Introduction to Digital Image Processing - Term project

Implementation

of

"*Progressive color transfer for images of arbitrary dynamic range*"

Student dept. / name / ID: 電機四 / 陳柏翔 / 4109061012

1. Introduction

1.1 Overview

此專題的目標為實現論文[1]的圖像色彩轉移技術，而此處的色彩轉移指的是「將 A 圖像的色彩轉移成跟 B 圖像的色調相同」。[1]所提出的技術為一種漸進式的圖像直方圖(histogram)重塑(reshape)技術，相對過去其他調整圖像色彩的方法來的更加直觀且更好控制。

根據[1]的 Introduction 第 4 段所述，在給定欲調整色彩的“源圖像(source image)”及“目標圖像(target image)”後，使用者將可以選定色彩轉移的完整程度：在最不完整的情況下，源圖像將完全保持原樣；最完整的情況下，將會完整匹配(matching)目標圖像。

實際執行上，據 Introduction 第 3 段所述，我們將使用圖像的直方圖來分辨圖像中的“大區域”及“細節”。大區域由於像素(pixel)數值相近，在直方圖上會形成波峰(peak)，而細節部分則為直方圖中其他數值較小的區域。作者利用這樣的特性，取代需要手動選擇區域的人為操作，也能夠分別對影像中不同的部分進行色彩轉移。

在重塑直方圖的方法上，[1]的技術主要是以漸進式的重塑進行，會從最粗糙(histogram's bins 很寬)的匹配漸進到最細緻(histogram's bins 很細)的匹配。而粗糙與細緻的程度，是由一個倍率(scale)進行控制，倍率越小則越粗糙、越大則越精細，而使用者能夠調整的就是細緻的上限倍率S_{max}。設定S_{max}後，直方圖將經過一連串的倍率$k = [1, S_{max}]$重複進行重塑，這個過程直方圖變化如下方圖 1 所示。

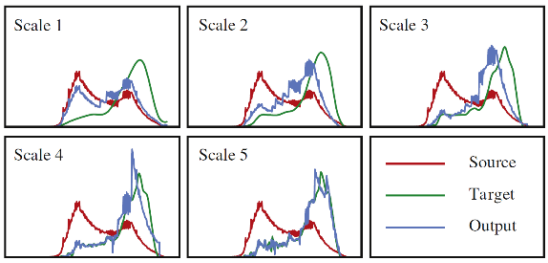

圖 1. 論文[1]中的 Fig. 4

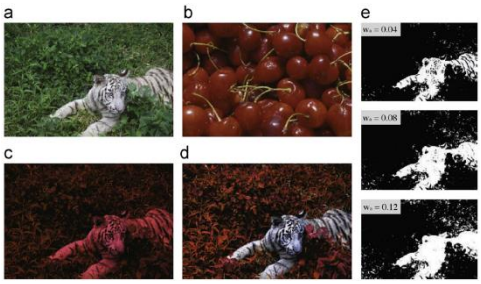

圖 2. 論文[1]中的 Fig. 7

除了全圖的色彩轉移外，論文中還提及了區域選擇的做法，如上方圖 2 所示。能夠將部分區域不考慮進直方圖重塑的過程中，單獨對其他部分進行轉換，後續也將實作這個部分。而值得注意的是，在論文 Region selection 章節的描述中，區域的選擇是以灰白色(achromatic colors)為中心的，也就是根據[1]中的 Eq. 18 計算的方法只能夠選取到灰白色區域(白色老虎)。

1.2 Algorithm analysis

以下根據[1]中 Fig.8 的虛擬碼(pseudo code)以及 Eqs.1~15，並依照虛擬碼的流程逐一分析演算法重要的各個部分，以及實作前所遇到的問題。

Histogram. 在演算法開始之前，[1]中的 Eqs. 1~5 在介紹直方圖的計算方法，並假設直方圖中 bins 的寬度為V，數量由$max(I)$、$min(I)$與V共同控制。由於在[1]當中整篇未提及V的設置，而$max(I)$與$min(I)$如果理解為圖像中的數值最小值與最大值，也將會與過去計算直方圖的方式上有很大的差異。在此專題實作中，我將比照過去講義所學來計算直方圖，也就是理解為 $V = 1$ (除了直方圖重新取樣(resampling)的部分)、$min(I) = 0$、$max(I) = 255$。

Convert color space. 演算法的第一步，會將源圖與目標圖都轉換到 CIELab 色彩空間。由於 CIELab 空間的值域並非 0~255，但是使用 python 的 cv2 套件進行轉換後不論是 La*b*哪個頻道(channel)都仍會是 0~255。參考[2]並自行測試後，發現到轉換完的數值在 a*與 b*(用於表示顏色)頻道中，如果顏色越貼近灰白，則越數值越靠近 128，此在後續實作過程中會有影響。

Compute S_{max}. 藉由[1]中的 Eq. 7 計算 S_{max} 的值，在不同頻道上將會有各自的 S_{max}，但是如上段所述，由於各頻道值域皆為 0~255，所以相同。

Compute histograms. 同 Histogram.所述。

Down- and up-sample. 在[1]中的 3.2 節第 3 段段末有描述到，此步驟指的是將直方圖先進行下取樣，再上取樣回原本的解析度。此步驟也是“Progressive”中最重要的其中一部分。

Find peaks. 雖然命名稱為“找波峰”但實際上找的是波谷，並利用波谷分隔出各個波峰區塊，這步驟由[1]中的 Eqs.8~9 計算得出。而此處有個疑點將在 1.3 節提出。

Region transfer. 藉由轉移平均(μ)及標準差(σ)的方式，把區域中的源直方圖轉換得更為接近目標直方圖，計算方式如[1]的 Eqs.10~12 所示。此處有兩個待討論的問題點於 1.3 節提出。

Histogram matching. 此步驟為圖像轉換回 RGB 色彩空間前的最後一步驟，此處將如過去課堂所學進行操作，不須用參考[1]的 Eqs.13~15。值得注意的是，如果使用者將 perc 設為 100%，那麼如[1]的 Introduction 第 4 段所述，最上限將會是完整的匹配直方圖，也就應該要如同直接做直方圖匹配(不漸進式進行)一樣，這樣便可以透過一般的直方圖匹配(histogram matching)來確認實作結果的正確性，結果應該要跟 perc 設為 100%時相同。

1.3 Issues

A. Find peaks. 在[1]的虛擬碼中 function FindPeaks() 理論上應該按照該文 Eqs. 8~9 以差分的方法計算梯度，但在虛擬碼中卻是先將原直方圖函數分為正與負(理論上原始的直方圖也不會有負值)、再計算梯度，這樣的做法將在第一步就失去梯度，且無法得到 R_{min} 集合。

B. Region transfer. 與 A.相似，在[1]的虛擬碼中 function RegionTransfer() 的 $w_{t,k}$與$w_{s,k}$理應按照該文 3.2 節第 6 段 $w_{s,k} = k/S_{max}$，但是虛擬碼的函數卻是將 k/S_{max} 傳入，並設為w_t而非w_s，此處為第一個疑點。

後續將再討論，我認為上述疑點應該要如同虛擬碼將 w_t 設為 k/S_{max} 才更符合色彩轉移的目的，但也因此衍伸新的問題：當使用者將 perc 設為 100%時， k/S_{max} 終將等於 1，也就是會使 $w_{t,k} = 1, w_{s,k} = 0$，進而造成[1]的 Eq.12 (同式(1)) 算出無限大的結果。

$$h_{o,k}(i) = \left(h_{s,k}(i) - w_{s,k}\mu_{s,k}(j)\right)\frac{w_{t,k}\sigma_{t,k}(j)}{w_{s,k}\sigma_{s,k}(j)} + w_{t,k}\mu_{t,k}(j) \quad (1)$$

同理，當要轉移的區間內只有 1 根 bin 的時候將使$\sigma_{s,k} = 0$，同樣計算出無限大的明顯錯誤結果。而後續將討論此問題的解決方法。

此外，另外會介紹我認為較簡單的另一種直方圖匹配方法，而就結果來看，我認為成果也同樣良好、同時還能解決匹配上發生的其他問題(如：匹配過度造成的大波峰)。

C. For each level k in perc/S_{max}. 此為虛擬碼的一行代碼。首先，由[1]的 3.2 節描述 $k \in [1, S_{max}]$ 以及當 perc=20%時 $k = \{1, \dots, 0.2S_{max}\}$，可以確定此行代碼不應該是 perc/$S_{max}$ 而應該是 perc*S_{max} 才正確。

其次，文中並未提及“level”具體為何，假設 level 指的是所有從 1 到 perc*S_{max} 的所有整數，那麼當我們將 B_{min} 設為 10 (與論文一致)時，S_{max} 將僅有 4，造成 k 不論 perc 如何調整，最多就只有 4 種可能。如果使用者將 perc 從 0%漸進不斷調整到 100%，在過程中其實就只有 4 種不同的輸出而已，與[1]的 Fig. 16 中 15%~100%多種不同的輸出不符合。因此可以確定 k 不一定為整數而是可以為小數，後續將會描述我所設置的“each level”具體為多少數值。

2. Method description

2.1 Solving issues

A. Find peaks. 對於直方圖梯度計算的問題，我選擇按照[1]中 Eqs. 8~9 進行計算以解決虛擬碼裡梯度完全消失的問題。由於離散的計算方式，會造成每次對一維的數據進行差分(求導)時，數據長度就會減 1；且由於差分求導是計算兩個數值中間的斜率，所以這個斜率代表的值不是位於原本數值的所在位置，而是兩者的中間。

在 function FindPeaks() 中兩次求導後，就會使原直方圖中的最頭與最尾位置的數值消失，因此我的作法是在求得局部最小值(local minimum)的位置 R_{min} 集合後，在最開頭及最尾端分別加上頭尾位置，使得區域涵蓋完整的直方圖範圍。

B. Region transfer. 對於第一個疑點(k/S_{max} 為 $w_{s,k}$ 還是 $w_{t,k}$)，考慮到在 perc 為 100%時要完全匹配，而 k 最後會等於 S_{max}，式(1)會成為完全轉移平均值與標準差的狀態。如果此時是 $w_{s,k} = k/S_{max} = 1$ ，則會變成完全不轉移，此情況與目標完全相反，因此正確應為 $w_{t,k} = k/S_{max}$，而並非[1]的 3.2 節第 6 段所述的 $w_{s,k}$。

在衍伸的問題上 $w_{s,k} = 0$ 與 $\sigma_{s,k} = 0$ 造成 $h_{o,k}(i)$ 無限大，前者表示完全轉移平均及標準差，後者表示只需要轉移一個 bin。考慮兩種情況後，我認為前者是由於式(1)並非正確進行轉移平均值與標準差所造成。假設 $w_{s,k}$ 與 $w_{t,k}$ 分別代表源直方圖與目標直方圖的權重比例，那麼理論上應先將 $h_{s,k}(i)$ 正規化，再重新按照權重計算 $h_{o,k}(i)$ 才會比較理想：

$$h_{o,k}(i) = \left(h_{s,k}(i) - \mu_{s,k}(j)\right)\frac{w_{t,k}\sigma_{t,k}(j)+w_{s,k}\sigma_{s,k}(j)}{\sigma_{s,k}(j)} + w_{t,k}\mu_{t,k}(j) + w_{s,k}\mu_{s,k}(j) \quad (2)$$

以上式(2)為我個人所想而調整後的公式，藉由減去 $\mu_{s,k}(j)$ 並除以 $\sigma_{s,k}(j)$ 對 $h_{s,k}(i)$ 做正規化，再乘上權重考量下的新標準差 $w_{t,k}\sigma_{t,k}(j) + w_{s,k}\sigma_{s,k}(j)$，以及新平均值 $w_{t,k}\mu_{t,k}(j) + w_{s,k}\mu_{s,k}(j)$ 計算出 $h_{o,k}(i)$，完全解決了 $w_{s,k} = 0$ 時發生的問題。

而對於後者問題的情況，考慮如果僅轉移一根 bin 的情形，那其實轉換後也應為一根 bin。因此不論前後都不會有標準差($\sigma = 0$)，這也使得我們可以直接將標準差項移除：

$$h_{o,k}(i) = h_{s,k}(i) - \mu_{s,k}(j) + w_{t,k}\mu_{t,k}(j) + w_{s,k}\mu_{s,k}(j) \text{ for } \sigma_{s,k}(j) = 0 \quad (3)$$

以上為對式(1)進行的調整，後續將在第 3 節比對式(1)與式(2, 3)不同方法下的成果。

然而，儘管此作法在結果上看起來已經明顯修正原方法的錯誤，但實際執行程式時如果將 B_{min} 設置的過小(約< 50)，仍會造成轉換後的直方圖有明顯的異常，為此我使用兩種解決方案：第一種方法是直接將 B_{min} 設置大一些(約 75)來解決；第二種方法則完全改寫 RegionTransfer()函數，使用簡單的線性組合方式做直方圖轉換：

$$h_{o,k}(i) = w_{s,k}h_{s,k}(i) + w_{t,k}h_{t,k}(i) \tag{4}$$

式(4)為我想法下的做法，與式(1)或式(2, 3)最大的差別在於，此轉換使得所有直方圖中的$h_{o,k}(i)$都以相同的比例介於$h_{s,k}(i)$與$h_{t,k}(i)$之間，不會有不穩定的轉換發生。後續亦將於第 3 節與式(1)及式(2, 3)進行比較。

C. For each level k in perc/S_{max}. 對於"level"的定義可以為小數的結論，我將倍率 k 的集合改為：

$$k = \{0.1S_{max}, 0.2S_{max}, \dots, perc * S_{max}\} \tag{5}$$

因此當 perc 設為 100%時，將會有 10 種可能的倍率與輸出。考慮到多樣性的輸出，最初希望能夠有 100 種可能的倍率，但在成果上會使得源直方圖太快收斂到目標直方圖的形狀，所以我最終採用式(5)的方案。

2.2 Python implementation

以下將逐步介紹我實作的 Python code 對應到 Pseudo code 中的各個部分。

A. ReshapeHistogram(Is, It, perc). 除了原本的源圖像 Is、目標圖像 It 以及轉換比率 perc 等參數以外，我加入了 Region selection [1] Eq.18 的權重 Wa、及是否使用式(4)算法的 simple_transfer 等參數。另外，如果 perc 為 0，則直接回傳源圖像。

```
def Reshape_Histogram(Is: np.ndarray, It: np.ndarray, perc: int, Wa: float=-1, simple_transfer=False) -> np.ndarray:
    if perc == 0: return Is
```

B. Convert Is, It to CIELab color space.

```
    source_lab = cv2.cvtColor(Is, cv2.COLOR_BGR2Lab)
    target_lab = cv2.cvtColor(It, cv2.COLOR_BGR2Lab)
    del Is, It
```

C. Initialization. 此步驟不在虛擬碼上，我將輸出圖像以及 Bins 進行初始化，並且根據[1] Eq.18 進行 Region selection。其中，mask 為 1 的部分才會參與整個過程及直方圖的匹配。值得注意的是同先前 1.2 節所述，此處實作是以 128 為中心計算。

```
    Io = np.empty_like(source_lab)
    Bins = np.array([CH_RANGE[ic][1] - CH_RANGE[ic][0] for ic in range(3)])

    mask = np.ones_like(source_lab[:, :, 0], dtype=int)
    th1 = Wa * (source_lab[:, :, 1].max() - source_lab[:, :, 1].min()) + np.min(source_lab[:, :, 1] - 128)
    th2 = Wa * (source_lab[:, :, 2].max() - source_lab[:, :, 2].min()) + np.min(source_lab[:, :, 2] - 128)
    mask[(abs(source_lab[:, :, 1] - 128) <= th1) | (abs(source_lab[:, :, 2] - 128) <= th2)] = 0
```

D. Compute Smax. 同上方 1.2 節所述。而此處我將 B_MIN 設為 70。

```
    Smax = compute_Smax(Bins, B_MIN)
```

```
def compute_Smax(Bins: np.ndarray, Bmin: int) -> np.ndarray:
    """ Compute Smax according to Eq.7 , Note: Smax has 3 variables for each channel """
    Smax = np.floor(np.log2(Bins / Bmin))
    return Smax
```

E. Compute histogram Hs, Ht from Is(ic), It(ic).

```
    for ic in range(3):
        Hs = histogram(source_lab[mask == 1, ic], CH_RANGE[ic], False)
        Ht = histogram(target_lab[:, :, ic], CH_RANGE[ic], False)
```

```
def histogram(img: np.ndarray, value_range: tuple, normalize: bool) -> np.ndarray:
    """ Compute histogram of input image """
    hist, _ = np.histogram(
        img, np.arange(value_range[0], value_range[1] + 1),
        (value_range[0], value_range[1]),
        density=normalize
    )
    return hist
```

F. For each level k. 根據式(5)的方式計算 k 的集合。STEP 設為 0.1。

```
        levels = list(np.arange(STEP, perc/100 + STEP, STEP) * Smax[ic])
        for k in tqdm(levels):
```

G. Down- and up-sample. 先計算倍率並決定 B_k，再進行下與上取樣。

```
            scale = round(k, 2)
            Bk = int(Bins[ic] * pow(2, scale - Smax[ic]))

            Hs_k, Ht_k = resample_histogram(Hs, Bk, True), resample_histogram(Ht, Bk, True)
            Hs_k, Ht_k = resample_histogram(Hs_k, Bins[ic], True), resample_histogram(Ht_k, Bins[ic], True)
```

H. Rmin ← FINDPEAKS(H, k). 同上方 2.1 節所述得到 R_{min} 集合。註：程式碼有兩行使用此函數，在此為方便展示而放在一起。(虛擬碼中的 Hs,k'為此處的變數 Hs_kp)

```
            Rmin_t = findpeaks(Ht_k)
            Rmin_s = findpeaks(Hs_kp)
```

```
def findpeaks(Hist: np.ndarray) -> np.ndarray:
    """ Search Rmin using Eq.8 & Eq.9 """
    H_grad = grad(Hist)
    H_grad2 = grad(H_grad)

    Rmin = []
    for i in range(H_grad2.shape[0]):
        if H_grad[i]*H_grad[i+1] < 0 and H_grad2[i] > 0:
            Rmin.append(i+1)

    Rmin.insert(0, 0)
    Rmin.append(Hist.shape[0])
    return np.array(Rmin, dtype=int)
```

```
def grad(arr: np.ndarray) -> np.ndarray:
    """ Compute gradients (Eq.8) """
    grad = np.empty(arr.shape[0] - 1, dtype=arr.dtype)
    for i in range(arr.shape[0] - 1):
        grad[i] = arr[i] - arr[i+1]
    return grad
```

I. **RegionTransfer(…) for each min-bound region.** 在 RegionTransfer 的函數中，我增加了一個參數可以選擇是否使用式(4)進行計算。註：程式碼有兩行使用此函數，在此為方便展示而放在一起。

```
            Hs_kp = np.empty_like(Hs_k)
            for m in range(Rmin_t.shape[0] - 1):
                Hs_kp[Rmin_t[m]:Rmin_t[m+1]] = RegionTransfer(
                    Hs_k[Rmin_t[m]:Rmin_t[m+1]],
                    Ht_k[Rmin_t[m]:Rmin_t[m+1]],
                    scale / Smax[ic],
                    simple_transfer
                )

            Ho_k = np.empty_like(Hs_k)
            for m in range(Rmin_s.shape[0] - 1):
                Ho_k[Rmin_s[m]:Rmin_s[m+1]] = RegionTransfer(
                    Hs_kp[Rmin_s[m]:Rmin_s[m+1]],
                    Ht_k[Rmin_s[m]:Rmin_s[m+1]],
                    scale / Smax[ic],
                    simple_transfer
                )
```

```
def RegionTransfer(Hs: np.ndarray, Ht: np.ndarray, wt: float, simple_transfer=False) -> np.ndarray:
    """ Region transfer function using adjusted Eqs.10~12 """
    ws = 1 - wt
    Hs_avg, Ht_avg = Hs.mean(), Ht.mean()
    Hs_std, Ht_std = Hs.std(), Ht.std()

    if simple_transfer:
        Ho = Hs * ws + Ht * wt
    else:
        Ho = Hs.copy()
        if round(Hs_std, 8) != 0:
            Ho = (Ho - Hs_avg) * (wt * Ht_std + ws * Hs_std) / Hs_std + wt * Ht_avg + ws * Hs_avg
        else:
            Ho = Ho - Hs_avg + wt * Ht_avg + ws * Hs_avg
    return Ho
```

J. **Io(ic) ← HistMatch(Is(ic), Ho).** 這邊考慮到 Region selection 的部分而有做些微調整。

```
        Hs = histogram(source_lab[mask == 1, ic], CH_RANGE[ic], False)
        Io[:, :, ic] = histogram_matching(
            source_lab[:, :, ic],
            Hs,
            Ho_k,
            CH_RANGE[ic],
            source_lab.shape[0] * source_lab.shape[1],
            mask
        )
```

```
def histogram_matching(Is: np.ndarray, Hs: np.ndarray, Ho: np.ndarray, value_range: tuple, pixel_count: int, Im: np.ndarray) -> np.ndarray:
    """ Performing histogram matching """
    Imin, Imax = value_range[0], value_range[1]
    Hs_sum, Ho_sum = Hs.sum(), Ho.sum()
    Hs = Hs * pixel_count / Hs_sum
    Ho = Ho * pixel_count / Ho_sum
    Ho, Hs = Ho.cumsum().round(), Hs.cumsum().round()

    new_pix_value = np.interp(Hs, Ho, np.arange(Imin, Imax))
    Io, Im = Is.ravel(), Im.ravel()
    Io[Im == 1] = new_pix_value[Io[Im == 1]]
    Io = np.reshape(Io, Is.shape).round()

    return Io.astype(dtype=np.uint8)
```

K. Others – Contrast modify. 在[1]中 3.3 節可見作者在細部操作上，對原始圖像和最終圖像使用 Bilateral filtering，用來改善對比度的做法。由於該方法在運算上花費時間較長，因此我改成 Median filtering 並採用同樣的方式消除明顯的對比度。wc 在此專題實驗中一律設為 0.1。

```
        output_img = Reshape_Histogram(source_img, target_img, perc, W_A, SHOW_HIST)
        output_img = contrast_modify(source_img, output_img, 0.1)
```

```
def contrast_modify(Is: np.ndarray, Io: np.ndarray, wc: float) -> np.ndarray:
    """ Smooth the image """
    Is, Io = Is.copy(), Io.copy()
    Is, Io = Is.astype(dtype=np.float32), Io.astype(dtype=np.float32)
    Ires_s = cv2.medianBlur(Is, 5)
    Ires_o = cv2.medianBlur(Io, 5)
    Io = Io + wc * (Ires_s - Ires_o)
    Io[Io < 0], Io[Io > 255] = 0, 255
    return Io.astype(dtype=np.uint8)
```

L. Others – Full reshaped Mean Absolute Error. 為了檢測實作結果的正確性，考慮 perc 為 100%完全轉換的特殊情況，理論上在源圖像直方圖上會被完整匹配到與目標圖像相同，也就是跟普通的 Histogram matching 有一樣的結果，因此我將兩種結果進行 MAE 的計算，雖然無法對所有可能的 perc 都做檢測，但至少仍可以驗證一種結果(perc=100%)的準確度。

```
    error = full_reshaped_MAE(source_img, target_img, LINEAR)
    print(f">> MAE of image with 100% reshaped histogram = {error}")
```

```
def full_reshaped_MAE(Is: np.ndarray, It: np.ndarray, simple_transfer=False) -> np.ndarray:
    """ Compute MAE of "100% reshaped histogram" & "matched histogram using histogram matching" """
    # Using paper's algorithm
    test = Reshape_Histogram(Is, It, 100, Wa=-1, simple_transfer=simple_transfer)

    # Normal histogram matching
    source_lab = cv2.cvtColor(Is, cv2.COLOR_BGR2Lab)
    target_lab = cv2.cvtColor(It, cv2.COLOR_BGR2Lab)
    Io = np.empty_like(source_lab)
    for ic in range(3):
        Hs = histogram(source_lab[:, :, ic], CH_RANGE[ic], False)
        Ht = histogram(target_lab[:, :, ic], CH_RANGE[ic], False)
        mask = np.ones_like(source_lab[:, :, ic])
        Io[:, :, ic] = histogram_matching(
            source_lab[:, :, ic], Hs, Ht, CH_RANGE[ic],
            source_lab.shape[0] * source_lab.shape[1], mask
        )
    truth = cv2.cvtColor(Io, cv2.COLOR_Lab2BGR)

    # Compute mean absolute error
    diff = test - truth
    mae = np.mean(abs(diff))
    return mae
```

3. Result comparison and discussion

實驗中，大部分的圖像都在 perc 大於 60%後呈現差不多的結果，因此我設置 perc 從 10%、15%、20%、40%到 60%與 100%等六個數值進行實驗。下方實驗參數設置為：STEP = 0.1, B_MIN = 70, W_A = -1

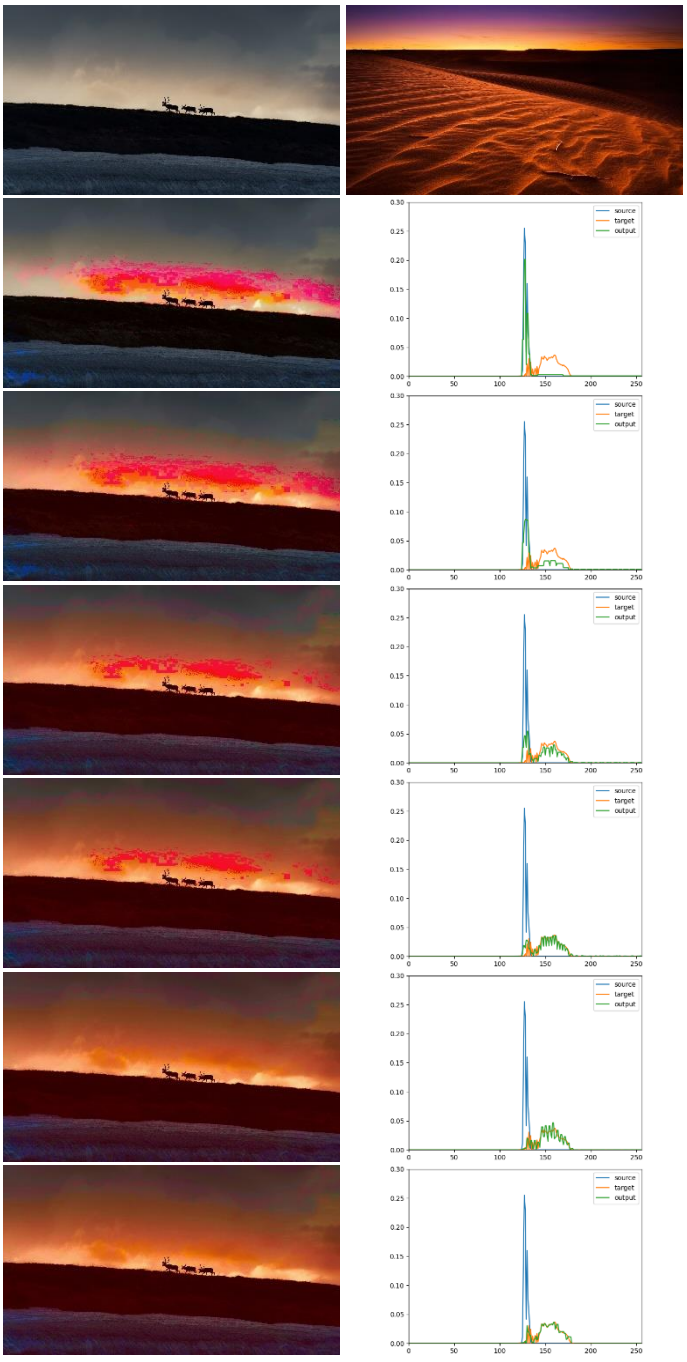

圖 3. 實驗成果 1。最上方兩張圖分別為源圖像(左)與目標圖像(右)。下方分別為 perc = 10%, 15%, 20%, 40%, 60%, 100%的輸出圖像(左)與 a*頻道上的直方圖匹配結果(右)。此實驗結果 MAE = 24.43。

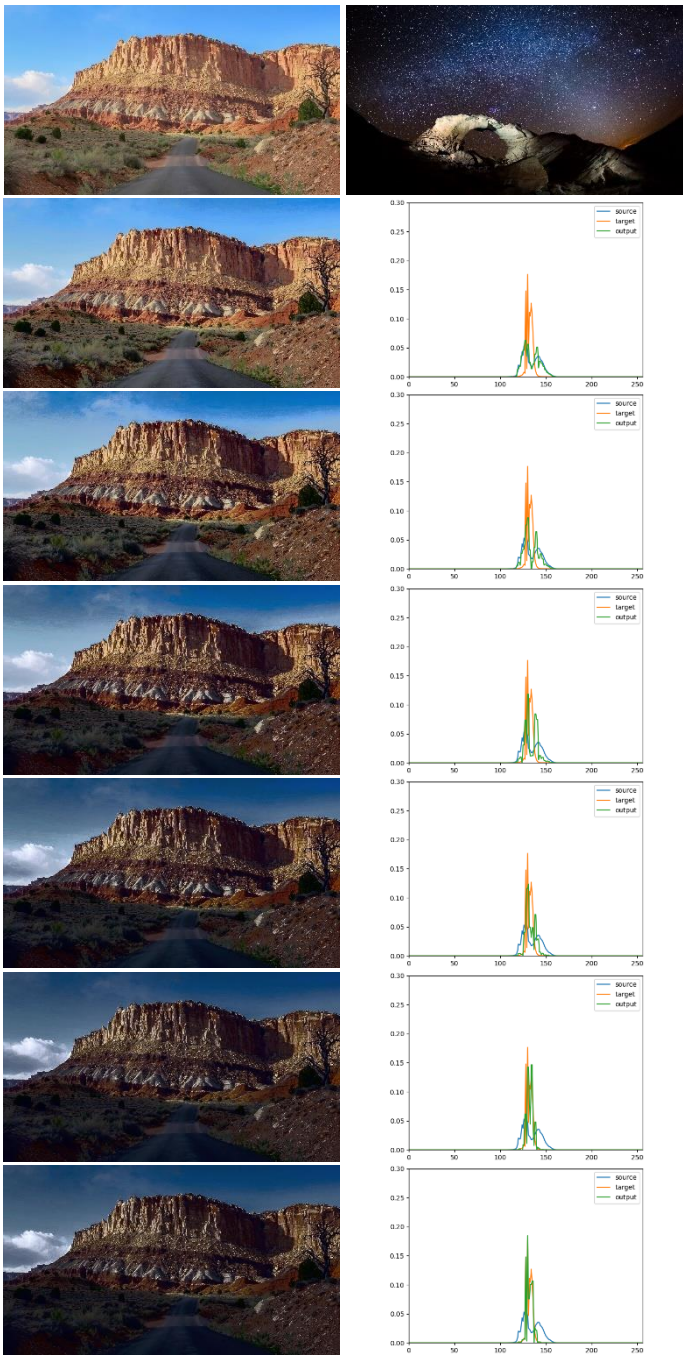

圖 4. 實驗成果 2。最上方兩張圖分別為源圖像(左)與目標圖像(右)。下方分別為 perc = 10%, 15%, 20%, 40%, 60%, 100%的輸出圖像(左)與 a*頻道上的直方圖匹配結果(右)。此實驗結果 MAE = 38.22。

由圖 3 可見 perc=10%, 15%, 20%, 40%時，輸出圖像中有許多明顯的方型區塊顏色較顯突兀。造成這樣的原因，是受到 R_{min} 集合不同的影響，而造成轉移平均與標準差時，轉移後應該為零的區域卻不為零(圖 5)。

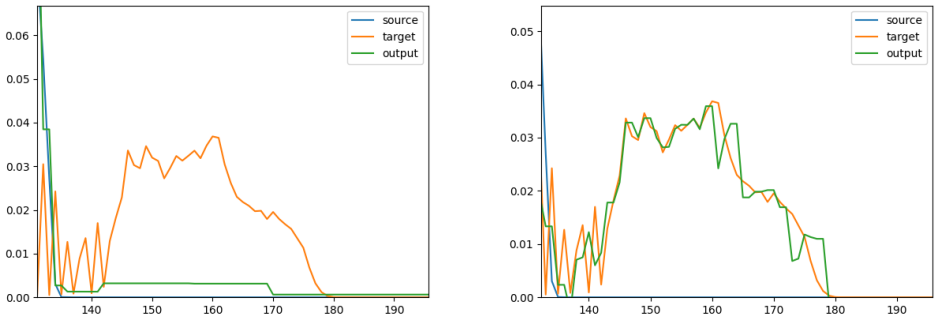

圖 5. a*頻道上的直方圖比較。左圖為 perc=10%、而右圖為 perc=100%的情況，圖中“output”直方圖在數值大於 180 的位置有明顯的差異，將會造成最後的 transfer function 有明顯的差異。

對於此問題造成的原因，是出在演算法本身使用 R_{min} 集合搭配轉移平均與標準差的方式，這種方式會使的集合所切出來的區塊有機率發生錯誤的轉移，此錯誤所指的是：在源圖像(圖 5 的“source”)直方圖及目標圖像(圖 5 的“target”)直方圖在大於數值 180 以上的 bins 為零的區域，卻轉移出非零的結果。因此我了解到最好的解決方案，就是使用式(4)的計算方法進行轉換，當兩者皆的 bins 為零時，絕對不可能轉移出非零的成果。

圖 6. 式(4)對於 perc=10%(左上), 15%(右上), 20%(左下), 40%(右下)的輸出圖像。對比於圖 3 有明顯改善(顏色突兀區塊)的結果。

即使這樣的結果看似完全否定了[1]的 Region transfer 想法，而可能會無法達到漸進式轉換的效果。不過實際上，此算法仍可以藉由不同倍率($perc*S_{max}$)下之 B_k 所進行的下取樣再上取樣來達到漸進式的成果，因此我認為此方案即使看似簡單的線性組合，卻是目前相對而言最好的計算方式。

接下來討論對於 Region selection 的實驗結果，我將參數設置為：STEP = 0.1, B_MIN = 70, W_A = 0.15

圖 7. 實驗成果 3。最上方兩張圖分別為源圖像(左)與目標圖像(右)。下方分別為 perc = 10%, 15%, 20%, 40%, 60%, 100%的輸出圖像(先左至右、後上至下)。

Region selection 如圖 7 所示，圖像中老虎的白色毛皮區塊沒有參與色彩轉移過程。在 perc=10%時，主要為老虎的黑色斑紋被轉換為紅色；perc=20%至 100%則漸漸將背景草皮轉換為與目標圖像相同的櫻桃色，成果非常理想。

以上討論完此 Python 實作內容的所有功能，最後將比對回原本[1]的計算方法，即式(1)與我在此報告中所想的式(2)(3)及式(4)的成果比對：

圖 8. 比對測試用之源圖像(左)與目標圖像(右)。

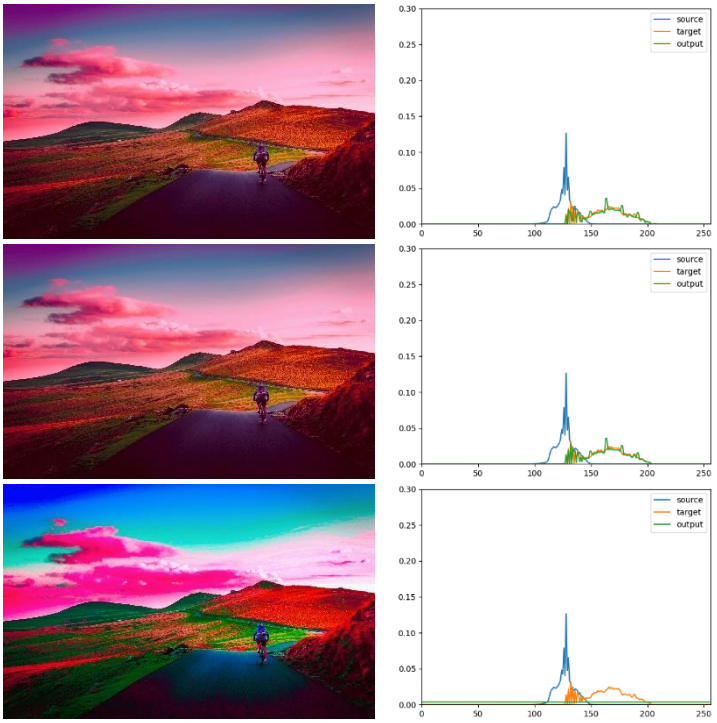

圖 9. 比對不同 Region transfer 算法之成果。由上至下分別為式(2)(3)；式(4)；式(1)所得的輸出圖像。右方三張圖皆為 a*頻道上的直方圖成果。

圖 9 顯示的成果中，可見式(1)的輸出圖像顏色明顯異常，從直方圖也能看到變成類似於 Histogram equalization 的成果。此外，當 perc=100%時，會發生先前於 1.3 節中討論的問題，而得到圖 10 的結果，此結果驗證了式(1)所可能發生的問題。

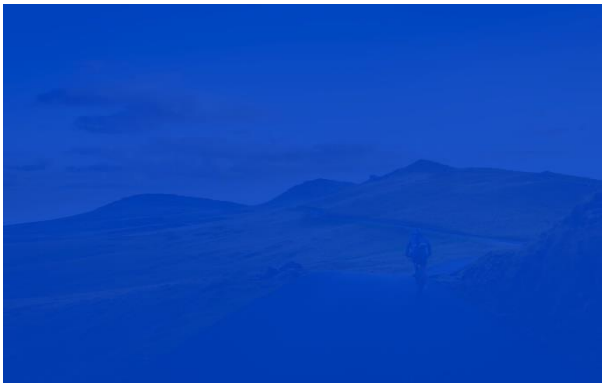

圖 10. 式(1)對於 perc=100%時的輸出圖像。MRE=119.91，整體呈現藍色的畫面。

4. Conclusion

在此實作專題中，參照[1]的演算法並經過許多調整和修改後，我完整實現了[1]的內容，從漸進式的色彩轉移、圖像區域選擇，到對比度的修正都有實作於 Python 程式碼中。

而在此報告中，首先於 1.1 節簡單探討論文[1]的內容與實作目標；並於 1.2 節釐清原先沒有被清楚敘述的部分；再於 1.3 節探討[1]的演算法會發生的問題；並在 2.1 節描述我解決問題的方式；最後，則是 2.2 節的 Python 程式碼說明，以及第 3 節的成果比對，並在第 3 節內說明式(4)為最佳計算方法的原因。

經過這次的專題實作，讓我瞭解並學習到了色彩轉移的演算法，由於我未來將至他校繼續往這方面(影像處理)學習與發展，所以對我來說是一個很有幫助的經驗。

然而，在這次的實作過程中，有太多的疑點與部分敘述矛盾充斥於論文[1]中，使得我的實作內容逐漸傾向獨自發想並修改其中的算法。讓我不太確定這樣的成果是否真的有達到[1]的目標。不過我認為很有趣，謝謝教授與助教這學期的教學及用心。

5. References

[1] Pouli, T., & Reinhard, E. (2011). Progressive color transfer for images of arbitrary dynamic range. *Computers & Graphics*, *35*(1), 67-80.

[2] Converting an OpenCV BGR 8-bit Image to CIE L*a*b* [https://stackoverflow.com/questions/11386556/converting-an-opencv-bgr-8-bit-image-to-cie-lab]

註：此報告內容與內含程式碼皆由 電機四 陳柏翔 4109061012 獨自完成。