110612117 張仲瑜 HW3

I. Method

A. Spatial domain

```
74    c = 3
75    img = cv2.imread('Q1.jpg', 1)
76    cross_kernel = np.array([
77        [0, -1*c, 0],
78        [-1*c, 1+4*c, -1*c],
79        [0, -1*c, 0]
80    ])
81    # cross_kernel = np.array([
82    #     [0, 0, -1*c, 0, 0],
83    #     [0, -1*c, -2*c, -1*c, 0],
84    #     [-1*c, -2*c, 1+16*c, -2*c, -1*c],
85    #     [0, -1*c, -2*c, -1*c, 0],
86    #     [0, 0, -1*c, 0, 0]
87    # ])
```

74~87: 在函式外建立 filter，圖中為參考上下左右的 3x3 laplacian kernal，c 為 Highboost Filtering 的參數

```
10  def convovle(image, kernel):
11      image_height, image_width, num_channel = image.shape
12      kernel_height, kernel_width = kernel.shape
13      kernel = np.flipud(np.fliplr(kernel))
14      padding_height = kernel_height //2
15      padding_width = kernel_width //2
16      padded_image = np.pad(image, ((padding_height, padding_height), (padding_width, padding_width), (0 , 0)), mode='constant')
17      output_image = np.zeros_like(image)
18      for c in range(num_channel):
19          for y in range(image_height):
20              for x in range(image_width):
21                  pixel = np.sum(kernel * padded_image[y:y+kernel_height, x:x+kernel_width, c])
22                  output_image[y, x, c] = np.clip(pixel, 0, 255)
23      return output_image
24
25  def apply_spatial_domain_filter(image, image_name, kernel):
26      filtered_image = convovle(image, kernel)
27      #filtered_image = cv2.filter2D(image, -1, kernel)
28      show_image(f'{image_name} Filtered Image', filtered_image)
```

16: 將原圖在四個方向 padding 出 kernel 長(寬)1/2、數值為 0 的 Pixels

18~23:

使用迴圈對原圖每個 channel 內的 pixel 做 pixelwise 的相乘後 clipping 回 0~255，由於 13 行已將 kernel 翻轉，此處等於做 convolution

B. Frequency Domain

```
34      for c in range(num_channel):
35          channel = image[:, :, c]
36          denormMax = np.max(channel)
37          denormMin = np.min(channel)
38          channel = (channel - np.min(channel)) / (np.max(channel) - np.min(channel)) #normalize
39          f = np.fft.fft2(channel)
40          fshift = np.fft.fftshift(f)
```

34~37: 將輸入 channel 標準化

39~40: 傅立葉轉換並移到中央

```
46          cenRow, cenCol = image_height//2, image_width//2
47          filter = np.zeros_like(channel)
48          for i in range(image_height):
49              for j in range(image_width):
50                  filter[i, j] = -4 * (np.pi**2) * ((i-cenRow)**2 + (j-cenCol)**2)
```

47~50: 根據課本公式製作 Laplacian filter

```
56          Lap = fshift * filter
57          Lap_ishift = np.fft.ifftshift(Lap)
58          channel_back = np.fft.ifft2(Lap_ishift).real
59          channel_back = channel_back / np.max(channel_back)
60          new_channel = channel - k * channel_back
61          new_channel = new_channel * (denormMax - denormMin) + denormMin
62          new_channel = np.clip(new_channel, 0, 255)
63          all_channel_back.append(new_channel)
```

56~59: 相乘、移回並轉換回空間域

59~60: 將 channel back 以避免色差的方式標準化

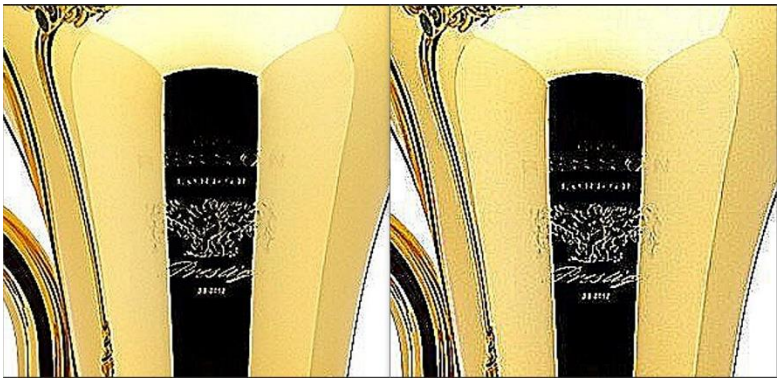
再用原圖減掉來達成銳利化

61~63: 反標準化後再 clip 回 0~255 的區間，並疊回 channel 區

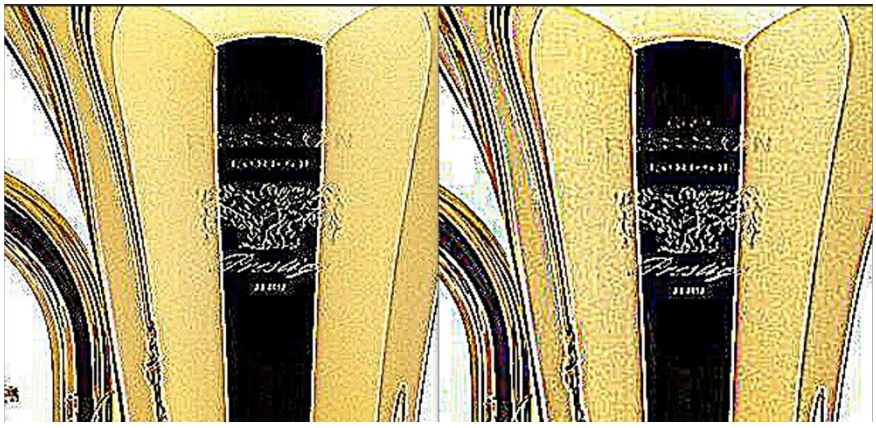
II. Result

A. 空間域中不同 kernel 類型(比對左右)與大小(比對上下)的成果

1. 3x3 cross (左) vs around (右)



2. 5x5 cross (左) vs around (右)



B. 頻率域中不同相減倍率的影響(Highboost Filtering k 值)

(左上原圖、左下 1 倍、右上 5 倍、右下 10 倍)

III. Feedback: 我選擇了具有金屬光澤的上低音號作為觀察對象，並做了關於 Laplacian sharpening 中不同 kernel 大小、類型與參數的影響。在頻率域的實作上我遇到不少的困難，包括轉換後相乘的值域調整以相減、正規化的數值紀錄順序(要先在轉換前記起來，否則要反正歸化時會遺失)等，都花費了不少時間來除錯、避免屢次出現的黑畫面。銳利化後的上低音號看起來格外逼真，銳利的效果如同時間為樂器帶來的鏽斑與撞傷，為照片增添了幾分真實感。