Computer Vision HW3 Report

作者：105062509 / 羅右鈞

註：此 report 原先在 dropbox paper 上編輯，直接到 https://paper.dropbox.com/doc/Computer-Vision-HW3-Report-4tRkfkMKrFXet0F3Efu1G 就能看到更美的排版。

函式說明

此部分解釋各個 part 都會用到的 function，像是 kmeans, mean shift 等主要的演算法，而這些 function 都會集中在 Class Segment.m 中。

註：另外還有與其他 part 主程式同階層目錄的 “RGB2Luv.m”, “Luv2RGB.m” 是用來做 RGB, Luv 之間的轉換，此部分並不詳細解釋。

---

function [img_segmented, color_bar] = optimize_kmeans(img, num_cluster, num_epoch, init_centroids)

- 功能概述：這個 function 是 kmeans function (主要跑 kmeans 演算法的 function，會在下面詳述) 的 wrapper，主要為 part 1 所用，可以指定跑 num_epoch 次的 kmeans，並從中選取 cost 最低 (根據 sum squared distance 的 objective function)，也就是 performance 最好的 kmeans 當作最終結果。
- 輸入變數：
  - img：要 segement 的圖，真正在處理的時候會 reshape 成 N 筆 D 維 的 2D 資料
  - num_cluster：cluster 的數量
  - num_epoch：指定要做幾次 kmeans
  - init_centroids：centroids 的初始值，如果沒給，就 random 產生。size 為 KxD
- 輸出變數：
  - img_segmented：segemented 過的結果圖
  - color_bar：centroids (cluster center) 的顏色圖

[centroids, cluster_indexs, min_distances] = kmeans(img, num_cluster, init_centroids)

- 功能概述：加速過的 kmeans 演算法
- 輸入變數：
  - img：要 segement 的圖，真正在處理的時候會 reshape 成 N 筆 D 維 的 2D data points (N 為 pixel 數量)
  - num_cluster：cluster 的數量，即 K

    - init_centroids：centroids 的初始值，如果沒給，就 random 產生。size 為 KxD
- 輸出變數：
    - centroids：cluster centers 的顏色，size 為 KxD
    - cluster_indexs：每個 data point (圖片的話就是每個 pixel) 所對應的 cluster index，size 為 Nx1
    - min_distances：每個 data point 與屬於它們的 cluster center 的距離，size 為 Nx1
- 做法：
    - 使用 initCentroids 的 helper function (會下面做更多的詳述) 從 data points 中 random sample K 個 data points
    - 對每個點都去找離他最近的 centroid，距離是根據 sum squared distance 的 objective function，找到後就更新它們的 cluster index
    - 對每個 cluster 計算 mean 值，並且當作新的 centroid
    - 如果 centroid 有變化，則重複上述動作 (步驟 2, 3)，如果沒有變化就停止 (這邊可以設定 threshold，也就是變化如果小於某個 threshold 就停止，但實務上不用設定 threshold 都能夠收斂)
- 補充說明：這個 function 是加速過的 kmeans，加速方法就是將所有能夠 vectorized 的計算都 vectorized，也就是減少迴圈，變成矩陣運算，還有用 mask 來更新矩陣的 tricks。我有留著第一次實作但沒加速過的 kmeans，放在與其他 part 程式同目錄下，叫做 “slow_kmeans.m”，留著給助教做比對。

---

function centroids = initCentroids(points, k)

- 功能概述：helper function，從 data points 中 random sample k 個 data points
- 輸入變數：
    - points： NxD 的 2D 資料（N 為資料筆數，D 為資料維度）
    - k：sample 個數
- 輸出變數：
    - centroids：k 個 D 維的 sample points

---

function [centroids, cluster_indexs] = mean_shift(img, color_bandwidth, spatial_bandwidth)

- 功能概述：加速過的 mean shift 演算法
- 輸入變數：
    - img：要 segement 的圖，真正在處理的時候會 reshape 成 N 筆 D 維 的 2D data points (N 為 pixel 數量；因為這題有考慮到 spatial information，因此 D 可能有 5 個，即 R, G, B, X, Y)

    - color_bandwidth：R, G, B 三個維度的 distance 的 bandwidth，這個 bandwidth 其實可以視作以某個點為圓心，圓圈之內的半徑，超出這個半徑的點就不考慮一起算 mean。
    - spatial_bandwidth：X, Y 兩個維度的 distance 的 bandwidth，這個 bandwidth 其實可以視作以某個點為圓心，圓圈之內的半徑，超出這個半徑的點就不考慮一起算 mean。如果沒給，那就只考慮 color bandwidth。
- 輸出變數：
    - centroids：cluster centers 的顏色，size 為 KxD (K 為 cluster 數量，D 為資料維度)
    - cluster_indexs : 每個 data point (圖片的話就是每個 pixel) 所對應的 cluster index，size 為 Nx1
- 做法：
    - 從 unvisited points 中選出一點做 mean shift（每一個被選出的點來做 mean shift 其實就是產生一個 cluster，到後面會跟已產生的 cluster 計算距離，若距離太近就合併）
    - 做 mean shift 跟 kmeans 很像，以選出的這一點為初始的 cluster center，找出在 color bandwidth 與 spatial bandwidth 以內的所有 data points，對他們計算 mean 當作新的 cluster center (質心) ，物理意義上就是往密度高的方向移動，並把它們都標註 visited 且紀錄 (累計) 它們被 visited 的次數，即 visit count
    - 重複上述步驟一直到這個質心收斂為止，收斂的質心其實就是 cluster 的 centroid
    - 跟先前產生出來的 cluster center 比較距離，如果在 color bandwidth 與 spatial bandwidth 之內就合併，如此可以把相像的點是做同個 cluster
    - 完成了 mean shift 之後，開始為每個 visited point 更新 cluster 種類，以在某個 cluster 中最多次的 visit count 視為 visited points 屬於該 cluster
    - 如果還有 unvisited points，重複整個步驟直到所有的 points 都被 visited 過

---

function img_segmented = slow_recreateImage(img, centroids, cluster_indexs)
function img_segmented = recreateImage(img, centroids, cluster_indexs)

- 功能概述：helper function，從 centroids, cluster_indexs 重建 image
- 補充：recreateImage 是加速版的 slow_recreateImage，將兩層迴圈變成矩陣運算，slow_recreateImage 在本作業並不會用到，而是都用加速版的 recreateImage 來重建 image。

---

function color_bar = createColorBar(centroids)

- 功能概述：helper function，將 centroids 重建成 color bar 圖片，跟 img_segmented 一起呈現結果。

各題結果

Part 1-A

做法：使用 Segment.optimize_kmeans，參數分別給題目指定的參數下去跑

結果：

- K = 3 ; 跑 50 次選出最好 (檔案位置：”result_images/part_1a_result1.jpg”)

- K = 7 ; 跑 50 次選出最好 (檔案位置：”result_images/part_1a_result2.jpg”)

- K = 11 ; 跑 50 次選出最好 (檔案位置：”result_images/part_1a_result3.jpg”)

Part 1-B

做法：使用 Segment.optimize_kmeans，參數分別給題目指定的參數下去跑，差別在於事先手動指定 centroids 的 RGB 值

結果：

- K = 3；手動初始 centroids (檔案位置："result_images/part_1b_result1.jpg")

- K = 7；手動初始 centroids (檔案位置："result_images/part_1b_result2.jpg")

- K = 11；手動初始 centroids (檔案位置：”result_images/part_1b_result3.jpg”)

Part 1-C

做法：使用 Segment.optimize_kmeans，參數分別給題目指定的參數下去跑，這次跑 kmeans 之前事先使用 RGB2Luv() 將 color space 由 RGB 轉成 Luv，最後再轉回 RGB 將結果呈現

結果：

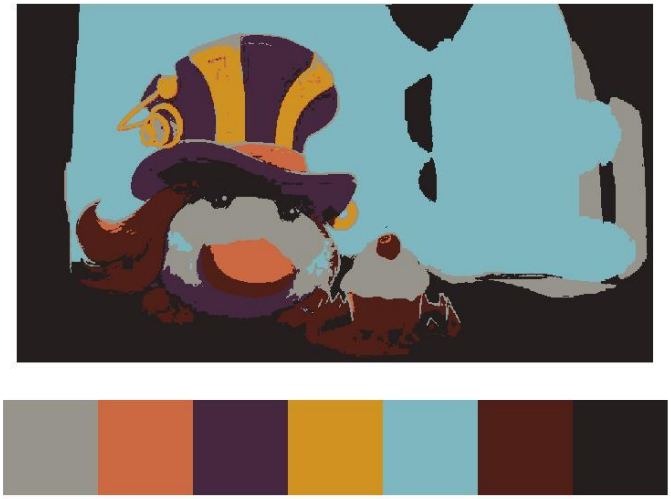
- K = 3 ; 跑 50 次選出最好 (檔案位置：”result_images/part_1b_result1(rand50).jpg”)

- K = 7 ; 跑 50 次選出最好 (檔案位置：”result_images/part_1b_result2(rand50).jpg”)

- K = 11 ; 跑 50 次選出最好 (檔案位置：”result_images/part_1b_result3(rand50).jpg”)

- K = 3；手動初始 centroids (檔案位置：”result_images/part_1c_result4(manual).jpg”)

- K = 7；手動初始 centroids (檔案位置：”result_images/part_1c_result5(manaul).jpg”)



- K = 11；手動初始 centroids (檔案位置：”result_images/part_1c_result6(manual).jpg”)

Part 1-D

由實驗結果可看出使用 Luv 會比使用 RGB 的切割效果來得好，尤其從這張圖來看，舌頭的部分有著滿顯著的差異，至於為何 Luv 會比較好，直觀上是因為 Luv 會比 RGB 接近人類視覺 model，因此許多 vision 的應用不太會使用 RGB 當作 feature，而是會使用更接近人類視覺的 color space feature。而 K 的部分從實驗結果可以看出對切割效果影響很大，如果要將圖片有很好切割的效果，K 不能選太少，但過多也不好，等於有切跟沒切差不多，而 K 越大，kmeans 也會需要跑更多的時間。

Part 2-A

注意：因為我的 matlab 版本較舊，不支援 VideoReader, VideoWriter，因此以 mmreader, addframe 完成此項作業 (有與助教確認過可以使用)

做法：

- 使用 mmreader()，與 read 將影片轉成 4-D uint8 的 data (前三個維度跟一班圖片一樣，第四個就是時間，有很多張圖片)
- 事先手動選兩個 centroids color，分別是前景的黃色，背景的藍色

- 對每個 image 跑 Segment.kmeans()，跑完後得到一張前後景都能分離的圖片，再利用 mask 的技巧把前景貼到新的背景圖，並使用 im2frame() 將 image 轉成 frame 後，用 addframe() 加到 avi file 中

結果： 由於影片檔案超出 100 MB 無法上傳，我另外上傳至 google drive，助教可點選連結 https://drive.google.com/open?id=0B2waMVL4IgWqTFRJWDVLdjhUU28 前往觀看，
或是跑一次程式，影片會在同目錄下生成，檔名為 “part_2a_result.avi”。

Part 2-B

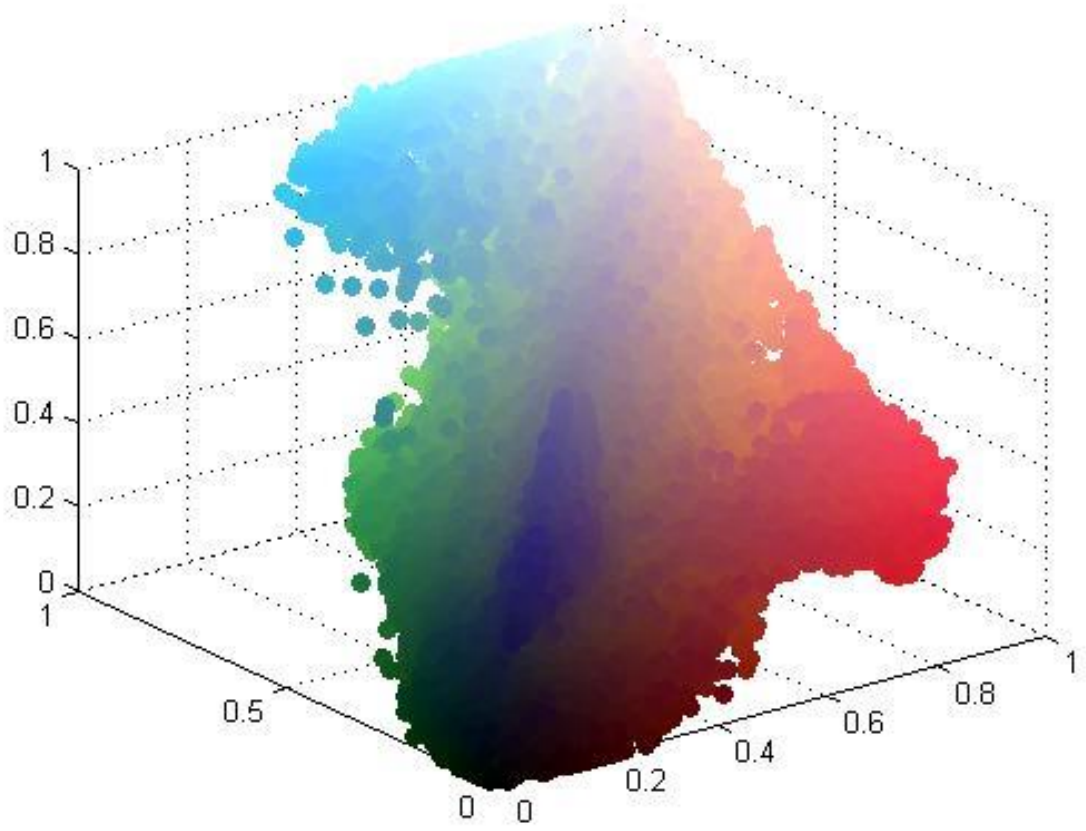
在這次作業中所使用的 kmeans 都已經使用 vectorized implementation 加速過，在解釋 function kmeans 時有提到，可去程式碼中看看。主要就是把 centroids 使用 repmat 變成與 data points 同 size 的矩陣，就可以直接算出每個 data points 到 centroids 的距離，再利用 mask 的技巧更新 centroids 的值，避免使用 for loop 對每個 data point 算它到 centroid 的距離。

Part 3-A

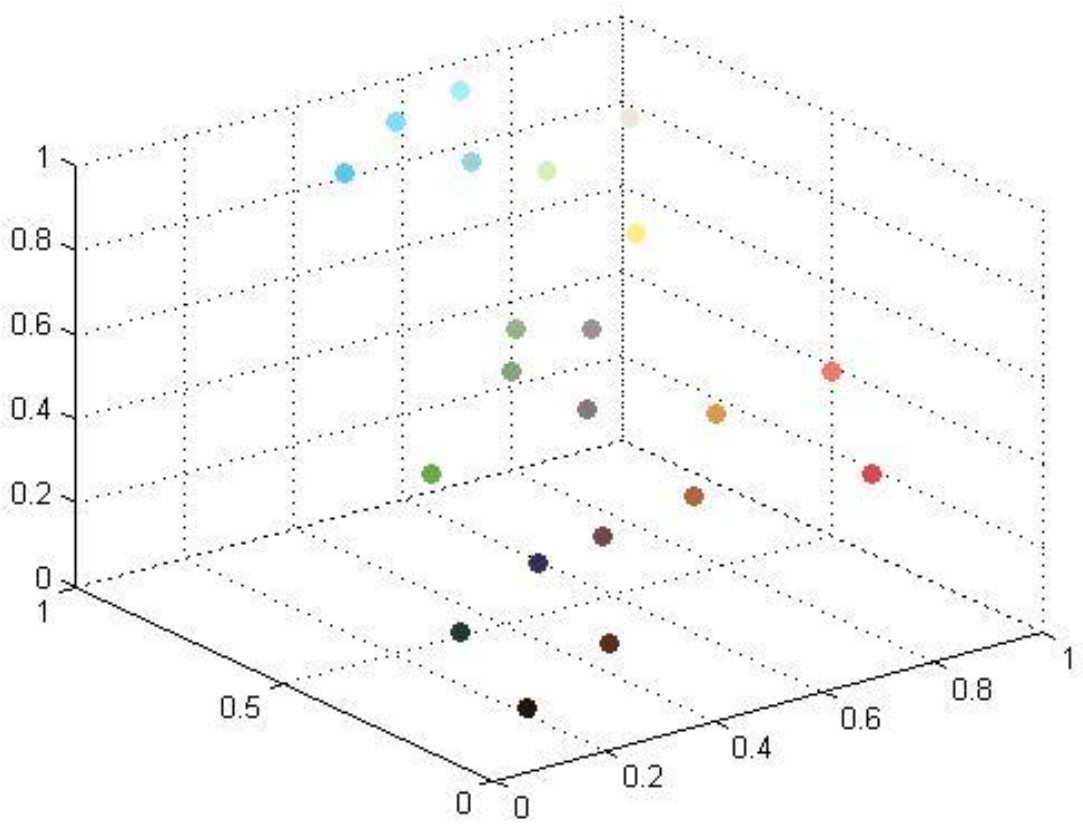
做法：使用 Segment.mean_shift，bandwidth 設定 0.01 來跑，並用 scatter3 function 將 pixel distribution 視覺化 (可能要跑一段時間)
結果：

- 檔案位置：”result_images/part_3a_result1.jpg”

- Pixel distribution before clustering (檔案位置：”result_images/part_3a_result2.jpg”)



- Pixel distribution after clustering (檔案位置：”result_images/part_3a_result3.jpg”)



Part 3-B

做法：將 X, Y 資訊加入第四, 五維度，使用 Segment.mean_shift，color bandwidth 設定 0.01 ，spatial bandwidth 設定 1000 來跑

結果：

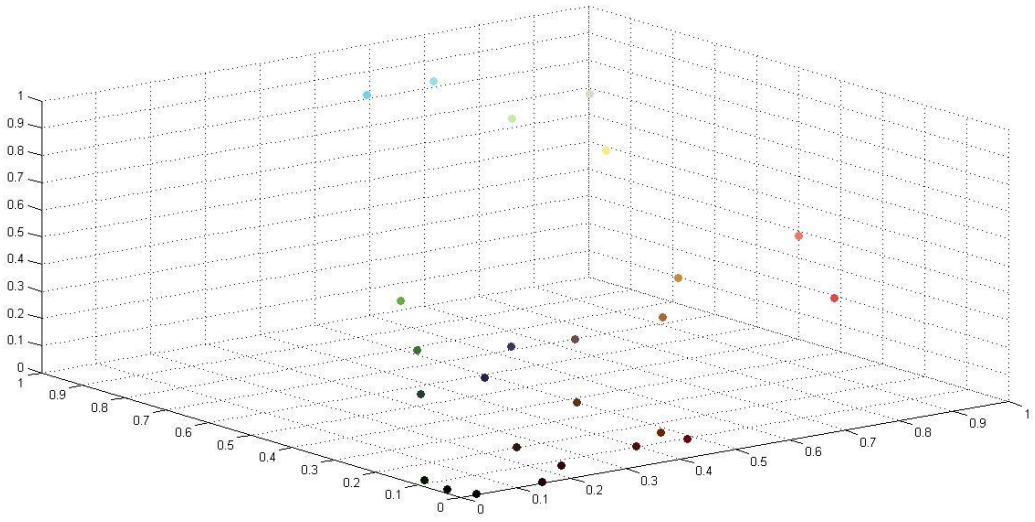
- 檔案位置：”result_images/part_3b_result.jpg”

Part 3-C

結果：

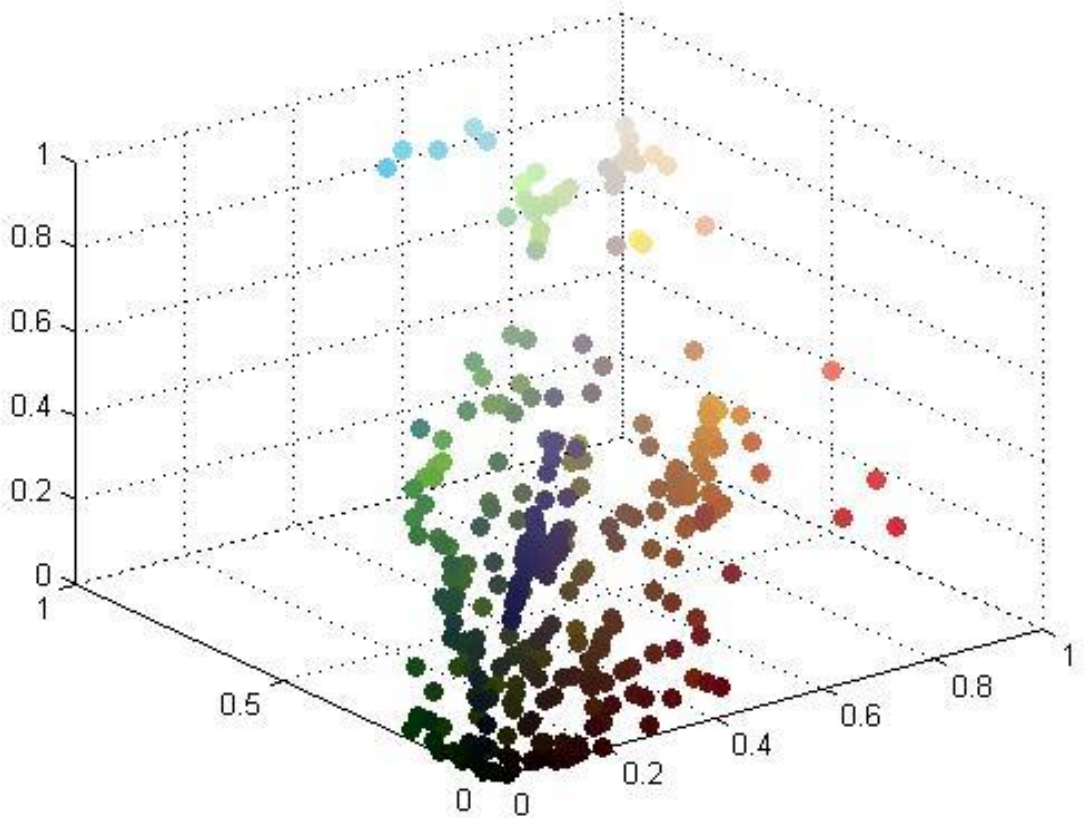
- Cluster using Luv color space (color bandwidth = 100)
  檔案位置：”result_images/part_3c_result1.jpg”

- Pixel distribution after clustering using Luv color space (color bandwidth = 100)
  檔案位置：”result_images/part_3c_result1_dist.jpg”



- Cluster using Luv color space and spatial information (color bandwidth = 100 ; spatial bandwidth = 1000)
  檔案位置：”result_images/part_3c_result2.jpg”

- Pixel distribution after clustering using Luv space and spatial information (color bandwidth = 100 ; spatial bandwidth = 1000)
  檔案位置：”result_images/part_3c_result2_dist.jpg”



Part 3-D

結果：

- Cluster using RGB color space and spatial information (color bandwidth = 0.01 ; spatial bandwidth = 1000)
  檔案位置：”result_images/part_3d_result_c001_s1000”

- Cluster using RGB color space and spatial information (color bandwidth = 0.01 ; spatial bandwidth = 5000)
  檔案位置：”result_images/part_3d_result_c001_s5000”

- Cluster using RGB color space and spatial information (color bandwidth = 0.02 ; spatial bandwidth = 1000)
  檔案位置：”result_images/part_3d_result_c002_s1000”

討論：從實驗結果可以看出，不管是 color 或是 spatial bandwidth，越大則 cluster 越少，因此也就跑得比較快，在這個實驗之中我挑選不會跑太久 (但還是要跑十分鐘至二十分鐘) 來跑出比較好比較的結果，color bandwidth 如果越小，則我們會看到更多不同的顏色的 cluster，而 spatial bandwidth 主要是可以看出照片之中有一小團一小團的 cluster，會造成這個結果也是因為超出一定的 spatial bandwidth，在計算 cluster centroid 時就只會考慮 x, y 範圍內距離相近的點，如果值越小，則一小團一小團的 cluster 範圍就越小。