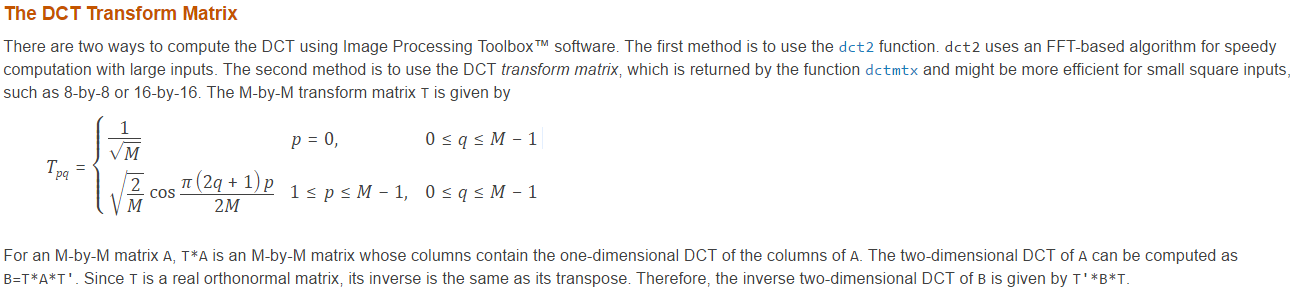
**CS3570 Introduction to Multimedia**

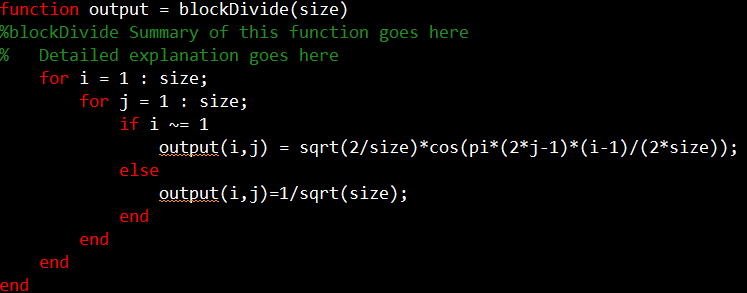
Homework #1 Report

103062234 張克齊

1. DCT image compression

[實作方法]



 <- Calculate Tpq

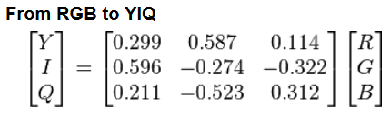
在Matlab官方文件中，有提到實作DCT的方法有兩種，一種是利用FFT-based algorithm，另一種則是利用DCT transform matrix來完成，我的方法是利用後者。因此，我們要先利用上圖公式計算出8 x 8的transform matrix，這樣我們就可以利用B = Tpq \* A \* Tpq' 計算出DCT後的結果，其中Tpq'是Tpq的transpose matrix；做這個矩陣相乘時我們要利用到blockproc()這個函式，可以幫助我們快速的處理分塊而不用使用到for迴圈，用法是要先建立一個structure表示我們想要處理的函數，然後在利用blockproc表示我們運算時要切的大小，切除來的就是上面的block\_struct.data的值，再進行func中的運算。

做IDCT前，我們要先對做完DCT的結果做題目要求的處理，就是先建造一個8 x 8的mask並依照指定的n給值 = 1，對DCT轉換後的結果做點乘，這樣才能再利用跟DCT一樣的方法做IDCT的轉換。轉換的方式就是從B = Tpq \* A \* Tpq' 變成 A = Tpq' \* B \* Tpq (Linear algebra method)。

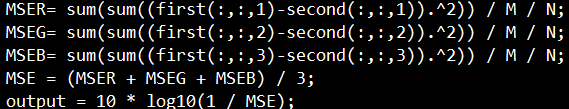
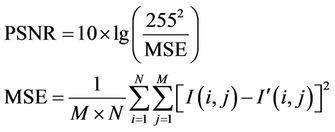
 <- applying mask

 <- A = Tpq\* B \* Tpq

至於YIQ的部分很容易，透過公式創立一個matrix K並取出input image的RGB各項進行矩陣相乘，做回IYIQ的部分就是變成inv(K)與YIQ各項進行矩陣相乘，也就是從YIQ = K \*RGB 變成inv(K) \* YIQ = RGB，就能得到結果。



PSNR的部分也是利用公式，但要注意的是因為彩色圖有RGB，所以要對三個維度分開計算在取平均，才是我們想要的MSE。圖中的的255是指峰值，但因為我們傳進去的是single (0~1)所以應該改成1。



[結果圖]

(a) DCT results:



n = 2, PSNR: 



n = 4, PSNR: 



n = 8, PSNR: 

(b) YIQ color model



n = 2, PSNR:  n = 4, PSNR: 



n = 8, PSNR: 

[結論]

(1) n = 8 時的討論: 我一開始在做n = 8時，做出來的結果是300左右，但是理想中mask 8 x 8在block 8 x 8上作用應該會使PSNR中MSE為0(因為圖片所有部分都保留)，答案要是Inf.才對。後來與同學討論發現應該是因為在做PSNR運算時，我們要先將圖片取成uint8 type再轉成single，才可以解決matlab中浮點數精度的問題，把一些接近於0的數給變成0，就能得到正確的PSNR值。



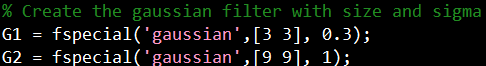
(2) PSNR的意義: PSNR愈大，代表圖片的失真現象較小，因為n取愈大代表我們圖片保留的部分愈多，所以PSNR會隨著n變大而變大；當n = 8時，代表著圖片被全保留，就是跟原圖一樣，所以PSNR無限大。(a)和(b)的結果相同，因為我們可以知道他只是從RGB的表示法經過線性轉換成YIQ，並沒有對圖片直接產生影響。

(3) (a)和(b)的差異: RGB的方式雖然和YIQ一樣，但因為資料量比較大，所以透過公式中那個3 x 3的矩陣(K)運算後，不僅能保留原本的圖片，也能達到稍微壓縮資料的效果。

2. Image filtering

[實作方法]

這一題是要實現filtering的功能，分成利用了Gaussian和Median的方法。

(a) Gaussian blurring: 首先依照題目提示我們可以利用fspecial()的函式創建出我們想要的Gaussian masks。接著，我們要運用卷積(CONV)的概念，算出每一點filter後的值，我對於邊界的處理是運用忽略法，就是邊界範圍的值不做CONV而直接取原圖的值，而運算卷積的技巧就是要先找出filter的中間點，接著一一對應Input算出中心點經過CONV處理後的值。

(b) Median filtering: 邊界處理的部分跟Gaussian一樣可以用忽略法，但在做運算的部分就有所不同；因為Median是給想劃分區塊並抓出該區塊的中位數，代表劃的區塊中間那個位置對應的值，所以我用了一個K存下當前我要處理的方塊。 然後，我利用reshape()的將K轉成column等於1，這樣就能利用matlab中的median() 函式算出正確的中位數(詳細原因請看討論)。

[結果圖]

(a) Gaussian blur filter:



mask sizes 3 x 3 mask sizes 9 x 9

(b) Median filter:

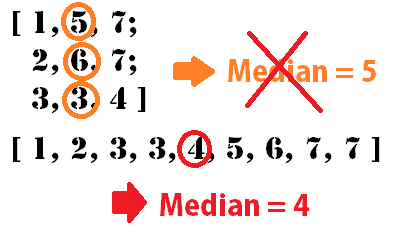


mask sizes 3 x 3 mask sizes 9 x 9

[討論]

(1) 邊界處理的方法: 有關邊界處理的方法有很多，像是這邊的忽略方法或是zero-padding等。忽略法的優點在於程式簡潔而且不會對原本的matrix進行操作，但缺點是會產生像是上面結果圖所呈現的邊界明顯的問題；採用zero-padding的話，必須依據filter的size在image matrix的四周向外補零，直到第一次filter的中心落在image第一個位置，這樣就不會產生忽略法造成邊界未處理的缺點，但是就必須對原本的matrix進行操作。

(2) 對matrix取中位數的方法: 在實作方法中，有提到要將原本方陣轉成total\_row\*1的矩陣才能做median而非對row和column各做一次median，這是因為再取中位數的時候並不能截成不同的段落取完再整合，因為它只代表在這個段落而非整個段落的中位數，如以下我自己舉的反例，第二個有經過sort比較好判斷，實際是直接跑內建函式median()。



(3) Gaussian blurring和Median filtering比較: 從得到的結果來看，我們可以發現當mask的size愈大時對圖片的影響會更大；更進一步來說，當mask大小為9 x9的時候，Median filtering比較能夠濾掉所謂的胡椒鹽雜訊，但也可以發現整張圖片相較於經過Gaussian blurring變得較為模糊，原因是因為Median filtering只取了中位數的點，而Gaussian blurring則要進行CONV的運算才能得到該點的值，所以不難理解後者對原圖保留的較多，結果也較不模糊。

3. Interpolation

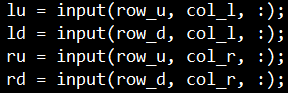
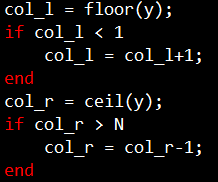
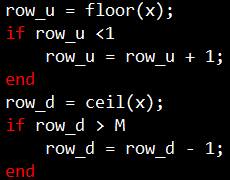
[實作方法]

這一題主要實作兩個interpolation的方法，Nearest Neighbor和Bilinear。

(a) Nearest Neighbor Interpolation (簡稱NN): 這個方法是很基本的方法，概念是將input依照指定的比例放大縮小，並將坐標系也一起縮放，但這樣會產生有些點落在分數的問題，NN的概念就是找到那些分數的點最接近坐標系的哪個整數點，而在color channel的部分就直接沿用該對應點的顏色。



(b) Bilinear Interpolation: 如上圖我們可以知道我們的目標是要找到對應點周圍的四個點並做interpolation，但要注意我們不能只取floor()和ceil()找到四個點，因為會有圖片邊界的問題，所以我們還要對取完的點做判斷，這樣我們就能正確取出我們想要做interpolation的四個點和手繪圖中的a和b。最後，找到四個點和a, b後，就能利用手繪圖中推得算式求出結果。





其中還要注意一開始取的點，我這邊是參照網路上的方法(Stack Overflow)來初始化，做了這樣的操作能確保所有的圖片邊界都會處理到，如下圖。

<http://stackoverflow.com/questions/1550878/nearest-neighbor-interpolation-algorithm-in-matlab>

https://scontent-hkg3-1.xx.fbcdn.net/v/t34.0-12/17580147_1285315578226569_1364908125_n.png?oh=907f7336bee4f89cfca2f08beeb0aa0e&oe=58DB87E9

https://scontent-hkg3-1.xx.fbcdn.net/v/t34.0-12/17580202_1285315728226554_60378655_n.png?oh=22c824ee43c41d6ac00990c31635bed2&oe=58DC4520

[結果圖]

(a) Nearest-neighbor interpolation



PSNR: 

(b) Bilinear interpolation



PSNR: 

[討論]

(1)兩種方法的比較: 因為做NN時，我們是直接強制將原本縮放後小數的點對應到整數點上，從計算上來看就會很明顯地知道這個方法產生的圖片鋸齒狀會很嚴重，尤其是倍率高的時候；而當我們做Bilinear時，我們會利用到他四周的四個點來進行內插運算，因此得到的值會較為精確，產生的結果也會較為平滑。兩者共同的是因為都不是取道最精確的點，所以原本顏色會些微跑掉，縮放後的圖片整體看起來都是模糊的。

(2) PSNR的意義: PSNR值愈高時，代表圖片的失真愈少，算是一種客觀的評比數據。所以在這一題中，我們可以知道經過Bilinear縮放後的PSNR應該要比經過NN縮放後的PSNR還來的高，因為我們算到比較精確的值，因此對於圖片的失真也較小。