Computer Vision HW3 Report

102062209 邱政凱

這次的作業要求我們實作的是 Image Segmentation。

第一題要我們用 K-means Clustering 的方式去把圖像做分群。K-means clustering 的基本概念很簡單，就是藉由一開始隨機散佈 k 個起始的群中心，把色彩空間中所有像素都歸類到最近的群中心，接著再分別把每個群中所有元素的平均值求出來，當成新的群中心，然後繼續把所有像素歸類到最近的新的群中心….不斷重複這樣的過程直到收斂。

第一題 A 部分要我們分別設定 3、7、11 個群並且個別做 50 次的 initial random sampling，然後最後個別取出 objective function 值最小的一次結果。一開始隨機取樣的部分我是用 k-means++，也就是比較均勻分布的起始隨機取樣方式。整個實作過程其實不算太複雜，但是第一次我實做出來的結過用了 4 個 for 迴圈，執行起來的速度慢的可觀，根本不可能跑出結果。後來重新改 code，盡可能把需要迴圈的東西用 Matlab 的矩陣和向量直接做運算以加速，最後終於把執行時間大幅壓縮了。(11 個群 pick 50 次大約 3~5 分鐘可以完成)。分割完之後，我把每個群的所有像素都用該群的中心點的顏色做替換。

以下是我的分群結果：

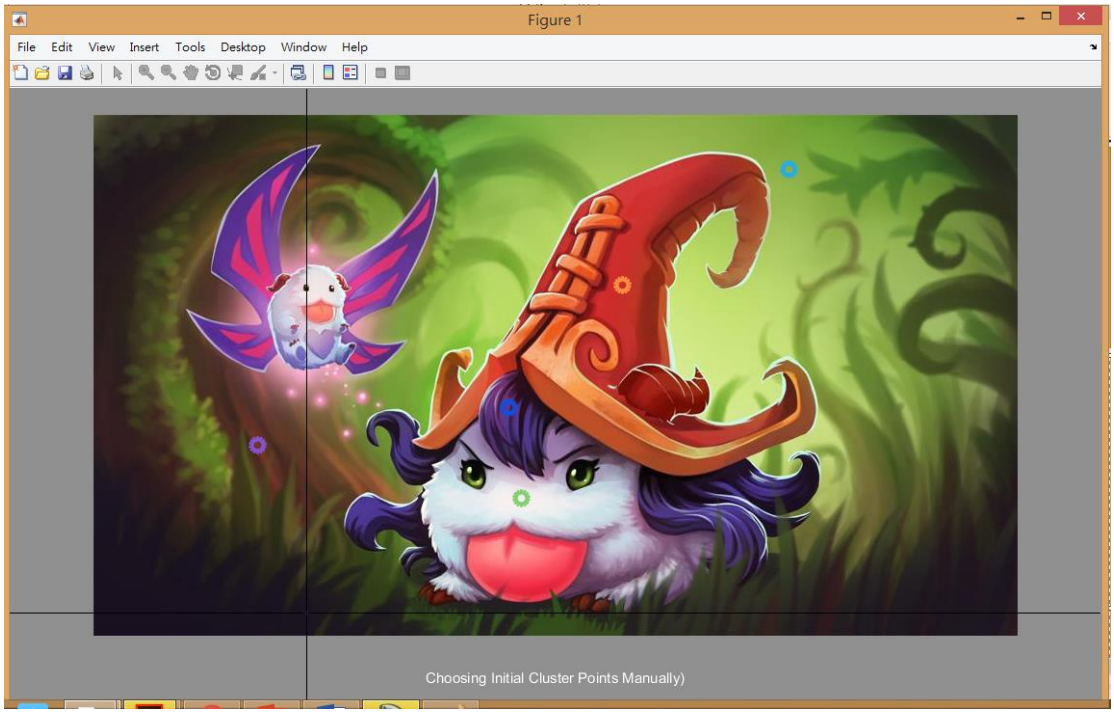
K=3, 50 Picks。

K=7, 50Picks。

K=11，50Picks。

我用這張圖顯示了 K=11 的分群中我所找到的 11 個群中心點的像素。

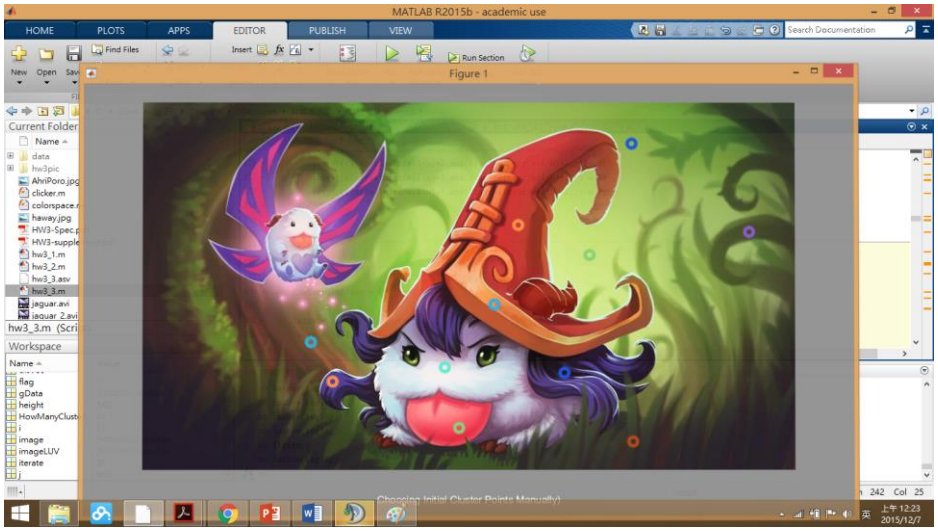
第一題的 B 部分則要我們直接手動選取一開始的群起始點，而非用 K means++ 的方式隨機選取再取最小 Objective function 的 Pick。
以下是我的結果：



(這是我的起始群中心選取介面，圖中有圓圈的點表示我的選取點)

手動選取 k=3 的 initial center 的結果。

手動選取 k=7 的 initial center 的結果。



K=11，這邊展示了我手動選取的 11 個點(圖上有圓圈的位置)。

而這張圖展示的就是上面那張圖的 11 個點當起始中心最後分群出來的結果。

這是上面那張圖收斂後的 11 個群中心點元素。

C 的部分則是要我們把 A 部分跟 B 部分用 LUV Color space 再做一次。我使用網路上開放的 code 來做色彩空間的轉換。(不過我只有單向轉換，因為我只要知道那些元素是最後收斂的群中心就好了，之後直接從原 image 把對應像素取出就好，不需要再把 Luv 轉回 RGB)。

以下是 A 部分用 LUV 重做的結果：

Luv color space，K=3，50 Picks。

Luv color space，K=7，50 Picks。

Luv color space，K=11，50 Picks。

D 部分，比較：

我們可以看到把圖片分成 3、7、11 群的 segmentation 結果各自大相逕庭。K=3 的狀況下顏色幾乎無法反映出原圖的色調和色彩分布，頂多只能藉由深淺稍微猜想原圖的明亮分布。而到了 k=7，已經可以看出原圖大致上的色調和色彩感了，不過還是可以看到很多部分因為群數不足互相共用顏色而顯得有些突兀。不過到了 k=11，除了原圖色彩感有出來外，也有了一點漸層的感覺，顏色間的區別也變得更明顯了。

另外是 Random Pick 的部分。由於我的 random Pick 是採用了 50 次的 initial guess，再取最佳解出來。而手動選取的部分我主要是選擇比較大塊或主要的顏

色當作起始點(相信這也是比較好的 GUESS)。於是我們可以看到，儘管是手動選取的點，最後做出來收斂後的結果其實跟隨機選取多次取最好的一次的結果差不多，可以想見其實 K MEANS 分群的過程近似的起始點會收斂至相似的解。(雖然我 K=11 用紅框標出來的最終中心點在空間上的位置有點差別，不過仔細看一下就可以發現其實他們對應的像素顏色幾乎都是一樣的。)

另外是 Luv color space 的部分。K=3 的時候我們發現整個結果跟 RGB 完全不一樣，而且有些畫面結構會因為這樣的分群而模糊掉。K=7 的時候畫面結構回來了，可以看出原圖大致的色彩感和色調了。K=11 的時候畫面的細節又跑出來更多了。依據我的觀察，我認為 Luv 跟 RGB 的分群結果的差別在於(以 K=11 當範例)：RGB 分群的結果比較有連續感，整個畫面色彩給人的感覺是比較滑順(看怪物的毛、頭髮以及後面背景的植物漸層可以看出)，然而 RGB Color space 必然會收斂到比較大群的色彩元素的中心，一些比較少但是也很重要的色彩資訊就可能遺失。然而 Luv 卻可以保留畫面中一些也很重要但是數量比較少的顏色(像是左方棕色的樹、怪物的舌頭反光、怪物的指甲)。

第二題要求我們把影片中的前景和背景用 K means 分離並且替換成自己選定的圖片。

我的做法很簡單，就是對影片中每個 frame 都用 kmeans++，10picks 取最佳的分群解果。不過這題有個 trick—該怎麼知道你分出來的群哪個是前景哪個是背景？我的做法是跑這支程式的時候會讓 user 選擇三個代表前景的顏色點和三個代表背景的顏色點。分群完的結果，把每個群中心去計算跟三個前景點的 total 距離和三個背景點的 total 距離，前景點距離比較小的話就把該群標記為前景群，反之標記為背景群。之後再遍歷整張圖片，把每個被標記為背景群的群中像素都直接替換成我的圖片(圖片大小要先縮放成跟原影片一樣大)。
不過我自己有發現這種作法一些問題，像是 k 的數量選得不好結果會有問題。一開始 User input 的前後景代表點也不能隨便亂抓，不然結果也會爛掉。

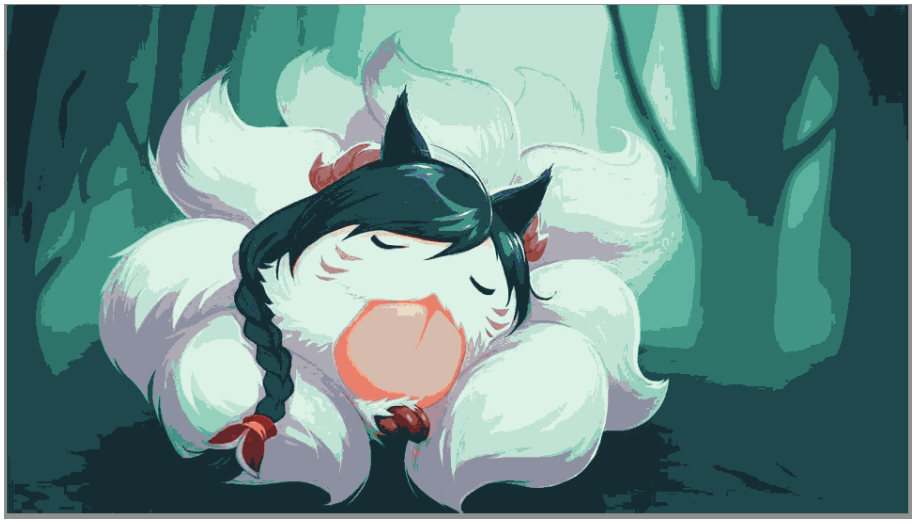
第三題則是要求我們用 Mean Shift 去做 Image segmentation。

Mean Shift 的概念就是針對每個像素，去計算每個落在 band Width 內的所有其他元素，並求出這些落在 bandwidth 內的原素的平均值，並且把當前 window 的中心點一道這個新的平均值上，重複這個過程直到收斂(window 移動距離<epislon)。以上作法是因為題目要求我們做的是 Uniform kernel 而做的化簡，若是其他 kernel，落在 bandwidth 內的元素要算的是一個加權的平均值。Mean shift 的好處就是不需要使用者指定要有幾個群，而且參數只有一個，也就是 kernel bandwidth，而且有明確的物理意義。

A 部分叫我們直接再 RGB 三維空間上做 Mean shift。因為我的圖片是先做過 im2double normalize 到 0~1 的，所以 bandwidth 也會比較小一點。另外就是如果逐象素去做 mean shift，每個象素都要迭代數十次才會收斂，這樣要跑過每個像素實在是太花時間了，所以我做了一點加速，也就是每次 mean shift 收斂到空間中某點後，距離該收斂點 h(kernel bandwidth)以內的所有元素，我直接假定他們最後都會收斂到這同一個收斂點上。同樣地，分群完之後我把屬於每個群的所有象素都替換成該象素所屬的群中心的象素顏色。我的收斂條件是 mean shift vector 的 norm 平方小於 0.1。

以下是結果：

Bandwidth h = 0.3。



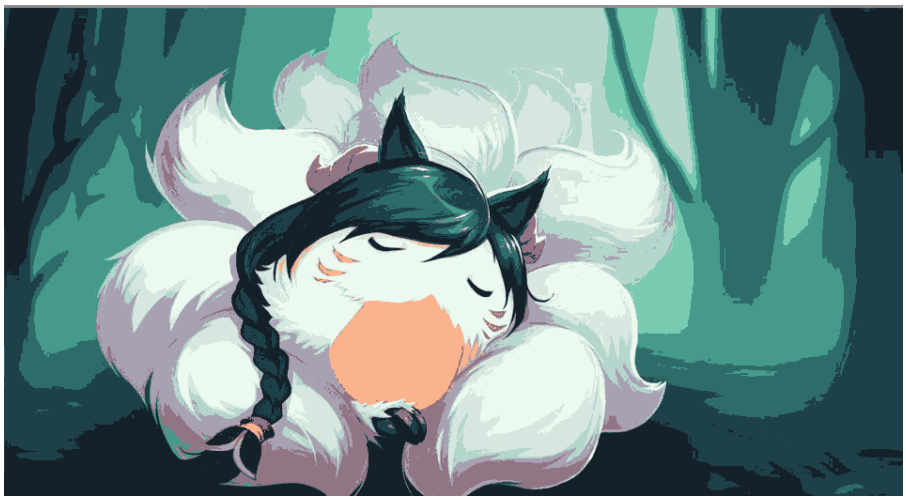
Bandwidth h = 0.2

Bandwidth h=0.1。

這邊要再提一下，用 meanshift 對所有象素點都找到對應的收斂點之後，比較所有的收斂點，若有收斂點之間的距離小於 bandwidth h 的距離，則把兩個收斂點 merge，對應的群也融合成一群。以上 h=0.3 時大約有 10 個群、h=0.2 時大約有 17 個群、h=0.1 時群數已經破百，分群就有點沒意義了。

依我看來，h=0.2 算是個比較合理又適當的 kernel bandwidth。

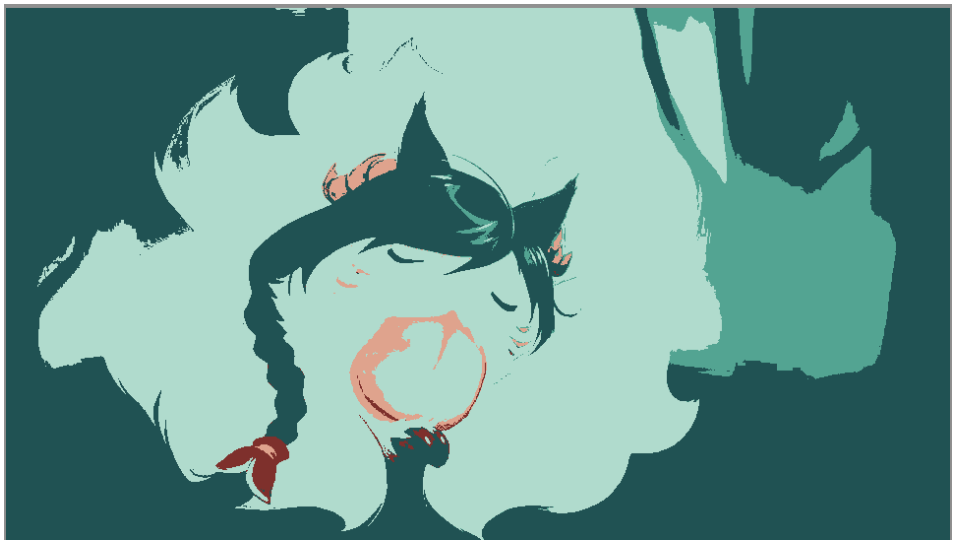
為了比較，我要在這邊把 h=0.3(10 個群)的結果跟用 kmeans，k=10 的分群做個簡單的比較：



雖然說兩種演算法分群的規則不同，也不能說讓 mean shift 的最後群數跟 k means 一樣就可以直接比較，不過我們大致上還是可以看出來，mean shift 可以保留數量少但是也很重要的顏色(辮子上的紅色裝是，舌頭的陰影…)。而 K means 則容易忽略這些比較少但是很重要的顏色。

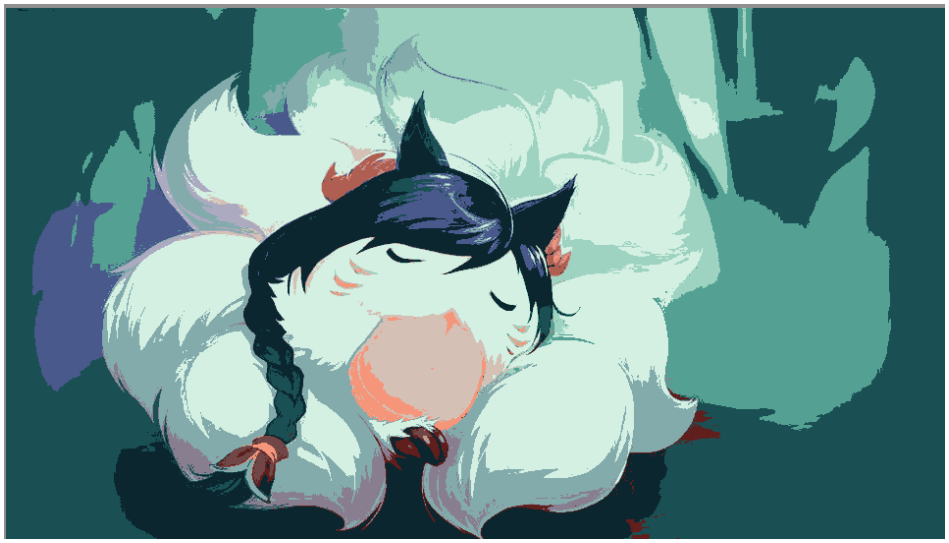
B 部分則要我們把 Spatial feature 也考慮進去，對五維向量(R,G,B,X,Y)做 Mean Shift。我的做法是把 X,Y 直接做 Normalize，也就是把空間資訊轉換到 0~1 之間(跟 double image 的 range 一致)，然後直接用同一個 bandwidth，而非把 color 跟 spatial 的 bandwidth 分開來計算。

以下是結果和比較(D 部分)：



Bandwidth h=0.5 ◦ k~= 5

Bandwidth h=0.4 ◦ k~=8



Bandwidth h=0.3 ◦ k~=14

Bandwidth h=0.2　k~=35

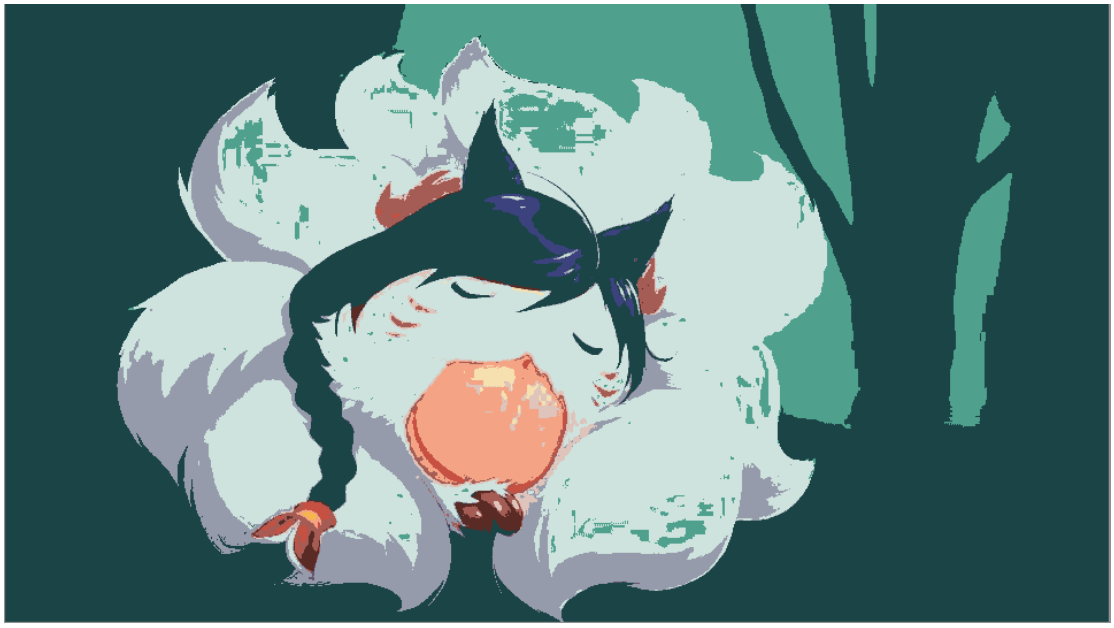
Bandwidth h=0.1 k>100

依我感覺，h=0.3 是個比較適切的 bandwidth。

不同 h 比較的部分，我們可以很直接地看出隨著 h 的變小，分群的數量會越來越多，代表著隨著 bandwidth 的減小，每個點越來越不容易偵測到同樣的 mean shift vector，也就越來越不容易收斂到同一點，造成了群數變多。畫面結構也會越來越清楚，不過當群數過多分群的行為就不太有意義了。然而我們可以看到就算是 h=0.5，k 只有 5 群的狀況下，mean shift 還是很堅持要保留髮帶的顏色(汗)

加入 spatial feature 之後的一個特點可以從 Bandwidth h=0.2 跟 part a 的 h=0.3 的比較看出來。Part A 是 17 群，Part B 只有 14 群，不過 Part B 的部分我們可以看到頭髮不小的藍色部分被保留了，可以想成是因為再 mean shift 收斂的過程有考慮到鄰近性的問題，而不是一味的往偏綠色的地方收斂過去。

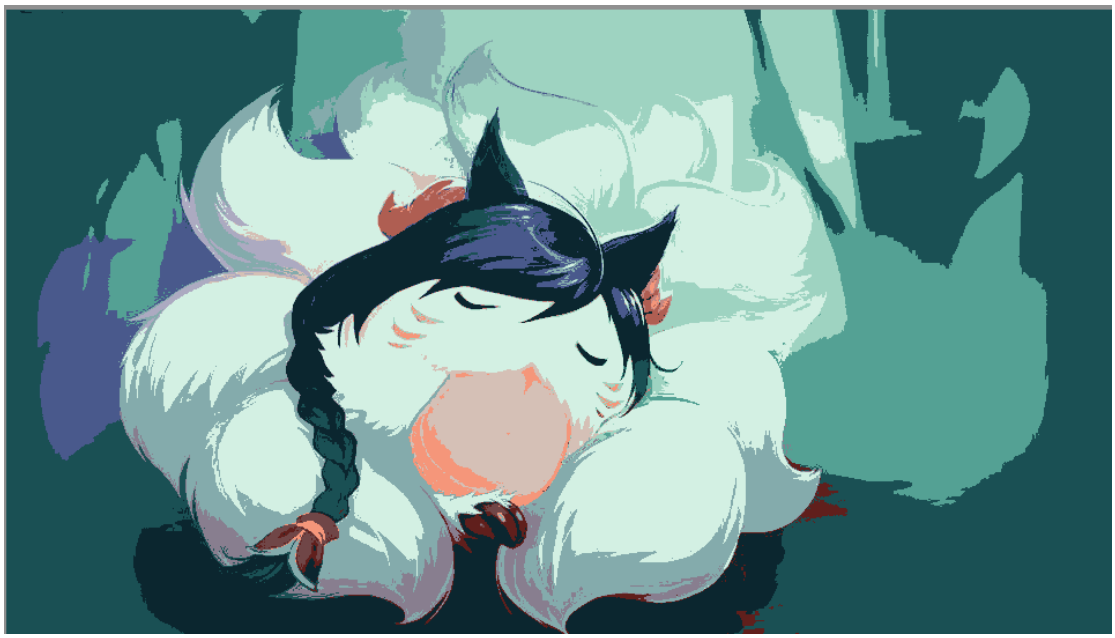
C 部分則要我們再針對 A 部分和 B 部分用 LUV Color space 再做一次。跟第一題不一樣的是換了個 Color space 之後 bandwidth 要重新選取。
我的做法是把圖片從 RGB 轉到 Luv 之後先對 Luv 的數值做 normalize 到 0~1 之間，分群完之後在反 normalize 回去，最後再轉換回 RGB。



這是 3-A 用 Luv 做的結果。K=12。

跟原本的比較起來算是再幾乎一樣的 k(分群數量)中找出了更多的特徵顏色，不過也可以看到背景的漸層感不見了，這邊的結果跟第一題 k means 分群時的 rgb luv color space 的比較不謀而合。

這是 3-B 用 Luv 做的部分，h=0.15。這邊的 k=20。



和原圖的 h=0.3(k=16)做比較，可以發現整個畫面的顏色比較有跳躍感和飽和感，而不像原圖的顏色平滑、色調一致。