常见的图片压缩编码

图片（即静态图像）压缩很重要，举个例子，一张 800X800 大小的普通图片，如果每个像素 32bit 表示（RGBA），那么，存储图片需要的空间大小是 800 X 800 X 4 = 2560000 Byte，大约 2.44M，压缩以后可达到 KB 级，现在互联网上绝大部分图片都使用了 JPEG（Joint Photographic Experts Group）压缩编码技术，使用 JPEG 标准压缩的图片文件，被称为“JPEG 文件”，这种文件的扩展名通常是 JPG、JPEG、JPE、JFIF 以及 JIF，在这些文件格式中，以 JPG 的使用最为广泛。

JPGE 压缩编码

JPEG 压缩比率通常在 10：1 到 40：1 之间，能够获得很高的压缩比是因为使用了有损压缩技术，所谓有损压缩，就是把原始数据中不重要的部分去掉，以便可以用更小的体积保存，这个原理其实很常见，比如 12759.200000000001 这个数，如果我们用 12759.2 来保存，就是一种“有损”的保存方法，因为小数点后面的那个“0.000000000001”属于不重要的部分，所以可以被忽略掉。

图像是由很多个独立的像素点组成的，比如有些图像的尺寸为 640X480，就表示水平有 640 个像素点，垂直有 480 个像素点，可以知道两个相邻的点，会有很多的色彩是很接近的，JPEG 压缩中尽量少记录这些不需要的数据，也即达到了压缩的效果。

另外，图像信号的频谱线一般在 0-6MHz 范围内，而且一幅图像内，包含了各种频率的分量。但包含的大多数为低频频谱，只在占图像区域比例很低的图像边缘的信号中才含有高频的谱线。这个是 JPEG 图像压缩的理论依据。因此具体的方法就是，在对图像做数字处理时，可根据频谱因素分配比特数，对包含信息量大的低频谱区域分配较多的比特数，对包含信息量低的高频谱区域分配较少的比特数，而图像质量并没有可察觉的损伤，达到数据压缩的目的。

JPEG 整个压缩过程基本上也是遵循这个步骤：

- 把数据分为“重要部分”和“不重要部分”
- 滤掉不重要的部分
- 保存

JPEG（全称是 Joint Photographic Experts Group）具有调节图像质量的功能，允许用不同的压缩比例对文件进行压缩，支持多种压缩级别，压缩比越大，品质就越低；相反地，压缩比越小，品质就越好。

JPEG2000 作为 JPEG 的升级版，其对应的文件扩展名为.jp2，其压缩率比 JPEG 高约 30%左右，同时支持有损和无损压缩。JPEG2000 格式有一个极其重要的特征在于它能实现渐进传输，即先传输图像的轮廓，然后逐步传输数据，不断提高图像质量，让图像由朦胧到清晰显示。此外，JPEG2000 还支持所谓的"感兴趣区域" 特性，可以任意指定影像上感兴趣区域的压缩质量，还可以选择指定的部分先解压缩。通常被认为是用来取代 JPEG。它的主要优点是在压缩得较多的情况下，影像质素要比相同压缩程度的 JPEG 好。JPEG2000 其实在 2000 年出现，发展至今天，不少影像处理软件都支援 JPEG2000，但是并没有流行起来，大众用家仍然喜爱使用 JPEG。

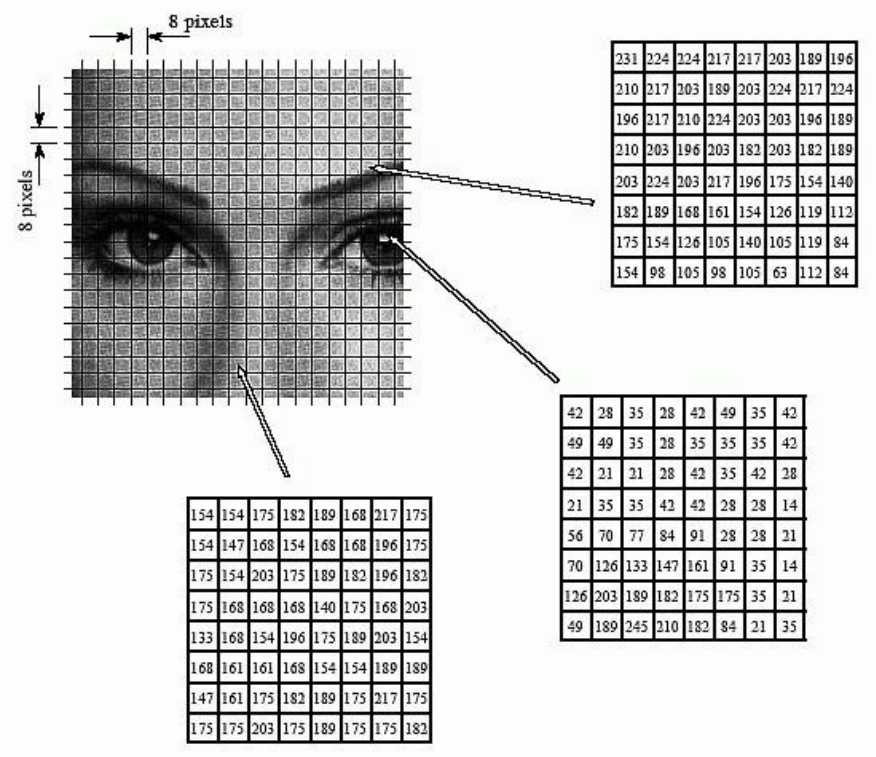
JPEG 算法首先将 RGB 分量转化成亮度分量和色差分量，同时丢失一半的色彩信息（空间分辨率减半）。然后，用 DCT 来进行块变换编码，舍弃高频的系数，并对余下的系数进

行量化以进一步减小数据量。最后，使用 RLE 行程编码和 Huffman 编码来完成压缩任务。

JPEG 具体的压缩编码过程：

(1) 图像分割

将原始图像按照 8*8 像素（pixel，图像是指由一个个的像素小方格组成的，这些小方格都有一个明确的位置和被分配的色彩数值，是整个图像中不可分割的单位或者是元素，像素如果用 RGB 表示，需要 3 个 byte）进行分成一个个的小块, 每个小块里有 64pixels，这些小块在整个压缩过程中都是单独被处理的，相邻的小块像素往往非常相似。



(2)色彩空间转换

大家熟悉的 RGB（红绿蓝）色彩模型用于显示屏的显示，因为人眼对亮度的敏感程度要高于对色彩的敏感程度，比如熄灯时，灯光瞬间由亮变为暗，我们可以在暗光下渐渐地看清周围的事物，而看不清周围事物的颜色，所以 JPEG 图像的颜色时并非采用 RGB 模式，而是 YCbCr 模式，其中，Y 表示的是像素的亮度（亮和暗，即黑—白信号），Cb 和 Cr 表示的是像素的色度和饱和度。在 YCbCr 模型中，Cb 通道和 Cr 通道中所包含的信息量远远少于 Y 通道中包含的信息量，JPEG 在压缩图像时所进行的色彩空间转换就是将 RGB 转换为 YCbCr。

Y = 0.299R+0.587G+0.114B

Cb = -0.1687R-0.3313G+0.5B+128

Cr = 0.5R=0.418G-0.0813B+128

(3)采样

Y、Cb、Cr 信号是分开存储的，Y 信号是黑白信号，是以全分辨率存储的。但是，由于人眼对于彩色信息的敏感度较低，色度信号并不是用全分辨率存储的。色度信号分辨率最高的格式是 4:4:4，也就是说，每 4 点 Y 采样，就有相对应的 4 点 Cb 和 4 点 Cr，即在 2x2 的单元中，分别有 4 个 Y，4 个 Cb，4 个 Cr 值，用 12 个字节进行存储。4:1:1，就是说，每 4 点 Y 采样，就有 1 点 Cb 和 1 点 Cr，即在 2x2 的单元中的值分别有 4 个 Y、1 个 Cb、1 个 Cr，

只要用 6 个字节就可以存储了。JPEG 的压缩算法主要对 Cb 和 Cr 通道中的数据进行缩减取样。

(4)DCT 变换

DCT（Discrete Cosine Transform）是将图像信号在频率域上进行变换，分离出高频和低频信息的处理过程。

分割的每一个小块图像中有低频和高频分量，图像中的高频分量，指的是图像强度（亮度/灰度）变化剧烈的地方，也就是我们常说的边缘（轮廓）；图像中的低频分量，指的是图像强度（亮度/灰度）变换平缓的地方，也就是大片色块的地方。每一个小块中高频分量比较小，相应的高频分量的 DCT 系数经常接近于 0，再加上高频分量中只包含了图像的细微的细节变化信息，而人眼对这种高频成分的失真不太敏感，因此考虑将这一些高频成分予以抛弃，从而降低需要传输的数据量。操作以后，传送 DCT 变换系数的所需要的编码长度要远远小于传送图像像素的编码长度。到达接收端之后通过反离散余弦变换就可以得到原来的数据，虽然这么做存在一定的失真，但人眼是可接受的，而且对这种微小的变换是不敏感的。假设有一个 8×8 的小块，其亮度值

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

由于一个字节的表示范围是 0~255，为了减小绝对值波动，先把数值移位一下，变成-128~127。

$$g = \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix} \begin{matrix} \overset{x}{\longrightarrow} \\ \\ \\ \\ \\ \\ \\ \downarrow y. \end{matrix}$$

接着，根据 DCT 变换公式，得结果。

$$G = \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.12 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.87 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix} \begin{matrix} \overset{u}{\longrightarrow} \\ \\ \\ \\ \\ \\ \\ \downarrow v. \end{matrix}$$

根据亮度量化表量化后得到的量化系数矩阵 。

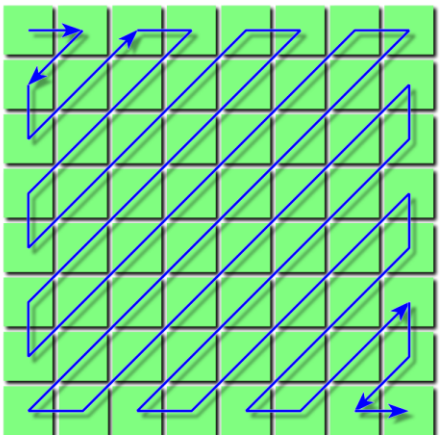
$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121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

量化过程实际上是简单地把频率领域上每个成份，除以一个对于该成份的常数，且接着四舍五入取最接近的整数。获得量化结果：

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

经 DCT 变换后，系数大多数集中矩阵的左上角，即低频分量区，因此采用 Z 字形按频率的高低顺序读出，就是把量化后的二维矩阵转变成一个一维数组，以方便后面的霍夫曼压缩。按这种顺序读出的一维数

{-26,-3,0,-3,-2,-6,2,-4,1,-3,0,1,5,,1,2,-1,1,-1,2,0,0,0,0,0,-1,-1,0,0,0,0,…,0,0}



最后一步对这个数组进行哈夫曼编码，哈弗曼几乎是所有压缩算法的基础，它的基本原理是根据数据中元素的使用频率，调整元素的编码长度，以得到更高的压缩比。举个例子，比如这段数据“AABCBABBCDBBDDBAABDBBDABBBBDDEDBD”，这段数据里面包含了 33 个字符，每种字符出现的次数统计如下：

字符	A	B	C	D	E
次数	6	15	2	9	1

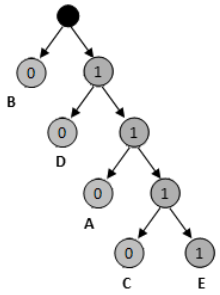
如果我们用我们常见的定长编码，每个字符都是 3 个 bit。

字符	A	B	C	D	E
编码	001	010	011	100	101

那么这段文字共需要 3*33 = 99 个 bit 来保存，但如果我们根据字符出现的概率，使用如下的编码

字符	A	B	C	D	E
编码	110	0	1110	10	1111

那么这段文字共需要 3*6 + 1*15 + 4*2 + 2*9 + 4*1 = 63 个 bit 来保存，压缩比为 63%，哈弗曼编码一般都是使用二叉树来生成的，这样得到的编码符合前缀规则，也就是较短的编码不能够是较长编码的前缀，比如上面这个编码，就是由下面的这颗二叉树生成的。



经过哈弗曼编码，并且序列化后最后得到压缩编码后的 JPEG 数据。解码过程与压缩过程正好相反。

参考文献：

https://blog.csdn.net/newchenxf/article/details/51719597/
http://blog.sina.com.cn/s/blog_61e10f020101hl0a.html
https://www.cnblogs.com/Arvin-JIN/p/9133745.html
https://www.cnblogs.com/tuotuteng/p/4645969.html
https://blog.csdn.net/garrylea/article/details/78536775