班級：資工三
學號：110590034
姓名：楊榮鈞

1-1. Convert the color image to the grayscale image
我是使用 numpy 的 dot 去將 color image 的 array 的值根據 ppt 給的 Formula $((0.3\times R)+(0.59\times G)+(0.11\times B))$ 轉乘 grayscale image。

```
def rgb_to_gray(image):
    '''
    1-1. Convert color image to grayscale image
    '''
    gray_image = np.dot(image[..., :3], [0.3, 0.59, 0.11])
    return gray_image.astype(np.uint8)
```

1-2. Convert the grayscale image to the binary image
利用 numpy 的 where 去將 grayscale image 的 array 的值根據 threshold (決定要 255 還是 0) 轉成 binary image。

```
def gray_to_binary(gray_image, threshold):
    '''
    1-2. Convert grayscale image to binary image
    '''
    binary_image = np.where(gray_image > threshold, 255, 0)
    return binary_image.astype(np.uint8)
```

1-3. Convert the color image to the index-color image
我是先用 numpy 的 zeros_like 創造出跟原本圖片一樣大小的陣列，然後再呼叫 find_closest_color 的 function 得出最相近的顏色的 index，並把最相近的顏色放進相對應的 pixel。
find_closest_color 的 function 是利用 numpy 的 function 實現 Euclidean Distance 去尋找出最相近的顏色，並使用 numpy 的 argmin 回傳最相近的顏色的 index。

```python
# color_count = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def find_closest_color(pixel):
    '''
    Convert each pixel to its closest color in the custom colormap
    '''
    custom_colormap = np.array([
        [241, 199, 106],# Yellow
        [240, 230, 140],# Khaki
        [0, 0, 255],    # Blue
        [0, 255, 255],  # Cyan
        [222, 184, 135],# BurlyWood
        [255, 255, 224],# LightYellow
        [128, 0, 0],    # Maroon
        [0, 128, 0],    # Green
        [0, 0, 128],    # Navy
        [0, 128, 128],  # Teal
        [198, 110, 72], # Brown
        [128, 128, 0],  # Olive
        [0, 0, 0],      # Black
        [128, 128, 128],# Gray
        [192, 192, 192],# Silver
        [255, 255, 255] # White
    ])
    distances = np.linalg.norm(custom_colormap - pixel, axis=1)
    # color_count[np.argmin(distances)] += 1
    return np.argmin(distances)
```

```python
def color_to_index(image):
    '''
    1-3. Convert color image to index-color image
    '''
    custom_colormap = np.array([
        [241, 199, 106],# Yellow
        [240, 230, 140],# Khaki
        [0, 0, 255],    # Blue
        [0, 255, 255],  # Cyan
        [222, 184, 135],# BurlyWood
        [255, 255, 224],# LightYellow
        [128, 0, 0],    # Maroon
        [0, 128, 0],    # Green
        [0, 0, 128],    # Navy
        [0, 128, 128],  # Teal
        [198, 110, 72], # Brown
        [128, 128, 0],  # Olive
        [0, 0, 0],      # Black
        [128, 128, 128],# Gray
        [192, 192, 192],# Silver
        [255, 255, 255] # White
    ])
    index_image = np.zeros_like(image, dtype=np.uint8)
    # print(image)
    # print(index_image)
    # print(image.shape[0])
    # print(image.shape[1])
    # print(image.shape)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            # print('a= ', image[i, j])
            index_image[i, j] = custom_colormap[find_closest_color(image[i, j])]
            # print(index_image[i, j])
    # for i in range(image.shape[1]):
    #     for j in range(image.shape[0]):
    #         index_image[i, j] = custom_colormap[index_image[i, j]]
    # print(image)
    # print(index_image)
    return index_image
```

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

這邊我是先用 shape 找出原本圖片的高和寬和縮放之後的高和寬，然後用 numpy 的 zeros 建立一個新的 image array。

接下來利用縮放後的高和寬去找出原本圖片的 pixel 的位置，並把那個 pixel 的

value 放到新的 image array 的指定 pixel 中。

```
def resize_no_interpolation(image, scale):
    '''
    2-1. Define function for resizing without interpolation
    '''
    height, width = image.shape[:2]

    new_height = int(height * scale)
    new_width = int(width * scale)

    resized_image = np.zeros((new_height, new_width, 3), dtype=np.uint8)

    for new_i in range(new_height):
        for new_j in range(new_width):
            original_i = int(new_i / scale)
            original_j = int(new_j / scale)
            resized_image[new_i, new_j] = image[original_i, original_j]

    return resized_image
```

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

我先利用 numpy 的 zeros 去創建一個縮放後的 image array，然後再使用 bilinear interpolation 的方式去實現縮放圖片。
其中我是利用 scale 推算出原圖的周圍的四個 pixel，然後再算出 interpolation 的權重，最後在用 bilinear interpolation 算出縮放後的 image array 的 pixel 的 value 要是什麼值。

```
def bilinear_interpolation(image, scale):
    '''
    2-2. Define function for bilinear interpolation
    '''
    height, width = image.shape[:2]

    new_height = int(height * scale)
    new_width = int(width * scale)

    resized_image = np.zeros((new_height, new_width, 3), dtype=np.uint8)

    for new_i in range(new_height):
        for new_j in range(new_width):
            original_i = new_i / scale
            original_j = new_j / scale

            # Get the surrounding four pixels in the original image
            top_left = image[int(original_i), int(original_j)]
            top_right = image[int(original_i), min(int(original_j) + 1, width - 1)]
            bottom_left = image[min(int(original_i) + 1, height - 1), int(original_j)]
            bottom_right = image[min(int(original_i) + 1, height - 1), min(int(original_j) + 1, width - 1)]

            # Calculate the weights for interpolation
            dx = original_j - int(original_j)
            dy = original_i - int(original_i)

            # Perform bilinear interpolation
            top_interpolation = (1 - dx) * top_left + dx * top_right
            bottom_interpolation = (1 - dx) * bottom_left + dx * bottom_right
            resized_image[new_i, new_j] = (1 - dy) * top_interpolation + dy * bottom_interpolation

    return resized_image
```

Result images

img1_q1-1.png

我預期的結果跟最後的輸出相同，因為是用 PPT 上給的 Formula 轉出來的，所以跟我預想的差不多。

img1_q1-2.png

這張圖我預期的結果與最後的輸出有一點差距，我預期的情況是應該可以把所有的 m&m 巧克力球的形狀和包裝都用出來，但是真正的輸出是一些顏色比較亮的巧克力球會因為 threshold 的關係而沒辦法完整的標示出來。

若是將 threshold 調太低或是太高，則會出現不想要的資訊（像是桌子的紋路）或是更多巧克力球沒辦法完整的標示出來。

img1_q1-3.png

colormap

```
custom_colormap = np.array([
    [241, 199, 106],# Yellow
    [240, 230, 140],# Khaki
    [0, 0, 255],    # Blue
    [0, 255, 255],  # Cyan
    [222, 184, 135],# BurlyWood
    [255, 255, 224],# LightYellow
    [128, 0, 0],    # Maroon
    [0, 128, 0],    # Green
    [0, 0, 128],    # Navy
    [0, 128, 128],  # Teal
    [198, 110, 72], # Brown
    [128, 128, 0],  # Olive
    [0, 0, 0],      # Black
    [128, 128, 128],# Gray
    [192, 192, 192],# Silver
    [255, 255, 255] # White
])
```

這張我的預期結果與真正的輸出不太一樣，雖然輪廓有標示出來，但由於我的 colormap 的一些非預期的顏色的 rgb 的 value 跟 img1.png 的一些 pixel 的 Euclidean Distance 的距離比較短，導致跟我預想的顏色不太相同（像是我預期桌面可能要偏向卡其色，但那一些 pixel 和銀色和灰色的 rgb 的 value 更相近，所以最後 index image 的桌面顏色偏向灰色和銀色）。

img1_q2-1-half.png

這張圖的預期結果和最後的輸出跟我想像得差不多，由於縮小 1/2 倍沒有用到 interpolation，所以巧克力球和包裝鋸齒的情況感覺比較明顯，當這張圖強行放大回原本的大小的時候，可以明顯感受到失真的感覺。

img1_q2-1-double.png

這張圖的預期結果和最後的輸出跟我想像得相似，由於放大 2 倍，並且沒有使用 interpolation 的關係，所以可以感受到巧克力球和包裝鋸齒的情況也相對明顯。

img1_q2-2-half.png

這張圖的預期結果和最後的輸出結果相似，只是跟前面沒有用 bilinear interpolation 的結果比起來，感覺色彩有比較明顯一些（相近於原圖）。

img1_q2-2-double.png

這張圖的預期結果和最後的輸出結果也差不多，只是跟前面沒有用 bilinear interpolation 的結果比起來，感覺鋸齒狀導致巧克力球和包裝周圍有點糊糊的樣子有比較少一些。

img2_q1-1.png

我預期的結果跟最後的輸出相同，因為是用 PPT 上給的 Formula 轉出來的，所以跟我預想的差不多。

img2_q1-2.png

這張圖的預期結果和最後的輸出差不多，我覺得 threshold 調在 155 左右時，可以明確地讓貓咪顯示出來。

img2_q1-3.png

colormap

```
custom_colormap = np.array([
    [241, 199, 106],# Yellow
    [240, 230, 140],# Khaki
    [0, 0, 255],    # Blue
    [0, 255, 255],  # Cyan
    [222, 184, 135],# BurlyWood
    [255, 255, 224],# LightYellow
    [128, 0, 0],    # Maroon
    [0, 128, 0],    # Green
    [0, 0, 128],    # Navy
    [0, 128, 128],  # Teal
    [198, 110, 72], # Brown
    [128, 128, 0],  # Olive
    [0, 0, 0],      # Black
    [128, 128, 128],# Gray
    [192, 192, 192],# Silver
    [255, 255, 255] # White
])
```

這張圖的預期結果和最後的輸出有一些差距，雖然我的 colormap 的顏色再利用 Euclidean Distance 找出最相近的顏色之後，他可以把貓貓完整的顯示出來，但是貓的左邊影子和後面的背景會因為我的 colormap 的顏色跟原圖比起來有一些差距，導致他會出現偏黃色和紅色的結果。

img2_q2-1-half.png

這張圖的預期結果和最後的輸出一樣，由於貓的毛和帽子的毛比較多，所以鋸齒狀的邊緣相對感受不太出來，因此感覺跟原圖的差距只有縮小 1/2 倍而已。

img2_q2-1-double.png

這張圖的預期結果和最後輸出一樣，貓咪的邊緣有明顯糊糊的感覺（鋸齒狀），然後後面的地板有種 720p 的感受，明顯感到跟原圖的像素的一些落差。

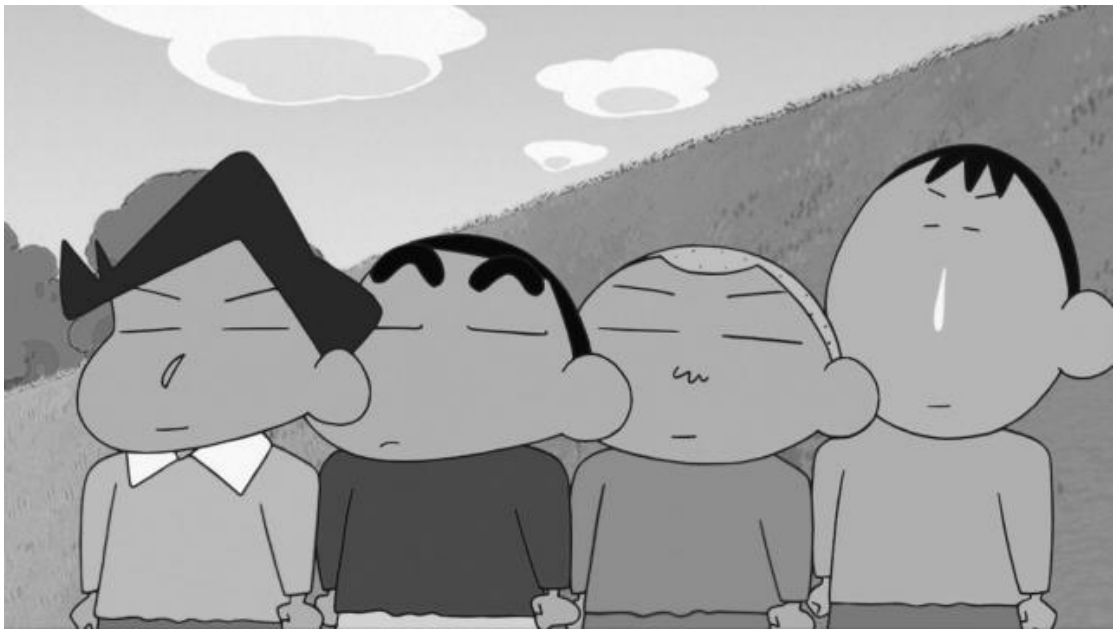
img2_q2-2-half.png

這張圖的預期結果和最後的輸出一樣，由於有使用 bilinear interpolation 的關係，可以明顯的看出一些毛（一根一根的那種）相對於沒有使用 interpolation 的圖來說，毛有更明顯一點。

img2_q2-2-double.png

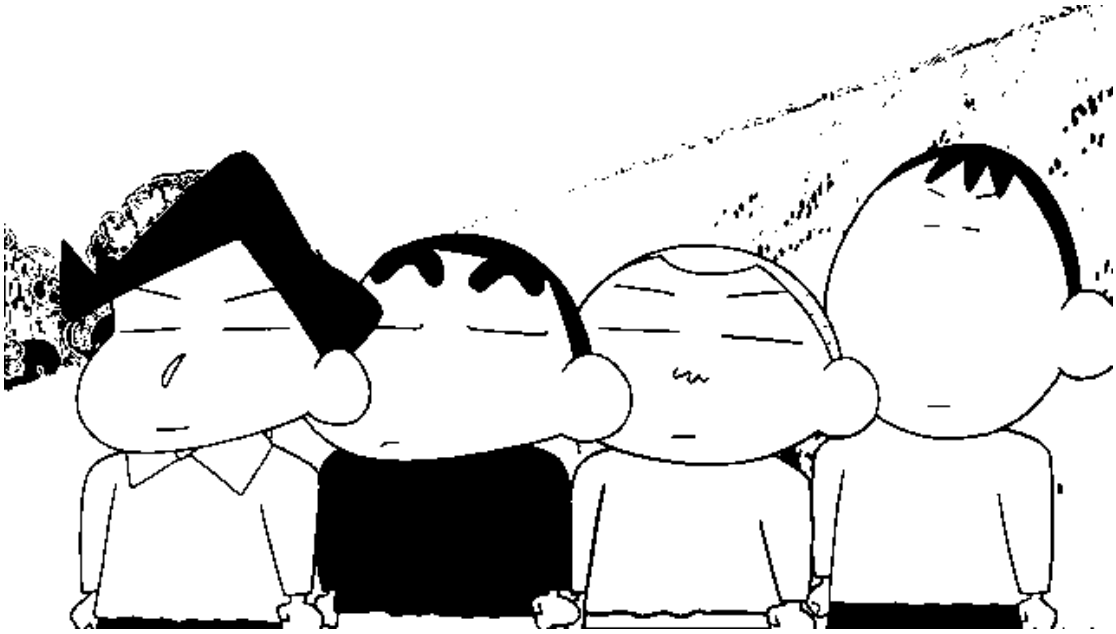
這張圖的預期結果和最後的輸出一樣，由於有用 bilinear interpolation 的關係，相對於沒有使用 interpolation 的圖來說，貓毛和帽子的毛有更明顯一點。然後背景的地板相對於沒有使用 interpolation 的圖感覺更有更高的解析度。

img3_q1-1.png



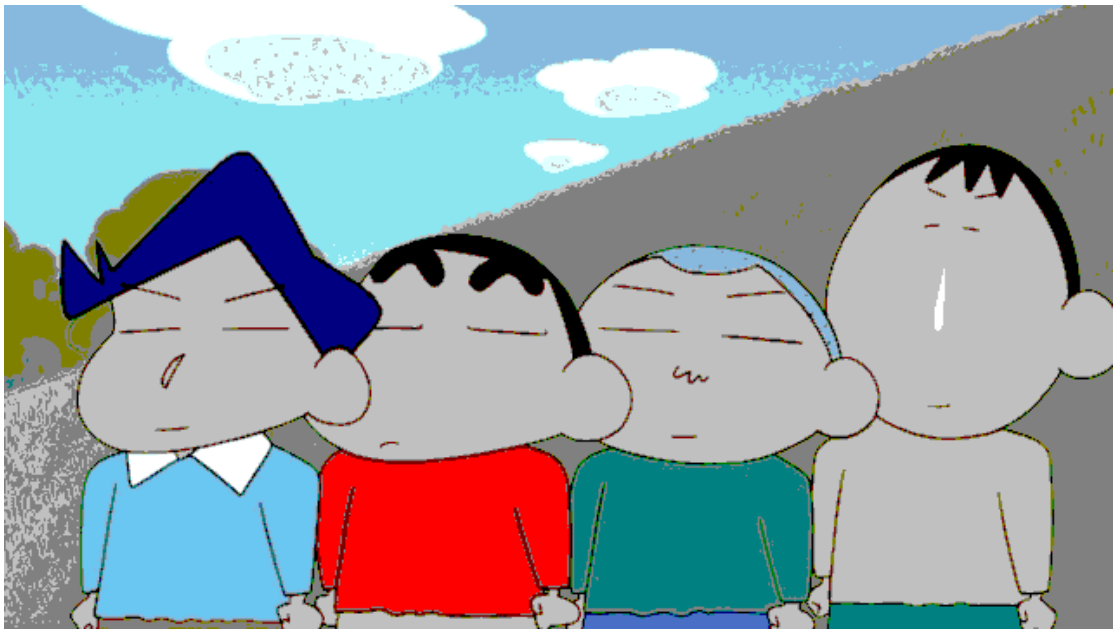
我預期的結果跟最後的輸出相同，因為是用 PPT 上給的 Formula 轉出來的，所以跟我預想的差不多。

img3_q1-2.png



這張圖的預期結果與最後的輸出差不多，在我把 threshold 調到 123 的時候（a lucky number）剛好可以把 4 個人物都標示出來，其中在調 threshold 的難點是風間頭上的草叢，因為他的灰階圖片的顏色，所以很難把那個草叢利用 threshold 清掉。

img3_q1-3.png



colormap

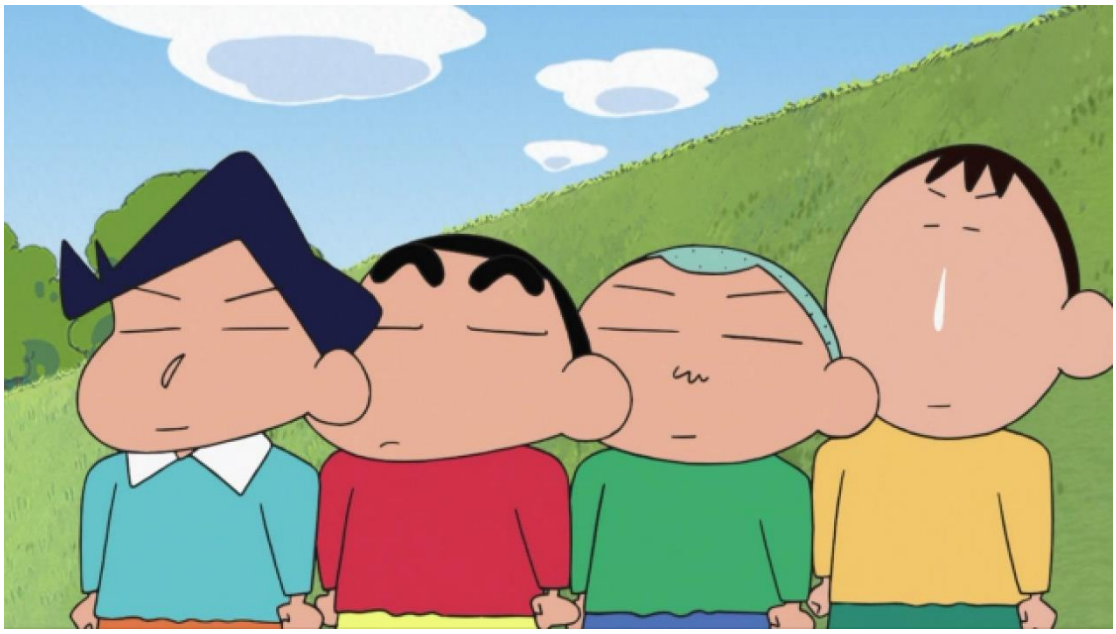
```
custom_colormap = np.array([
    [241, 199, 106],# Yellow
    [240, 230, 140],# Khaki
    [0, 0, 255],    # Blue
    [0, 255, 255],  # Cyan
    [222, 184, 135],# BurlyWood
    [255, 255, 224],# LightYellow
    [128, 0, 0],    # Maroon
    [0, 128, 0],    # Green
    [0, 0, 128],    # Navy
    [0, 128, 128],  # Teal
    [198, 110, 72], # Brown
    [128, 128, 0],  # Olive
    [0, 0, 0],      # Black
    [128, 128, 128],# Gray
    [192, 192, 192],# Silver
    [255, 255, 255] # White
])
```

這張圖的預期結果與最後的輸出相差的有點大，雖然 colormap 裡面的顏色可以讓圖片裡面的人物背景的輪廓顯示出來，但由於我的顏色似乎跟原圖比起來，有些顏色跟我所預想的不一樣，像是阿呆的衣服應該要是黃色，但根據 Euclidean Distance 的計算結果後衣服的 pixel 相對於黃色更接近於銀色。

img3_q2-1-half.png

這張圖的預期結果跟最後的輸出差不多，從 4 人的眉毛、正男的頭髮的界線和衣服等等，可以明顯的看出鋸齒狀。

img3_q2-1-double.png

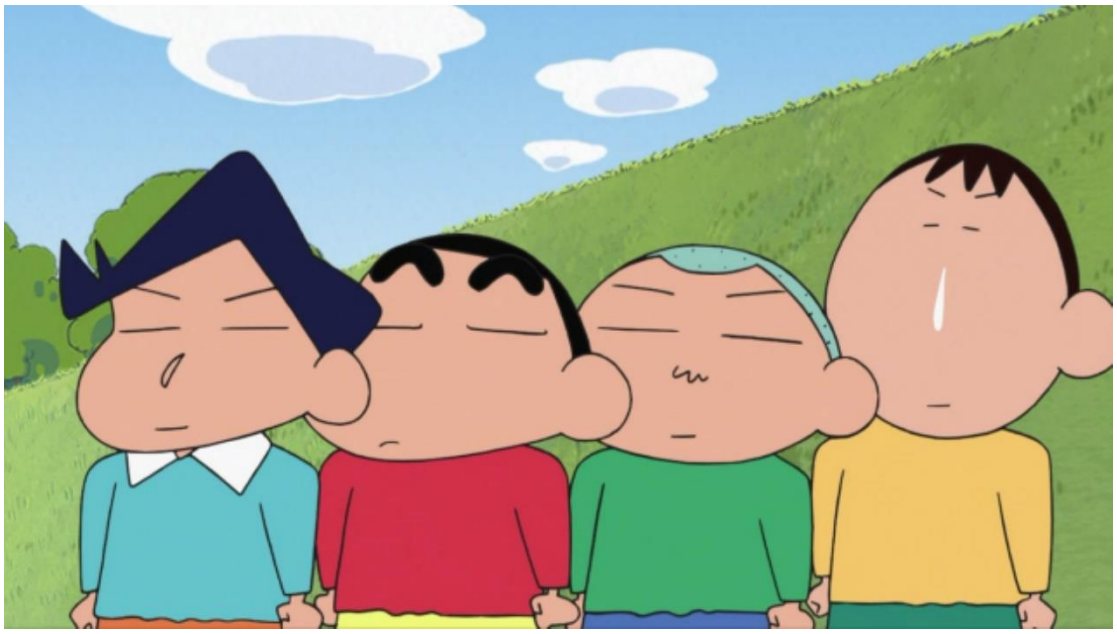


這張圖的預期結果與最後的輸出相似，跟縮小 1/2 倍比起來，鋸齒狀有相對不明顯，但是可以明顯的感受到一點模糊的感覺。

img3_q2-2-half.png

這張圖的預期結果跟最後輸出相似，然後跟沒用 interpolation 的圖比起來，似乎有顏色有比較豐滿一點，但也可以辨識出一些鋸齒狀。

img3_q2-2-double.png



這張圖的預期結果和最後的輸出一樣，他有比沒用 interpolation 的圖還要清晰一點，跟縮小的圖比起來鋸齒狀相對比較不嚴重。

最後的結果總結：

最後做完 hw1 之後，我覺得 hw1 的第一個難點是 grayscale image 轉 binary image，其中困難的地方在於找 threshold 的時候，要先考慮到圖片想要著重的東西，再來才是考慮如何消除雜訊（非重點的各種東西），在消除雜訊時，常常會遇到他的 value 跟想要著重的東西的 value 相近，使 threshold 調太高或太低都不適合，導致沒辦法完整的消除雜訊。

第二個難點是 color image 轉 index-color image，在我把 rgb(0, 0, 0)到 rgb(255, 255, 255)分成 16 份塞進 colormap 之後，我發現再轉成 index-color image 時，有些顏色是用不到的。因此我又根據三張圖片比較相似的顏色替換掉沒有到的顏色，只是再放進去之後我發現這些顏色不一定會像我想的一樣讓 index-color image 有那些顏色，因為原本的 colormap 的一些顏色比起我後來替換掉的顏色來說，他更接近原圖的顏色，所以導致 index-color image 沒有呈現出我想像的圖片。

在做完 hw1 後我有試著用 KNN 去找 colormap，在這個過程中我了解到定義 colormap 的用意跟使用 colormap 還原圖片有點差別，因為定義 colormap 是想要用現有的顏色去做圖片的辨識，要是想要還原圖片的話，直接用 KNN 去抓取 colormap 就可以更好的還原。因此在 hw1 中我決定還是用自己定義的 colormap 去轉 index-color image。