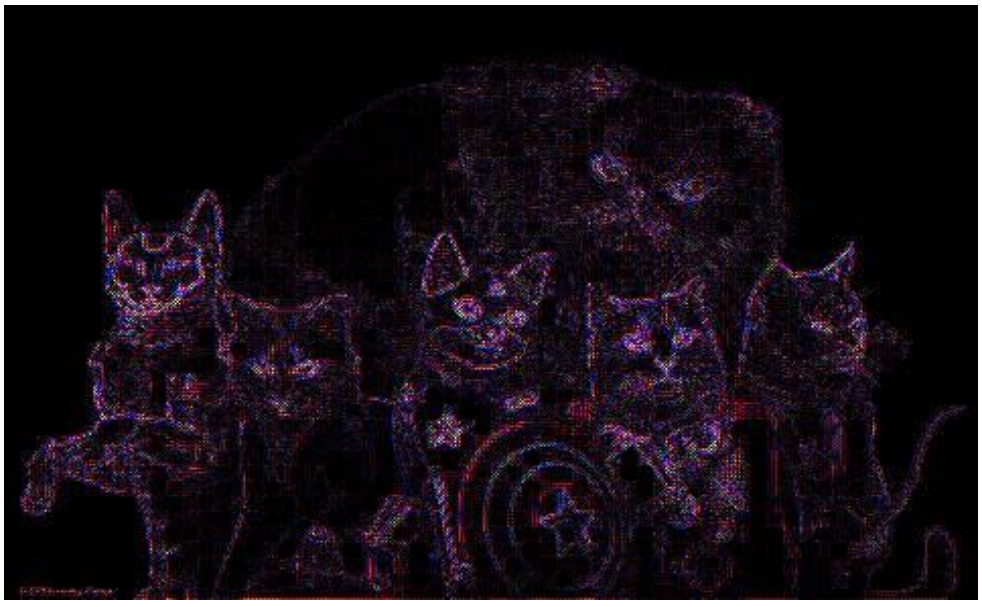
Introduction to Multimedia

HW1 Report

第一題的(a)部分，要求用 Nearest Neighbor Algorithm 實作 Demosaicing，我的作法是開一個相同大小的 TEMP 新陣列，針對原圖的每一個像素(i,j)，去判斷這個像素的非零值是 RGB 的哪個，如果是 R，則直接把 R 填入 TEMP(i,j)的 R，G 跟 B 則分別取周遭四個取平均，填入 TEMP(i,j)的 G 或 B。如果原圖(i,j)非零值是 B 也是一樣的做法，比較特別的是原圖(i,j)非零值是 G 的狀況，因為他周遭的 RB 排法可能有兩種，所以 R 跟 B 都是分別取上下左右四個相加然後除以”二”(畢竟不管如何周遭四個像素裡只會有兩個有對應的 R 或 B)。

Demosaicing



第一題(B):

上圖是 Absolute color Difference，已乘以五倍來突顯結果。

PSNR(Peak Signal to Noise Ratio)可以用來度量一張處理過的圖與原圖之間的相似度，公式是 10 * log10(信號最大值^2/MSE)，MSE 是原圖像與處理圖像的均方誤差，整體而言 PSNR 代表的意義是圖片最大表示可能的信號值(此處為 255)與兩張圖之間均方差的比值，單位是 dB(也就是為何要取對數的原因)，是一項廣泛被應用於客觀衡量影像處理品質的方式。然而 PSNR 並不能絕對描述一張圖的視覺品質，畢竟人類觀看圖片時會受到許多各種因素影響，對誤差的敏感度並不是絕對的。

這張圖計算出來的 PSNR 值略微多於 31，去網路上查了一下發現這樣子的 PSNR 值是合理的，一般影像處理的 PSNR 介於 20~40，30 以上代表有一定的還原程度了。

討論：

實作時一開始是用 uint8 去儲存這張圖，但是輸出的結果雖然有點接近，但是會有奇怪的斑紋(類似 Pattern Dithering 的那種)，可是一旦改成用 double 去儲存這張圖，結果瞬間清晰許多，

讓我對 Matlab 裡面處理不同類型變數的方式感到有點好奇，畢竟我也沒有對 uint8 做負數運算，結果為何會不同其實讓我有點困惑。(後來我認為可能是因為小數值後面的值被捨棄掉的問題)。另外值得一提的是，其實我一開始是把每個像素周遭八個取平均計算，但是這樣反而試出來的 PSNR 比前述做法還要低，所以採用才上述作法)

第二題(A)

這一題要求我們實作 Thresholding。基於想要嘗試與實驗的心態，我試了 3 種 Dithering 的方式：Random(0~255) 、Average、Pattern，以及最基本的題目所要求的 Threshold 來做比較。(Threshold 值經過一些嘗試之後我還是覺得 128 對我選的圖片來說比較適合，畢竟我的原圖算是畫面上有各種明暗程度的圖片，太極端的 Threshold 值會讓圖片失去很多細節)。

當像素低於算出來的門檻值時，將它設成 0，大於等於時設成 255。(全黑全白)

拿指定的Lena圖來說，Random是針對每一個像素都取0~255的一個隨機值Random來當門檻值。產生的結果雖然還是看得出原圖的脈絡但是顯得相當紊亂，不過也不失為一種有趣的畫面效果(有點像點畫)。Average如其名是把所有像素的灰階值取平均以此當門檻值，不過以Lena圖來說平均跟128實在太接近了，導致結果跟128門檻值的Dithering結果實在太像了。Pattern則是使用了一個3*3Thresholding Matrix [200 250 100;220 150 200;10 150 50]來對每個3*3為單位的九個像素分別做門檻處理，結果會讓畫面看起來整個畫面上都會有方方正正的格子，看起來有種對稱整齊的美感。128如其名就是以128(中間值)當作門檻值做Dithering。自選圖的部分比較不一樣的地方是它的Average值偏離128較多，呈現出來的結果跟128門檻值差滿多的。

Lena128

LenaAverageDithering

Lena Random

Lena Pattern

自選圖128

自選圖 Average

自選圖Random

自選圖 Pattern

第二題的(B)的部分，用了兩種指定的MASK去做Error Diffusion，設置一變數error，原像素值(i,j)<=127時error值就直接等於像素值，>127時則把像素值(i,j)-255當成error值。做errorDiffusion把error值Diffuse出去，把周圍像素的值加上以特定比例(題目上所給的兩個2*3矩陣[0 0 7,和3*5矩陣)乘上中心像素的error值，讓整張圖的像素值分布得以變得比較平均，好處是影像比較不會有好幾個大塊大塊的黑色或白色區塊。邊界值的處理我用的方法是類似於直接在原圖的周圍全都塞入值為0的像素。以結果來說2*3的mask做出來的結果，像素的排列會比較整齊一點，畫面上的顆粒似乎看起來比較小顆而3*5的mask的像素排列就比較流動一點，畫面上的顆粒看起來就比較大顆(顆粒可以看成是黑色像素的聚集方式，較密或較鬆)。不過以整體來說，不論哪種diffusion的mask，處理出來的畫面放大看都是四處散布的顆粒，跟其他不是error Diffusion的Dithering(除了random)以外相比畫面比較不會那麼'濃稠'。

如果要我說哪種處理方法最'好'，其實我覺得有點弔詭，畢竟每個處理方式處理出來的效果我覺得都各有其特色，很難說哪個比較好。(不過如果單論對於原圖的還原程度來說，ErrorDiffusion的效果會算是最好的，可以一定程度的還原出原圖的陰影等細節)

Lena ErrorDiffusion 2*3

Lena ErrorDiffusion 3*5

自選圖 errorDiffusion 2*3mask

自選圖 ErrorDiffusion 3*5mask

第三題(A)，用Nearest Neighbor做Upsampling放大四倍，直接就是用
oriscale(i,j) = ori(round((i-0.5)/s+0.5),round((j-0.5)/s+0.5));
(oriscale為4x原圖，ori為原圖)。
的算法，對放大圖的每一個像素(i,j)都去尋找原圖中某個對應像素的值抓出來使用。然後為了矩陣取值的考量，把坐標軸偏離0.5做計算，才不會round出index是0或是>size的狀況。
第三題的(B)部分，改用Bilinear Interpolation的方式做Upsampling。雙線性內差的意義就是把x,y兩個維度分別做線性內插，以幾何意義來考量可以想像成放大圖每個點對應回原圖之後的點跟周圍四個整數點所圍出來的面積當作比例乘上原圖四個點的像素值，就是放大圖裡每個像素(i,j)最後的像素值。(用程式來描述的話 a=i/scale,b=j/scale;
左上比例= (1-(a-floor(a)))*(1-(b-floor(b)));
右上比例= (1-(a-floor(a)))*(b-floor(b));
左下比例= (a-floor(a))*(1-(b-floor(b)));
右下比例 = (a-floor(a))*(b-floor(b));
oriscale(i,j) = 左上*ori(x,y) + 右上*ori(x,y+1) + 左下*ori(x+1,y)+ 右下*ori(x+1,y+1);
(oriscale是4x原圖，ori是原圖。

Nearest Neighbor

Bilinear Interpolation

(C) 比較兩者的不同的話，可以看出 Nearrest Neighbor 在放大時(如下頁)邊線的鋸齒會頗嚴重，精細的部分會嚴重失真，而 Bilinear 鋸齒和失真狀況比較不那麼嚴重(畢竟用了內插來把放大過後像素中間的空隙依照相鄰像素的比例補上了)，放大看整體會呈現出一種朦朧美，當然如果不放大看的話其實結果差異並不大，如果比起圖片細節更在意處理速度的話 Nearest Neighbor 不失為一個不錯的選擇。

 f

上圖為 Nearest Neighbor 的局部放大，下圖為 Bilinear 局部放大。