University of Colorado at Boulder ECEN 4532

Instructor: Mike Perkins

JPEG Image Processing

(Joint Photographic Experts Group)

Lab 5
Lab report due on April 15, 2019

This is a Python only lab. Each student must turn in their own lab report and programs.

1 Introduction

In this lab you will implement parts of the JPEG image compression algorithm.

1.1 Why compress images

Analog images, such as camera pictures or chest X-Rays, are usually digitized and compressed to save space. Images exist not only in a two-dimensional (2-D) form, but also as three-dimensional (3-D) entities (e.g. medical MRI images), or as spatio-temporal sequences (videos). The availability of images in digital form offers many advantages:

- Digital images can be stored and transmitted very efficiently
- They can be further processed. Examples of post processing include: enhancement, de-noising, zooming, etc.
- Computer assisted diagnosis can be performed on digital images. Such higher level processing can involve feature extraction, decision making, classification, etc.
- Images can be stored in large data bases, where they can be indexed and efficiently retrieved.

However the very large size of uncompressed images and videos are basic hurdles to any attempt at storing, transmitting, or searching for them in a database. Table 1 shows representative sizes of image and video files. For each application, we calculated the compression ratio required for a realistic usage under normal conditions. It is clear from this table that the ability to decrease the amount of bits necessary to represent an image, or a sequence of images, will have a dramatic impact in a wide variety of domains, including but not limited to: multimedia technology, medical imaging, geophysical exploration, satellite imagery, etc.

Applications	horiz x vert	frame/s	size	ratio
Digital camera	768 x 512		375kB	10
Fax	1728 x 1100		240 kB	20
CD	352 x 240	30	7.6 Mb/s	50
HDTV	1920 x 1080	30	186 Mb/s	80
VideoPhone	176 x 144	15	1.1 Mb/s	300

Table 1: Typical size of images or sequence of images, and the compression ratio required for a realistic usage under normal conditions.

2 Structure of a compression system

A compression system can be decomposed into three components: (1) an encoder, (2) a decoder, and (3) and a channel. The channel is often out of the compression algorithm developer's control. Examples of channels include the following: wireless, satellite, ethernet, CD, modem, optic fiber, ISDN, computer memory, etc.

In this lab we will assume that the channel reliably communicates the compressed data, and we will ignore the possibility of channel coding errors. It is possible to combat channel errors, by using error protection codes, but we will not explore that here.

2.1 Encoder

A transform-based encoding algorithm, such as JPEG, can be decomposed into a series of stages:

1. Pre-processing. A number of operations can be performed on the image prior to compression. The goal of pre-processing is to prepare the image in order to minimize the coding complexity, and to increase the compression efficiency. Examples of pre-possessing operations are:
    - filtering
    - padding with zeros

- symmetric extension on the boundaries
- tiling (cutting into smaller blocks) to enable the coding of images of arbitrarily large size

2. Transform. The goal of the transform is to reduce the correlation among image pixels, and describe most of the image content with a smaller set of significant coefficients. A good transform "packs" most of the energy into a few coefficients.
3. Quantization. The output of the transform are real-valued coefficients. In order to decrease the amount of information needed to describe these coefficients, they are represented with a much smaller set of values. Quantization is a non-reversible distortion-introducing process that permits a compression algorithm to balance the reconstructed image quality against the number of bits required to code the image.
4. Entropy coding. Entropy coding of the output of a quantizer permits a description of the outputs in an efficient way using the bits "0" and "1". The basic concept is that more common symbols are assigned shorter code words, while less common symbols are assigned longer ones.

2.2 Decoder

The decoder reconstructs the image by inverting each of the encoder processes 4), 3), and 2). While entropy coding does not introduce distortion, quantization does. A final post-processing is often performed on the reconstructed image in order to conceal coding artifacts and improve the visual appearance.

2.3 Important parameters of a compression system

Some of the fundamental parameters of a compression system are:

- Compression efficiency: it measures the compression ratio (often quoted in bits per pixel: bpp),
- Fidelity: this is the distortion due to the coding. While everybody agrees that the PSNR (Peak Signal to Noise Ratio) is a global parameter, often but not always correlated with perceived visual quality, there is currently no universally

accepted definition of a better quantitative measure of visual distortion. For this reason visual inspection of reconstructed images remains one criterion for the evaluation of a compression method.

- Complexity: this is a key parameter for a number of applications where real-time compression is required.
- Memory. Some applications will require an encoder that can work with a limited memory.
- Coding delay: real-time applications such as video-phone cannot tolerate a coding delay that is longer than a couple of frames.
- Robustness. This is a key feature in areas such as wireless communication where channel coding errors can have a catastrophic effect on the reconstructed image.

3 Transform coding

Transform coding involves applying a linear transformation from the image space to a transformed space. The transformed values are referred to as coefficients. When an appropriate transform is used, the transform coefficients will be less correlated than the original samples. Another way to express the same idea is that a small number of transformed coefficients will carry most of the energy of the image, and the rest of the coefficients can be quantized to zero. Most transforms used for compression operate in the frequency domain (e.g. the Fourier Transform). Fast algorithms exist for these transform types (often based on the FFT). Furthermore, for images, a frequency domain transform can be shown to be nearly as optimal as the theoretical best transform.

From a practical point of view, some aspects of the human visual system are well understand in the frequency domain, so it is easier to work directly in this domain. Known facts about human visual system sensitivity to different spatial frequencies can be incorporated.

The theoretically optimal transform is known as the Karhunen-Loève transform (KLT), and is a function of the image statistics. Assuming the images are realizations of a wide sense stationary process, then the KLT diagonalizes the correlation matrix. The KLT is the transform that packs the maximum amount of energy into the fewest number of transform coefficients.

While the KLT transform depends on knowledge of the correlation matrix of the input, and is therefore rarely used in practice, this result provides a theoretical justification for the performance of transform coding. For certain inputs the Discrete Cosine Transform provides an approximation to the data-dependent KLT.

We note that researchers have also looked into wavelet-based image compression algorithms. The wavelet transform is an orthonormal transform that provides a very efficient decorrelation of images, and can under certain circumstances provide an approximation to the KLT.

3.1 Discrete Cosine based coders

There exists four different versions of the discrete cosine transform {Rao90}. The DCT is important for several different reasons. First, the DCT provides a good approximation to the KLT if the image has a Markov behavior. In practice, the DCT provides excellent energy compaction for highly correlated data. From a practical point of view, the DCT can be computed using an FFT, but unlike the FFT, the DCT coefficients are real. In 1988, a DCT based algorithm won in blind subjective assessments for the definition of a new image compression standard for continuous-tone images (Joint Photographic Experts Group - JPEG).

4 Reading, writing, and displaying images

4.1 Test images

In this lab, we will be using standard test images available in the appropriate Canvas module.

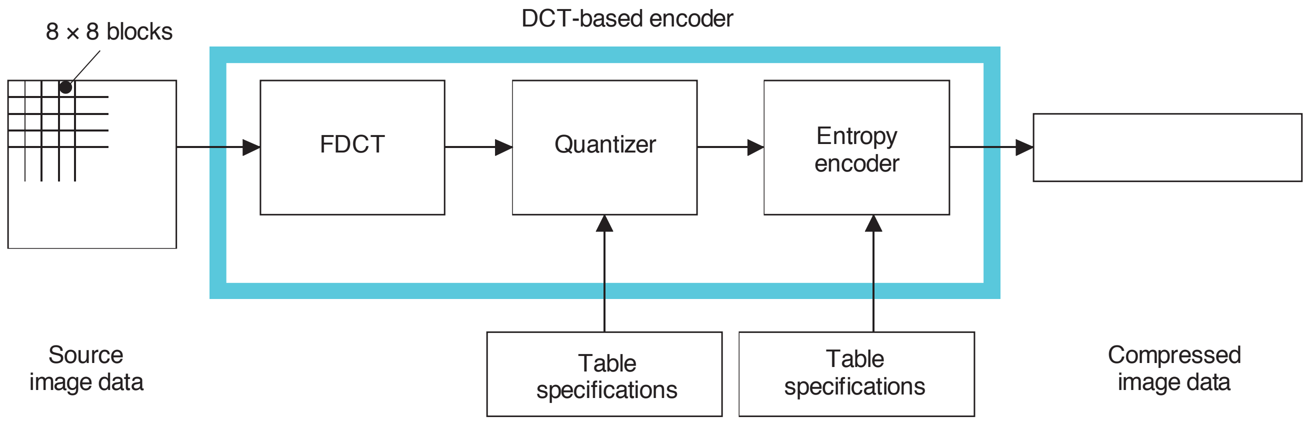


Figure 1: The different stages in the JPEG compression algorithm

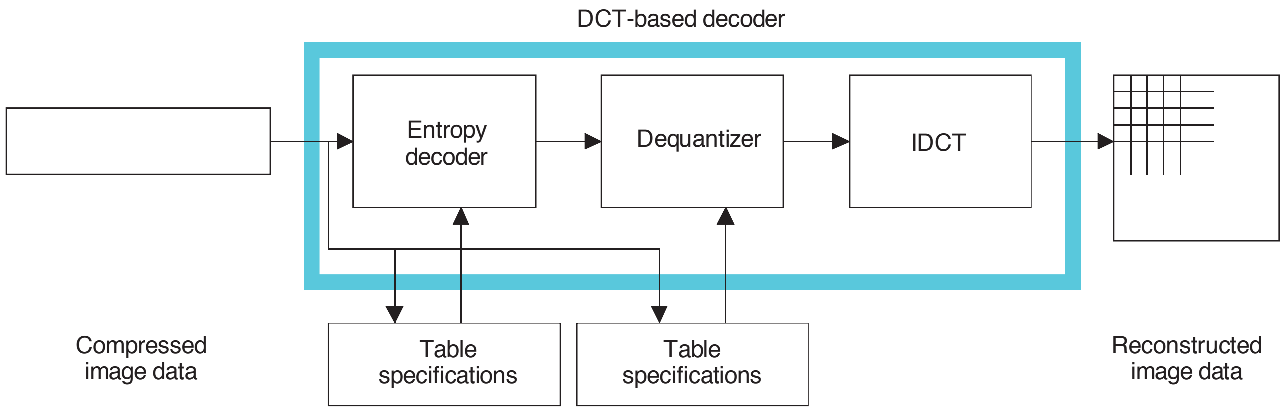


Figure 2: The different stages in the JPEG decoding algorithm

5 The JPEG compression algorithm

Figures 1 and 2 show the three main steps of the JPEG compression/de-compression algorithm. We will implement two stages: forward and inverse DCT, and forward and inverse quantization. Each decoder step implements the inverse of the corresponding operation performed by the encoder.

5.1 The forward and inverse discrete cosine transform.

The original image is divided into non overlapping blocks of size 8×8. The blocks are scanned in left-to-right and top-to-bottom order. Each block is transformed using a two-dimensional discrete cosine transform. For blocks of size 8×8 this transform is given by:

$$F_{u,v} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}, \quad u, v = 0, \dots 7 \tag{1}$$

where C_v follows the same rule as C_u and

$$C_u = \begin{cases} 1 & \text{if } u \neq 0, \\ \frac{1}{\sqrt{2}} & \text{if } u = 0. \end{cases} \tag{2}$$

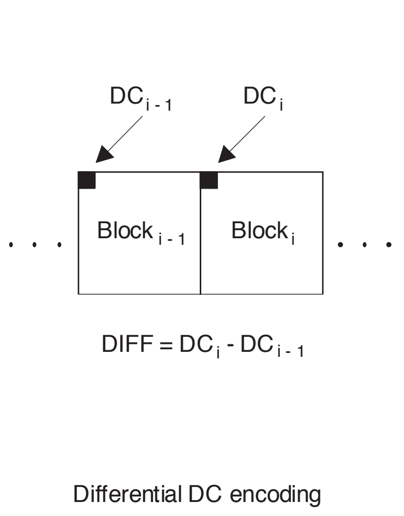
The coefficients $F_{u,v}$ are coarsely approximated over a small set of possible values (quantization), and replaced by $\widetilde{F}_{u,v}$. The *quantization* process introduces distortion and therefore contributes to the loss in image quality.

Given the coefficients $\widetilde{F}_{u,v}$, one can use the inverse discrete cosine transform to reconstruct an estimate of the block using

$$\widetilde{f}(x,y) = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u \; C_v \; \widetilde{F}_{u,v} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}, \quad x, y = 0, \dots 7 \tag{3}$$

where C_v follows the same rule as C_u and

$$C_u = \begin{cases} 1 & \text{if } u \neq 0, \\ \frac{1}{\sqrt{2}} & \text{if } u = 0. \end{cases} \tag{4}$$



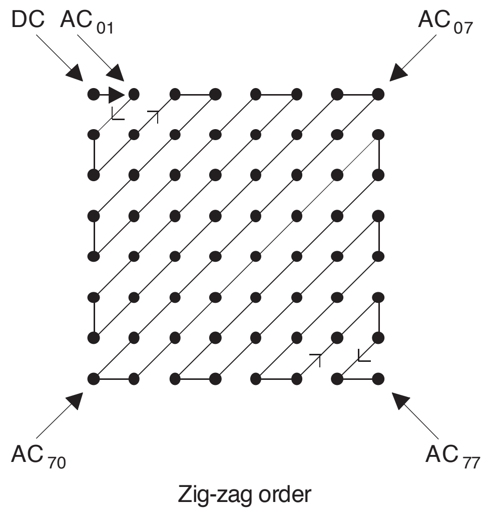


Figure 3: Zig-zag order for mapping the two-dimensional frequencies to a one dimensional array.

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

Figure 4: The quantization table Q

Assignment

1. Write a Python function `dctmgr` that implements the following functionality:
   (a) Its input is a luminance (gray-level) image, it divides it into non overlapping 8×8 blocks, and DCT transforms each block according to (1). You should write a forward 2-D DCT routine using 1-D transforms first applied to the rows and then to the columns as discussed in class (use the 1-D routine from `scipy.fftpack`).
   (b) The DCT coefficients for the entire image are returned in a matrix `coeff` of size $64 \times N_{\text{blocks}}$, where N_{blocks} is the number of 8×8 blocks inside the image.
   (c) For a given block `b`, the coefficients of the column `coeff[:,b]` are organized according to the zig-zag order shown in Fig. 3. That is,

       `coeff[0,b] = F(0, 0), coeff[1, b] = F(0,1), coeff[2,b] = F(1,0),...`
   (d) The zero-frequency "DC" coefficients, $F(0,0)$, for each block are then replaced by their differentially encoded values (i.e each DC coefficient is predicted by either 0 if it is the start of a new row, or by the DC coefficient of the block to its left). As an example:
       `coeff[0, 0]` $= F_0(0,0) - 0$,
       `coeff[0, 1]` $= F_1(0,0) - F_0(0,0)$,
       `coeff[0, 2]` $= F_2(0,0) - F_1(0,0)$,
       ...
       where $F_i(0,0)$ is the zero frequency DCT coefficient of block i prior to differential encoding. Image blocks should be processed in a left-to-right and top-to-bottom order. When starting each new row of 8-by-8 blocks, the predictor of the DC coefficient is reset to 0 (in other words, don't use the last block of one row to predict the first block of the following row).
2. Write the Python function `idctmgr` that implements the following functionality:
   (a) Its input is a matrix `coeff` of size $64 \times N_{\text{blocks}}$, where N_{blocks} is the number of 8×8 blocks inside the image.
   (b) For each block `b`, the coefficients in the column `coeff[:,b]` are used to reconstruct the block according to (3). You should write an inverse 2-D DCT routine using `scipy.fftpack` as above. Keep in mind that you will have to undo the prediction used for the DC coefficients to recover their true values.
   (c) The output of the function is a luminance image

5.2 Quantization and inverse quantization

The effect of the DCT is to create many small coefficients that are close to zero, and a small number of large coefficients. We can think of the DCT as a rotation of the space $\mathbb{R}^{64}$ that aligns the coordinate system along the directions associated with the largest variance in the distribution of the coefficients. In order to benefit from this redistribution of the energy, we need to simplify the representation of the floating numbers that are used to describe the DCT coefficients. This task is achieved by the quantization step.

The goal of quantization is to represent a continuum of values with a finite (and preferably) small set of symbols. In theory, a scalar quantizer can be defined as a map from the set of real numbers to the set of integers, or any subset thereof. In practice, real numbers are represented in a computer as floating point numbers and have only a fixed number of digits, and thus a fixed precision. In this context a scalar quantizer is a map from a large set of discrete values, to a much smaller set of codewords.

Quantization permits us to control the performance of the compression by performing a reduction of the set of possible values that can be used to represent the initial samples. In principle quantization is the only mechanism in the JPEG compression system where irrecoverable loss is introduced.

In the JPEG compression standard, the quantization of the DCT coefficients *other than the DC coefficient* is performed as follows

$$F^q_{u,v} = \operatorname{floor}\left(\frac{F_{u,v}}{\texttt{loss-factor} \times Q_{u,v}} + 0.5\right). \tag{5}$$

$Q_{u,v}$ is the entry (u, v) of the standard quantization table shown in Fig. 4, and the scaling factor `loss-factor` is a user supplied parameter that measures the quality of the compressed image. In our specific implementation approach, the fact that the DC coefficient is not quantized means that the DC coefficient prediction residuals stored in the first row of the $64 \times N_{blocks}$ coefficient matrix are *not* quantized.

As an example of the impact of the `loss-factor`, the choice of `loss-factor` $= 2$ should yield an image that is visually identical to the original image, while `loss-factor = 10` produces an image of lower quality.

The inverse quantization of the DCT coefficients is defined by the map

$$\widetilde{F}_{u,v} = F^q_{u,v} \times \texttt{loss-factor} \times Q_{u,v}. \tag{6}$$

We note that the compression becomes lossy during the quantization step performed in (5) that creates many zero coefficients as `loss-factor` becomes larger. The quantization step $Q_{u,v}$ varies as a function of the frequency. As a result, it becomes possible to spend fewer bits for the high frequency coefficients than for the low frequency coefficients. Several quantization tables have been proposed. In this lab we use the table Q defined in Fig. 4.

Assignment

3. Implement the quantizer and inverse quantizer, as defined by (5) and (6), and integrate it with the DCT and IDCT transforms.
4. For the three test images, display reconstructed images for `loss-factor = 1`, `loss-factor = 10`, and `loss-factor = 20`.
5. For the three test images, compute the Peak Signal to Noise Ratio (PSNR) between the original image and the reconstructed image,

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{255^2}{\frac{1}{N^2}\sum_{i,j=0}^{N-1}|f(i,j)-\widetilde{f}(i,j)|^2}\right), \tag{7}$$

   for the above 3 values of `loss-factor`. Use the program `img_psnr.py` in the lab module on Canvas.

5.3 Variable length and runlength coding

After zig-zag ordering and quantization, we expect the sequence of quantized coefficients within each block to contain mostly zeros, except maybe for the first few coefficients. If the original image is encoded using 8 bits per pixel, then the intensity of each pixel is in the range $[0, 2^8-1]$, and it is possible to compute the maximum and minimum value that any DCT coefficient or DC prediction residual can assume.

When a quantized coefficient is a zero (and this should happen often as the `loss-factor` is increased), the coefficient is not coded. Instead, we count the number of zeros separating this non-zero coefficient from the previous non-zero coefficient and efficiently communicate the length of this "run of zeros" to the decoder. Conceptually, for each non-zero coefficient, we precede it with a codeword that encodes the number of zeros between this coefficient and the previous non-zero coefficient.

Finally, JPEG uses an End Of Bock (EOB) symbol to indicate that all remaining coefficients in the current block are zero.

To create the input to an entropy encoder, we will process each picture's coefficients into triplets of numbers (with EOB sequences inserted between blocks). The triplets are as follows:

$$\left[\texttt{nZeros}\ \ \texttt{nBits}\ \ \texttt{value}\right] \tag{8}$$

where

1. `nZeros` is the number of zeros skipped since the last non zero coefficient. We have (in a slight departure from the true standard):

   $$\texttt{nZeros} \in [0, 63] \tag{9}$$

   Of course, `nzeros` is 0 for the DC coefficient prediction residuals, as this is the first symbol in a block.

2. `nBits` is the number of bits needed to represent the value of a coefficient using two's complement:

   $$\texttt{value} \in [-2^{\texttt{nBits-1}}, 2^{\texttt{nBits -1}} - 1]. \tag{10}$$

   Because the original image $f(x, y)$ is represented using 8 bits per pixel, we need `nBits` $= 11$ to represent `value` for the AC coefficients and `nBits` $= 12$ to represent the DC prediction residuals.

3. `value` is the actual value of the coefficient and lies in the range

   $$\texttt{value} \in [-2^{\texttt{nBits -1}}, 2^{\texttt{nBits -1}} - 1] \tag{11}$$

   where `nBits` =12 for the DC coefficient, and `nBits` =11 otherwise.

4. When there are no more non-zero coefficients remaining to be encoded in a block, then we insert a special EOB sequence of three zeros. You can think of this as `nZeros` $= 0$, `nBits = 0`, and `value = 0`. The triplets for the next block immediately follow the EOB sequence of the preceding block.

Assignment

6. Implement the triplet pre-processor and its corresponding inverse-processor. The input parameter to the pre-processor should be the $64 \times N_{blocks}$ array. The output should be a 2D matrix of signed 16 bit integers, `symb`. The matrix `symb` will have 3 columns. You should store sequentially in each row of the matrix the values `nZeros, nBits,` and `value` for each non-zero coefficient (including the DC coefficient prediction residual). A row of 3 zeroes should be inserted at the end of each block to indicate to the post-processor where block boundaries lie. All columns of your coefficient array are processed into a single large 2D matrix of triplets. The number of rows in `symb` will be variable and depend on how the `loss-factor` is chosen.
   (a) Name your pre-processor function `run_bits_value`. You will pass to this function your coefficient array, and it will return a single 2D matrix `symb` representing the entire picture as described above.
   (b) Name your inverse-processor function `irun_bits_value`. You will pass to this function `symb`, and it will return a coefficient array.

5.4 Entropy coding

The aim of entropy coding is to provide a lossless representation of a set of symbols with a description whose average length is minimal. This can be achieved by using a variable length coding mechanism: the most frequent outcomes of the data are encoded with the shortest descriptions, while longer descriptions are used for less frequent outcomes. (Intuitively, for example, if you are designing a code for the English alphabet, you should assign a short code word to the letter 'E' and a long codeword to the letter 'Q'.)

Two entropy coding algorithms are supported in the JPEG standard: Huffman coding, and arithmetic coding. Arithmetic encoding is more efficient, because it can adapt itself to the statistics of the quantized coefficients. In this lab we will not implement either approach. However, we will estimate the entropy of each triplet value: `nZeros, nBits, value`. In other words, we will estimate the entropy of each column of the matrix `symb`. We can then imagine designing Huffman codes for each triplet value, and using those codes to complete our picture encoding.

The entropy of a random variable, X, that assumes the values $\{x_1, x_2, ..., x_N\}$ with probabilities $\{p_1, p_2, ..., p_n\}$ is given by:

$$H(X) = \sum_{n=1}^{N} p_n \log_2 \left(\frac{1}{p_n}\right) \tag{12}$$

Think of rolling a dice with N sides, and X as describing the outcome. We assume that X will be rolled over and over. The probabilities, p_n, can be estimated by computing their relative frequencies (the dice might be unfair). For convenience, we define $0 \log_2(1/0) = 0$.

The entropy gives a value in bits. It can be shown that a lossless code can be designed that allows a coder to use $H(X)$ bits to encode each dice roll, and that no lossless code exists that uses fewer than $H(X)$ bits per roll.

Assignment

7. You have already written a program to output a list of triplet values for a picture. Now, write a program to estimate the entropy separately for each component of a triplet value: `nZeros, nBits, value` (include the EOB triplet in your estimate). Add the three entropies together to estimate the number of bits required to code a triplet. Multiply this estimate by the number of rows in the matrix `symb` to estimate the total number of bits required to code the picture. You will now evaluate the compression ratio of the coder. For the three test images, for every value of the parameter `loss-factor = {1,..50}`, determine the number of bits required by our hypothetical entropy coder to represent the picture, and use this to compute a compression ratio:

$$\text{compression\_ratio} = \frac{8 \times \text{number\_of\_rows} \times \text{number\_of\_columns}}{\text{total\_number\_of\_bits}} \tag{13}$$

   Plot a curve that gives the compression ratio as a function of the `loss-factor`.

A Appendix: technical supplements

The following sections provide brief theoretical descriptions of the three main components of the JPEG algorithm.

A.1 Discrete Cosine Transforms

One can construct four different cosine and sine transforms, that are all based on a Fourier transform. As opposed to the discrete Fourier transform that assumes a periodic signal, the cosine/sine transform does not require a periodic signal. The principle of the different constructions is to extend the input signal in a periodic manner, using odd or even extensions. Let us examine the construction of the DCT II for instance. We consider a series of samples $x_n, n = 0, \dots N-1$. We extend the samples for $n = -N, \dots -1$ by mirror symmetry across the half integer $-1/2$:

$$x_n = x_{-n-1} \quad , \quad n = -N, \dots, -1 \tag{14}$$

Fig 5 demonstrates the extension. We obtain then a periodic sequence of size $2N$, and we can calculate its discrete Fourier transform: $\hat{x_k} \quad , \quad k = -N, \dots N-1$. Note that because the extended sequence is symmetric around -1/2, we know that the Fourier coefficients will be real (up to a phase shift, that would not exist if we were to calculate the symmetry with respect to 0), and even. We need only to compute the coefficients for $k = 0, \dots, N-1$.

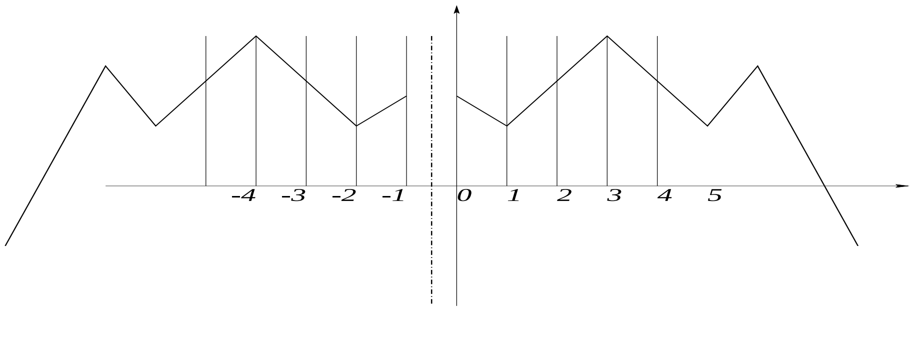


Figure 5: The original signal is extended with an even parity

$$\hat{x}_k = \sum_{n=-N}^{N} x_n e^{-\frac{2\pi\, i\, kn}{2N}} \tag{15}$$

We cut the sum into two parts:

$$\hat{x}_k = \sum_{n=-N}^{-1} x_n e^{-\frac{2\pi\, i\, kn}{2N}} + \sum_{n=0}^{N} x_n e^{-\frac{2\pi\, i\, kn}{2N}} \tag{16}$$

We can rewrite the first sum, using the mirror symmetry:

$$\sum_{n=-N}^{-1} x_n e^{-\frac{2\pi\, i\, kn}{2N}} = \sum_{n=1}^{N} x_{-n} e^{\frac{2\pi\, i\, kn}{2N}} = \sum_{n=0}^{N-1} x_n e^{\frac{2\pi\, i\, k(n+1)}{2N}} = e^{\frac{\pi\, i\, k}{2N}} \sum_{n=0}^{N-1} x_n e^{\frac{2\pi\, i\, k(n+1/2)}{2N}} \tag{17}$$

Now we can rewrite the second sum as follows:

$$\sum_{n=0}^{N} x_n e^{-\frac{2\pi\, i\, kn}{2N}} = e^{\frac{\pi\, i\, k}{2N}} \sum_{n=0}^{N} x_n e^{-\frac{2\pi\, i\, k(n+1/2)}{2N}} \tag{18}$$

And if we put the two pieces back together, we get:

$$\begin{aligned} \hat{x}_k &= e^{\frac{\pi\, i\, k}{2N}} \sum_{n=0}^{N-1} x_n e^{\frac{2\pi\, i\, k(n+1/2)}{2N}} + e^{\frac{\pi\, i\, k}{2N}} \sum_{n=0}^{N} x_n e^{-\frac{2\pi\, i\, k(n+1/2)}{2N}} \\ &= e^{\frac{\pi\, i\, k}{2N}} \sum_{n=0}^{N-1} x_n \cos\frac{\pi\ k(n+1/2)}{N} \end{aligned} \tag{19}$$

We then define the discrete cosine transform II of a vector x_n, $n = 0, \ldots,$N-1 as follows :

$$y_k = \sum_{n=0}^{N-1} x_n \cos\frac{\pi\ k(n+1/2)}{N} \qquad k = 0, \ldots, N-1 \tag{20}$$

Obviously we could have extended the initial signal using different polarity. Another example is provided in Fig 6 where the original signal is extended with an even parity on the right end side, and then with odd parity on the left end side.

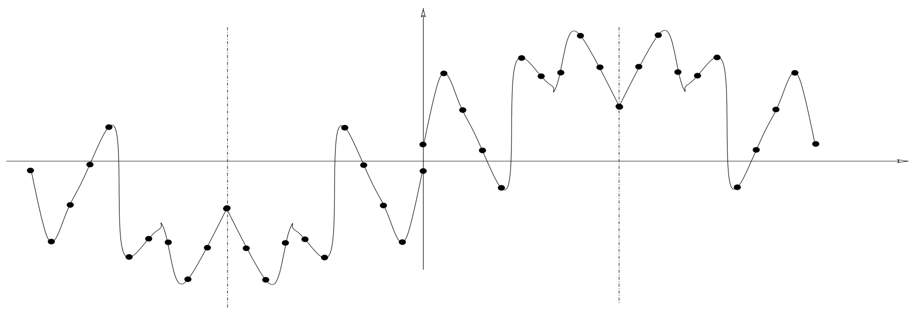


Figure 6: The original signal is extended with an even parity on the right end side, and then with odd parity on the left end side.

Definition 1 *One defines four discrete cosine transforms* $\mathbf{C}^I$,$\mathbf{C}^{II}$,$\mathbf{C}^{III}$,$\mathbf{C}^{IV}$. *The coefficients of these matrices are given given by :*

- *DCT-I*

$$\begin{aligned} &\mathbb{R}^{N+1} \to \mathbb{R}^{N+1} \\ C^I_{N+1}(k,n) =& b(k)b(n)\sqrt{\frac{2}{N}} \cos\frac{\pi kn}{N} \end{aligned} \tag{21}$$

- *DCT-II*

$$\begin{aligned} &\mathbb{R}^{N} \to \mathbb{R}^{N} \\ C^{II}_{N}(k,n) =& b(k)\sqrt{\frac{2}{N}} \cos\frac{\pi k(n+1/2)}{N} \end{aligned} \tag{22}$$

- *DCT-III*

$$\begin{aligned} &\mathbb{R}^{N} \to \mathbb{R}^{N} \\ C^{III}_{N}(k,n) =& b(n)\sqrt{\frac{2}{N}} \cos\frac{\pi (k+1/2)n}{N} \end{aligned} \tag{23}$$

- *DCT-IV*

$$\begin{aligned} &\mathbb{R}^{N} \to \mathbb{R}^{N} \\ C^{IV}_{N}(k,n) =& \sqrt{\frac{2}{N}} \cos\frac{\pi (k+1/2)(n+1/2)}{N} \end{aligned} \tag{24}$$

where the normalization constant $b(n)$ *is defined by*

$$b(n) = \begin{cases} 0 & \textit{if} \quad n < 0 \quad \textit{or} \quad n > N \\ 1/\sqrt{2} & \textit{if} \quad n = 0 \quad \textit{or} \quad n = N \\ 1 & \textit{if} \quad 1 \le n \le N-1 \end{cases} \tag{25}$$

In this definition, k plays the rôle of the frequency index (row index in the matrix), whereas n plays the rôle of the time index (column index in the matrix).

The construction just described can be performed in the continuous domain. Instead of working with discrete samples, we calculate Fourier series of properly extended functions, and expand them using the family $\{e^{i\,\pi\,kx}\ ,\ k = 0, 1, \dots\}$. We obtain the following four orthonormal bases.

Theorem 1 *Each of the four sets constitutes an orthonormal basis of* $L^2[0,1]$*:*

$$\begin{gathered}
\left\{\sqrt{2}\,\sin(k+1/2)\pi x\ ,\ k=0,1,2,\ldots\right\}\\
\left\{\sqrt{2}\,\sin k\pi x\ ,\ k=1,2,\ldots\right\}\\
\left\{\sqrt{2}\,\cos(k+1/2)\pi x\ ,\ k=0,1,2,\ldots\right\}\\
\left\{1,\sqrt{2}\,\cos k\pi x\ ,\ k=1,2,\ldots\right\}
\end{gathered} \tag{26}$$

Figure 7 displays the first three basis functions of the last set.

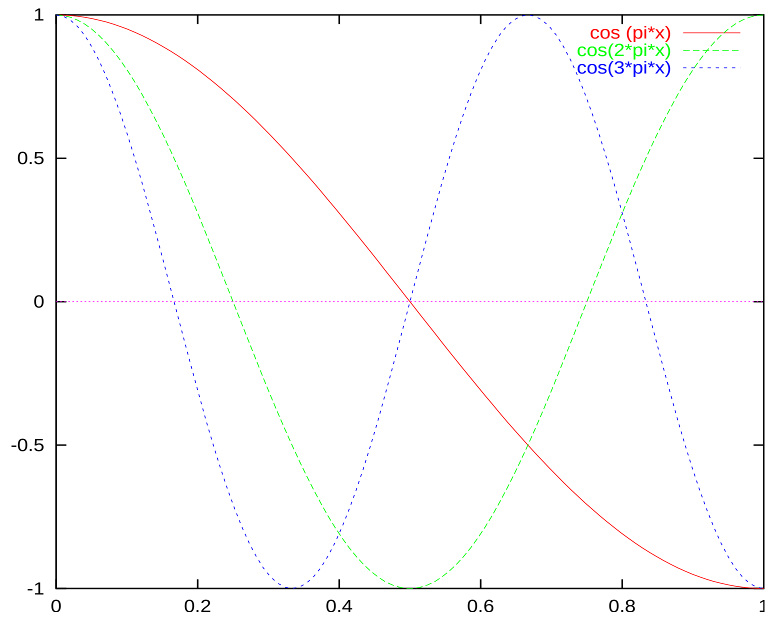


Figure 7: First three basis functions of the DCT-II

We can sample the four sets using integers or half integers, and we obtain four different discrete cosine transforms, and four discrete sine transforms.

A.2 Fast computation of the discrete cosine transform

There exists a fast algorithm to calculate DCT transforms. The algorithm is based on a formula that expresses the DCT as an FFT of half length, up to a phase shift.

For instance, the DCT-IV coefficients, $\hat{x}(j), \quad j = 0, \ldots, N-1$ of the sequence $x(n), \quad n = 0, \ldots, N-1$ are given by:

$$\begin{aligned}\hat{x}(j) =& \mathrm{Re}\left(e^{-\dfrac{ij\pi}{2N}} \sum_{n=0}^{N/2-1} y(n) e^{-\dfrac{2i\pi jn}{N}} \right) \\ \hat{x}(N-j-1) =& -\mathrm{Im}\left(e^{-\dfrac{ij\pi}{2N}} \sum_{n=0}^{N/2-1} y(n) e^{-\dfrac{2i\pi jn}{N}} \right)\end{aligned} \tag{27}$$

with

$$y(n) = (x(2n) + i\, x(N-2n-1))\, e^{-\dfrac{i(n+1/4)\pi}{N}} \tag{28}$$

B Quantization

For a given budget (measured in number of bits, or compression ratio), one wishes to design the optimal quantizer in order to obtain the best reconstructed image. A very large body of work has been expended on the problem of the optimal design of quantizer {Gersho92}.

B.1 Rate distortion

Consider the following problem: let X be a random variable that is being approximated by $Q(X)$. $\hat{X}$ takes only a finite number $N = 2^R$ of values, and thus can be represented using R bits. The question is then this: what is the optimal set of values that Q should take in order to minimize the error between X and $Q(X)$? For instance, let X be Gaussian distributed, $X \sim N(0, \sigma^2)$, and let us assume that $R = 1$. If one has only one bit to approximate X, and if one measures the error of quantization using the expected squared error, $E(X - Q(X))^2$, then $Q(X)$ should be defined as follows :

$$Q(x) = \begin{cases} \sqrt{\frac{2}{\pi}}, & \text{if} \quad x \geq 0 \\ -\sqrt{\frac{2}{\pi}}, & \text{if} \quad x < 0 \end{cases} \tag{29}$$

Indeed, X is approximated by the mean of the Gaussian distribution over the positive or negative values. If one has two bits, or 4 values to describe X, then the problem becomes more complicated.

B.2 Scalar quantization

A scalar quantizer will process each sample individually. Let X be a random variable that takes its values in an interval $[a, b]$. Let $f(x)$ be the probability density function of X. Let $Q(X)$ be the reconstructed value after scalar quantization. We measure the distortion with the mean squared error :

$$D = E[\|X - Q(X)\|^2] = \int (x - Q(x))^2 dx \tag{30}$$

A scalar quantizer is a piece-wise constant function. The regions where the functions is constant are called the bins or quantizer cells (see Fig 8). One divides $[a, b]$ into N quantizers bins, that need not have the same size. We have :

$$Q(x) = Q_j \quad \text{if} \quad x \in [x_j, x_{j+1}) \tag{31}$$

Let $\Delta_j = x_{j+1} - x_j$ be the size of the cell j.

B.2.1 Uniform scalar quantization

The simplest form is a uniform scalar quantizer, where all bins have the same size. Figure 8 shows an example of a uniform scalar quantizer, where the central bin, called the dead zone (where all values are quantized to zero) has a size twice as large as the size of the other bins.

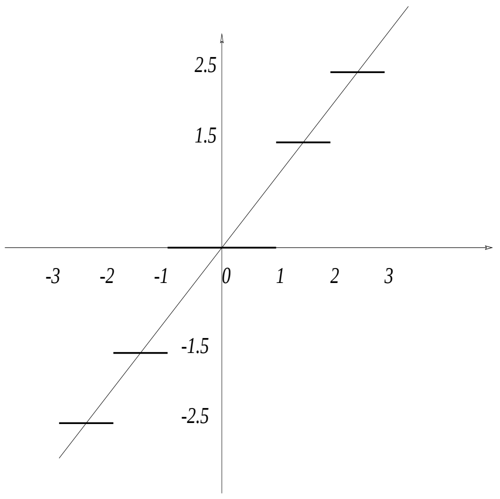


Figure 8: Uniform scalar quantizer, with a dead zone twice as large as the size of the other bins.

B.2.2 High resolution scalar quantization

In the following we examine the properties of the scalar quantizers under the following conditions :

1. the probability density function $f(x)$ of the source is smooth,
2. the number of bins N is large,
3. the size of the quantizers bins, Δ_j, is very small,
4. the p.d.f $f(x)$ in a cell is approximately constant. In particular, one has

$$f(x) = \frac{p_j}{\Delta_j} \quad \text{if} \quad x \in [x_j, x_{j+1}) \tag{32}$$

   where $p_j = P(X \in [x_j, x_{j+1}))$.

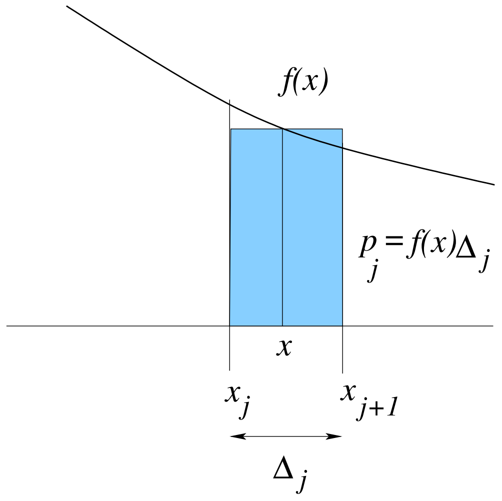


Figure 9: High resolution quantization

If these assumptions are true, we will say that the quantizer operates in the "high resolution" mode. Under these circumstances, one can compute the distortion D as a function of p_j and Δ_j.

$$D = \sum_{j=0}^{N-1} \int_{x_j}^{x_{j+1}} (x - Q(x))^2 f(x) dx \tag{33}$$

but on $[x_j, x_{j+1}]$, $f(x) = p_j/\Delta_j$, and $Q(x) = Q_j$ so one gets :

$$D = \sum_{j=0}^{N-1} \frac{p_j}{\Delta_j} \int_{x_j}^{x_{j+1}} (x - Q_j)^2 dx \tag{34}$$

Clearly, the integral $\int_{x_j}^{x_{j+1}} (x - Q_j)^2 dx$ is minimum if one chooses $Q_j = x_j + \Delta_j/2$. Then the integral is simply $\Delta_j^3/12$. Therefore the distortion becomes :

$$D = \frac{1}{12} \sum_{j=0}^{N-1} p_j \Delta_j^2 \tag{35}$$

In the case where all quantization cells are equal to Δ, the quantizer is uniform and one gets :

$$D = \frac{1}{12} \Delta^2 \sum_{j=0}^{N-1} p_j = \frac{1}{12} \Delta^2 \tag{36}$$

Let us normalize the source X, and define $Y = X/\sigma$. Y takes its values in $[a/\sigma, b/\sigma]$, and let us define

$$c = \frac{b-a}{\sigma} \tag{37}$$

c measures the size of the "normalized" range of X. We can then express D as a function of R.

$$D = \frac{\Delta^2}{12} = (\frac{b-a}{\sigma})^2 \sigma^2 \frac{1}{12N^2} \tag{38}$$

but $N = 2^R$, and thus

$$D = \frac{c^2}{12}\sigma^2 2^{-2R} \tag{39}$$

The distortion increases as σ increases, and it decays exponentially as a function of the rate R.

C Entropy coding

We very briefly describe in this section the principles of entropy coding. A very good reference on the subject is provided in {Cover91}.

The aim of entropy coding is to provide a representation of a set of symbols with a description whose average length is minimal. This can be achieved by using a variable length coding mechanism: the most frequent outcomes of the data is encoded with the shortest description, and longer descriptions are kept for less frequent realizations.

We consider a source vector $\mathbf{X} = (X_0, X_1, \ldots, X_{N-1})$, where the X_i are random variables that take their values in an alphabet $\mathcal{A} = \{a_0, a_1, \ldots, a_{M-1}\}$. We assume that the X_i are identically distributed with a probability distribution p.

An encoder α is defined as a map from the alphabet $\mathcal{A}$ to the set of binary sequences:

$$\alpha \ : \mathcal{A} \longrightarrow \{0,1\}^* \tag{40}$$

A decoder β is defined as a map from the the set of binary sequences back to the alphabet:

$$\beta \ : \{0,1\}^* \longrightarrow \mathcal{A} \tag{41}$$

A minimum property required for the decoder is that one should be able to recover the original data perfectly:

$$\beta(\alpha(a)) = a \tag{42}$$

We define the length $l(\alpha(a))$ of an encoded symbol $\alpha(a)$ as the number of bits (0, and 1) in $\alpha(a)$. The average length of the code over the alphabet is

$$\bar{l}(\alpha) = \sum_{a \in \mathcal{A}} p(a) l(\alpha(a)) \tag{43}$$

Encoders can generate fixed length sequences, or variable length sequences.

Example 1 *Let X be a random variables that takes its value in $\{0, 1, 2, 3\}$, with the probabilities :*

$$\begin{aligned} P(X = 0) &= 1/2 \\ P(X = 1) &= 1/4 \\ P(X = 2) &= 1/8 \\ P(X = 3) &= 1/8 \end{aligned}$$

We construct the following code α,

$$\begin{aligned} \alpha(0) &= 0 \\ \alpha(1) &= 10 \\ \alpha(2) &= 110 \\ \alpha(3) &= 111 \end{aligned} \tag{44}$$

The average code length is

$$\bar{l}(\alpha) = \frac{1}{2} + \frac{2}{4} + \frac{6}{8} = 1.75$$

We note that any sequence of bits that was generated by the coder by concatenating the codewords can be uniquely decoded to generate the original sequence of symbols. For instance, the sequence 110010101110 *is decoded as* 312241.

Example 2 *Let X be a random variables that takes its value in $\{0, 1, 2, 3\}$. We construct the following code α,*

$$\begin{aligned} \alpha(0) &= 0 \\ \alpha(1) &= 01 \\ \alpha(2) &= 010 \\ \alpha(3) &= 10 \end{aligned} \tag{45}$$

This code can generate sequences of bits that can be decoded in multiple ways. The code is not useful, because it creates ambiguity in the decoding process. For instance the sequence 01010 *can be interpreted as decoded in the following ways :*

$$\begin{array}{rcl} 0,10,10 & \rightarrow & 0,3,3 \\ 01,0,10 & \rightarrow & 1,0,3 \\ 01,010 & \rightarrow & 1,3 \\ 010,10 & \rightarrow & 3,1 \end{array} \tag{46}$$

The ambiguity stems from the fact that the beginning (prefix) of some codewords coincide with already existing codewords.

Definition 2 *A code is called a prefix code, or an instantaneous code if no codeword is a prefix of any other codeword*

The code defined in example 1 is instantaneous, while the code defined in example 2 is not.

As shown in Fig. 10, the codewords of a prefix code can be organized in a binary tree.

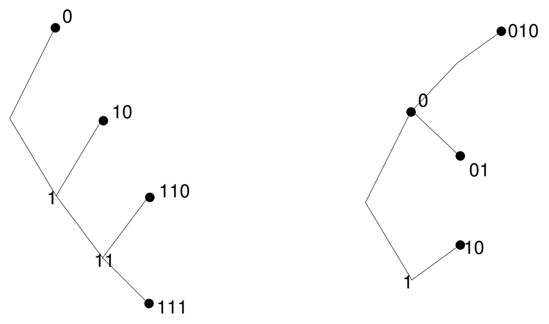


Figure 10: Left : prefix code of example 1. Right : the code of example 2 is clearly not a prefix code.

The fact that the code is instantaneous can be described by the fact that the codewords cannot have any children in the tree (see Fig. 10).

Definition 3 *We define a uniquely decodable code as a code such that, for any valid encoded sequence of finite length, there exists only one possible input symbol.*

The goal of entropy encoding is to build a uniquely decodable encoder, that minimizes the average code length. As one can expect, there is a lower bound on the average length of a uniquely decodable code. This lower bound is given by he Kraft inequality:

Lemma 1 (Kraft Inequality) *If a code is uniquely decodable, then*

$$\sum_{a \in \mathcal{A}} 2^{-l(\alpha(a))} \leq 1 \tag{47}$$

Example 3 *Let us assume that the each X_i can take L values with a uniform probability $p = 1/L$. If one codes each X_i with $\log_2 L = -\log_2 p$ bits, then*

$$\sum_{l=1}^{L} 2^{-\log_2 L} = \sum_{l=1}^{L} \frac{1}{L} = 1 \tag{48}$$

and the Kraft inequality is satisfied. In fact , one can get a more precise estimate of the average length of a code as a function of the complexity of the source (measured with the entropy).

Definition 4 *Let X be a random variable that takes its values in the alphabet $\mathcal{A}$, and let p be the probability distribution of X. We define the entropy of X as:*

$$H(X) = -\sum_{a \in \mathcal{A}} p(a) \log_2 p(a) \tag{49}$$

where $\log_2$ is the logarithm in base 2.

Example 4 *Let*

$$X = \begin{cases} 1 & \textit{with probability} \quad p \\ 0 & \textit{with probability} \quad 1-p \end{cases} \tag{50}$$

Then

$$H(X) = -p \log_2 p - (1-p) log_2 (1-p) \tag{51}$$

We can then state Shannon's lossless coding theorem.

Lemma 2 (Shannon 1948) *For any uniquely decodable scalar lossless code α we have*

$$\bar{l}(\alpha) \geq H(X) \tag{52}$$

and there exists a code such that

$$\bar{l}(\alpha) < H(X) + 1 \tag{53}$$

We will not give the complete proof of the theorem, but we will show that $\bar{l}(\alpha) \geq H(x)$. From Kraft inequality we have

$$\frac{1}{\sum_{b\in\mathcal{A}} 2^{-l(\alpha(b))}} \geq 1 \tag{54}$$

and therefore

$$\frac{2^{-l(\alpha(a))}}{\sum_{b\in\mathcal{A}} 2^{-l(\alpha(b))}} \geq 2^{-l(\alpha(a))}. \tag{55}$$

Therefore

$$\bar{l}(\alpha) = \sum_{a\in\mathcal{A}} p(a)l(a) = -\sum_{a\in\mathcal{A}} p(a)\log_2 2^{-l(\alpha(a))} \geq -\sum_{a\in\mathcal{A}} p(a)\log_2 \frac{2^{-l(\alpha(a))}}{\sum_{b\in\mathcal{A}} 2^{-l(\alpha(b))}} \tag{56}$$

The quantity

$$q(a) = \frac{2^{-l(\alpha(a))}}{\sum_{b\in\mathcal{A}} 2^{-l(\alpha(b))}} \tag{57}$$

can be interpreted as a probability distribution over the alphabet $\mathcal{A}$. We can use the inequality

$$\sum_{a\in\mathcal{A}} p(a)\log_2 \frac{1}{q(a)} \geq \sum_{a\in\mathcal{A}} p(a)\log_2 \frac{1}{p(a)} \tag{58}$$

to conclude that

$$\bar{l}(\alpha) \geq -\sum_{a\in\mathcal{A}} p(a)\log_2 p(a) = H(X). \tag{59}$$

Example 5 *For the code of example 1, we have* $H(X) = -(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{4}\log_2\frac{1}{4} + \frac{1}{8}\log_2\frac{1}{8} + \frac{1}{8}\log_2\frac{1}{8}) = 1.75$. *Therefore we have* $\bar{l}(\alpha) = H(X)$.

If one is coding a block of stationary random variables $\mathbf{X} = (X_0, \ldots, X_{N-1})$ then the entropy rate is defined as

$$\overline{H} = \min_N \frac{H(\mathbf{X})}{N} = \lim_{N\to\infty} \frac{H(\mathbf{X})}{N} \tag{60}$$

and there exists a prefix code for which the average length per symbol is bounded by

$$\overline{H}(\mathbf{X}) \leq \frac{1}{N}\bar{l}(\alpha) < \overline{H}(\mathbf{X}) + \frac{1}{N} \tag{61}$$

The performance of the coder can in principle be improved by coding blocks of larger size. But the increase in block size result in a major increase in complexity.