

# JPEG Image Compression

蔡意維、陳博彥

這份報告分成兩個部分，第一個部分探討不同 block size 對於 JPEG 壓縮所造成影響 (JPEG 在壓縮圖片時，會先將 RGB 色彩描述改成 YCbCr 格式，再將圖片切成許多相同大小的 block 分別進行壓縮)，第二部分在探討，如何在保留一定影像品質得前提下，讓影像壓縮變得困難。

第一部分：

## JPEG Compression with Different Sizes of Blocks

JPEG 標準是以8x8的 block 進行壓縮，將原圖片切成許多8x8的 block 後，程式會針對每個 block 的數值做 discrete cosine transform，再將 transform 完的數值做 quantization (將數值除以某個參數再四捨五入，並在 decode 時乘回來)，隨後再將 quantize 後的 DC term (每個 block 最左上角的像素) 和 AC term (DC term 以外那些) 分別進行壓縮。其中細節在 survey presentation 時已經講過，在此就不贅述，詳情可參照 JPEG 標準規定。

我們嘗試用8x8 (JPEG 標準)、16x16、4x4三種 block size 進行試驗，結果呈現在下表中。從下表中我們可以得知，就檔案大小而言，8x8的壓縮效果最好，16x16與4x4分別比8x8大了9.6%和5.3%左右，差異不小。就影像品質(與原圖算 SSIM)而言，block size 越大，影像品質越好，但差異微乎其微。總和兩個觀測標準，8x8的影像品質不輸給其他，但檔案大小較小，也難怪會被選為壓縮表準。

注：SSIM ( Structural SIMilarity ) 是一個用來衡量影像視覺相似度的指標，數值會在1跟-1之間，兩張完全一樣的圖片會有1的 SSIM 。

PNG file(原圖)	8x8	16x16	4x4
			
768KB SSIM: 0.9937	32.0KB SSIM: 0.9939	35.0KB SSIM: 0.9939	33.9KB SSIM: 0.9926
			
768KB SSIM: 0.9920	31.7KB SSIM: 0.9920	35.4KB SSIM: 0.9922	33.6KB SSIM: 0.9907
			
768KB SSIM: 0.9838	33.5KB SSIM: 0.9838	36.2KB SSIM: 0.9842	34.9KB SSIM: 0.9822

在縮的過程中，我們需要使用到 quantization table，quantization table 的大小應與 block size 一樣。在8x8的壓縮中，我們採用 JPEG 的標準 quantization table，在16x16的壓縮中，我們將標準 quantization table 的每一格分裂成同樣數值的四格，藉此生成16x16的 quantization table，在4x4的壓縮中，我們則將標準 quantization table 的相鄰四格取平均，藉此生成4x4的 quantization table。以下為三個大小的 quantization table。

8x8 quantization table (luminance):

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	103	100	99

4x4 quantization table (luminance):

12.7500	14.7500	37	56.7500
14.5000	22.7500	58.7500	66.7500
24.7500	53	90.5000	96.2500
69.2500	89.5000	109.7500	105

16x16 quantization table (luminance):

16	16	11	11	10	10	16	16	24	24	40	40	51	51	61	61
16	16	11	11	10	10	16	16	24	24	40	40	51	51	61	61
12	12	12	12	14	14	19	19	26	26	58	58	60	60	55	55
12	12	12	12	14	14	19	19	26	26	58	58	60	60	55	55
14	14	13	13	16	16	24	24	40	40	57	57	69	69	56	56
14	14	13	13	16	16	24	24	40	40	57	57	69	69	56	56
14	14	17	17	22	22	29	29	51	51	87	87	80	80	62	62
14	14	17	17	22	22	29	29	51	51	87	87	80	80	62	62
18	18	22	22	37	37	56	56	68	68	109	109	103	103	77	77
18	18	22	22	37	37	56	56	68	68	109	109	103	103	77	77
24	24	35	35	55	55	64	64	81	81	104	104	113	113	92	92
24	24	35	35	55	55	64	64	81	81	104	104	113	113	92	92
49	49	64	64	78	78	87	87	103	103	121	121	120	120	101	101
49	49	64	64	78	78	87	87	103	103	121	121	120	120	101	101
72	72	92	92	95	95	98	98	112	112	103	103	100	100	99	99
72	72	92	92	95	95	98	98	112	112	103	103	100	100	99	99

討論：

至於到底為什麼 block size 會影響 JPEG 檔案大小呢？由於 JPEG 壓縮時，會把 Y、Cb、Cr 三個色彩維度，和 DC、AC 項係數分別壓縮，壓縮檯的大小則為六 ( 3x2 ) 個部分的總和，我們可以透過觀測六者分別的大小來獲得更多的瞭解。

以下為以512x512的 Lena.bmp ( 第一張表的第一張圖 ) 為例，使用8x8, 16x16或4x4方格來壓縮之下，DC 和 AC term 的 Huffman code 的 bit 長度。

	8x8	16x16	4x4
DC(Y)	23449	7677	75080
DC(Cb)	4167	1583	11528
DC(Cr)	3943	1421	11360
AC(Y)	140405	146270	145295
AC(Cb)	8317	10160	9400
AC(Cr)	8495	9908	9186

由上表可知，若以16x16的方格壓縮，則能有效減少 DC term 長度，但 AC term 為255項的 zero run-length coding，會有較多複雜的 term 而使 code 變長；以4x4的方格壓縮，則因為需要取較多方格，DC 和 AC term 的 code 皆會變長。

16x16的 bit rate 較8x8低，檔案壓縮率卻沒有8x8好，顯示 bit rate 和檔案大小沒有絕對關係。

未來可研究方向：

本次實驗中，16x16或4x4的 quantization table 表皆以8x8做簡易修改得之，Huffman code 的對照表也是直接挪用8x8的。若能設計更好的 quantization table，16x16、8x8的 SSIM值有望提升；若能夠統計出符合16x16或4x4特性的 Huffman code 對照表，檔案大小則有望下降。總之，16x16和4x4的壓縮還有進步空間，可以是未來繼續研究之方向。

程式碼：

這部分的程式碼儲存在 code/part1/中。

compression.m 可讀取影像並以8x8方格壓縮得到 Huffman code，inv\_compression.m 可讀取 Huffman code 並還原成解壓縮後的影像；

16x16以 compression16.m, inv\_compression16.m 實作；

4x4以 compression4.m, inv\_compression4.m 實作；

SSIM.m 可得到兩張影像的結構相似度，參數  $c_1, c_2$  設為 0.01, 0.03。

第二部分：

### Making Compression Harder – By an Annealing Approach

想像我們是不懷好意的攻擊者，我們收到一張圖片，並且想在不大幅改變影像內容的前提下，使得這張圖片很難被 JPEG 壓縮，我們該如何做呢？

用正式一點的方式來定義這個題目，假設我們收到的圖片是  $\text{image}$ ，要產生的  $\text{output}$  是  $\text{image}'$ ，題目的定義如下：

Maximize  $\text{JPEG\_Size}(\text{image}')$

Subject to  $\text{Visual\_Similarity}(\text{image}', \text{image}) > \text{threshold}$

首先，該如何量化  $\text{Visual\_Similarity}$  呢？我們依然採用 structural similarity (SSIM) 當作標準，條件式即為  $\text{SSIM}(\text{image}', \text{image}) > \text{threshold}$ 。

演算法：

我們採用 simulated annealing 的方式來完成這樣任務，pseudocode 如下：

```
Initial temperature  $T > 0$ , annealing rate  $0 < r < 1$ 
image=original_image
while ( time_used < time_limit ):
    for 1 <= i <= P:
        image' = add_noise ( image )
        if SSIM( image', original_image ) > threshold:
            improvement = JPEG_size(image') - JPEG_size(image)
            if improvement > 0:
                image=image'
            else:
                image=image' for probability exp( improvement/T )
        T=r*T
return image
```

其中， $T$ 、 $r$ 、 $P$  是可以調整的參數，**add\_noise** 是在圖片上加上雜訊，**threshold** 是 SSIM 的門檻，接下來，我們會對這些設定做討論。

### (1) 改變 T、r、P

改變這幾個參數對於實驗結果會有什麼影響呢？在經過多次實驗後，我們發覺這幾個參數對於實驗結果的影響微乎其微，他們只會影響收斂速度，但不影響最後產出的圖片樣式。

### (2) 調整 add\_noise 方式

#### 方式一：Gaussian noise

顧名思義，這個方式就是在每個像素的 RGB 數值上，都加上 iid Gaussian noise，Gaussian 的平均是0，變異數是要手動調整的，如果變異數設的太小，收斂時間會大幅提升，如果變異數太大，生出來的影像跟原圖的 SSIM 大於 threshold 的機率會很低，因此演算法也會卡住，遲遲不更新生成的圖片。Threshold 越大，就要用越小的變異數來搭配，反之亦然。

#### 方式二：Uniform noise

顧名思義，這個方式就是在每個像素的 RGB 數值上，都加上 iid 的 uniform noise，這個 noise 的範圍是在正負 k 之間。與 Gaussian 十分類似的是，如果 k 設的太小，收斂時間會大幅提升，如果 k 太大，生出來的影像跟原圖的 SSIM 大於 threshold 的機率會很低，因此演算法也會卡住，遲遲不更新生成的圖片。Threshold 越大，就要用越小的 k 來搭配，反之亦然。

至於到底哪個方式比較好呢？實驗結果顯示，兩個方式得到的最終結果都差不多（長相差不多，檔案大小也差不多），但 Gaussian 通常收斂得比較快。

由以下的比較表，我們可以看見，Gaussian 和 uniform 兩種方式所生出來的圖片看起來幾乎一樣，檔案大小也一樣，但 uniform 所花費的 iteration 數則比 Gaussian 多上不少。整而言 Gaussian 是比較好的方式。

	原圖	Gaussian	Uniform
圖片			
File size (kb)	13	21	21
花費的iteration數	0	36	53

### (3) 調整 Threshold :

Threshold 越低，演算法就可以盡可能得亂改圖片內容來加大檔案大小，但生成的圖片則會跟原圖差異較大。下表比較不同 threshold 下所生成的圖

片，其中 threshold 為 1 的那張就是原圖。

圖片					
Threshold	1	0.9	0.8	0.7	0
File size (kb)	13	17	21	28	31

未來可研究方向：

這次的實驗室採用 annealing 的方式，比較偏向 heuristic approach。但事實上，我們可能可以根據 JPEG 的壓縮設定，直接 construct 出一張大小更大的圖片（用解的，而非用試的）。例如，觀測 JPEG 壓縮時的 Huffman code 表，看哪個 symbol 長度最長，就生成許多那個 symbol。這也是未來可以繼續嘗試之方向。

程式碼：

本實驗是由 `code/part2/annealing.py` 所完成的。