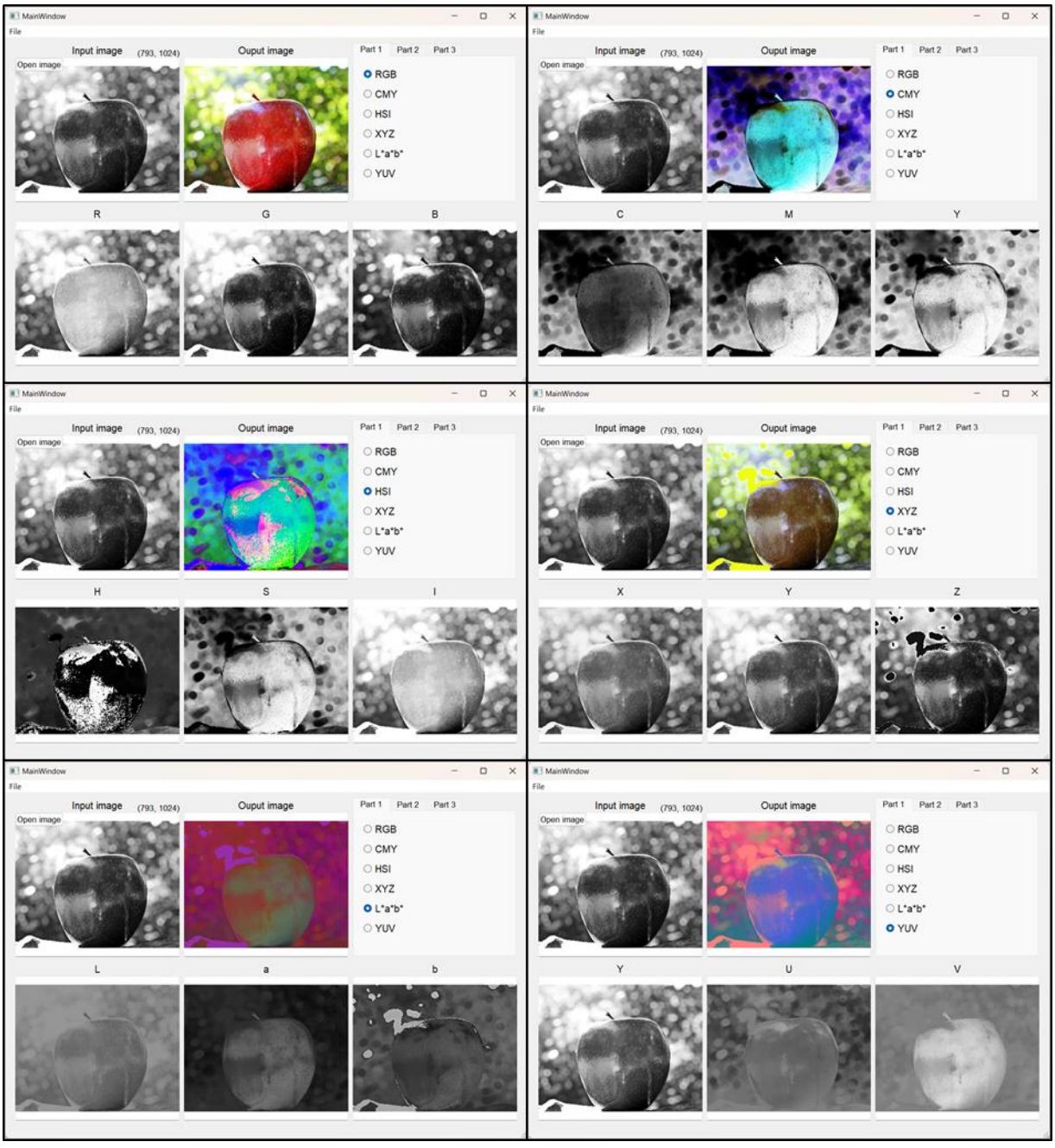
Principles and Applications of Digital Image Processing

Homework 5

R11631012 林雲

## Part 1: (40%) Color Model Conversion

轉換 RGB 至不同色彩空間之算法，實作可檢視 ./code/image_processing.py。程式之介面與執行結果如下圖所示。

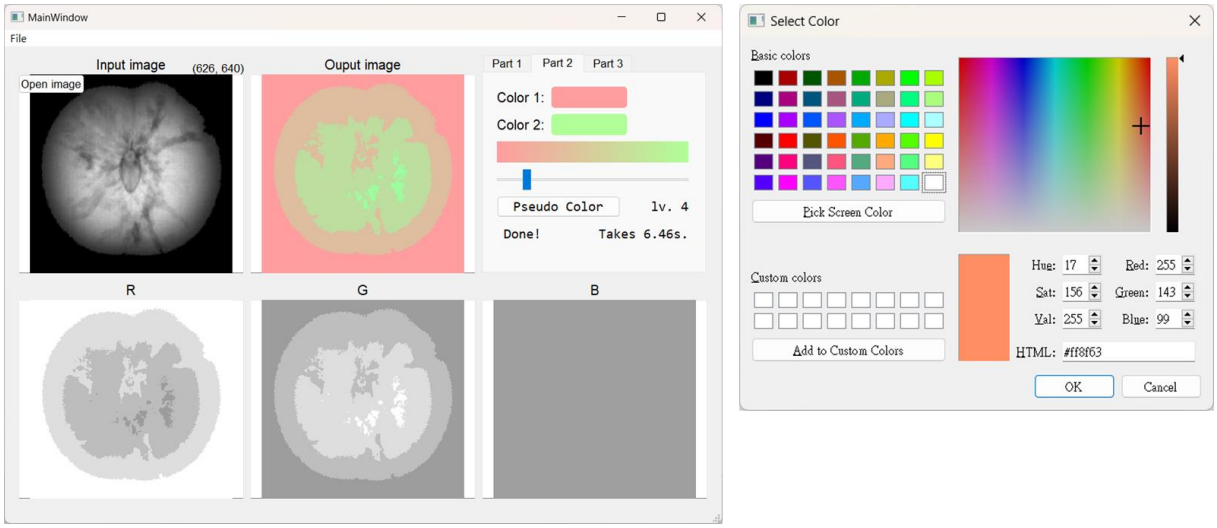



圖一、不同色彩空間之影像。

- RGB 影像顯示在其三個 channel 可看到 R 中的蘋果偏白的，此跟彩色影像中的蘋果主要為紅色相關。
- CMY 影像有一種負片的感覺，這是因為 CMY 為 RGB 的互補影像，也因此各個 channel 的深淺也都和 RGB channel 相反。
- HSI 色彩空間利用色調、色彩飽和度、亮度三者來描述色彩。H 為角度來定義色彩，而紅色的角度接近 0，因此在蘋果部分呈現較深的顏色；在 S 的部分可觀察到後面亮光的部分，由於較不飽和因此呈現黑色斑塊；而 I 的部分則可以看到呈現類似灰階的效果。
- XYZ 色彩空間可以用來表達人類眼睛可見的所有色彩，但其可以表達的顏色就為廣泛。而由顯示結果可以看出他的三個 channel 都和 RGB 相當類似，但蘋果看起來更加像日常肉眼所見。
- L*a*b*影像式利用數值化的方式來描述人眼所見的影像，而我們可以由結果觀察出他非常適合用來分離影像中的前景與背景，顯現影像中的深度。
- YUV 色彩空間將亮度資訊儲存在 Y，而色度信號則是儲存在 U 和 V 中，也因此所顯示的 Y 和其灰階影像相當類似。

## Part 2: (30%) Pseudo-color Image

圖二為操作介面，使用者可以點有顏色的方框呼叫色彩盤選擇自身喜好的顏色。並能透過調整滑桿自訂色彩階段數。




圖二、Pseudo-color 之影像。

Pseudo-color 的實作會參考所選的兩個顏色及色彩階段數，將兩顏色依據階段數分層。最後灰階影像以相對應的數值轉換為 pseudo-color 影像。透過設定兩種顏色的數值，將灰階影像轉換成不同顏色的影像，此外，若顏色分層數量過少，影像資訊將遺失。以下為實作的程式碼：

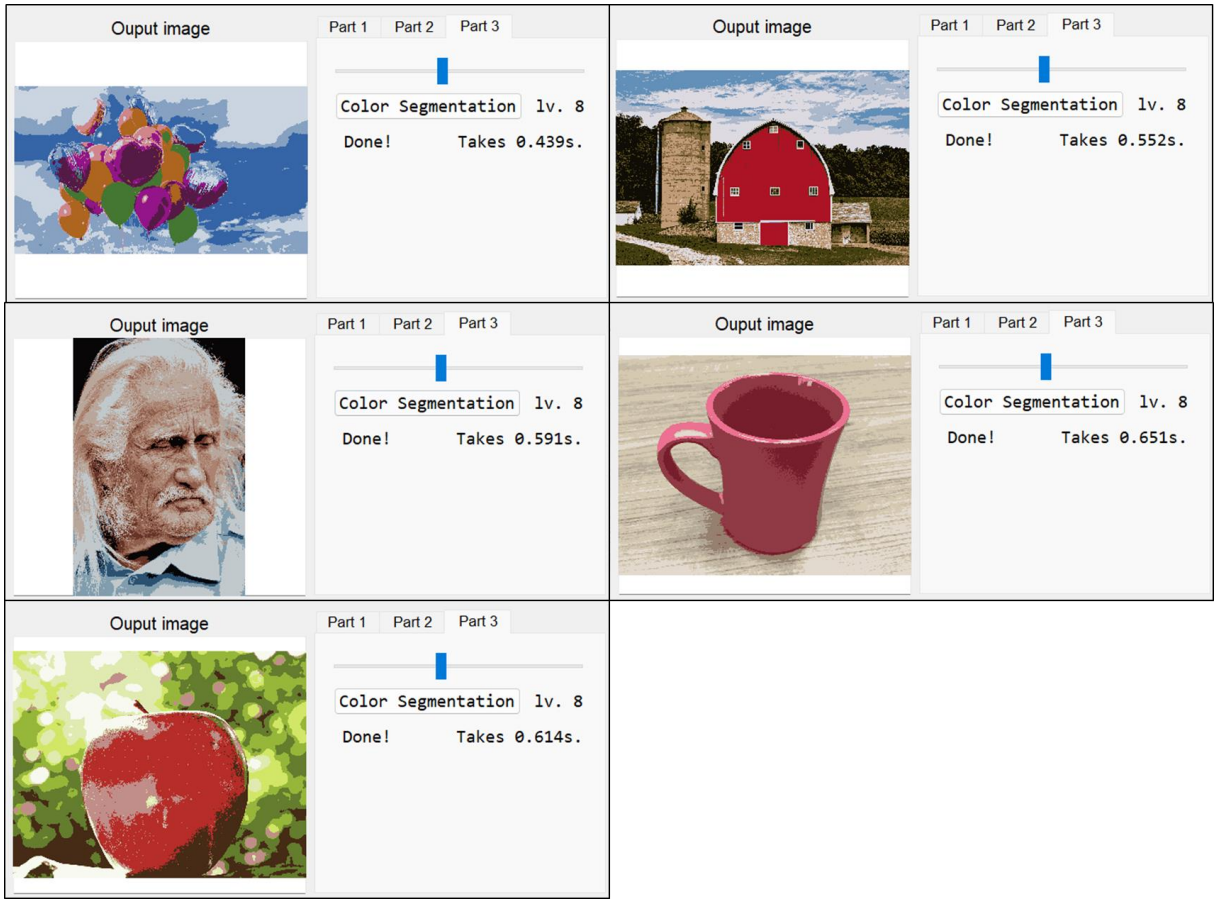
```
def pseudo(img, level, color_start, color_end, img_path):
    level = level - 1
    r, g, b = cv2.split(np.float32(img))
    rs = color_start.red()
    gs = color_start.green()
    bs = color_start.blue()
    re = color_end.red()
    ge = color_end.green()
    be = color_end.blue()
    gray = 0.299*r + 0.587*g + 0.114*b
    stepr = (re-rs)/level
    stepg = (ge-gs)/level
    stepb = (be-bs)/level
    output_img = np.zeros(img.shape, dtype=np.uint8)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            output_img[i, j, 0] = np.around(
                rs + np.round(gray[i, j]*level/255)*stepr)
            output_img[i, j, 1] = np.around(
                gs + np.round(gray[i, j]*level/255)*stepg)
            output_img[i, j, 2] = np.around(
                bs + np.round(gray[i, j]*level/255)*stepb)
    output_path = save_output(
        output_img[:, :, [2, 1, 0]], img_path, 'RGB', split=True)
    return output_path
```

## Part 3: (30%) Color Segmentation

參考 OpenCV 網站，使用 cv2.kmeans()實作 color segmentation。

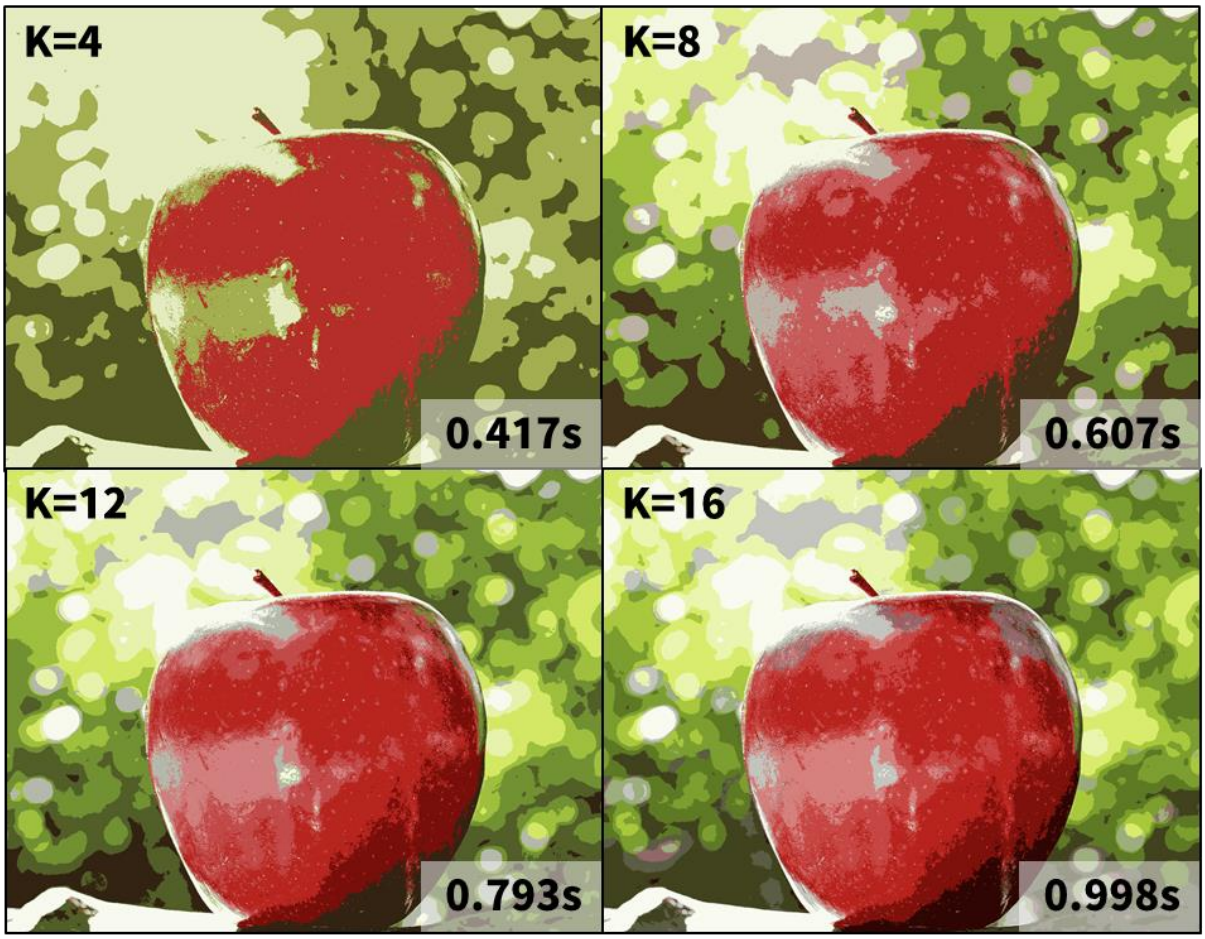
```python
def segmentation(img, k, img_path):
    # reshape to m*n 3-d points.
    img_point = img.reshape((-1, 3))
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
    flags = cv2.KMEANS_RANDOM_CENTERS
    compactness, label, center = cv2.kmeans(
        np.float32(img_point), k, None, criteria, 10, flags)
    center = np.uint8(center)
    result = center[label.flatten()]
    output_img = result.reshape((img.shape))
    output_path = save_output(
        output_img[:, :, [2, 1, 0]], img_path, 'RGB', split=True)
    return output_path
```

比較不同複雜度影像對相同 k 值的表現。圖三是五張不同像素大小之影像，從小到大依序為氣球、房子、老人、杯子和蘋果。見結果可發現影像越大，所耗時越多。但其初始點使用隨機決定，每次皆不固定，略為影響計算時間。




圖三、不同解析度影像用 Color Segmentation。

使用不同 k 對 RGB 影像做 Color segmentation。當 k 越小的時候，分群結果會和原始影像有較大的差異；當 k 越大影像會和原始影像越來越接近，但須更多計算時間。此一影像處理方式可以用於壓縮圖像大小，k 值越小則影像大小越小，流失之資訊約多。反之變大能有較佳的色彩表現。




圖四、影像用不同 k 做 Color Segmentation。

Color Segmentation 勉強可將影像中的物體分離。觀看結果可發現 L*a*b*與 RGB 的結果較佳，HSI 則效果不彰。


圖五、RGB、HSI、L*a*b*做 Color Segmentation。