JPEG-L : a lightweight JPEG for Mars exploration missions

ISAE-Supaéro Image Processing (ISIP) group

30 novembre 2021

1 Lossy compression : JPEG-L

Unlike lossless compression, lossy compression applies to non-sensitive data for which we do not require the file with the exact information, but a distorted version of the data is sufficient. This is mainly the case for audio data, images and video, for which a human receiver can tolerate compression, sometimes without even noticing the difference. In the following, we focus on image compression, and more specifically on JPEG.

1.1 Principle of lossy compression

The idea behind lossy compression is to approximate the data with a distorted data which requires fewer bits to transmit. One of the most basic and intuitive form of lossy compression, is quantization, which instead of producing a signal with a very high number of levels , for instance the real valued interval $[-3.5 : 3.5]$ which requires a very high number of bits to code each of its levels, produces a signal with few quantization levels, for instance the integer interval $[-3 : 3]$ which requires only 3 bits to code the 7 energy levels.

The quality of the signal will hence depend on the design of the quantizer, and hence, one could control the quality and size of the produced file by controlling the number of quantization levels.

In practice, there exists a huge number of lossy compression algorithms, each tailored to the type of data they are meant for, and no theoretical framework allows a mathematical construction of such schemes, hence we will only concentrate on describing JPEG-L for image compression.

1.2 Basic components of JPEG-L

As most lossy compression algorithms, JPEG-L combines three basic steps as shown in Figure 1

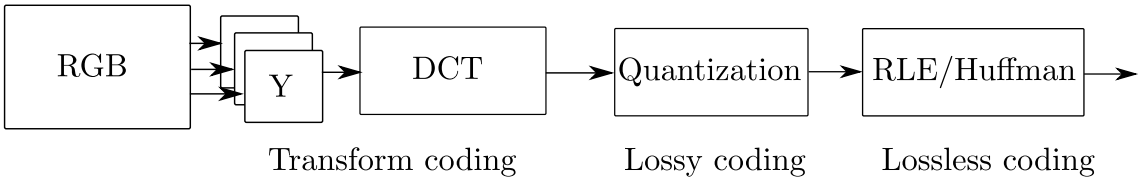

FIGURE 1 – JPEG-L block diagram

1) **Transform coding** which aims, through an invertible transformation, at mapping the data into equivalent data but which present the advantage of being easily compressable. In the case of JPEG-L, transform coding includes a mapping from RGB to YCbCr, and a Discrete Cosing Transform (DCT)

2) A **lossy coding** step which is the only part of the algorithm where information is lost. In the case of JPEG-L, lossy coding will be performed through a rounding operation, i.e., quantization.

3) A **lossless coding** part to compress even more the result. In the case of JPEG-L, two lossless codes are used : Run Length Encoding (RLE) and Huffman coding.

1.2.1 Transform coding of JPEG-L

1) RGB $\rightarrow$ Y Cb Cr. A color image is in fact three parallel matrices each corresponding to a color *plane* R G or B. The first part of transform coding is to transform these three plans through an invertible linear transformation into the so-called Y Cb and Cr planes.

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ Cb &= 128 - 0.1687R - 0.3312G + 0.5B \\ Cr &= 128 + 0.5R - 0.4288G - 0.08B. \end{aligned}$$

The Y plane represents the luminescence of the image (how much energy is carried per pixel), and the Cb and Cr are the chrominance of the blue and red components (basically the shade of the color). It turns out that the human eye is much more sensitive to the luminescence Y than to the chrominances Cr and Cb, hence, one could compress the chrominances more than the luminescence Y. In the following hence, we will treat the three planes separately, but using the same tools. In most cases, it is advisable to use different compression parameters for the three planes, namely, quantization factor (see quantization section), and Huffman coding trees.

2) Discrete Cosine Transform (DCT). Consider a matrix of $NtimesN$ pixels. DCT could be thought of as a 2-dimensional discrete spatial Fourier transform, which maps a matrix $NtimesN$ into an $NtimesN$ matrix where the upper left corner of the DCT represent the low frequencies (energy or DC components) and the lower right part, the high frequencies (AC components). When applying a DCT to a matrix, one seeks to recognize the different spatial frequencies present in the image. In the case of JPEG-L, the DCT is applied by chunks of 8×8 matrices. Hereafter an example of the processing of a chunk of8×8 pixels through the DCT, one can see that the energy level (the DC component) is the main component (= 1186) while all other frequencies are pretty low.

This indicates that this chunk was in fact a group of pixel with similar colors.

$$\begin{bmatrix} 150 & 148 & 148 & 147 & 147 & 148 & 148 & 150 \\ 150 & 148 & 148 & 147 & 146 & 148 & 148 & 150 \\ 148 & 148 & 147 & 147 & 147 & 147 & 147 & 148 \\ 148 & 148 & 147 & 147 & 147 & 147 & 147 & 148 \\ 149 & 148 & 150 & 150 & 150 & 150 & 148 & 148 \\ 148 & 148 & 150 & 150 & 150 & 150 & 148 & 148 \\ 148 & 148 & 150 & 150 & 150 & 148 & 147 & 147 \\ 148 & 148 & 150 & 150 & 150 & 148 & 148 & 147 \end{bmatrix} \Rightarrow \begin{bmatrix} 1186.3 & 1.309 & 0 & 0 & 0.89 & 0 & 0.7 & 0 \\ -2.2 & 0 & 6 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3.05 & 0 & -1 & 0 & 0 & 0 & 1.99 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1.99 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.05 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now one can see clearly that compressing this matrix might be more advantageous than the original matrix due to the fact that there are many zeros in the obtained matrix.

The inverse DCT is defined then as the inverse 2-D spatial Fourier transform.

1.2.2 Quantization of JPEG-L

The values given in the DCT matrix are real values, and thus, in order to be able to run a Huffman code, or a RLE, one needs to map these into integer values, and thus, quantization is needed. Moreover, knowing that the human eye is more sensitive to low frequencies than to high frequencies, then one would want to compress less the low frequencies, and compress more the high frequencies. Hence, a typical compression operation is given by the following transformation

$$Q_{uv} = \text{Round}\left(\frac{T}{Q}\right)$$

where T is the output of the DCT, the division is done point wise between T and the compression matrix Q given by

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

The inverse quantization consist in the following operation

$$\hat{T} = Q_{uv}.Q$$

1.2.3 Separation of DC and AC components

Prior to applying entropy coding or Run Length Encoding (RLE), we will first separate DC and AC components of the 8×8 blocks. The DC components are the upper left corner of the matrix, while the AC components are the remaining 63 coefficients.

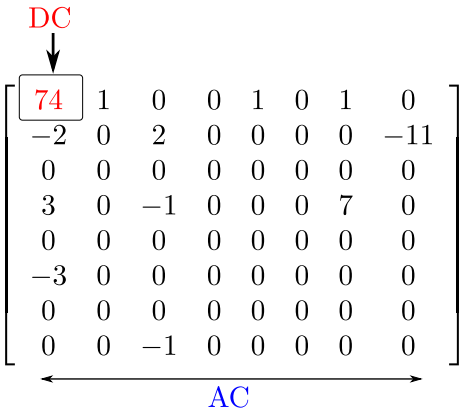

FIGURE 2 – DC and AC components

This separation is performed because DC and AC components have different behaviours and thus, will each have a dedicated processing. DC components are often large, and do not vary that much from two consecutive blocks of 8×8 pixels, whereas AC components are variable even with the same block, and present the property of being quite low as compared to the DC components.

1.2.4 DC components : DPCM then Hufman

DC components will undergo a series of processing techniques as shown in figure 3.

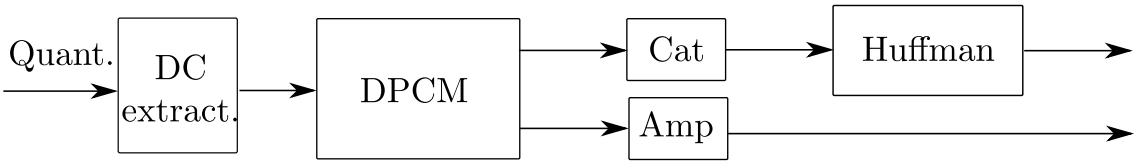

FIGURE 3 – DC processing

1) Assume that the image was composed of 100 blocks of 8×8 blocks of pixels. Then, assume that we collected a vector containing the 100 DC components corresponding to these 100 blocks.

Since DC components do not vary much, one could perform Differential Pulse Modulation Coding on them (DPCM) which consists in coding the differences between two consecutive values, instead of coding each value separately.

Hence if the DC vector is given by

$$DC = [74, 73, 72, 75, 77, 88, ...] \tag{1}$$

then we could use the vector given by

$$DC_{DPCM} = [74, -1, -2, 1, 3, 14, ...] \tag{2}$$

which would require fewer bits to code the result.

2) Throughout JPEG-L, be it for AC or DC components, we will use the so-called *category-amplitude* tables, which allow to code efficiently integer values (positive and negative). We will cut the set of integers into intervals of values having the same lower power of 2, which we will categories. For instance -1 and 1 belong to category 1, $-3, -2, 2$, and

3 belong to category 2, while $-7, -6, -5, -4, 4, 5, 6$, and7 belong to category 3. Note for each category i, there are 2^i distinct integers, coded each over i bits, which we call the amplitude. Table 1 summarizes the mapping.

Values v	Category $c(v)$	Amplitude code $a(v)$	# of bits of $a(v)$
0	0	-	0
-1, 1	1	0,1	1
-3,-2,2,3	2	00,01,10,11	2
-7,-6,-5,-4, 4, 5, 6, 7	3	000,001,010,011,100,101,110,111	3

TABLE 1 – Category amplitude mapping for JPEG-L

We will limit in JPEG-L the max category to 16, knowing that 2^{16} is almost an impossible value to encounter. 3) Huffman coding : Having now obtained the categories and amplitudes of all 100 DC components, and given that amplitudes are already mapped into a binary stream, what remains is to apply a **Huffman code on the categories of the DC components**, which are integers in $[0 : 16]$.

1.2.5 AC components : Zigzag, RLE then Hufman

As for AC components, the processing as shown in Figure 4 is going to be different from DC components, since AC components have a different behaviour.

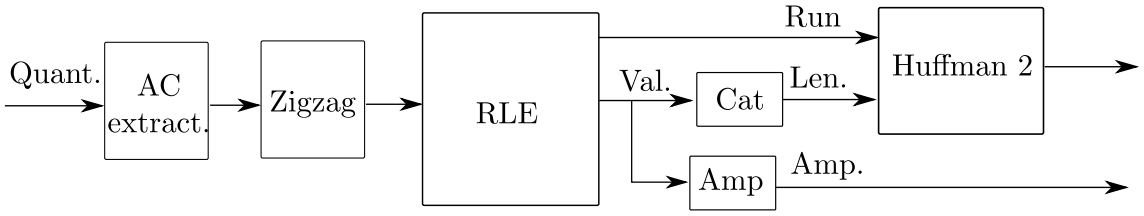

FIGURE 4 – AC processing

1) Zigzag reading : since in compression, we always benefit from consecutive equal values, and since the AC components have the structure shown in Figure 5, then one could read this AC matrix in a Zigzag ordering fashion, and thus, make the result contain a big number of zeros.

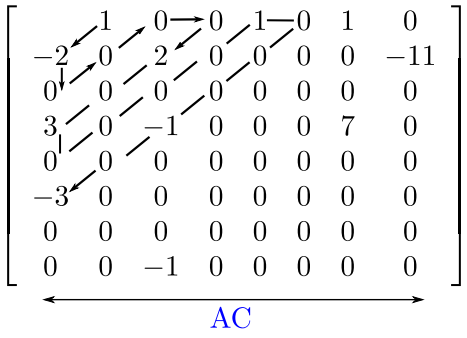

FIGURE 5 – Zigzag reading of AC components

2) Run Length Encoding. Run Length encoding is a coding scheme which is very well tailored to deal with long sequences of zeros. To this end, it transforms and AC stream

given for instance by

$$AC = [1, -2, 0, 0, 0, 2, 0, 3, 0, 0, 0, 0, 1] \tag{3}$$

into a stream of runs and values given (r, v) where r is the number of consecutive 0 before the value v, which would in our case give

$$AC_{RL} = [(0, 1), (0, -2), (3, 2), (1, 3), (4, 1), ...] \tag{4}$$

In JPEG-L, the runs are limited to 15, so if there are longer runs of 0, they are split in runs of 15 consecutive 0.

After RLE is performed, we transform the length values v in (r, v) using their categories and amplitudes $v \to c(v), a(v)$ as given by Table 1. Hence, the output of RLE is of the form :

$$AC_{RL,c} = [(0, 1, 1), (0, 2, 01), (3, 2, 10), (1, 2, 11), (4, 1, 1), ...] \tag{5}$$

The pairs $(r, c(v))$ are termed the run-length pairs of the AC coefficient, while $a(v)$ is termed the amplitude of the AC coefficient.

4) Huffman coding : given that the amplitudes of the length $a(v)$ are already binary streams, we only need to compress using Huffman coding the run-length pairs $(r, c(v))$, using a 2 dimensional Huffman code.