JPEG-LS

The lossless mode of JPEG is inefficient and often is not even implemented. As a result, the ISO decided to develop a new standard for the ***lossless (or near-lossless)*** **compression of *continuous-tone images*.** The result became popularly known as JPEG-LS. This method is not simply an extension or a modification of JPEG. It is a new method, designed to be simple and fast. ***It does not use the DCT, does not use arithmetic coding, and uses quantization in a limited way, and only in its near-lossless option.***

JPEG-LS examines several of the previously seen neighbors of the current pixel, uses them as the *context* of the pixel, uses the context to predict the pixel and to select a probability distribution out of several such distributions, and uses that distribution to encode the prediction error with a special Golomb code. There is also a run mode, where the length of a run of identical pixels is encoded.

Vector Quantization

This is a generalization of the scalar quantization method. It is used for both image and sound compression. In practice, vector quantization is commonly used to compress data that has been digitized from an analog source, such as sampled sound and scanned images (drawings or photographs). Such data is called ***Digitally Sampled Analog Data (DSAD).*** It is a lossy compression method.

Vector quantization is based on two facts:

i. We know that compression methods that compress strings, rather than individual symbols, can, in principle, produce better results.
ii. Adjacent items in an image and in digitized sound are correlated. There is a good chance that the near neighbors of a pixel P will have the same values as

P or very similar values. Also, consecutive audio samples rarely differ by much.

The basic vector quantization procedure is illustrated in the following figure says that the encoder finds the closest code vector to the input vector and outputs the associated index. On the decoder side, exactly the same codebook is used. When the code index of the input vector is received, *a simple table lookup is performed* to determine the reconstruction vector.

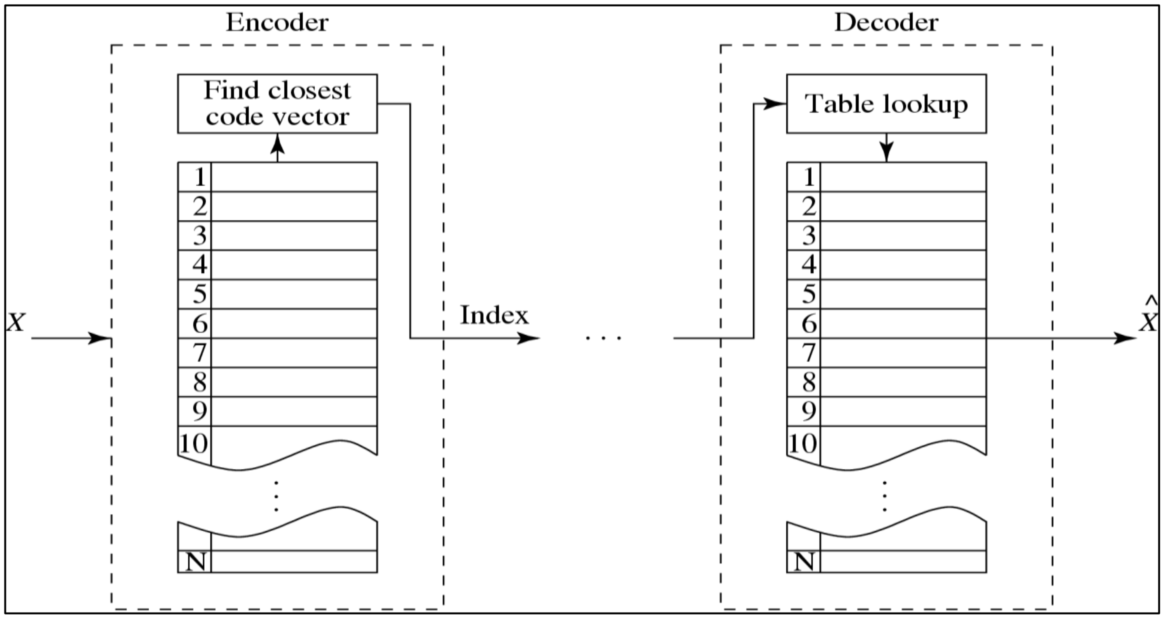


Figure (3) Basic vector quantization procedure

Now how to build this codebook, there are many ways, LBG algorithm is the basis of many vector quantization methods for the compression of images and sound.

Its main steps are the following:

<u>Step 0:</u> Select a threshold value ϵ and set k = 0 and $D^{(-1)} = \infty$. Start with an initial codebook with entries $C_i^{(k)}$(where k is currently zero, but will be incremented in each iteration). Denote the image blocks byB_i (these blocks are also called *training vectors*, since the algorithm uses them to find the best codebook entries).

Step 1: Pick up a codebook entry $C_i^{(k)}$. Find all the image blocks $B_m$that are closer to $C_i$than to any other C_j . Phrased more precisely; find the set of all B_m that satisfy

$$d(B_m, C_i) < d(B_m, C_j) for\ all\ \ i \neq j$$

This set (or *partition*) is denoted by $P_i^{(k)}$. Repeat for all values of *i*. It may happen that some partitions will be empty, and we deal with this problem below.

Step 2: Select an *i* and calculate the distortion $D_i^{(k)}$ between codebook entry $C_i^{(k)}$ and the set of training vectors (partition) $P_i^{(k)}$found for it in Step 1. Repeat for all *i*, then calculate the average $D^{(k)}$of all the $D_i^{(k)}$. A distortion $D_i^{(k)}$ for a certain *i* is calculated by computing the distances $d(C_i^{(k)}, B_m)$ for all the blocks $B_m$in partition $P_i^{(k)}$, then computing the average distance.

Step 3: If $\left(D^{(k-1)} - D^{(k)}\right)/D^{(k)} \ \leq \epsilon$ halt. The output of the algorithm is the last set of codebook entries $C_i^{(k)}$. This set can now be used to (lossy) compress the image with vector quantization. In the first iteration *k* is zero, so $D^{(k-1)} = D^{(-1)} = \infty > \epsilon$. This guarantees that the algorithm will not stop at the first iteration.

Step 4: Increment *k* by 1 and calculate new codebook entries $C_i^{(k)}$; each equals the average of the image blocks (training vectors) in partition $P_i^{(k-1)}$ that was computed in Step 1. (This is how the codebook entries are adapted to the particular image.) Go to Step 1.

Example:

Our example assumes an image consisting of 24 pixels, organized in the 12 blocks (each has 2 pixels that can be plotted on paper as 2D points)

$B1$ = (32, 32),

$B2$ = (60, 32),

$B3$ = (32, 50),

$B4$ = (60, 50),

$B5$ = (60, 150),

$B6$ = (70, 140),

$B7$ = (200, 210),

$B8$ = (200, 32),

$B9$ = (200, 40),

$B10$ = (200, 50),

$B11$ = (215, 50),

and $B12$ = (215, 35).

It is clear that the 12 points are concentrated in four regions. We select an initial codebook with the four entries

$C_1^{(0)}$= (70, 40),

$C_2^{(0)}$ = (60, 120),

$C_3^{(0)}$= (210, 200),

and $C_4^{(0)}$= (225, 50)

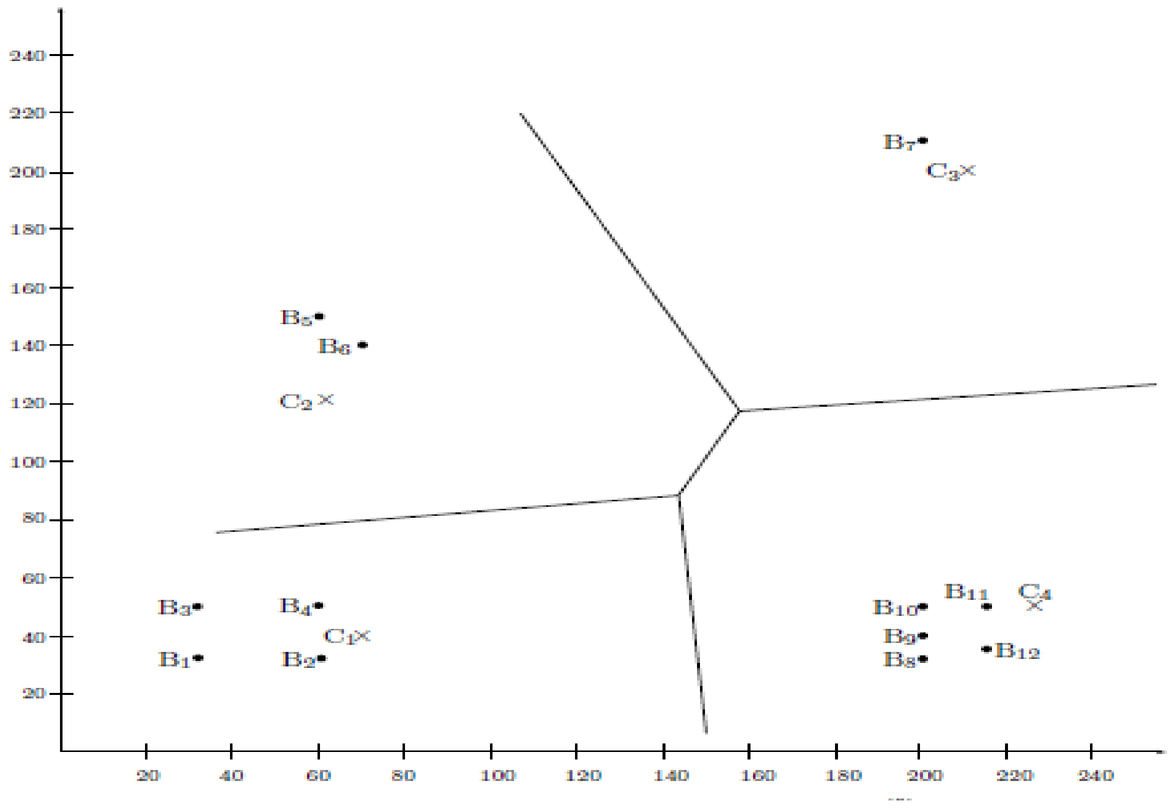
(shown as × in the diagram). These entries were selected more or less at random but we show later how the LBG algorithm selects them methodically, one by one. Because of the graphical nature of the data, it is easy to determine the four initial partitions. They are

$P_1^{(0)}$= (B1,B2,B3,B4),

$P_2^{(0)}$= = (B5,B6),

$P_3^{(0)}$= = (B7), and

$P_4^{(0)}$= (B8,B9,B10,B11,B12).



The table below to compute the $D_i^{(0)}$

$$\begin{array}{rrr}
\text{I:} & (70-32)^2+(40-32)^2 = 1508, & (70-60)^2+(40-32)^2 = 164, \\
 & (70-32)^2+(40-50)^2 = 1544, & (70-60)^2+(40-50)^2 = 200, \\
\text{II:} & (60-60)^2+(120-150)^2 = 900, & (60-70)^2+(120-140)^2 = 500, \\
\text{III:} & (210-200)^2+(200-210)^2 = 200, & \\
\text{IV:} & (225-200)^2+(50-32)^2 = 449, & (225-200)^2+(50-40)^2 = 725, \\
 & (225-200)^2+(50-50)^2 = 625, & (225-215)^2+(50-50)^2 = 100, \\
 & (225-215)^2+(50-35)^2 = 325. &
\end{array}$$

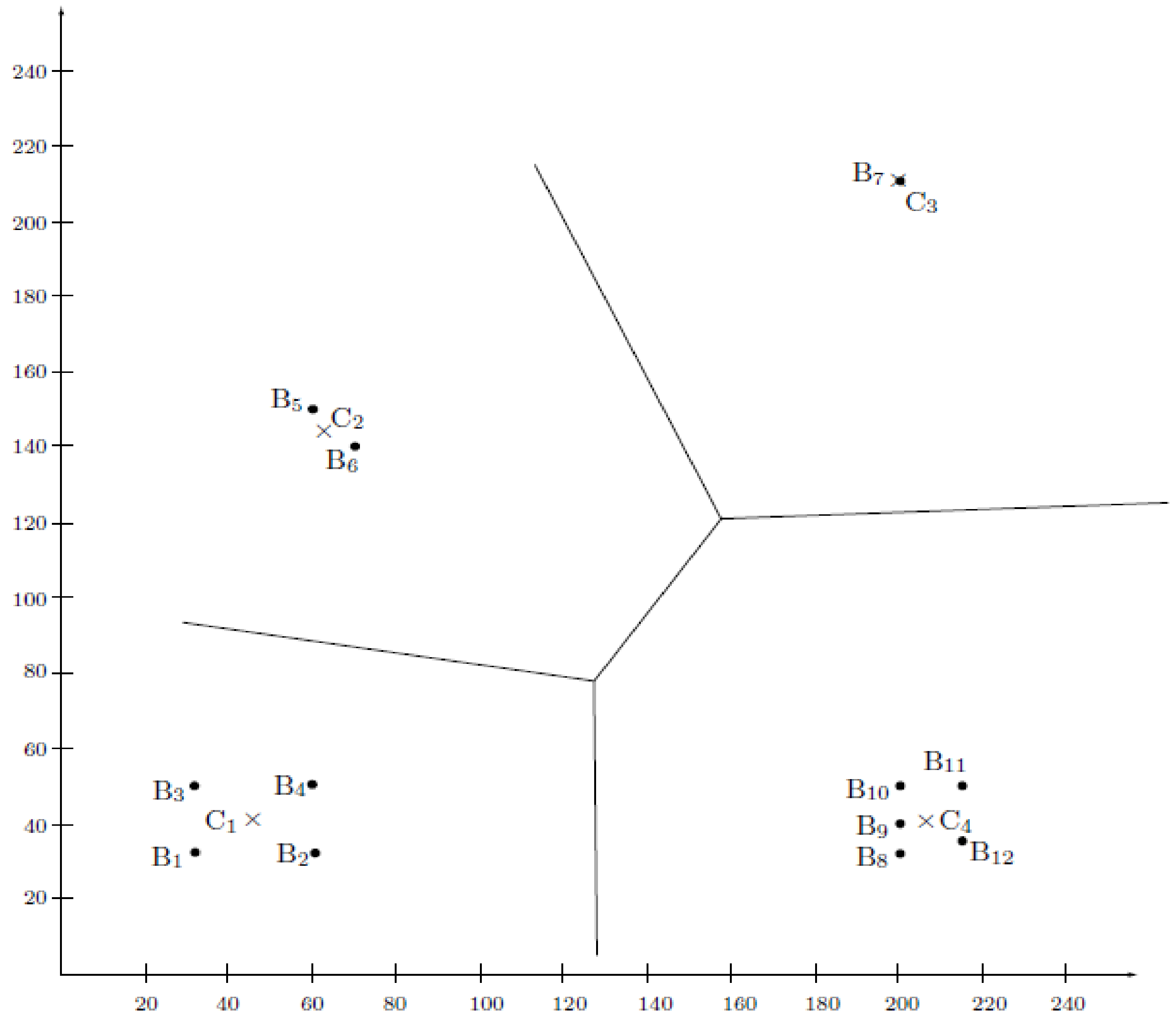
The Table above shows how the average distortion $D^{(0)}$is calculated for the first iteration (we use the Euclidean distance function). The result is

$$\begin{aligned}
D^{(0)} =&(1508+164+1544+200+900+500 \\
&+200+449+725+625+100+325)/12 \\
=&603.33.
\end{aligned}$$

Step 3 indicates no convergence, since $D(-1) = \infty$, so we increment k to 1 and calculate four new codebook entries $C(1)i$ (rounded to the nearest integer for simplicity)

$$\begin{aligned}
C_1^{(1)} &= (B_1+B_2+B_3+B_4)/4 = (46,41), \\
C_2^{(1)} &= (B_5+B_6)/2 = (65,145), \\
C_3^{(1)} &= B_7 = (200,210), \\
C_4^{(1)} &= (B_8+B_9+B_{10}+B_{11}+B_{12})/5 = (206,41).
\end{aligned}$$

They are shown in the Figure below.



240
220
200
180
160
140
120
100
80
60
40
20
20 40 60 80 100 120 140 160 180 200 220 240
B_7
C_3
B_5
C_2
B_6
B_3
B_4
C_1
B_1
B_2
B_{11}
B_{10}
B_9
C_4
B_8
B_{12}