Digital Image Processing (2017)

Homework 2 Report

{color manipulation}

0650253　賴茂睿

1. White balance [1]

本作業將照片做 white balance 處理通常有兩種方法，一是選擇圖片 color checker 的某個顏色，與真實的顏色做 linear transformation。因為 linear 的緣故，我們需要先將 data 從 non-linear RGB 轉為 linear RGB 才能做轉換，得到一個 3x3 的矩陣後，便可對照片中每個點(pixel)做 mapping。
而第二種方法，也是我本次所使用的方法，是直接使用白平衡的演算法，也就是所謂的 Auto White-Balance。我所使用到的有 gray world 和 gimp 兩種方法。[2,3]Gray world 以灰度世界假設為基礎，其假設為：對於一幅有著大量色彩變化的圖像，R,G,B，三個分量的平均值趨於同一灰度值 Gray。從物理意義上而言，灰色世界法假設自然界景物對於光線的平均反射的值在整體上是一個定值，這個定值近似為“灰色”。顏色平衡演算法將這一假設強制應用於待處理圖像，可以從圖像中消除環境光的影響，獲得原始場景圖像。

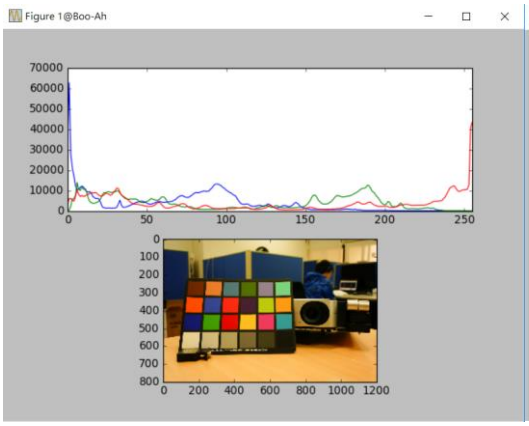
```
def gray_world(nimg):
    nimg = nimg.transpose(2, 0, 1).astype(np.uint32)
    mu_g = np.average(nimg[1])
    nimg[0] = np.minimum(nimg[0]*(mu_g/np.average(nimg[0])),255)
    nimg[2] = np.minimum(nimg[2]*(mu_g/np.average(nimg[2])),255)
    return  nimg.transpose(1, 2, 0).astype(np.uint8)
```

我們可以將 gray world 與老師上課所說的 retinex 做比較，得到更多啟發

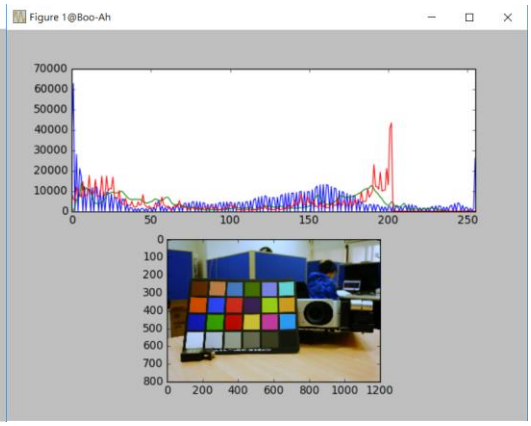
```
def retinex(nimg):
    nimg = nimg.transpose(2, 0, 1).astype(np.uint32)
    mu_g = nimg[1].max()
    nimg[0] = np.minimum(nimg[0]*(mu_g/float(nimg[0].max())),255)
    nimg[2] = np.minimum(nimg[2]*(mu_g/float(nimg[2].max())),255)
    return nimg.transpose(1, 2, 0).astype(np.uint8)
```

Retinex 是將人類視神經的概念引入演算法，因為人類是藉由強度最強的物體，相對比較出物體的顏色。而 gray world 便是將強度最強的概念，替換成平均值的想法。
Gimp 的白平衡原理，則是把 RGB 個顏色，除了上下 1% 外，盡量線性拉長。其概念為先將 color histogram 在鄰近邊界的數值剔除，再將色調盡量拉成平均分佈，讓色彩整體來說比較平均，而實作上我使用上下 5%。

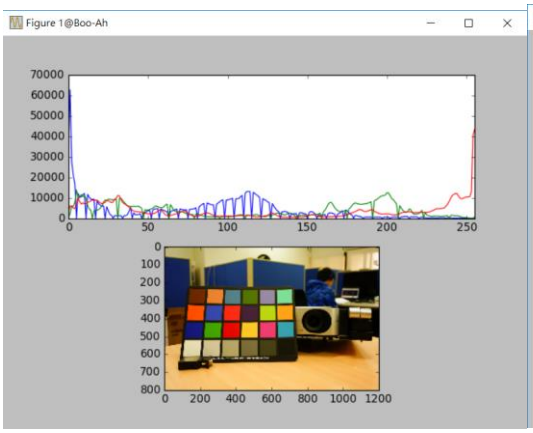
以下是我使用不同 Auto white-balance 演算法的效果及 color histogram :



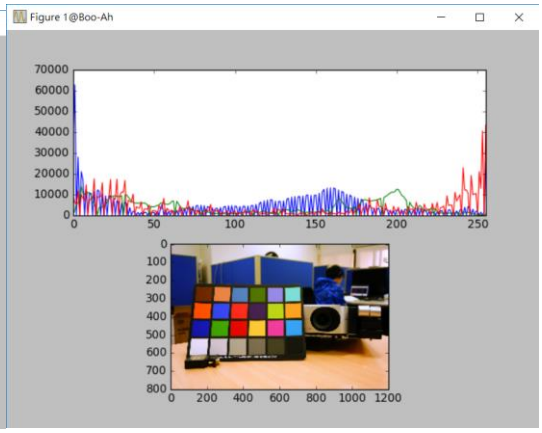
Original Image



Gray world



Gimp



Gray world + Gimp

Discussion :

我們可以發現，當 histogram 中紅色的值在 50~200 的範圍內較低時，色調偏向褐色。從 gray world 的處理可以看出，效果比較像我們所需要的，像是在日光燈下的效果。

而由 histogram 也可以了解為何 gray world 的成效較好，因為白光可以視為相同強度的 R,G,B，所以若是我們在 G,B 的值較高的時候，也使的 R 的值較高，便有機會產生近似白光的效果。

2. Color transformation[4,5]

作業的第二題要求我們改變圖片的色調，因此我將 sRGB 座標轉為 LAB 座標系，這樣我便可針對紅-綠,藍-黃,黑-白分別作調整。從 sRGB 轉換到 LAB，需先將 sRGB 經過 gamma effect，轉換成 linear RGB，再經過 Rec 709.的 3x3 transformation 得到 XYZ 值，最後再代入 CIE 1976 (L*,a*,b*)的公式中。而從 Lab 到 sRGB 則使用逆運算即可。如下圖所示 ：

CIE Lab

$$X = X_n f^{-1}\left(\frac{L^* + 16}{116} + \frac{a^*}{500}\right)$$
$$Y = Y_n f^{-1}\left(\frac{L^* + 16}{116}\right)$$
$$Z = Z_n f^{-1}\left(\frac{L^* + 16}{116} - \frac{b^*}{200}\right)$$

where

$$f^{-1}(t) = \begin{cases} t^3 & \text{if } t > \delta \\ 3\delta^2\left(t - \frac{4}{29}\right) & \text{otherwise} \end{cases}$$

$$L^* = 116\, f\left(\frac{Y}{Y_n}\right) - 16$$
$$a^* = 500\left(f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right)\right)$$
$$b^* = 200\left(f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right)\right)$$

where

$$f(t) = \begin{cases} \sqrt[3]{t} & \text{if } t > \delta^3 \\ \frac{t}{3\delta^2} + \frac{4}{29} & \text{otherwise} \end{cases}$$
$$\delta = \frac{6}{29}$$

CIE XYZ

$$\begin{bmatrix} R_{\text{linear}} \\ G_{\text{linear}} \\ B_{\text{linear}} \end{bmatrix} = \begin{bmatrix} 3.2406 & -1.5372 & -0.4986 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \begin{bmatrix} R_{\text{linear}} \\ G_{\text{linear}} \\ B_{\text{linear}} \end{bmatrix}$$

linear RGB

$$C_{\text{srgb}} = \begin{cases} 12.92 C_{\text{linear}}, & C_{\text{linear}} \le 0.0031308 \\ (1+a) C_{\text{linear}}^{1/2.4} - a, & C_{\text{linear}} > 0.0031308 \end{cases}$$

where $a = 0.055$ and where C is R, G, or B.

$$C_{\text{linear}} = \begin{cases} \frac{C_{\text{srgb}}}{12.92}, & C_{\text{srgb}} \le 0.04045 \\ \left(\frac{C_{\text{srgb}} + a}{1+a}\right)^{2.4}, & C_{\text{srgb}} > 0.04045 \end{cases}$$

sRGB

需要注意的是，運算 linear-RGB 到 sRGB 時，會遇到 invalid value encountered in power 的問題，這是因為 linear-RGB 為負值的原因。由下面的例子可以看出是多出虛數的緣故 ：

```
In [2]: numpy.array([-2.0+0j])**(1/2.4)
Out[2]: array([ 0.34548198+1.28935629j])
```

剩下的則是改變圖片色調的部分，因為要將草從綠色變為褐色，只需要調整”a”通道的部分，我找了一個可以涵蓋草和樹的 threshold 值，將草和樹”a”通道的數值向紅色平移，便可將其變成褐色的樣貌。接著，原先我想將天空變得更深藍，然而調整”b”值會發現無法將天空和雲有效做分離。因此，我最後決定藉由調整天空和雲的亮度的方式，來達到其效果。

Discussion:

起初，我想使用 opencv 來驗證我所 implement 的運算是否正確時，發現所對應的數值無法對應的很準確，因此我查看 opencv 的教程 [6]。發現 opencv 並未做 gamma effect 的轉換，而是在 transformation 的公式中，引入 delta 值並在最後對 Lab 做修正，如下圖所示 ：

- RGB ↔ CIE L*a*b* (`CV_BGR2Lab, CV_RGB2Lab, CV_Lab2BGR, CV_Lab2RGB`).

  In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$X \leftarrow X/X_n, \text{where} X_n = 0.950456$$

$$Z \leftarrow Z/Z_n, \text{where} Z_n = 1.088754$$

$$L \leftarrow \begin{cases} 116 * Y^{1/3} - 16 & \text{for } Y > 0.008856 \\ 903.3 * Y & \text{for } Y \le 0.008856 \end{cases}$$

$$a \leftarrow 500(f(X) - f(Y)) + delta$$

$$b \leftarrow 200(f(Y) - f(Z)) + delta$$

where

$$f(t) = \begin{cases} t^{1/3} & \text{for } t > 0.008856 \\ 7.787t + 16/116 & \text{for } t \le 0.008856 \end{cases}$$

and

$$delta = \begin{cases} 128 & \text{for 8-bit images} \\ 0 & \text{for floating-point images} \end{cases}$$

This outputs $0 \le L \le 100, -127 \le a \le 127, -127 \le b \le 127$. The values are then converted to the destination data type:

- 8-bit images

$$L \leftarrow L * 255/100, \; a \leftarrow a + 128, \; b \leftarrow b + 128$$

References

[1] https://stackoverflow.com/questions/1175393/white-balance-algorithm

[2] http://www.bkjia.com/ASPjc/924771.html

[3] http://weijr-note.blogspot.tw/2012/03/python.html

[4] https://zh.wikipedia.org/wiki/SRGB 色彩空间

[5] https://zh.wikipedia.org/wiki/Lab 色彩空间

[6] https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=cvtcolor#cv2.cvtColor