

#HW1 學號：110810006 姓名：林君曆

1. Image Quantization(binary, gray, index-color)

1-1. Convert the color image to the grayscale image

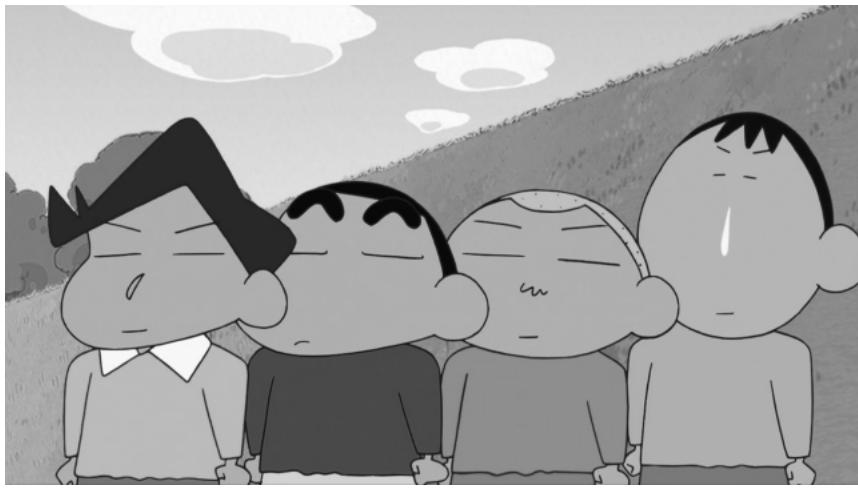
- Formula: $(0.3 \times R) + (0.59 \times G) + (0.11 \times B)$.

輸入：彩色圖像

方法：先初始化一張新的照片，接著遍歷過每個 pixel，套用提供的公式即可計算新的 grey level 的像素為多少

結果如下：





解釋：結果看起來非常自然

1-2. Convert the grayscale image to the binary image

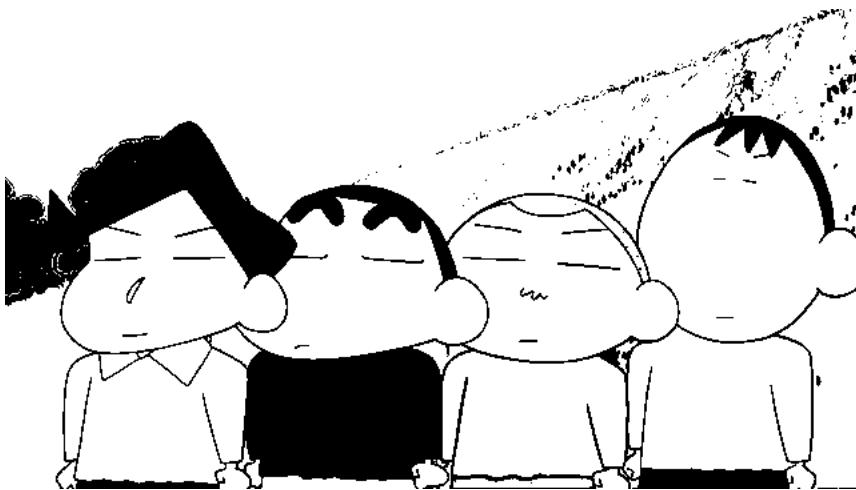
- Choose a appropriate threshold by yourself.

輸入：輸入為灰階圖像

方法：先初始化一張新的照片，接著遍歷過每個 pixel，我設的 threshold 為 128，即 0~255 的一半之處，對分的轉換效果比較好，如果小於 128 就設為 0（白），反之設為 255（黑）

結果如下：





說明：第一跟第三張照片看起來都還算自然，但第二張我猜是因為陰影的關係，沒辦法很好的劃分同個區塊去形成對比，這可能就是 **binary image** 的缺點，比如這紙袋的兩側因為陰影，一邊幾乎都白，一邊幾乎都黑，很難還原原圖的情況

1-3. Convert the color image to the index-color image

- Define your own colormap of 16 type colors.

輸入：彩色圖像以及 `color_map`

方法：先初始化一張新的照片，接著遍歷過每個 `pixel`，且計算點與 `color_map` 上面的不同顏色 RGB 值的距離，最後選取距離最相近的作為映射過後的顏色

`Color_map` :

```
[0, 0, 255], # 藍色  
[0, 255, 0], # 綠色  
[255, 0, 0], # 紅色  
[0, 255, 255], # 黃色  
[255, 0, 255], # 洋紅  
[255, 255, 0], # 青色  
[0, 0, 128], # 深藍色  
[0, 128, 0], # 深綠色  
[128, 0, 0], # 深紅色  
[128, 128, 0], # 淺黃色  
[128, 0, 128], # 紫色  
[0, 128, 128], # 灰綠色  
[192, 192, 192], # 銀色  
[128, 128, 128], # 灰色  
[255, 165, 0], # 橙色  
[255, 192, 203] # 粉紅色
```

結果如下：



說明：從我的 `color_map` 轉換過來的感覺很有趣，圖像都有種蠟筆畫出來的感覺，不過第一張跟第二張可能轉換的沒那麼好就是了

2. Resizing Image

2-1. Resizing image to $\frac{1}{2}$ and 2 times without interpolation

輸入：Image(原圖)、scale_factor(轉換倍率)

方法：我是依照 ppt 上面的範例做放大縮小的轉換，假設新的座標點是(i, j)，觀察過後可以發現縮放過後的照片的像素點，是對應到原圖的($(i/scale_factor), (j/scale_factor)$)，所以我遍歷過新的 size 的圖片的每一個像素點，去對應的原圖的像素點，就完成了照片的轉換了，結果如下：







說明：除了放大兩倍的 m&m 很顆粒感之外，都很自然

2-2. Resizing image to $\frac{1}{2}$ and 2 times with interpolation

- You can use bilinear or bicubic interpolation.

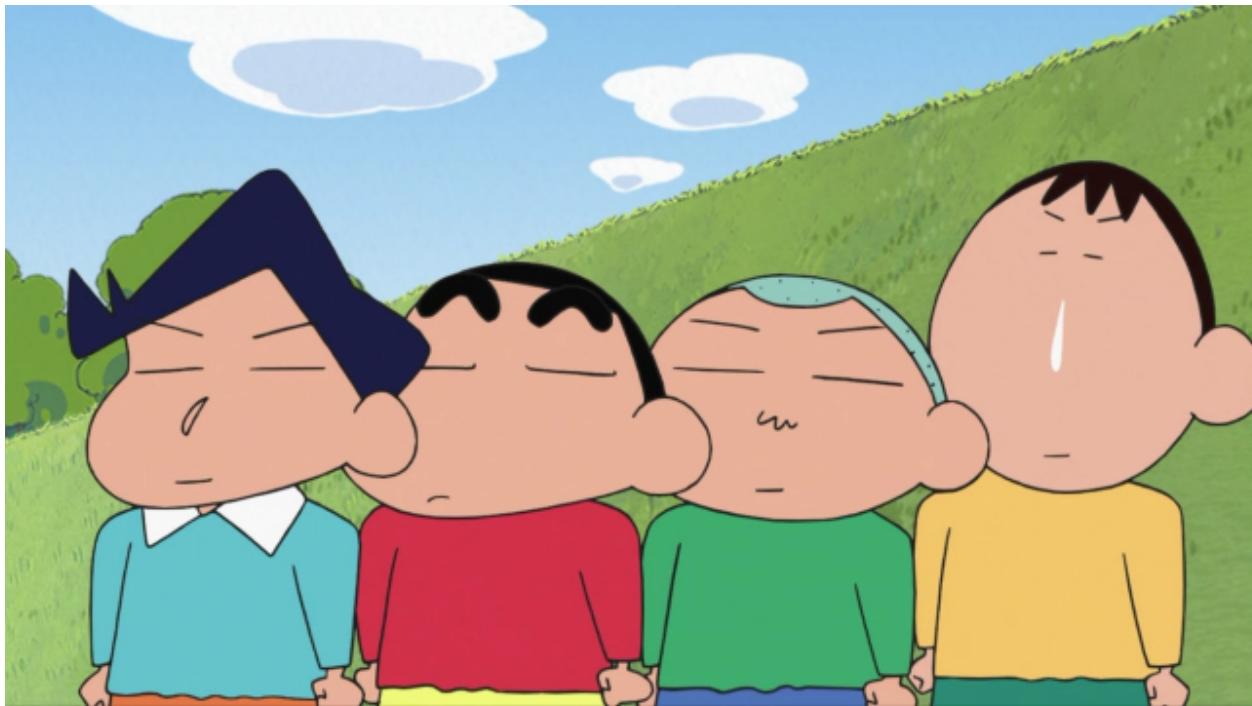
輸入：Image(原圖)、scale_factor(轉換倍率)

方法：使用 bilinear interpolation，先計算 resize 過後的寬跟高，以及縮放後的照片每個像素點在原圖的寬度 $step = 1/scale_factor$ 。遍歷每個縮放過後的照片 pixel，先計算出周圍的四個點，在算 ceil 的時候要注意不能超出 index，所以加了 min 函式處理這個情況，接著因為新的座標點可能剛好點在原圖的座標點上，所以需要個別來判斷這個情況，總共分四個情況（1）如果 x,y 都在點上，則直接照原本的 RGB 不用插值（2）如果 x 方向的座標有在點上，進行 y 方向的插值就好（3）如果 y 方向座標有在點上，進行 x 方向插值就好

（4）如果都不在點上就先做 x 再做 y 插值，就可以算出最後縮放後新的座標的 RGB 了
結果如下：







說明：使用插值法，可以觀察到第一張照片在放大的時候，是有比較平滑的，所以插值法某些情況真的能看出表現的比第一種方式好

程式碼撰寫補充：

撰寫程式碼的時候，想說作業雖然分為多個部分，但是都有點承接或是共同的地方，所以寫了 `ImageConverter` 跟 `ImageResizer` 並建立 `staticmethod`，然後透過 `HW1_1`, `HW1_2` 兩個 `class` 去一次處理三張圖片，雖然用這樣方式撰寫，但是 `function` 內部寫的還是比較專注在符合題目的需求，所以 `function` 的可用性感覺還是很差，下次會再評估一下是否要這樣寫

參考資料（關於插值法的教學）：

https://www.youtube.com/watch?v=hpqrDUuk7HY&list=PLjMXczUzEYcHvw5YYSU92WrY8IwhTuj7p&index=4&t=3074s&ab_channel=JosephRedmon