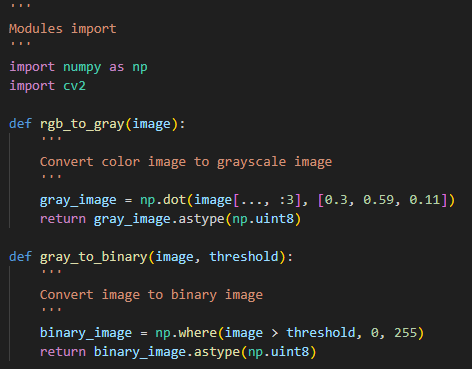
班級：資工三

學號：110590034

姓名：楊榮鈞



Function rgb\_to\_gray:

使用numpy的dot去實現。

rgb轉灰階是利用hw1的公式(0.3×𝑅)+(0.59×𝐺)+(0.11×𝐵)去轉成灰階。

最後用astype(np.uint8)是因為現在的的範圍是0~255。

回傳gray\_image。

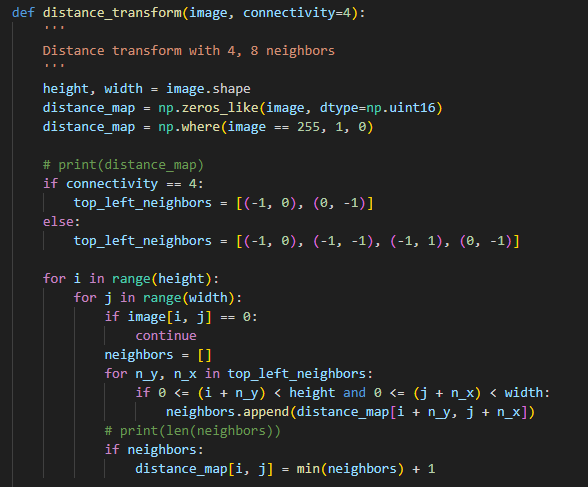
Function gray\_to\_binary:

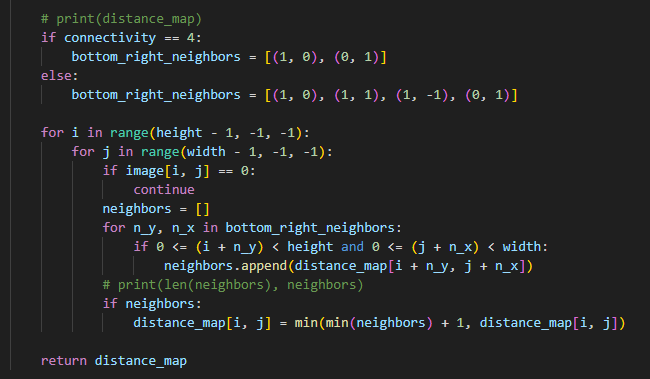
使用numpy的where去實現。

灰階轉binary是當大於threshold時，會設成0，要是小於等於threshold，則是設成255，也就是背景會設為黑色。

最後用astype(np.uint8)是因為現在的的範圍是0~255。

回傳binary\_image。





Function distance\_transform:

首先用shape找出圖片的高和寬，然後用numpy的zeros\_like建立distance\_map的array，其中dtype=np.unit16是因為不確定會有標記的distance會有多少，所以先設定np.unit16。

接著用np.where把binary的image的值複製到distance\_map上，從而進行初始化，其中255會換成1來標記。

然後我用來計算distance transform的方式是：

根據connectivity的值決定top\_left\_neighbors為哪些方向的值。

如果connectivity為4，top\_left\_neighbors會儲存top和left。

如果connectivity為8，top\_left\_neighbors會儲存top、top left、top right和left。

先從左上scan到右下，如果掃到的image[i, j]的value不為0的話，會根據top\_left\_neighbors找出image的neighbors，然後把image[i, j]的值設為neighbors的最小值+1。

根據connectivity的值決定bottom\_right\_neighbors為哪些方向的值。

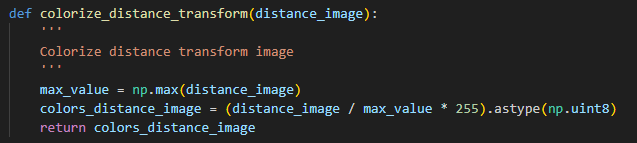
如果connectivity為4，bottom\_right\_neighbors會儲存bottom和right。

如果connectivity為8，bottom\_right\_neighbors會儲存bottom、bottom right、bottom left和right。

再從右下scan到左上，如果掃到的image[i, j]的value不為0的話，會根據bottom\_right\_neighbors找出image的neighbors，然後把image[i, j]的值設為neighbors最小值+1和image[i, j]的最小值，

（會跟自己本身的值找最小值的原因是image[i, j]本身的值是已經從左上scan到右下的結果，所以在右下scan到左上時，需要把從左上scan到右下的結果考慮進去。）

最後回傳distance\_map

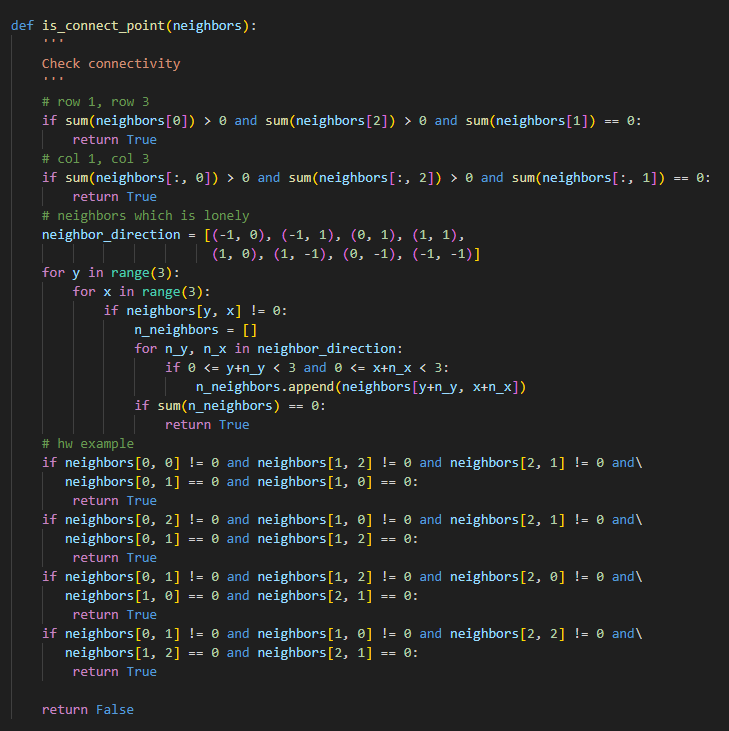


Function colorize\_distance\_transform:

首先會用np.max找出distance\_image的max\_value。

然後用max\_value去normalize再乘上255使distance\_image的value在0~255內，其中有使用astype(np.unit8)，因為現在distance\_image的value範圍是0~255。

回傳colors\_distance\_image。



Function is\_connect\_point:

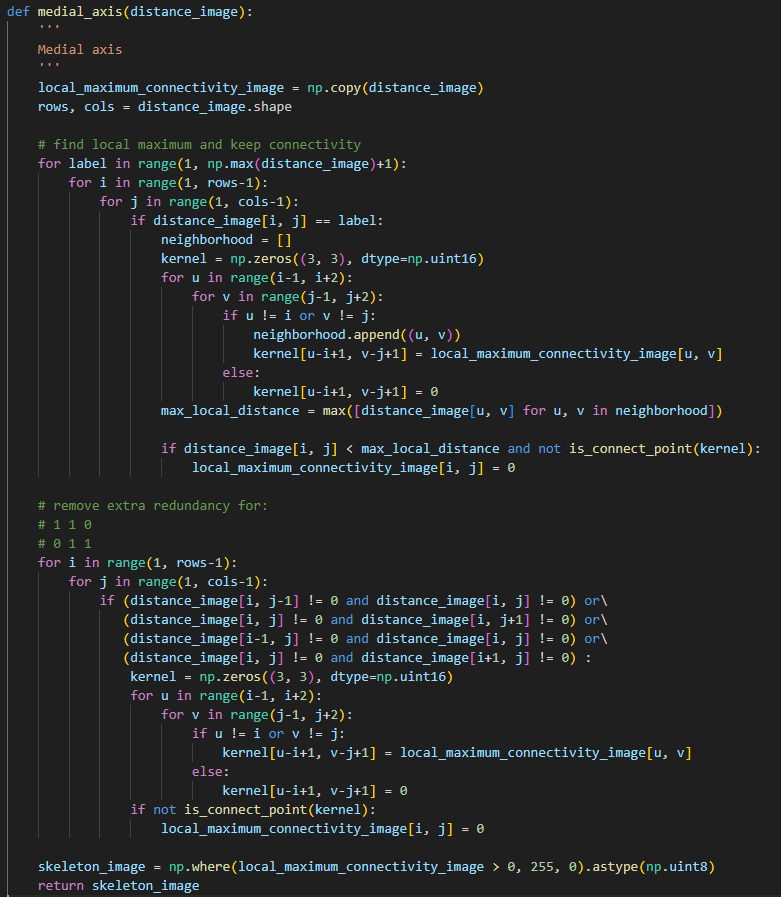
funciton主要功能是確認image的pixel是否為不可刪除的pixel（刪除會導致沒有connectivity）。以下condition都是從neighbor去看。

第一個condition是考慮到row1和row3有value且row2沒有value的狀況。

第二個condition是考慮到column1和column3有value且column2沒有value的狀況。

第三個condition是考慮到有neighbor周圍沒有自身的neighbor的狀況。（自身的neighbor不包括我們呼叫此function的點）

第四、五、六、七個condition是HW3.pdf中的作業的提示。



Function medial\_axis:

Function的主要功能是找到skeleton。

首先會用np.copy()複製distance transform過的image（or map）到local\_maximum\_connectivity\_image，然後用shape找到image的rows（height）跟cols（width）。

接下來有兩個大迴圈。第一個大迴圈我們會從最小的距離找local\_maximum到最大的距離，當我們找到對應的距離的點的時候，我們會儲存他們的neighbor到neighborhood和kernel中，然後我們會用neighborhood找到neighbor中距離最大的值，接著我們會判斷這個點是否小於neighbor中距離最大的值和是否為不是連接點（不可刪除的點），如果符合條件的話，就會把這個點的value設成0（把這個點刪掉）。

第二個大迴圈會檢查是否有出現一些可以刪除的點但是沒有刪掉的狀況，其中考慮到的會是左、右、上或下有連起來的狀況，我們一樣會把neighbors存到kernel再去判斷這個檢查的點是否可以刪除，如果可以我們就會把點設成0。

再跑完大迴圈後利用np.where把local\_maximum\_connectivity\_image非0的value設成255並用astype(np.uint8)設定value範圍，存到skeleton\_image中。

最後再回傳skeleton\_image。



Function image:

先用cv2的imread讀取圖片到image，然後利用rgb\_to\_gray把image轉成gray\_image，再利用gray\_to\_binary把gray\_image轉成binary\_image。

接下來用distance\_transform把binary\_image轉成distance\_image\_4和distance\_image\_8 (在參數設定connectivity=4，會使用4-connected，在參數設定connectivity=8，會使用8-connected)。

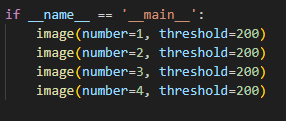
接著利用colorize\_distance\_transform將distance\_image的label根據normalize的結果填上顏色，把distance\_image\_4轉成colors\_distance\_image\_4還有distance\_image\_8轉成colors\_distance\_image\_8。

再來使用medial\_axis找到skeleton\_image。

利用cv2的imshow查看original image、gray image、binary image、colors\_distance\_image\_4、colors\_distance\_image\_8和skeleton\_image。

利用cv2的imwrite將colors\_distance\_image\_4、colors\_distance\_image\_8和skeleton\_image的結果存到results裡面。

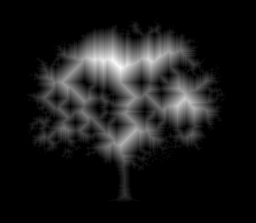
利用cv2的waitKey(0)和destroyAllWindows()將imshow用顯示出來的圖片關閉。



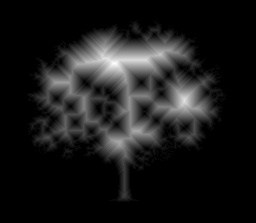
執行image，產生所有助教給的圖片的4-connected和8-connected的distance transform後上不同層次的顏色的圖片，還有Medial Axis的圖片。

Result images

img1\_q1-1\_4.jpg

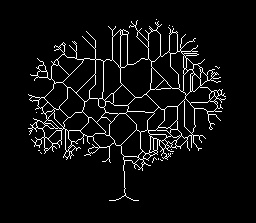


img1\_q1-1\_8.jpg



我預期的結果跟最後的輸出相同，從4-connect和8-connect的結果可以明顯看出來distance transform的結果，4-connect散出來的痕跡是偏向十字，8-connect散出來的痕跡偏向斜線。

img1\_q1-2.jpg



我預期的結果跟最後的輸出相同，可以從distance transform的結果看到一點medial axis的結果的雛型。

img2\_q1-1\_4.jpg



img2\_q1-1\_8.jpg



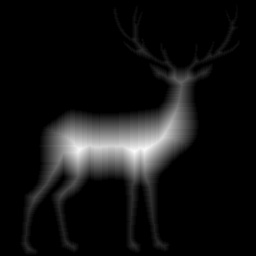
我預期的結果跟最後的輸出相同，8-connect的Machine Vision的V可以明顯感受到鋸齒狀。

img2\_q1-2.jpg

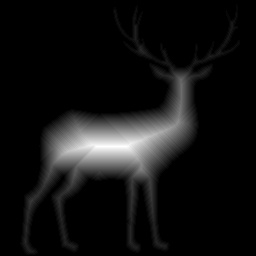


我預期的結果跟最後的輸出相同，可以從distance transform的結果看到一點medial axis的結果的雛型。

img3\_q1-1\_4.jpg

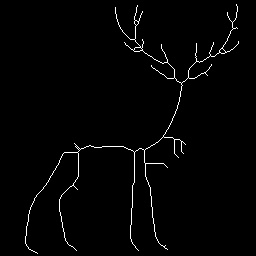


img3\_q1-1\_8.jpg



我預期的結果跟最後的輸出相同，4-connect的鹿的身體散出來的線偏向十字，8-connect散出來的線偏向斜線。

img3\_q1-2.jpg



我預期的結果跟最後的輸出相同，從distance transform的結果中可以看到一點medial axis的結果的雛型。

img4\_q1-1\_4.jpg

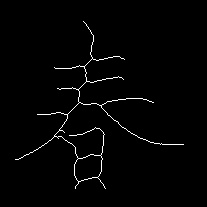


img4\_q1-1\_8.jpg



我預期的結果跟最後的輸出相同，4-connect的春上面的散出來的線偏向直線，8-connect的則是斜線。

img4\_q1-2.jpg



我預期的結果跟最後的輸出相同，從distance transform的結果中可以看到一點medial axis的結果的雛型。

最後的結果總結：

在一開始寫hw3的難點主要是找medial axis的部分，再找尋medial axis的時候有發現有些thinning的方式可以不用用尋找local maximum的方式就可以找到skeleton，像是Skeletonization-by-Zhang-Suen-Thinning-Algorithm，但是它的結果跟用尋找local maximum的方式（作業提到的做法）相差很多，從樹的圖片可以看到Skeletonization-by-Zhang-Suen-Thinning-Algorithm的樹葉有多捲起來形成圓圈的部分，而作業提到的方式則是以散發出去的線為居多。

從distance transform的結果來看4-connect和8-connect的散發結果4-connect會以十字的線較多，8-connect的則以斜線較多。再8-connect的distance transform結果來看，8-connect會比較接近原圖。

Skeletonization-by-Zhang-Suen-Thinning-Algorithm連結

<https://github.com/linbojin/Skeletonization-by-Zhang-Suen-Thinning-Algorithm/blob/master/thinning.py>