JPEG Codec Implementation

一.前言

這次的期末專題選擇 JPEG codec 的實作，會選擇此主題的主要原因在於這門課所教到的許多編碼技巧都實際應用在 JPEG 上，且 JPEG 現今仍被廣泛使用，為了讓自己能夠更了解理論與實作，希望透過此次 project，能夠了解 JPEG 的 format，以及 JPEG codec 的運作。並利用課堂教授的編碼技巧與理論實作 JEPG codec。

此報告將會分成以下幾個部分:介紹 JPEG 的規範以及壓縮方式，並介紹 JPEG 編解碼器的流程圖，為了解碼的工作順利進行，我們需要了解和讀取 JPEG 的檔案資料。有了以上的基礎後，再進入 Encoding 和 Decoding 的詳細介紹，最後顯示結果。

二. JPEG 介紹

JPEG 是第一個由國際標準化組織和國際電話電報諮詢委員會為靜態圖像制定的第一個國際數位影像壓縮標準，直到現在仍作為最常使用的影像壓縮標準，例如 Android 手機拍照，產生的檔案都是 jpg 檔，如下圖所示:

圖 1:使用手機拍照後生成的 jpg 格式檔案

JPEG 為失真壓縮演算法，代表影像的品質在壓縮後會降低並損失一些不重要的資訊，但同時能夠提高很高的壓縮比。JPEG 的壓縮算法可以分成四種:分別為 sequential、progressive、lossless 和 hierarchical。

Sequential 的方式為由上而下的解碼，如下圖所示:

圖 2:Sequential encoding(註:該圖檔為 gif 格式，在 pdf 檔案中為靜止圖片，因該 gif 動圖效果可在 PPT sequential encoding 找到同樣的)

實現較為簡單，但是當圖片的大小較大，或是網路速度較慢時，效率較低，會到圖片一行一行加載的效果。

Progressive 會掃描多次圖片，這些掃描順序儲存在 JPEG 文件中。打開文件的過程中，會先顯示整個圖片的模糊輪廓，隨著掃描次數的增加，圖片變得越來越清晰，如下圖:

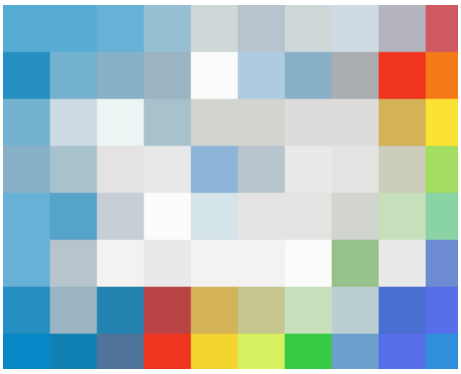

圖 3:Progressive encoding(註:該圖檔為 gif 格式，在 pdf 檔案中為靜止圖片，因該 gif 動圖效果可在 PPT progressive encoding 找到同樣的)

Lossless 是個無失真的圖像壓縮標準，從與目標最近的三個方格，通常為上方，左方以及左上方預測，並在預測誤差中使用 entropy coding。這個壓縮方式常被使用在醫療影像。

Hierarchical 的方式較為複雜，基本上和 progressive 相同，但會根據不同的 resolution 使用不同的 encoding.

此外，在 entropy coding 中可選擇 Huffman coding 或 Arithmetic coding 方式。

本次專案採用 Baseline JPEG，為最簡單也最廣泛被使用的 JPEG 方式，採用 sequential encoding 和 Huffman coding.

三.JPEG Codec Flow Graph

下圖為 JPEG 編碼和解碼的流程圖，左邊為原始影像，經過一系列步驟後壓縮成右邊的 JPEG 檔案，而將步驟反過來作則為解壓縮，將右邊的 JPEG 解壓為原始的影像。

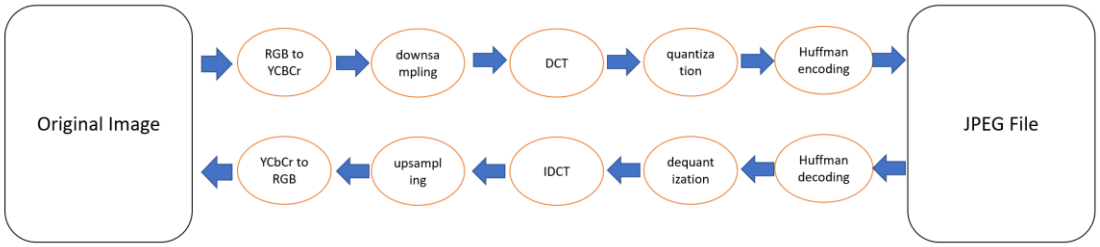

圖 4:JPEG 編解碼流程圖

根據上圖可以將壓縮分為五個步驟:第一步為 color space transformation，將 RGB 色域轉換為 YCbCr，進行降採樣(downsampling)、DCT、量化，最後霍夫曼編碼成 JPEG。

DCT 變換和 Huffman coding 都是完全可逆且無損的，為了提高壓縮率，baseline JPEG 會移除人眼感受不到的資訊，如將 RGB 轉換為 YCbCr，並在 DCT 變換後量化數據。

四.JPEG File Part

由於 JPEG 支援多種 coding 方式，因此必須記錄我們選擇的種類，因此一個 JPEG 檔案包含以下幾個區段:

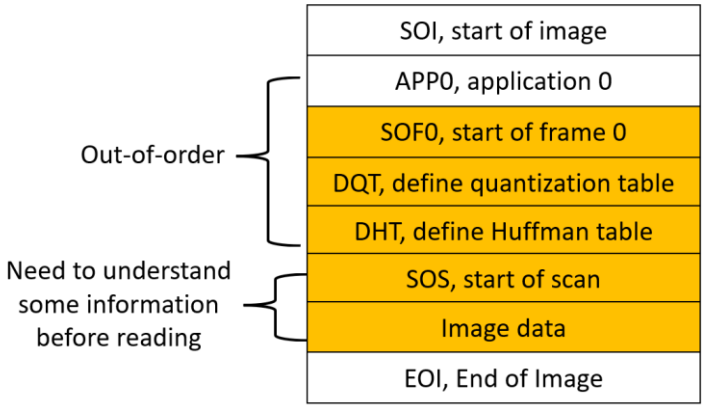

圖 5:JPEG 檔案架構圖

為了記錄算法，因此檔案會包含使用的算法(SOF0，0 代表 baseline)，量化

表(DQT)、Huffman table(DHT)、壓縮圖像數據(Image data)。但比較特別的是，JPEG 檔在開始和結尾處，分別有「檔案開始(SOI,start of image)」和「檔案結束(EOI,end of image)」的標記。另外為了遵循 JFIF 的規範，會多出一個區段 APP0，紀錄 JFIF 的額外資訊。採用哪種算法跟降採樣率放在同一個區段，壓縮圖像數據會在前面放置一個 SOS 區段，用以告知解碼器接下來該如何讀取。

APP0、SOF0、DQT 和 DHT 這四個沒有固定的順序，而 SOS 和 Image data 則要先瞭解一些訊息才能夠讀取數據。值得一提的是，實際上在解碼階段並不需要 APP0，以及 DQT 和 DHT 可能不只一個，下圖為一例子:

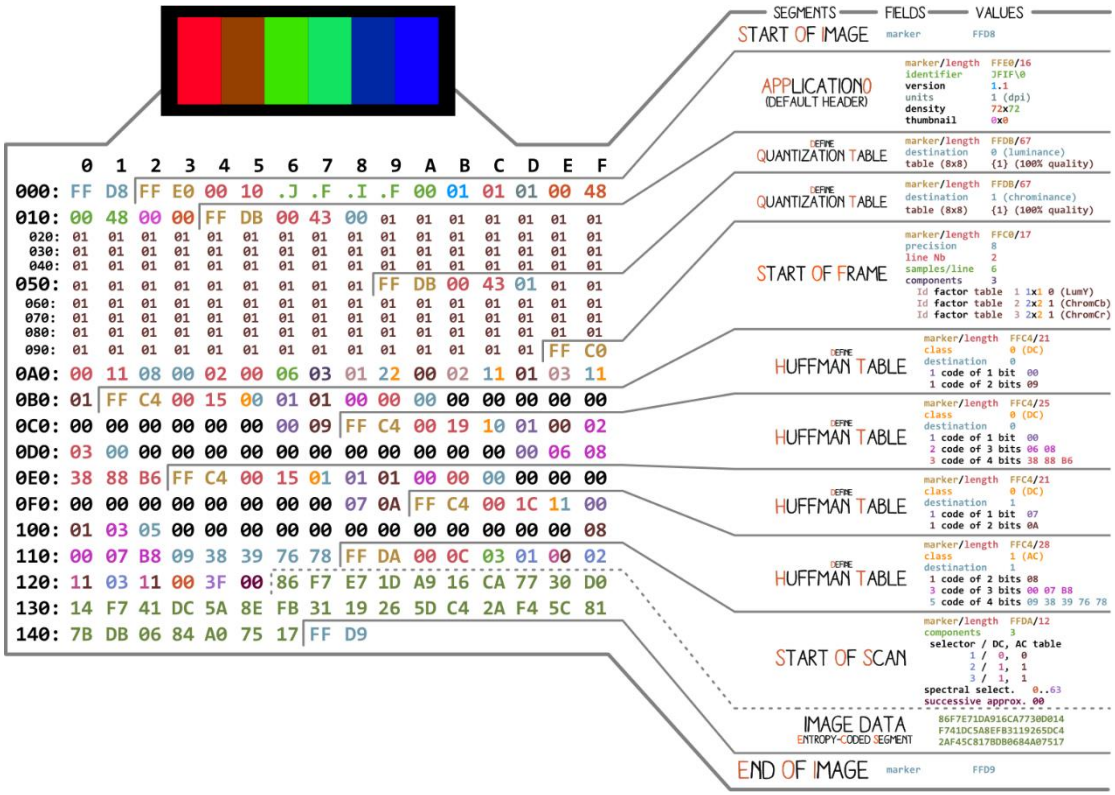

圖 6:JPEG 檔案結構圖。

透過上圖可以發現，每個區段開始的前置碼都一樣，而幾個比較重要區段的前置碼如下:

DQT	0xFFDB
DHT	0xFFC4
SOF0	0xFFC0
SOS	0xFFDA

圖 7:區段前置碼

之後便可透過這些前置碼，辨識接下來要讀取的是哪個區段，在之後的讀取檔案結構會派上用場。

五.Encoding JPEG

如前所述，Encoding 的過程牽涉到五個步驟:色域轉換、降採樣、DCT、量化、Huffman encoding，以下分別詳細說明:

1. RGB 轉換至 YCbCr

由於人眼對亮度的敏感程度高於對色彩的敏感程度，JPEG 利用此特性，在壓縮圖像時將亮度與顏色分開處理。JPEG 不會對亮度做太多改變，而人眼對顏色較不敏感，所以在人眼查覺到顏色失真之前，JPEG 對顏色進行壓縮處理，就算損失部分細節，我們也不易察覺。

YCbCr 中，Y 表示亮度(luminance)、Cb 和 Cr 表示彩度(chrominance)，將 RGB 轉換為 YCbCr 的公式如下:

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.334 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

圖 8:RGB 轉 YCbCr 公式

2. 降採樣(downsampling)

在 YCbCr 模型中，Cb 和 Cr 通道所包含的訊息量遠少於 Y 通道包含的訊息量，同時人眼對於亮度的敏感度高於色彩的敏感度，因此 JPEG 主要對 Cb 和 Cr 進行降採樣，又稱為縮減取樣。能夠達到很好的壓縮效能，但同時又不會損失我們能夠察覺到的明顯資訊。常見的採樣標準如下圖:

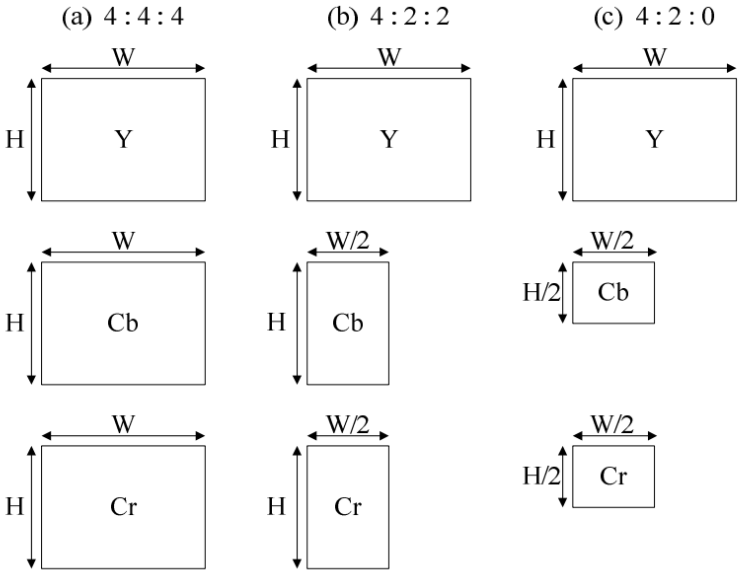

圖 9:降採樣格式

(1) 4:4:4 format:又稱為無縮減取樣。

(2) 4:2:2 format:每 4 個 Y 取樣，有 2 個 Cb 和 2 個 Cr 取樣，造成每行的像素減半，但每個 frame 的行數相同。

(3) **4:2:0 format:**對 Cb 和 Cr 分量在水平和垂直方向進行一半採樣，每 4 個 Y 採樣還有 1 個 Cb 和 1 個 Cr 採樣。

不論降採樣的格式為何，在解碼時，都需升採樣為 4:4:4 format。

3. DCT

JPEG 將圖像轉換為許多 8 * 8 的像素區塊(MCU, Minimum Coding Unit)，並將 DCT 套用在每個 block。使用二維 DCT-II 將時域空間變換是頻率域空間，一個經過 DCT 變換後的矩陣如下:

$$\begin{bmatrix} -415 & -33 & -58 & 35 & 58 & -51 & -15 & -12 \\ 5 & -34 & 49 & 18 & 27 & 1 & -5 & 3 \\ -46 & 14 & 80 & -35 & -50 & 19 & 7 & -18 \\ -53 & 21 & 34 & -20 & 2 & 34 & 36 & 12 \\ 9 & -2 & 9 & -5 & -32 & -15 & 45 & 37 \\ -8 & 15 & -16 & 7 & -8 & 11 & 4 & 7 \\ 19 & -28 & -2 & -26 & -2 & 7 & -44 & -21 \\ 18 & 25 & -12 & -44 & 35 & 48 & -37 & -3 \end{bmatrix}$$

圖 10:DCT 變換後的矩陣

其中，左上角稱為低頻區，右下角為高頻區。人眼對於高頻區較難分辨，但對於低頻區則有較好的辨識力，如下圖:

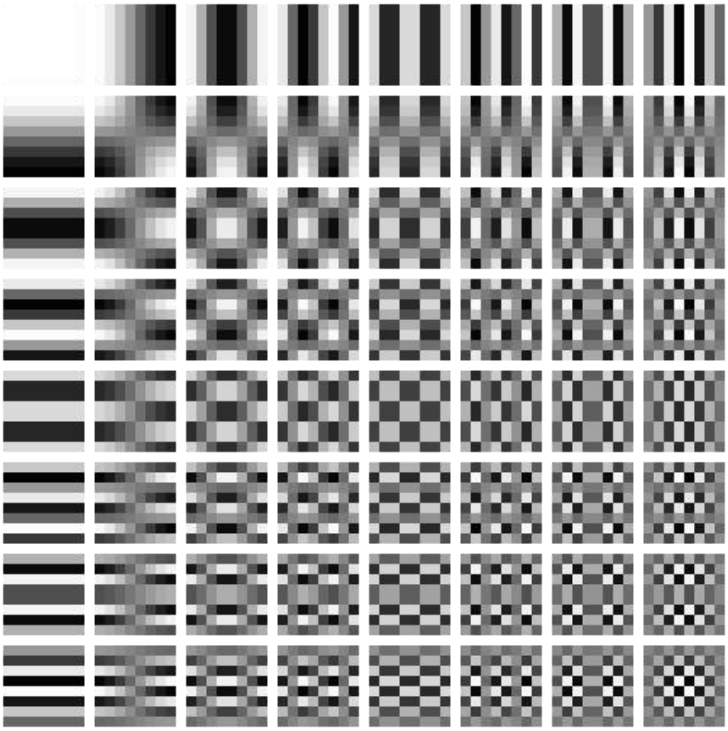

圖 11:DCT 餘弦

可以看到左上角低頻區域，當有顏色差異時我們可以很明確的辨識出，當越到右下角的高頻區，如果只有其中一個小區塊進行顏色改變，我們要花比較多的時間才能辨識出差異，或者甚是根本看不出差別。這也是為何要使用 DCT，為了後面的步驟方便處理資訊。

而 DCT 比 DFT 更適合圖像壓縮的原因在於，DCT 可以將變換訊號的能量集中在低頻率，而 DFT 不能。根據 Parseval 定理，能量在空間域和頻率域是相同的。如前所述，由於人眼對低頻的敏感度較低，我們可以關注在低頻區域並減少 DCT 後的高頻區域。

4. Quantization

做完 DCT 時，所得到的數字必須再進行處理。量化就是對 DCT 係數的一個優化過程。量化目標盡量將高頻區域的 DCT 係數減少至 0，產生的 0 越多，就能夠壓縮得越多。

在量化過程中，每個 8 * 8 的 DCT 矩陣的係數會除上一個各自對應的量化值，並做四捨五入。量化值為 DCT 矩陣每個係數位置對應的量化矩陣係數。如下式子:

$$F(u,v)_{Quantization} = round\left(\frac{F(u,v)}{Q(u,v)}\right)$$

假設有個 DCT 矩陣如下:

$$\begin{bmatrix} -415 & -33 & -58 & 35 & 58 & -51 & -15 & -12 \\ 5 & -34 & 49 & 18 & 27 & 1 & -5 & 3 \\ -46 & 14 & 80 & -35 & -50 & 19 & 7 & -18 \\ -53 & 21 & 34 & -20 & 2 & 34 & 36 & 12 \\ 9 & -2 & 9 & -5 & -32 & -15 & 45 & 37 \\ -8 & 15 & -16 & 7 & -8 & 11 & 4 & 7 \\ 19 & -28 & -2 & -26 & -2 & 7 & -44 & -21 \\ 18 & 25 & -12 & -44 & 35 & 48 & -37 & -3 \end{bmatrix}$$

量化矩陣為:

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

經過量化後所得的值為:

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -3 & 4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

量化過後，左上角的第一個稱為 DC 係數(以上圖為例: -26)，其餘剩下的 63 個係數稱為 AC 係數。此外，必須確保量化過後的係數在[0, 255]之間。

5. Zig-zag

觀察量化結果的矩陣可以發現，越往右下角(高頻區域)就出現越多的 0。我們可以使用一個方式掃描整個矩陣的元素，將低頻區域與高頻區域區分，這個方式稱為 zig-zag scan，掃描方式如下:

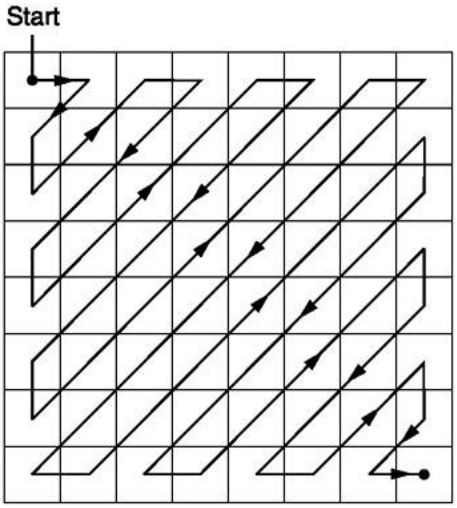

圖 12:Zig-zag scan

Zig-zag 會按照上圖的順序逐一讀取掃描矩陣中的每個元素，假設有一個矩陣經過 DCT 和量化後結果如下:

$$\begin{bmatrix} 15 & 14 & 10 & 9 \\ 13 & 11 & 8 & 0 \\ 12 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

圖 13:Zig-zag 矩陣

則經過 Zig-zag 掃描後的結果為:[15, 14, 13, 12, 11, 10, 9, 8, 0, 0… 0]。可以發現低頻區域係數都在儲存在前半部分，這對後續的編碼(如 Run-length encoding)有很大的幫助。

6. Run-length and Delta encoding

經過 Zig-zag 掃描後，會發現越到高頻區域後面基本上全部為 0，而可以使用 run-length 方式使用較少的位元表示後面所有的 0。Run-length 的編碼格式為(R, L) = (出現 0 的數目, 非零值)。以圖 13 為例，DC 值之後接著之 AC 值為 14(非零值)，而且 14 之前出現 0 的數目為零，因此可編為(0, 14)；接著 14 後面出現 13(非零值)，而且 13 之前出現 0 的數目為零，因此可編為(0, 13)；依此類推後，可依序編碼為(0, 12)、(0, 11)、….(0, 8); 從 8 之後之值皆為 0，因此可直接編 EOB 這個碼字，代表從 8 之後剩下的值皆為 0。

在 JPEG 標準中，對於每隔 8 * 8 block 採用 Delta encoding。利用 Delta encoding 的目的在於，由於每個 8 * 8 block 的 DC 值基本上都很接近，因此，與其儲存各 DC 值，倒不如儲存前後兩兩 DC 值之間的差值。透過此方式可

進一步達到料壓縮的目的。且硬體實現也非常容易完成。如果我們更改第一個 DCT 的係數，則後面所有的係數都會被影響到，但如果我們只更改最後一個 DCT 的係數，則影響的部分較少。圖像中的第一個 DC 值通常變化最大，通過 Delta encoding，可以使其餘的 DC 值接近 0。這對於後續的 Huffman encoding 中能夠達到更好的壓縮。

7. Huffman Encoding

Huffman encoding 為無失真編碼方式。出現機率越高者使用較短的編碼長度。透過此方式可達到資料壓縮的目的。在 JPEG 編碼中對於 DC 值與 AC 值分別有其相應的霍夫曼表。因此在編解碼的過程中，透過查表法可以輕易的實驗影像訊號的霍夫曼編碼。各表如下:

Size	Length	Codeword
0	2	0
1	3	10
2	3	11
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

圖 13:DC 值的霍夫曼表

Size	Length	Codeword
0	2	0
1	2	1
2	2	10
3	3	110
4	4	1110
5	5	11110
6	6	111110
7	7	1111110
8	8	11111110
9	9	111111110
10	10	1111111110
11	11	11111111110

圖 14:AC 值的霍夫曼表

在 JPEG 中，可包含最多 4 個霍夫曼表，並且儲存在資料中的 DHT 區域。DCT 係數儲存在兩個不同的霍夫曼表，其中一個只儲存 DC 值，另一個為 AC 值。這代表在解碼的過程中，必須將 DC 和 AC 值結合。且 DCT 中對於 luminance 和 chrominance 通道的資訊是分開儲存的，因此最後我們總共會得到 4 個霍夫曼表。

六.Encoding JPEG

解碼過程就是編碼的相反，但是解碼過程中，需要讀取檔案結構，因此需利用前面章節所描述的 JPEG File Part 輔助。

1. Huffman Decoding

在進行 decoding 前，首先要讀取 Huffman table。每個 DHT(Defined Huffman Table)包含以下幾個部分:

(1) Marker Identifier : 大小為 2 bytes，用來辨識 DHT 開頭，值為 0xFFC4。

(2) Length : 大小為 2 bytes，代表 Huffman table 的長度。

(3) HT information : 大小為 1 bytes，其中 bit 0 ~ 3 代表 Huffman table 的數字; bit 4 代表 HT 類型，0 為 DC table，1 為 AC table; bit 5 ~ 7 未使用則為 0。

(4) Number of Symbols : 大小為 16 bytes，代表編碼長度為 1~16 的符號個數。

(5) Symbols : 大小為 n bytes，包含按代碼長度遞增順序排列的符號的 table(n = 代碼總數)。

2. Dequantization

反量化前，一樣先讀取 Quantization table(QT)，QT 包含以下幾個部分:

(1) Marker Identifier :大小為 2 bytes，用來辨識 QT 開頭，為 0xFFDB。

(2) Length : 大小為 2 bytes，代表 QT 的長度。

(3) QT information : 大小為 1 bytes，其中 bit 0 ~ 3 代表 QT bits 數; bit 4 ~ 7 代表 QT 的精度，0 代表 8 精度，其餘為 16 精度。

(4) Bytes : 大小為 n bytes，代表 QT 值。

根據 JPEG 標準，共有 2 個 QT，一個 luminance 和一個 chrominance，兩者都以 0xFFDB 開頭。

3. Decoding Start of Frame(SOF)

檔案開頭包含以下資料:

(1) Marker Identifier : 2 bytes，以 0xFFC0 當作 SOF 開頭。

(2) Length : 2 bytes，代表長度。

(3) Data precision : 1 bytes，以 bit 為單位，通常為 8。

(4) Image Height : 2 bytes，值必大於 0。

(5) Image Width : 2 bytes，值必大於 0。
(6) Number of components : 1 bytes。1 代表灰階，3 代表色彩 YCbCr。
(7) Each components : 3 bytes，1 byte for component id，1 byte for sampling factor， 1 byte for QT number。

在這些資料中。我們會需要 image width 和 height 以及 QT number。當我們開始對真正的圖像解碼，會需要用到 width 和 height。

4. Decoding Start of Scan(SOS)

這個部分包含了真正的圖像資料，前面所做的步驟可以當作是在為此步做準備。接著要讀取壓縮圖像數據流，數據流可以看成一個接一個的 MCU 串:
[MCU1], [MUC2], [MCU3], [MCU4]…
而一個 MCU 又可以進一步切割成不同顏色分量的串聯，以 Y -> Cb -> Cr 的順序如下:
[Y1, Y2, Y3, Y4, Cb1, Cr1]….
這裡的 Y1, Y2…每個都是一個 8 * 8 的 block，接下來只要瞭解一個 block 如何讀取就成功。需要注意的是，JPEG 允許 scan data 出現 marker(0xFFXX) 來增加容錯性，為了和 marker 進行區別，寫入 0xFF 的資料會寫入成 0xFF00

讀取一個 block 的流程為:
(1) 讀取 DC 係數 : 不斷從數據流中讀取一個 bit ，直到可以對上直流霍夫曼表中的一個碼字，取出的對應信源編碼代表的是接下來還要讀入幾位，假如是 n 就繼續讀取 n bits。
(2) 讀取 AC 係數 : 再接着取出 bit 直到對上交流霍夫曼表的一個碼字，取出對應信源編碼。這個代表個意義為:
  (a) 較高的 4 bits 代表接下來的數值連續有幾個 0。
  (b) 較低的 4 bits 代表這些 0 之後跟著的數值的位數，如果為 n 就繼續取 n 個 bits，以下表解碼後，就是這些 0 之後跟著的數值。

實際數值	位數	碼字
-1,1	1	0,1
-3,-2,2,3	2	00,01,10,11
-7,-6,-5,-4,4,5,6,7	3	000,001,010,011,100,101,110,111
-15,……,-8,8,……,15	4	0000,……,0111,1000,……,1111
-31,……,-16,16,……,31	5	00000,……,01111,10000,……,11111
-(2^n -1),……-2^(n - 1),2^(n - 1),……2^n - 1	n <= 11	所有長度爲 n 的二進位

(3) 0x00 代表接下來的 AC 係數皆為 0。
(4) 0xF0 代表接下來有 16 個 0。
(5) 當 63 的 AC 係數或讀到 0x00 代表所有的 AC 係數已被讀取。

上一步所讀取道的 DC 係數，由於最前面使用 Delta encoding 的關係，其實只是前一個 block 的 DC 係數與當下的 block 的 DC 係數的差別。

5. Inverse Zig-zag

把 Zig-zag 的順序反過來作即可，如下虛擬碼:

```
let ZZ = [
    [ 0,  1,  5,  6, 14, 15, 27, 28 ],
    [ 2,  4,  7, 13, 16, 26, 29, 42 ],
    [ 3,  8, 12, 17, 25, 30, 41, 43 ],
    [ 9, 11, 18, 24, 31, 40, 44, 53 ],
    [ 10, 19, 23, 32, 39, 45, 52, 54 ],
    [ 20, 22, 33, 38, 46, 51, 55, 60 ],
    [ 21, 34, 37, 47, 50, 56, 59, 61 ],
    [ 35, 36, 48, 49, 57, 58, 62, 63 ]
];
let result[8][8];
for i in 0..8 {
    for j in 0..8 {
        let order = ZZ[i][j];
        result[i][j] = block[order / 8][order % 8];
    }
}
```

圖 15:反 Zig-zag 虛擬碼

6. IDCT(Inverse DCT)

進行 DCT 逆轉換，將公式以程式碼的形式表達:

```python
class DCT():
    def __init__(self):
        self.base = np.zeros(64)
        self.zigzag = np.array([
            [0, 1, 5, 6, 14, 15, 27, 28],
            [2, 4, 7, 13, 16, 26, 29, 42],
            [3, 8, 12, 17, 25, 30, 41, 43],
            [9, 11, 18, 24, 31, 40, 44, 53],
            [10, 19, 23, 32, 39, 45, 52, 54],
            [20, 22, 33, 38, 46, 51, 55, 60],
            [21, 34, 37, 47, 50, 56, 59, 61],
            [35, 36, 48, 49, 57, 58, 62, 63],
        ]).flatten()
        # Generate 2D-DCT matrix
        L = 8
        C = np.zeros((L, L))
        for k in range(L):
            for n in range(L):
                C[k, n] = np.sqrt(1/L)*np.cos(np.pi*k*(1/2+n)/L)
                if k != 0:
                    C[k, n] *= np.sqrt(2)
        self.dct = C

    def perform_DCT(self):
        self.base = np.kron(self.dct, self.dct) @ self.base

    def perform_IDCT(self):
        self.base = np.kron(self.dct.transpose(),
                            self.dct.transpose()) @ self.base

    def rearrange_using_zigzag(self):
        newidx = np.ones(64).astype('int8')
        for i in range(64):
            newidx[list(self.zigzag).index(i)] = i
        self.base = self.base[newidx]
```

圖 16:IDCT 程式碼

7. YCbCr to RGB

最後，將 YCbCr 色域轉換回原本的 RGB 色域，公式為原本轉換支逆轉換即可。

```
R = chomp(Y + 1.402*Cr + 128.0);
G = chomp(Y - 0.34414*Cb - 0.71414*Cr + 128.0);
B = chomp(Y + 1.772*Cb + 128.0);
```

圖 17:YCbCr 轉換 RGB

七.結果

將一張 1024 * 1024 的 png 檔案轉換為 jpg 檔案，如下

圖 18:原始的測試檔案(png)

經過轉換後的檔案如下:

圖 19:轉換後的檔案(jpg)

在經過解碼還原成 PNG 檔案:

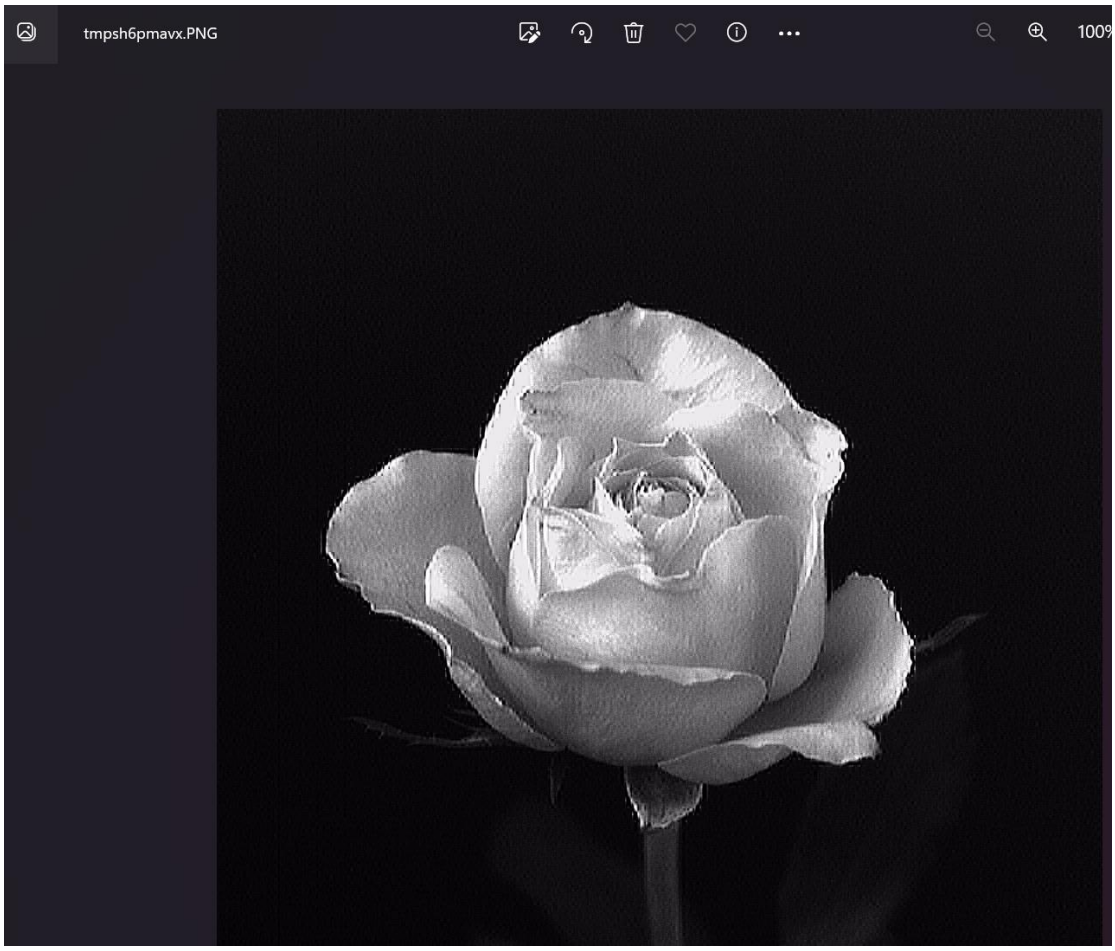

圖 20:解碼後的檔案(png)