Hw3_Color image enhancement (Due. 6/5)

409410014 資工三 柯柏旭 (hand in 6/2)

Technical description

測試環境

```
❯ uname -a
Linux hentci-Aspire-A515-52G 5.15.0-69-generic #76-Ubuntu SMP Fri Mar 17 17:19:29 UTC 2023
 x86_64 x86_64 x86_64 GNU/Linux
```

使用語言:

`python 3.10.6`

library-requirement:

```
matplotlib==3.7.1
opencv_python==4.7.0.72
Pillow==9.5.0
```

如何執行

```
python3 color_image_enhancement.py
```

或是

```
python color_image_enhancement.py
```

便會依序顯示以下內容:

```
共4張圖, 每張圖包含 original image 和 enhance 過後的 RGB, HSI, LAB 空間的 image
1. aloe.jpg (original, RGB, HSI, LAB)
2. church.jpg (original, RGB, HSI, LAB)
3. house.jpg (original, RGB, HSI, LAB)
4. kitchen.jpg (original, RGB, HSI, LAB)
```

如果要單獨看大張圖，可以把程式碼中標記的地方解註解後執行。

也另外實做了UI版本的程式，可以輸入:

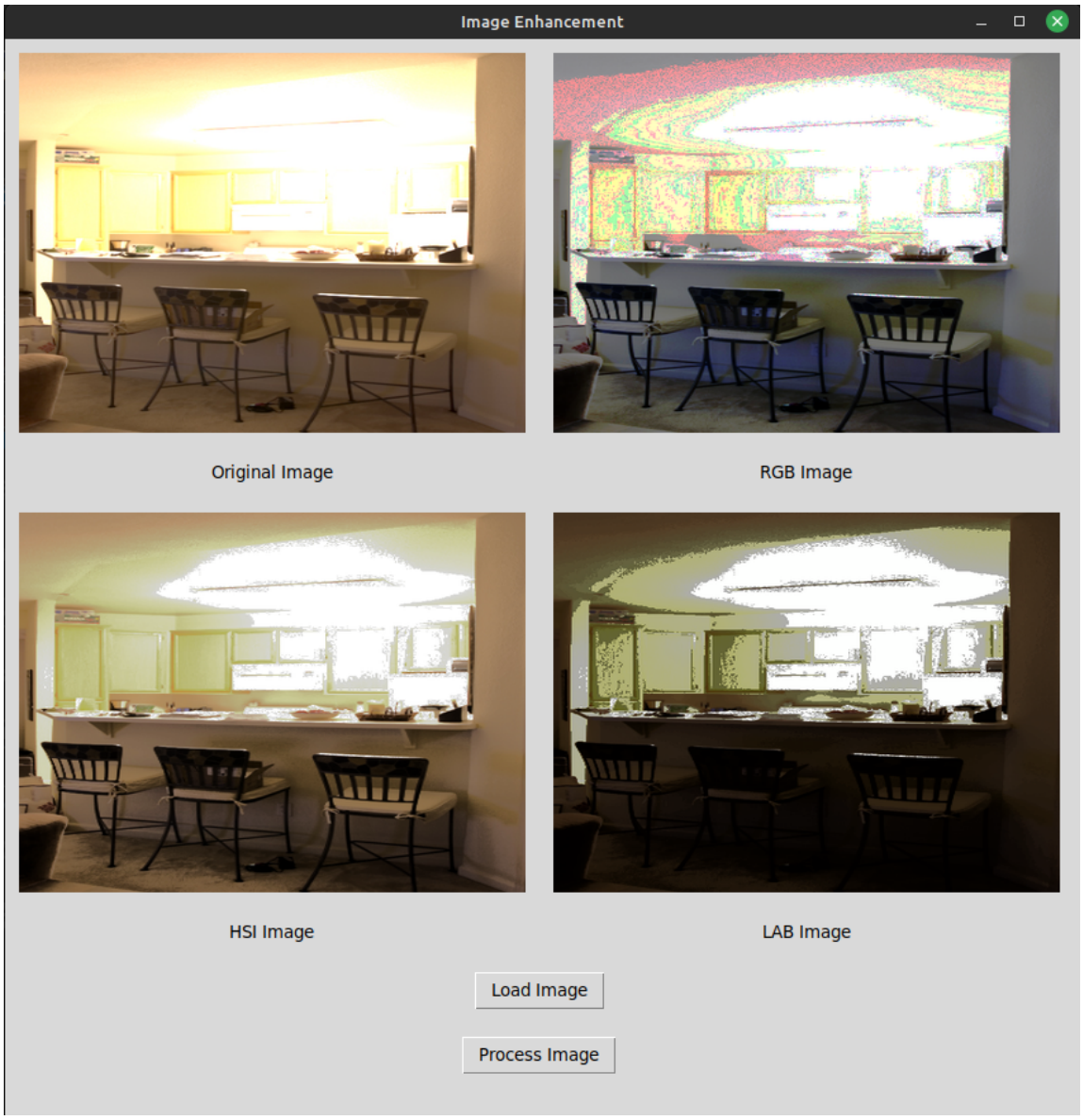
```
python3 UI_version.py
```

或是

```
python UI_version.py
```

執行後，從`Load Image`這個按鈕讀入欲執行的圖片，並且按下`Process Image`便會自動對圖片的各個色彩空間做enhancement。

示例圖:



程式碼解釋

1. Histogram-Equalization

圖片 enhancement 的方式使用 Histogram Equalization

主要可以分為以下步驟:

1. 計算每個 pixel value(0~255)的出現次數 (`original_histogram_cnt`)
2. 計算累積出現的數量(`cdf`)
3. 得到`cdf`後，代入公式:

$$h(v) = \text{round}\left(\frac{cdf(v) - cdf_{min}}{cdf_{max} - cdf_{min}} \times (L-1)\right)$$

4. 將原圖經過`new_histogram`查表得到新的pixel value，將結果存入`new_img`

基本上實做方式和`HW1`相同，但是需要注意的是要注意範圍及型態，不然之後轉回RGB可能圖會爆掉。

```
# implement RGB case
    if status == 1:
        height, width, channels = image.shape
    else:
    # implement HSI or LAB case (preprocessing)
        height = len(image)
        width = len(image[0])
        for i in range(height):
            for j in range(width):
                # HSI * 255 -> recover to 0 - 255
                if status == 2:
                    img[i][j][channel] = int(img[i][j][channel] * 255)
                    img[i][j][channel] = clamp_pixel_value(img[i][j][channel])
                else:
                    img[i][j][channel] = int(img[i][j][channel])
```

且HSI只需要做I的部份，LAB只需要做L的部份，而RGB三者都須做enhance:

```
# COLOR ENHANCEMENT
    result_RGB =
histogram_equalization(histogram_equalization(histogram_equalization(image_RGB,
0, 1), 1, 1), 2, 1)
    tmp_HSI = convert_HSI_to_RGB(histogram_equalization(image_HSI, 2, 2)) # only
process I channel
    tmp_LAB = convert_LAB_to_RGB(histogram_equalization(image_LAB, 0, 3, 101)) #
L max vale = 100, only process L channel
```

2. RGB to HSI

```
# Convert from RGB to HSI
def convert_RGB_to_HSI(image):
    [height, width, channels] = image.shape
    hsi_image = [[[0.0 for _ in range(3)] for _ in range(width)] for _ in
range(height)]
    for i in range(height):
        for j in range(width):
            [B, G, R] = image[i][j]
            [H, S, I] = [0, 0, 0]
            R /= 255.0
            G /= 255.0
            B /= 255.0
            # Calculate theta
            molecular = 0.5 * (R - G + R - B)
            denominator = math.sqrt((R - G) ** 2 + (R - B) * (G - B))
            if denominator == 0:  # Avoid division by 0
                theta = 0
            else:
                theta = math.acos(round(molecular / denominator, 6))
            # H value
            if B <= G:
                H = theta
            else:
```

```
                H = 2 * math.pi - theta
            # S value
            if R != 0 or G != 0 or B != 0:
                S = 1 - 3 / (R + G + B) * min(R, G, B)
            # I value
            I = (R + G + B) / 3.0
            hsi_image[i][j] = [H, S, I]
    return hsi_image
```

主要可以分為以下步驟:

1. 將RGB通道的像素值歸一化到0到1的範圍內。
2. 計算色相（H）的角度，根據藍色（B）和綠色（G）的值確定範圍。
3. 計算飽和度（S）的值，根據紅色（R）、綠色（G）和藍色（B）的值。
4. 計算亮度（I）的值，為紅色（R）、綠色（G）和藍色（B）的平均值。
5. 將計算得到的色相（H）、飽和度（S）和亮度（I）的值存儲到HSI圖像對應的像素位置。

3. HSI to RGB

```
# Convert from HSI to RGB
def convert_HSI_to_RGB(image):
    [height, width] = [len(image), len(image[0])]
    rgb_image = [[[0.0 for _ in range(3)] for _ in range(width)] for _ in
range(height)]
    for i in range(height):
        for j in range(width):
            [H, S, I] = image[i][j]
            [R, G, B] = [0.0, 0.0, 0.0]
            # RG sector (0° <= H < 120°)
            if H >= 0 and H < 2 * math.pi / 3:
                B = I * (1 - S)
                R = I * (1 + (S * math.cos(H) / math.cos(math.pi / 3 - H)))
                G = 3.0 * I - (R + B)
            # GB sector (120° <= H < 240°)
            elif H >= 2 * math.pi / 3 and H < 4 * math.pi / 3:
                H = H - 2 * math.pi / 3
                R = I * (1 - S)
                G = I * (1 + (S * math.cos(H) / math.cos(math.pi / 3 - H)))
                B = 3.0 * I - (R + G)
            # BR sector (240° <= H < 360°)
            elif H >= 4 * math.pi / 3 and H <= 2 * math.pi:
                H = H - 4 * math.pi / 3
                G = I * (1 - S)
                B = I * (1 + (S * math.cos(H) / math.cos(math.pi / 3 - H)))
                R = 3 * I - (G + B)
            rgb_image[i][j] = [clamp_pixel_value(B * 255), clamp_pixel_value(G *
255), clamp_pixel_value(R * 255)]
    return rgb_image
```

主要可以分為以下步驟:

1. 遍歷HSI圖像的每個像素。

2. 根據色相（H）、飽和度（S）和亮度（I）的值，計算紅色（R）、綠色（G）和藍色（B）的值。
3. 根據色相值的範圍，將計算得到的RGB值存儲到RGB圖像對應的像素位置。
4. 確保RGB值在0到255的範圍內。

4. RGB to LAB

```
# RGB to LAB function
def h(q):
    if q > 0.008856:
        return pow(q, 1 / 3)
    else:
        return 7.787 * q + 16 / 116

# Convert from RGB to LAB
def convert_RGB_to_LAB(image):
    [height, width, channels] = image.shape
    lab_image = [[[0.0 for _ in range(3)] for _ in range(width)] for _ in
range(height)]

    for i in range(height):
        for j in range(width):
            [B, G, R] = image[i][j]
            [L, a, b] = [0.0, 0.0, 0.0]
            R /= 255.0
            G /= 255.0
            B /= 255.0
            Xn = 0.95045
            Yn = 1.00000
            Zn = 1.08875
            # Convert to CIE XYZ
            X = 0.4124564 * R + 0.3575761 * G + 0.1804375 * B
            Y = 0.2126729 * R + 0.7151522 * G + 0.0721750 * B
            Z = 0.0193339 * R + 0.1191920 * G + 0.9503041 * B
            # Convert to CIE Lab
            X = X / Xn
            Y = Y / Yn
            Z = Z / Zn
            L = 116 * h(Y) - 16
            a = 500 * (h(X) - h(Y))
            b = 200 * (h(Y) - h(Z))
            lab_image[i][j] = [L, a, b]
    return lab_image
```

主要可以分為以下步驟:

1. 遍歷RGB圖像的每個像素。
2. 根據每個像素的紅色（R）、綠色（G）和藍色（B）值，將它們歸一化為0到1的範圍。
3. 根據CIE XYZ的轉換公式，計算出每個像素對應的XYZ值。
4. 將XYZ值除以參考值（Xn、Yn、Zn）以正規化。
5. 根據CIE Lab的轉換公式，計算出每個像素對應的L、a、b值。
6. 將計算得到的L、a、b值存儲到LAB圖像對應的像素位置。

5. LAB to RGB

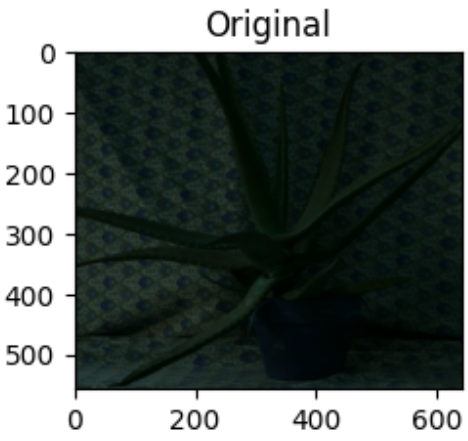
```
# LAB to RGB function
def f(t):
    if t > 0.008856:
        return pow(t, 3)
    else:
        return 3 * (0.008865) ** 2 * (t - (16 / 116))

# Convert from LAB to RGB
def convert_LAB_to_RGB(image):
    [height, width] = [len(image), len(image[0])]
    rgb_image = [[[0.0 for _ in range(3)] for _ in range(width)] for _ in
range(height)]
    for i in range(height):
        for j in range(width):
            [L, a, b] = image[i][j]
            [R, G, B] = [0.0, 0.0, 0.0]
            Xn = 0.95045
            Yn = 1.00000
            Zn = 1.08875
            # Convert to CIE XYZ
            X = Xn * f(1 / 116 * (L + 16) + 1 / 500 * a)
            Y = Yn * f(1 / 116 * (L + 16))
            Z = Zn * f(1 / 116 * (L + 16) - 1 / 200 * b)
            # Convert to RGB
            R = 3.2404542 * X - 1.5371385 * Y - 0.4985314 * Z
            G = -0.969266 * X + 1.8760108 * Y + 0.0415560 * Z
            B = 0.0556434 * X - 0.2040259 * Y + 1.0572252 * Z
            rgb_image[i][j] = [clamp_pixel_value(B * 255), clamp_pixel_value(G *
255), clamp_pixel_value(R * 255)]
    return rgb_image
```

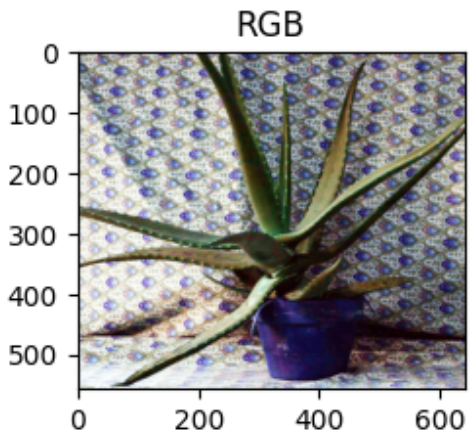
主要可以分為以下步驟:

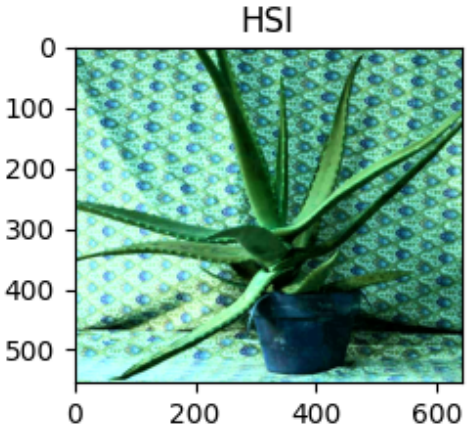
1. 遍歷LAB圖像的每個像素。
2. 根據每個像素的L、a和b值，計算對應的CIE XYZ值。
3. 根據CIE XYZ到RGB的轉換公式，計算出每個像素對應的R、G、B值。
4. 將計算得到的R、G、B值存儲到RGB圖像對應的像素位置。

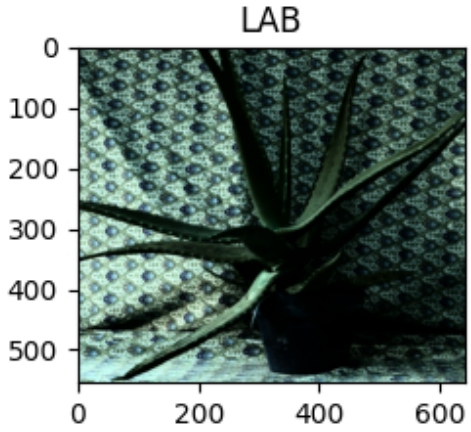
Experimental results

aloe.jpg


Original
0
100
200
300
400
500
0
200
400
600

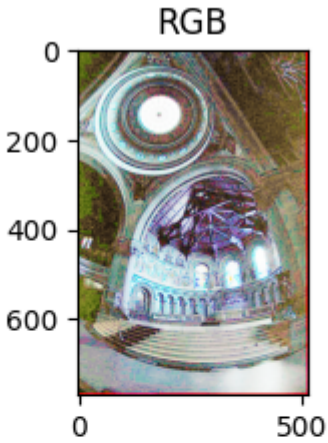

RGB
0
100
200
300
400
500
0
200
400
600

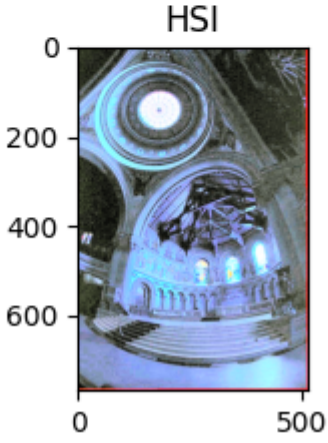

HSI
0
100
200
300
400
500
0
200
400
600

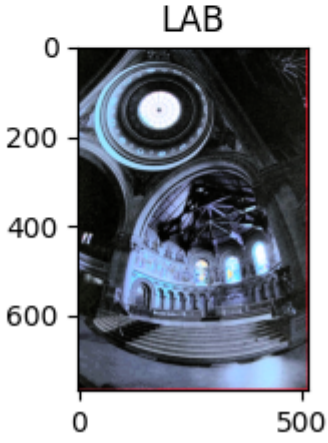

LAB
0
100
200
300
400
500
0
200
400
600

church.jpg

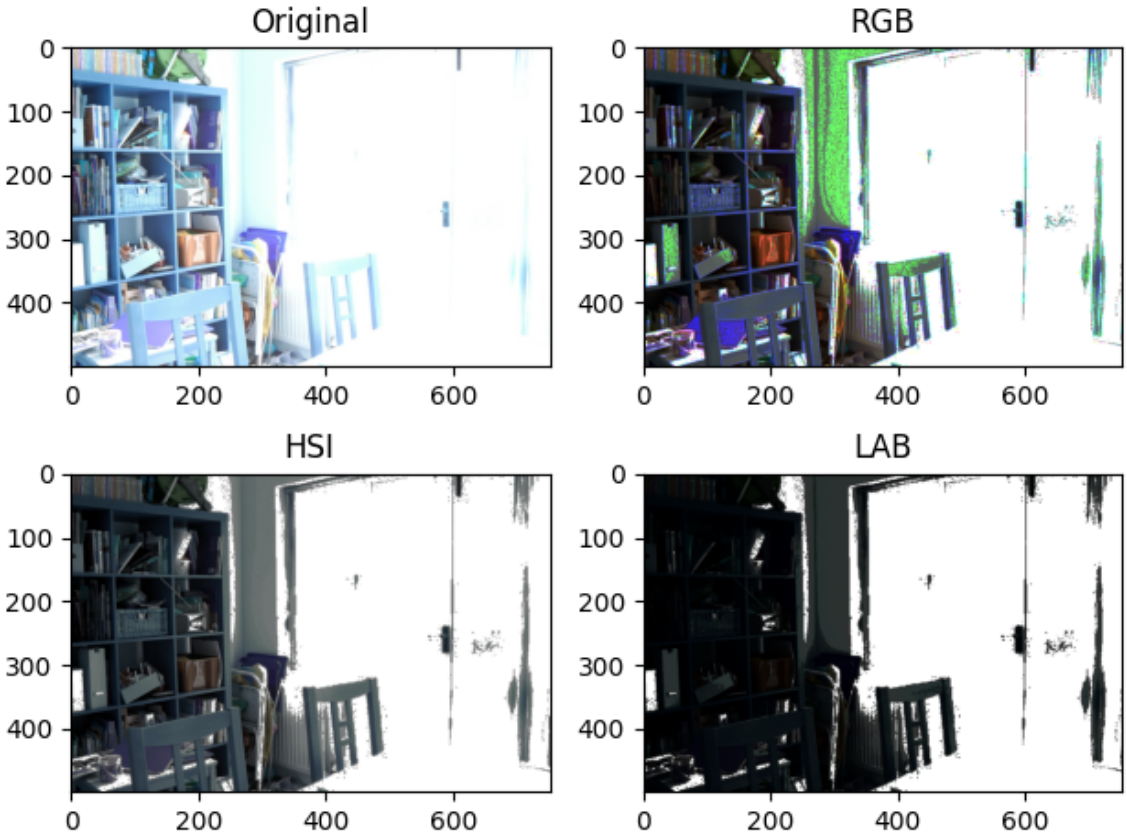

Original
0
200
400
600
0
500


RGB
0
200
400
600
0
500

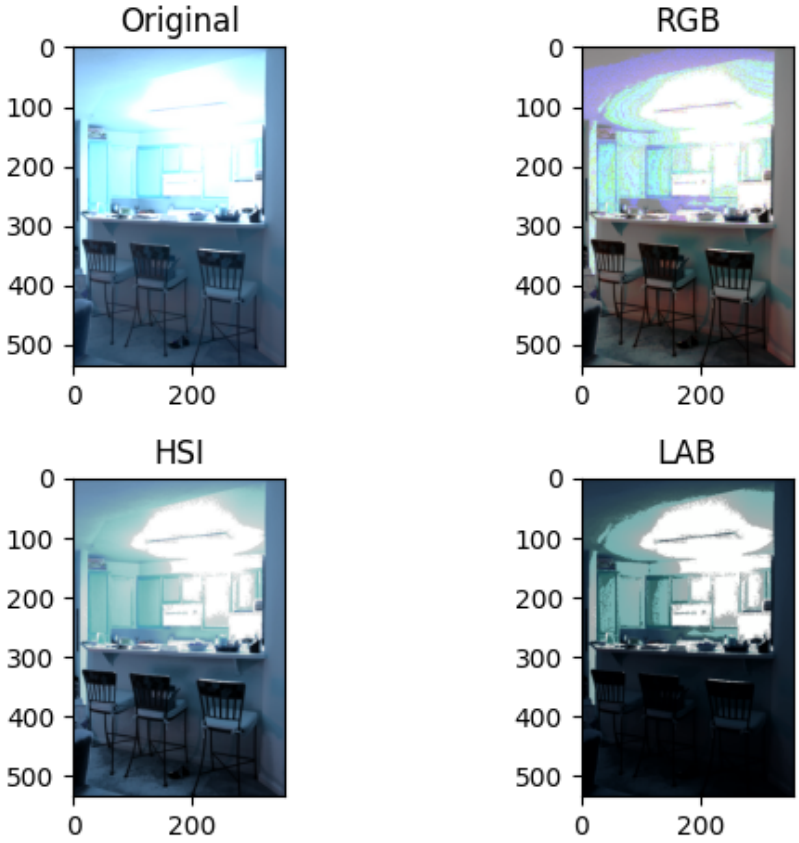

HSI
0
200
400
600
0
500


LAB
0
200
400
600
0
500

house.jpg



kitchen.jpg



Discussions

從上方的實驗結果與比較可以發現:

當對RGB、HSI和LAB圖像進行直方圖均衡化時，結果可能有所不同。

- RGB直方圖均衡化：
  在RGB空間中，直方圖均衡化將分別應用於每個通道（紅色、綠色和藍色）的直方圖。這可以增強圖像的整體對比度，使得亮度更均勻分佈。然而，這種方法忽略了顏色信息之間的關聯性。
- HSI直方圖均衡化：
  在HSI色彩空間中，直方圖均衡化主要應用於亮度（I）通道。通過對I通道進行均衡化，可以增強圖像的對比度和亮度分佈，同時保持色相（H）和飽和度（S）的不變性。這可以達到增強圖像亮度的效果，同時保持顏色信息的穩定。
- LAB直方圖均衡化：
  在LAB色彩空間中，直方圖均衡化主要應用於亮度（L）通道。通過對L通道進行均衡化，可以增強圖像的整體對比度和亮度分佈，同時保持顏色信息（a和b通道）的穩定。這可以在不影響顏色信息的情況下，提升圖像的亮度和對比度。

總結起來，RGB直方圖均衡化僅針對每個通道的亮度進行處理，HSI直方圖均衡化保持了色相和飽和度的不變性，而LAB直方圖均衡化則保持了色彩信息的穩定性。選擇使用哪種方法取決於具體應用和需求。每種方法都可以用於增強圖像的對比度和亮度分佈，但可能對顏色信息產生不同的影響。

心得

和前兩次相比，這次作業真的爆炸難。原本想說只要乖乖把老師ppt上的轉換公式都刻上去就好，但是在生圖的時候馬上報一堆bug。還好大部份都是邊界問題，這個問題蠻常導致在轉換的時候超出邊界導致圖無法顯示。

實驗結果也挺令人意外的，原本想說都會一致RGB強化後的結果會比較好，但是其實根據每張圖的不同都會有自己的需求。像是`aloe.jpg`的話RGB比較好，`church.jpg`的話感覺HSI和LAB比較好，`house.jpg`的話HSI比較舒服，`kitchen.jpg`的話也是HSI。感覺由亮到暗RGB都會處理的糊糊的，這種時候因為HSI保持色相（H）和飽和度（S）的不變性，所以就會強化的比較成功。

這次作業時間也比較充裕，因此也寫了UI版本來實做。原本想要使用pyinstaller把UI版本轉成.exe檔，不過freeze後沒想到變成1GB多，後來還是作罷。

References and Appendix

- RGB與HSI互換
- RGB與LAB互換
- 老師ppt