Introduction of Multimedia HW1

103062318蔡尚倫

1. DCT image compression

包含DCT\_image.m dct2d.m decode.m encode.m idct2d.m

PSNR.m(這裡用大寫避開build in function conflict)

RGB：(n=2 ,4,8)

DCT\_image.m：imread image，並且把它轉成double計算，call

encode.m -> decode.m -> imwrite image ->PSNR(original,image)

show PSNR

encode.m：利用雙層for loop 每8個為間距取得block的起始值，call dct2d.m，矩陣可以用(,)來取得的slice，這邊我分成R、G、B三種分開實作(後面發現其實可以更快的寫法)，接著創造出一個8X8 mask matrix，結著依照n傳入的參數決定要保留的數量，與上方一樣使用雙層for loop，分別對每個block做點乘(.\*)，用意是會一一對應，如果我mask上的值是1的話會保留0的話捨棄

dct2d.m：做DCT轉換再matlab因為可以做矩陣的相乘，所以描述公式就比較簡單，由原來的公式可以轉為 A\*input\*A’ ，A可用一維的DCT來表示，如此一來就能簡化公式的繁雜，要注意 n值得填寫，因為這次是以8X8為一個block，所以 n=8。

decode.m ：與encode相反，呼叫idct2d.m，也不用在做mask

idct2d.m：將原本公式相反即可還原 A’\*input\*A

PSNR.m：把公式套入即可，因為會重複用到寫成function

n=2 a2.png PSNR = 26.888657



n=4 a4.png PSNR = 33.372222



n=8 a8.png PSNR = Inf



(a) PSNR大小是 n=8 > n=4 > n=2 ，PSNR代表的就是與原本圖片的差距，因為DCT是將圖片轉成頻率，而題目要求取左上角是要保留低頻的資訊，而一張圖片通常有locality的特性，就是在該點附近的顏色”通常”會跟自己很像，而高頻代表的就是圖片上物體與物體的邊界，會是顏色變化最巨大的部分，所以濾掉高頻會使的圖片邊界模糊，如同n=2的圖片所展現的，但有時為了壓縮，其實n=4的圖片還原度已經表現的不錯。

YIQ(n=2 4 8)

主要與RGB做法相同，不同的是在DCT處理以前要從RGB domain轉成 YIQ domain，最後做完 encode decode之後要記得再轉回RGB domain

n=2 b2.png PSNR = 26.888434



n=4 b4.png PSNR = 33.372743



n=8 b8.png PSNR = Inf



(b) PSNR大小是 n=8 > n=4 >n=2，理由跟(a)小題一樣，PSNR代表的就是與原本圖片的差距，也可以稱做還原程度，保留越多資訊當然PSNR會越大，當兩者沒有差距時，則PSNR為INF

(c) 兩者出來的結果PSNR是相同的，理由我想是說你在不同的domain上丟棄資訊，一樣是丟棄等量的資訊量，所以在做還原圖片時，兩者出來的結果會非常類似，兩者domain轉換是線性的，所以在哪個domain做DCT，結果是相同的，這裡注意到的是可能會因為你轉換矩陣小數點的準確度，而造成PSNR有些微差距，大概0.00X左右，如果結果不是差太多，應該都是能接受的範圍

2. Image filtering

包含：imgfilter.m

Gaussian filter：(n=3 segma=0.3) (n=9 segma=1)

imgfilter.m：先用fspecial創出Gaussian mask，這裡只要上網去查document填入參數即可，接著創造一個大小是(rows+2\*(n-1)/2, cols+2\*(n-1)/2) zeros matrix，目的用來填入原本image，方便filter的移動，將原本的圖片填到正中央的位置，這樣filter最中心的點剛好就對到圖片的第一個pixel，接著跑兩層for loop，將mask與對應到的block做.\*(點乘)，並將該矩陣的數值全部加起來，assign到輸出的image，跑出loop，再image imwrite

n=3 segma=0.3 (此圖片有縮放過，原圖請參考資料夾) ta\_gaus3X3.jpg

n=9 segma=1 (此圖片有縮放過，原圖請參考資料夾)

ta\_gaus9X9.jpg

(a)由濾鏡(n)大小可得知，如果n很大的話，就會參考到周圍大部分點，而Gaussian filter可以看作是一種權重的分配，是由Gaussian函數產生，是一種常態分布，以自己這個pixel當作是常態分佈的最高點，segma大小代表你這個點的重要性，越低的話資料越集中，波峰也就越高，越大的話資料就開始往兩側分散，波峰相較而言就比較低。所以第一張圖片採用 n=3 segma=0.3基本上幾乎就是採用自己的那個點當作輸出結果，所以圖片會覺得跟原本很像，然而n=9 segma=1的則是濾鏡較大，參考到的周圍值就越多，自己的權重也沒有那麼高了，所以圖片會將原本的noise消除不少。

Median filter：(n=3) (n=9)

主要的操作跟Gaussian差不多，最主要是在filter的使用不一樣，這次是將濾鏡選取的範圍內，找出這些所有值的中位數，而3\*3=9與9\*9=81剛好都是基數，所以中位數很好找，用這個中位數來代表這個pixel的輸出

n=3 (此圖片有縮放過，原圖請參考資料夾)



ta\_med3X3.jpg

n=9 (此圖片有縮放過，原圖請參考資料夾)



ta\_med9X9.jpg

(b) n=3的圖片很成功的消除了noise，而且圖片還是保持的清晰，然而n=9 的圖片雖然消除了noise，但卻讓圖片過於模糊，讓人感覺好像丟了許多的資訊。兩者的差距在於濾鏡的大小，濾鏡太大會造成sample到的值太多，進而有很大的機率被周圍pixel取代。Gaussian與median兩者median表現較佳，原因我想在於noise顏色是白色，白色的值恰巧是最高的，所以使用median時，白色自然不太可能成為中位數，反觀Gaussian雖然利用自己pixel權重較大的特性，可是還是會被noise的高數值影響，造成圖片沒辦法有效的過濾掉noise。

3. Interpolation

包含：interpolation.m PSNR.m

Nearest neighbor：

interpolation.m：創一個長與寬都是原本圖片四倍的zeros matrix，接著用兩層for loop跑4\*original rows 和 4\*original cols，接著用sampling的方式來取得該pixel應該要對應到原來original image的pixel，只要將 i/4 and j/4取round即可，因為原本就是要找靠近的點，所以四捨五入是很好的方法，也是講義上寫的方式，如此一來就能取得放大後的圖片，最後 image imwrite，由於下方圖片縮放後會影響效果，所以換頁展示。

nearest neighbor a\_4X.png PSNR = 26.236858



Bilinear：

interpolation.m：前置作業與nearest neighbor一樣，最核心的演算是內插法來決定權重，與課本相同，這裡不分RGB三種來做，使的code較為簡潔，但會跑出warning，不過不影響結果。Sample完的x與y分別取floor 與 ceil，產生四個變數f\_x c\_x f\_y c\_y，比例的取得方式用 y\_rat =rem(j/4,1)，這樣的方法可以讓我們將整數部分消除，用mod也可以(因為確保兩個值會是正數)，接著分別取出四個pixel(RGB)取出，方法是例如img(f\_x,f\_y,:)，用冒號就能代表取得該維度的vector，這裡代表的是三原色RGB，如此一來不用像之前一樣分開來做，再用cat合併。接下來套用內插公式，分別先將左上與右上執行內插(x軸上的比例，記得要乘另外一邊的比例，這部分看code比較能理解)，再來將左下與右下座內插，得到的兩個新矩陣，再做一次內插，但這次上下合併，不要用錯比例。如此一來就能完成內插出來的RGB vector，將其assign給輸出圖片的pixel，這裡會出現前面所說的warning，簡單來說compiler說有可能會改變矩陣大小，可是你在寫code時確保不會發生即可。最後 image imwrite，由於下方圖片縮放後會影響效果，所以換頁展示。

bilinear aa\_4X.png PSNR = 26.101265



(a)(b)已在圖片上方顯示

(c)這兩張圖片的差距在於nearest neighbor因為是選用其中一個原本的pixel，所以正常顆粒感會很重，另外一個bilinear因為用內插法將周圍四個點做了權重的乘積，所以會發現該pixel不會出現在original image上，產生近視的模糊感，兩者的PSNR值相近。我想這裡的PSNR就不在是還原度的概念，因為從原本資訊少的original image要產生資訊多的放大圖片，本來就是會有些技巧去混淆觀看者，以為兩者是接近的，所以我覺得這裡PSNR的意義是放大效果的表現，越高代表與高解析度圖片的”pixel顏色”相似程度越高，畢竟他解析度高，擁有的資訊就是比較多，不太可能完全猜測使用到的顏色，只能藉由周圍pixel顏色的權重分配做出類似的顏色，自然PSNR就不會太高。