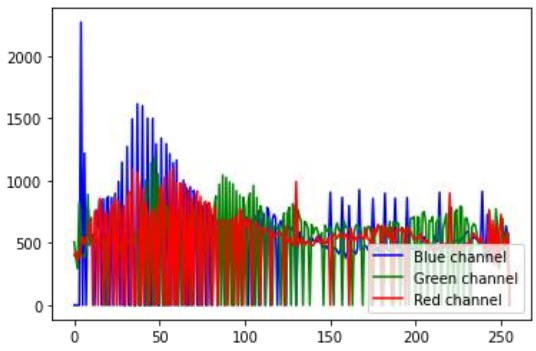


Origin histogram:

bear	turtle
	

Images after equalization:

Histogram after equalization:

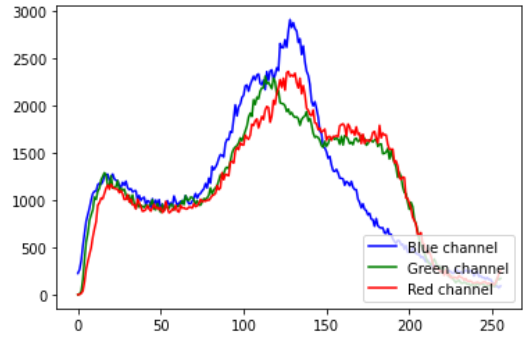
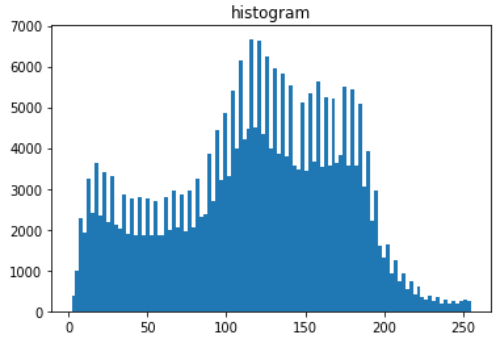
bear	turtle
	

3. 實驗三

將一圖片在 RGB 以及 gray scale 分別做 histogram equalization，比較在彩色以及灰階上操作的結果

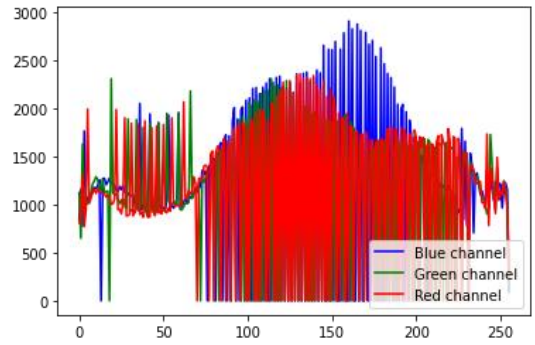
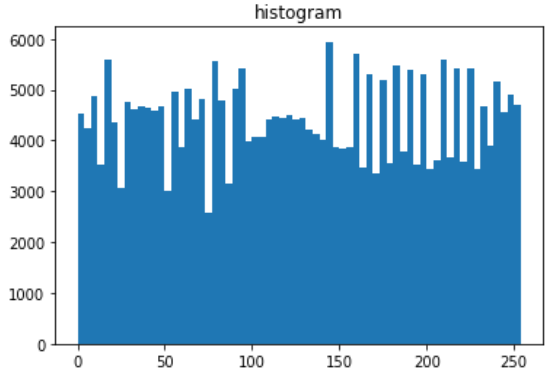
Origin images


Origin histogram

RGB	Gray scale
	

Images after equalization


Histogram after equalization

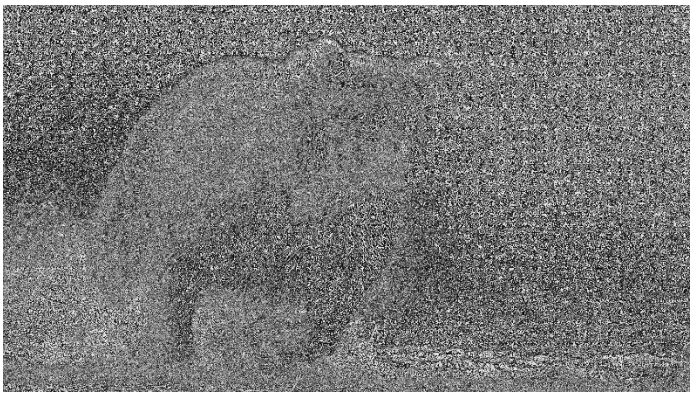
RGB	Gray scale
	

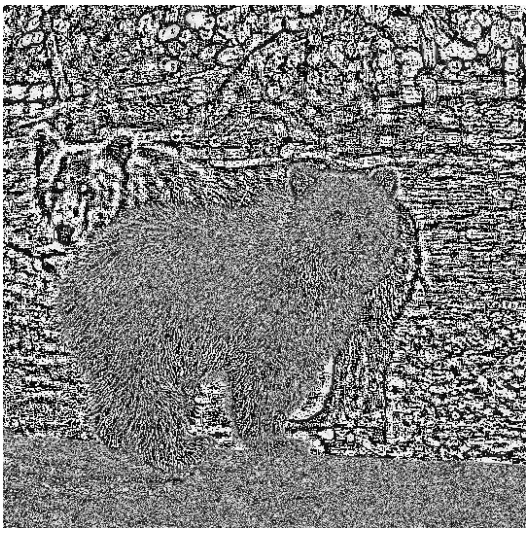
4. 實驗四

對兩張不同圖片套用 Laplacian filter，比較結果

Origin images

Images after Laplacian filter





4. Discussion

- 實驗一

  在加入高斯噪聲之後，整體而言圖片會變得比較模糊，圖片上看起來也會有一些顆粒感。比較三種 filter，可以發現 median filter 可以把物體整體的明暗重現出來，但是噪點也同樣變得更明顯了，而 mean filter 與 Gaussian filter 的效果接近，但是 Gaussian filter 的效果又再好一點，比較不會失去圖像細節。會有這樣的結果推測是因為 Gaussian filter 是 mean filter 的變形，會比較注重像素之間的空間關係，在平滑化的同時也可以不失去圖像細節。

  在加入隨機白噪之後，觀察三種 filter 的表現，可以發現 medain filter 對於白噪聲的處理相當好，幾乎還原回原圖了，而 mean filter 與 Gaussian filter 的表現則不理想，推測是因為像素與雜訊之間的數值差異

太大，取均值或是加權總合後仍被視為雜訊，而 median filter 是改取中值代替，可以有效改善雜訊過於唐突的情況。

- 實驗二

  比較兩張圖片在 histogram equalization 之後的結果，發現熊的效果更好，整體圖片有從黃昏場景變成白天的感覺，推測是因為在 equalization 後，RGB 三通道的分布比起烏龜更為均勻的關係。

- 實驗三

  將一張圖片在彩色以及灰階分別做 histogram equalization，並比較結果。從結果來看，可以發現灰階圖改善的效果更好，整體圖片變得更亮，而且分布也明顯變得更為均勻。相較起來，彩色的效果就沒那麼明顯，推測是因為灰階圖只需要改善一個通道，而 RGB 需要三個通道，且最後呈現的圖會是三通道疊在一起，呈現的圖會受到通道之間的互相影響，所以效果有限。

- 實驗四

  Laplacian filter 是一種用來做邊緣檢測的 filter。實驗四的作法是將兩張圖分別套用 Laplacian filter，發現第二張圖的效果較好，推測是因為第一張圖的顏色與背景較為相近，才會導致邊緣檢測的效果較不好。

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random

def Conv(img, kernel, RGB = False):
    new_row = img.shape[0] - kernel.shape[0] + 1
    new_col = img.shape[1] - kernel.shape[0] + 1
    if RGB == True:
        new_img = np.zeros(((new_row), (new_col), 3))
        for ch in range(3):
            for i in range(int(new_row)):
                for j in range(int(new_col)):
                    value = 0
                    for ki in range(3):
                        for kj in range(3):
                            value += img[:,:,ch][i+ki][j+kj] *
kernel[ki][kj]
                            new_img[i][j] = value
    else:
        new_img = np.zeros((int(new_row), int(new_col)))
        for i in range(int(new_row)):
            for j in range(int(new_col)):
                value = 0
                for ki in range(3):
                    for kj in range(3):
                        value += img[i+ki][j+kj] * kernel[ki][kj]
                        new_img[i][j] = value
    #print(new_img.shape)
    return new_img.astype('uint8')

def make_histogram(img, RGB = False):
    if RGB == True:
        hist_plot = np.zeros((3,256))
        for ch in range(3):
            his_list = []
            for i in range(img.shape[0]):
                for j in range(img.shape[1]):
```

```
                    his_list.append(img[i][j][ch])
            his_list = np.array(his_list)
            for x in range(256):
                hist_plot[ch][x] += sum(his_list == x)

        plt.plot(range(256), hist_plot[0], c = 'b', label = 'Blue
channel')
        plt.plot(range(256), hist_plot[1], c = 'g', label = 'Green
channel')
        plt.plot(range(256), hist_plot[2], c = 'r', label = 'Red
channel')
        plt.legend(loc = 'lower right')
        plt.show()
    else:
        his_list = []
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                his_list.append(img[i][j])
        his_list = np.array(his_list)
        plt.hist(his_list, bins='auto')
        plt.title('histogram')
        plt.show()

    return 0


def hist_norm(img, RGB = False):
    if RGB == True:
        for ch in range(3):
            hist_array = np.bincount(img[:,:,ch].flatten(),
minlength=256)
            num_pixel = np.sum(hist_array)
            hist_array = hist_array / num_pixel
            chist_array = np.cumsum(hist_array)
            transform = np.floor(255 * chist_array).astype(np.uint8)
            img_list = list(img[:,:,ch].flatten())
            new_img_ch = [transform[p] for p in img_list]
            new_img_ch = np.array(new_img_ch).reshape(img.shape[0],
img.shape[1], 1)
```

```
            if ch == 0:
                B_img = new_img_ch
            if ch == 1:
                G_img = new_img_ch
            if ch == 2:
                R_img = new_img_ch
        new_img = np.concatenate((B_img, G_img, R_img),
axis=2).astype('uint8')

    if RGB == False:
        hist_array = np.bincount(img[:,:].flatten(), minlength=256)
        num_pixel = np.sum(hist_array)
        hist_array = hist_array / num_pixel
        chist_array = np.cumsum(hist_array)
        transform = np.floor(255 * chist_array).astype(np.uint8)
        img_list = list(img[:,:].flatten())
        new_img = [transform[p] for p in img_list]
        new_img = np.array(new_img).reshape(img.shape[0],
img.shape[1]).astype('uint8')

    return new_img

def median_filter(data, filter_size):
    temp = []
    indexer = filter_size // 2
    data_final = []
    data_final = np.zeros((len(data),len(data[0])))
    for i in range(len(data)):

        for j in range(len(data[0])):

            for z in range(filter_size):
                if i + z - indexer < 0 or i + z - indexer > len(data) -
1:
                    for c in range(filter_size):
                        temp.append(0)
                else:
```

```
                    if j + z - indexer < 0 or j + indexer > len(data[0])
- 1:
                        temp.append(0)
                    else:
                        for k in range(filter_size):
                            temp.append(data[i + z - indexer][j + k -
indexer])

            temp.sort()
            data_final[i][j] = temp[len(temp) // 2]
            temp = []
    return data_final.astype('uint8')


def mean_filter(data, filter_size):
    temp = []
    indexer = filter_size // 2
    data_final = []
    data_final = np.zeros((len(data),len(data[0])))
    for i in range(len(data)):

        for j in range(len(data[0])):

            for z in range(filter_size):
                if i + z - indexer < 0 or i + z - indexer > len(data) -
1:
                    for c in range(filter_size):
                        temp.append(0)
                else:
                    if j + z - indexer < 0 or j + indexer > len(data[0])
- 1:
                        temp.append(0)
                    else:
                        for k in range(filter_size):
                            temp.append(data[i + z - indexer][j + k -
indexer])

            data_final[i][j] = sum(temp)/len(temp)
            temp = []
```

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

def gaussian_noise(img, mean=0, sigma=0.3):
    #normalize
    img = img / 255.0
    noise = np.random.normal(mean, sigma, img.shape)
    gaussian_out = img + noise
    gaussian_out = np.clip(gaussian_out, 0, 1)
    gaussian_out = np.uint8(gaussian_out*255)
    noise = np.uint8(noise*255)

    return gaussian_out

def sp_noise(image,prob):
    output = np.zeros(image.shape,np.uint8)
    thres = 1 - prob
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            rdn = random.random()
            if rdn < prob:
                output[i][j] = 255
            elif rdn > thres:
                output[i][j] = 255
            else:
                output[i][j] = image[i][j]
    return output


gaussian_filter = 1/16 * np.array([[1, 2, 1],
                                   [2, 4, 2],
                                   [1, 2, 1]])


img = cv2.imread('bear.jpg')
img = cv2.resize(img,None,fx=0.3,fy=0.3,interpolation=cv2.INTER_LINEAR)
img_g = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imwrite('blackbear.png', img_g)
img_g_n = sp_noise(img_g, 0.1)
img_median = median_filter(img_g_n, 3)
```

```
img_mean = mean_filter(img_g_n, 3)
img_gaussian = Conv(img_g_n, gaussian_filter)

fig, ax = plt.subplots(2,2,figsize=(10,6))
ax[0,0].set_title('image with noise')
ax[0,0].imshow(img_g_n, cmap='gray')
ax[0,1].set_title('median filter')
ax[0,1].imshow(img_median, cmap='gray')
ax[1,0].set_title('mean filter')
ax[1,0].imshow(img_mean, cmap='gray')
ax[1,1].set_title('Gaussian filter')
ax[1,1].imshow(img_gaussian, cmap='gray')
plt.show()

#histogram
turtle = cv2.imread('turtle.jpg')
turtle =
cv2.resize(turtle,None,fx=0.3,fy=0.3,interpolation=cv2.INTER_LINEAR)
turtle = hist_norm(turtle, True)
cv2.imwrite('norm_turtle.png', turtle)
bear = cv2.imread('bear2.jpg')
bear =
cv2.resize(bear,None,fx=0.3,fy=0.3,interpolation=cv2.INTER_LINEAR)
bear = hist_norm(bear, True)
cv2.imwrite('norm_bear.png', bear)
make_histogram(turtle, True)
make_histogram(bear, True)

#實驗 3
bear2 = cv2.imread("bear3.jpg")
bear2 =
cv2.resize(bear2,None,fx=0.5,fy=0.5,interpolation=cv2.INTER_LINEAR)
bear_gray = cv2.cvtColor(bear2, cv2.COLOR_BGR2GRAY)
cv2.imwrite('gray_bear3.png', bear_gray)
make_histogram(bear2, True)
make_histogram(bear_gray)
norm_bear2 = hist_norm(bear2, True)
norm_bear_gray = hist_norm(bear_gray)
```

```
make_histogram(norm_bear2, True)
make_histogram(norm_bear_gray)
cv2.imwrite('norm_bear2.png',norm_bear2)
cv2.imwrite('norm_bear_gray.png', norm_bear_gray)

#實驗 4
laplacian = np.array([[1, 1, 1],
                      [1, -8, 1],
                      [1, 1, 1]])

lap_bear1 = Conv(img_g, laplacian)
cv2.imwrite('lapbear.png', lap_bear1)
lap_bear2 = Conv(bear_gray, laplacian)
cv2.imwrite('lapbear2.png', lap_bear2)
```