

# Homework 1 Report

## Part 1. Demosaicing

- A. 檔案結構，在 `hw1_1.m` 這個 Script 裡面會呼叫兩個我們需要實作的函式 `Demosaicing` 和 `MyPSNR`。

```
hw1_1.m -----> Demosaicing.m
|-----> MyPSNR.m
```

- B. 本作業主要是要把已經做完 Bayer color filter 的圖片透過 NN(Nearest-Neighbor) 的演算法解析成原本的模樣。

因為 filter 是長這個樣子的：

```
|G|R|G|R|...
|B|G|B|G|...
|G|R|G|R|...
|B|G|B|G|...
```

觀察上圖以後，我們可以知道最接近要求的點附近最多只有八格，且對一個 pixel 而言，該點只會有 R, G, B 其中一種顏色；換句話說，除了該點的某顏色已經有數值以外，剩餘的其他兩色需要計算來求得答案。

如果該點是 G 有值，我們需要求出 R, B；R 的話，會從該點的上下兩個 R 取平均值，B 則是從該點的左右兩個 B 來取平均值。

如果該點是 R 有值，我們需要求出 G, B；G 的話，會從該點的上下左右四個 G 取平均值，B 則是從該點的左上、左下、右上、右下四點取平均值。

如果該點是 B 有值，我們需要求出 G, R；G 的話，會從該點的上下左右四個 G 取平均值，R 則是從該點的左上、左下、右上、右下四點取平均值。

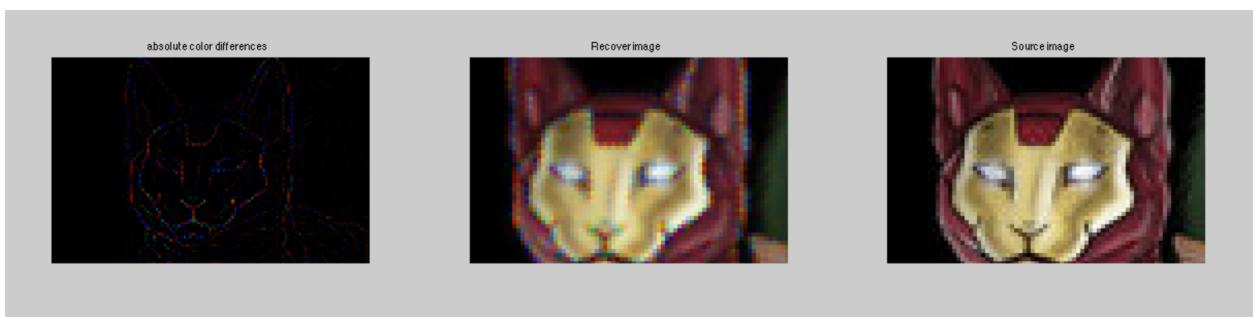
- C. 但是再分析一下輸入的圖片，剛好除了我們要取值以外的點會是空的，所以我要取的時候就直接把周圍的八格掃過一遍，取有找到值的平均。另外為了避免處理邊界的麻煩，我選擇不處理邊界的資料。（`Demosaicing` 的實作）

- D. PSNR 我是透過自己撰寫的 `MyPSNR` 來求得，大約是 30.5512 dB。根據查到的資料，PSNR 代表了有用訊號與雜訊間的比值，通常介於 30dB (watchable) 到 45dB (visually lossless) 是最佳的。

- E. 以下為結果圖：



展示圖片



- F. 可以見得在絕對顏色差那邊有些許的顏色出現。與原圖相比也顯得比較粗糙，畢竟是由周圍的 RGB 還原回來的。

## Part 2. Dithering

- A. 檔案結構，在 `hw1_2.m` 這個 Script 裡面會呼叫兩個我們需要實作的函式 `Thresholding` 和 `ErrorDiffusionDithering`(我將不同的 mask 透過這個函式的參數傳入)。

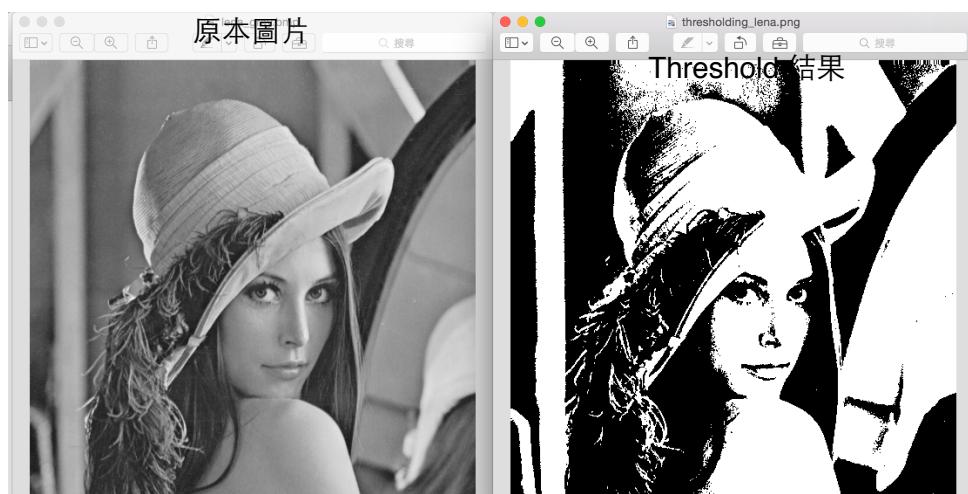
```
hw1_2.m -----> Thresholding.m
                  |-----> ErrorDiffusionDithering.m
```

輸出的圖片：

```
lena-----> thresholding_lena.png (經過 threshold 處理)
              |-----> EDD_lena_1.png (經過 mask #1)
              |-----> EDD_lena_2.png (經過 mask #2)
myimg-----> thresholding_myimg.png (經過 threshold 處理)
              |-----> EDD_myimg_1.png (經過 mask #1)
              |-----> EDD_myimg_2.png (經過 mask #2)
```

- B. 本作業主要是要把一個灰階的圖像轉成只有二原色（黑、白）的圖片，再透過 `Error Diffusion` 讓圖片重新處理成灰階。  
C. 在計算 `Threshold` 的時候，為了讓失真變少，所以我取所有灰階值的平均作為 `Threshold` 的值。因為我已經把圖片轉成 `double` 了，所以 `255, 0` 會用 `1, 0` 來表示。如果該點的灰階值比 `Threshold` 還大，我就取成 `1`，否則就取成 `0`。另外，在 `Error Diffusion` 的時候，我把 `mask #` 設為一個參數，使用者可以傳入 `1` 或 `2` 來決定使用哪一種模式來計算。

- D. `lena` 跑出來的結果圖片：



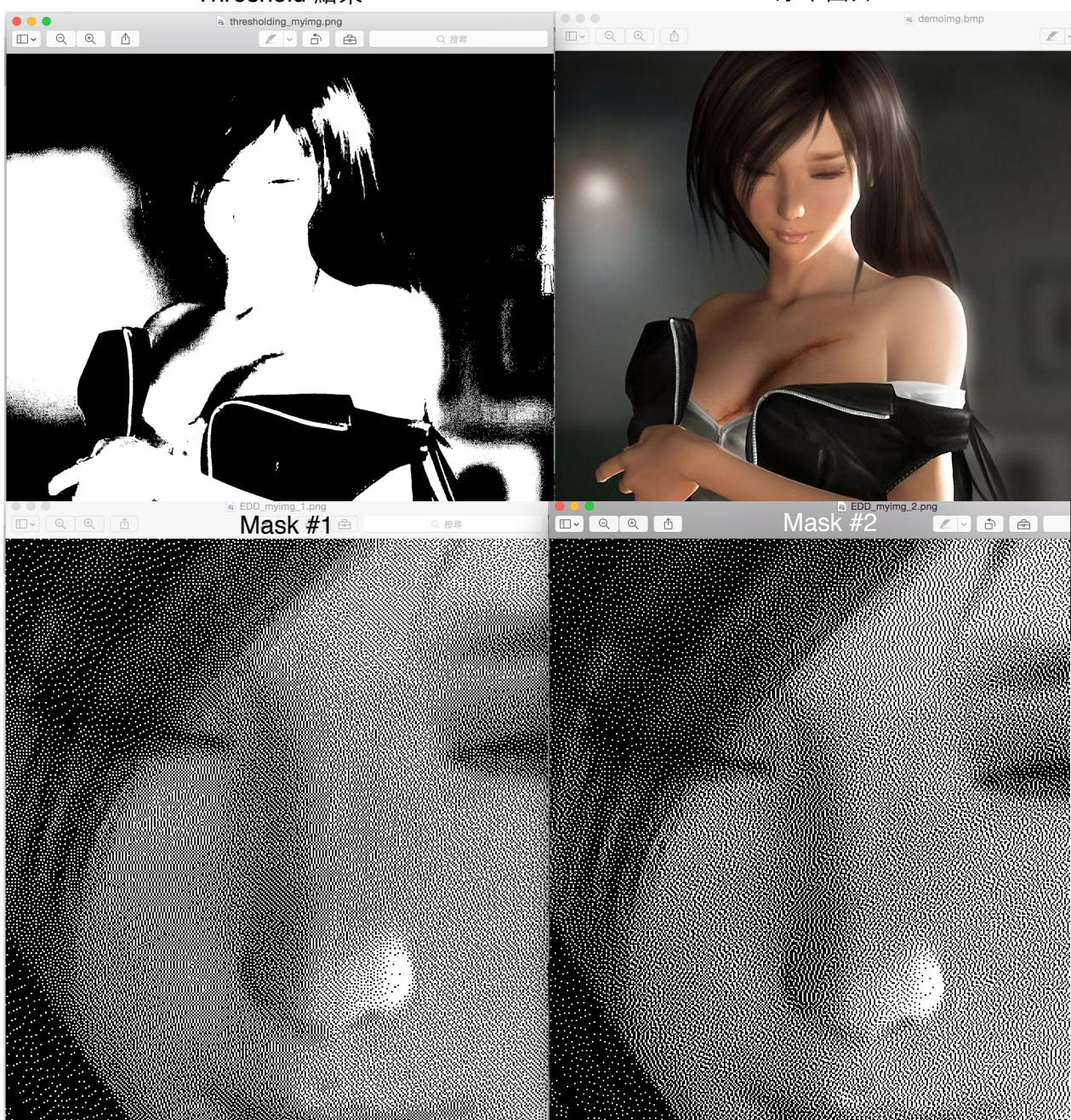


E. 在 lena 的圖片中，mask #1 與 mask #2 的效果沒有很顯著，大體上就是 mask #2 的輪廓比較明顯一些。

F. 自己的圖片部分：

Threshold 結果

原本圖片



- G. 我選擇的這張圖主要在人臉上的光澤比較多層次，希望可以更明顯展現出效果。可以發現在 mask #1 的時候人會比較扁平一些，紋路的塊狀也會比較明顯。在 mask #2 的時候，輪廓與景深會明顯一些，紋路也相較自然許多。

## Part 3. Interpolation

- A. 檔案結構，在 `hw1_3.m` 這個 Script 裡面會呼叫兩個我們需要實作的函式 `NN(nearest-neighbor interpolation)` 和 `BI(Bilinear interpolation)`。

```
hw1_3.m -----> NN.m
|-----> BI.m
```

輸出的圖片：

```
Cat-----> NN_Cat.png (By NNI)
|-----> BI_Cat.png (By Bilinear)
```

- B. 本作業主要是要把一張圖片透過兩種不同的演算法將其放大四倍。  
 C. 兩者共同的部分主要在處理邊界都不是很簡單，所以我都忽略邊界（不計算）。

D. Nearest neighbor interpolation:

透過直接對應的方式（大圖的位置直接除以四，並找出比較接近的小圖座標）取值。為了計算出比較靠近的點，我使用了 `round` 來做四捨五入取值。

E. Bilinear interpolation:

本方式透過權重的概念，離點越近的顏色的影響力越大，也是取對應到原本圖片的該點與其右、右下、下，三格來做計算。

F. 結果圖：



- G. 可以發現在 NNI 的部分，他的顆粒狀很明顯，畢竟概念上就是直接把原圖放大，因此效果不是很好。在 Bilinear 的部分，就相對圓滑，感覺上比較有漸層補充（每一個都會受到其周圍的權重而變），效果也就比較好看。當然在放大之後，跟原圖相比肯定是不夠好的。