Simple and Fast Ordered Codebook for Vector Quantization

K. Somasundaram¹ and S. Vimala²

¹Department of Computer Science and Applications, Gandhigram Rural Institute
Gandhigram–624 302, Tamil Nadu, India
E-mail: somasundaramk@yahoo.com

²Department of Computer Science, Mother Teresa Women’s University
Kodaikanal–624 102, Tamil Nadu, India
E-mail: vimalaharini@gmail.com

ABSTRACT—In this paper, we propose two fast and efficient methods to generate codebook for Vector Quantization (VQ). The computational complexity is very simple when compared to any one of the existing algorithms for generating codebook for VQ. In the first method, codevectors are selected at regular intervals from the training set. The second method is an improvement over the first one. In the second method, the training vectors are ordered by the sum of the elements in each training vector. From the ordered training set, codebook is generated by taking the training vectors at regular intervals. Experimental results on standard images show that both methods give better results in image quality and computation time than the popular LGB algorithm. Among the two, the second method gives better results than the first one.

KEYWORDS: Vector Quantization, Image Compression, Codebook Generation.

I. INTRODUCTION

Vector Quantization (VQ) is a lossy image compression technique and has applications in different areas: protein classification, secondary structure computation [1], speech recognition, face detection, pattern recognition, real-time video based event detection and Anomaly Intrusion Detection System, etc. [2]. VQ techniques have been used for a number of years for data compression. With its relatively simple structure and computational complexity, VQ has received great attention in the last decade. LBG (Linde Buzo Gray) [3] is the most widely referred VQ method for designing a codebook. VQ comprises of three stages codebook designing, encoding and decoding of image. The codebook generation is the important task in VQ.

There are several known methods for generating a codebook [4]. The most cited and widely used is the Generalized Lloyd Algorithm (GLA) [LBG] [3]. It starts with an initial solution, which is iteratively improved using two optimality criteria until a local minimum is reached. A different approach is to build the codebook hierarchically. The iterative splitting algorithm [5] starts with a codebook of size one, where the only code vector is the centroid of the entire training set. The codebook is then iteratively enlarged by a splitting procedure until it reaches the desired size. Another hierarchical algorithm, the Pairwise Nearest Neighbour (PNN) [6] uses an opposite, bottom-up approach to generate the codebook. It starts by initializing a codebook where each training vector is considered as its own code vector. Two code vectors are merged in each step of the algorithm and the process is repeated until the desired size of the codebook is reached. From the two hierarchical approaches [7], the PNN has higher potential because it gives better results with a simpler implementation. It can also be used to produce an initial codebook for the GLA or it can be embedded into other hybrid methods. The greatest deficiency of PNN is its slow speed.

Research efforts in codebook generation techniques have been concentrated in two directions: to generate a codebook that approaches global optimal solution, and to reduce the computational complexity of the LBG algorithm.

Many methods for reducing the time for codebook generation have appeared in literature. The Subspace Distortion method [8] attempts to reduce the computation by reducing the dimension of the distortion measure in the LBG algorithm. The PNN algorithm [7] generates a codebook by merging nearest training vector clusters until the desired number of codevectors is reached. The codebooks generated by both methods are slightly degraded even though the computation time is reduced by several times. The generation of better codebooks requires longer computation time and that fast codebook generation methods often suffer from degradation in overall distortion [11].
In this paper we propose two methods Simple Codebook Generation (SCG) and Ordered Codebook Generation (OCG). Experiments show both methods give better PSNR and time than that of LBG and certain other techniques.

The remaining part is organized as follows: In section 2, a brief outline of the existing methods for codebook generation is given. In section 3, the proposed methods and experimental results are given and in section 4, the conclusion is given.

II. EXISTING METHODS

Linde Buzo Gray (LBG) Algorithm

In this method, centroid of the entire training set is computed first. To the centroid, a constant error \( \{1,1,1,1,1,1,1,1,1,1\} \) is added to the centroid, i.e. two different vectors \( v1 \) and \( v2 \) are obtained by adding the constant error to and subtracting the constant error from the centroid respectively. Two clusters are formed by grouping the nearest vectors of \( v1 \) and \( v2 \) using the minimum distance method. The centroids of these two clusters are computed and again the constant error is added to the centroids to get further codevectors. These steps are repeated until the desired size of codebook is generated.

Kekre’s Proportionate Error Algorithm (KPE) [10]

In this method, a proportionate error is added to the centroids of the clusters. The error value is decided based on the components of the centroid. The minimum \( c_j \) of the components of the codevector \( \{c_1, c_2, c_3, …, c_i\} \) is computed. The error value \( e_i \) is calculated as:

\[
\text{if } c_i/c_j \leq 10 \text{ then assign } e_i = c_i/c_j \\
\text{else assign } e_i = 10
\]

where, \( c_i \) is the individual component of the codevector and \( c_j \) is the minimum of the components. \( e_i \) is the component of the error vector \( \{e_1, e_2, …, e_k\} \). Every time when the new clusters are formed, the error vector is generated and is added to the codevectors to form the new codevectors.

Kekre’s Efficient Fast Algorithm (KEFA)[10]

In this method, the image is divided into blocks and blocks are converted to the codevectors of size \( k \). Giving a matrix \( T \) of size \( M \times k \) consisting of \( M \) number of image training vectors of dimension \( k \), each row of the matrix is the image training vector of dimension \( k \). The training vectors are sorted with respect to the first column of the matrix \( T \) and the entire matrix is considered as one single cluster. The median of the matrix \( T \) is chosen as codevector for the codebook. The matrix is then divided into two equal parts and each of the part is again sorted based on the second column of the matrix and we obtain two clusters both consisting of equal number of vectors. The median of both the parts are picked up as codevectors for the codebook. Now the size of the codebook is two and two other clusters are formed. These clusters are again divided into four other parts and the median of the four parts are selected to give further codevectors. The above steps are repeated until a codebook of desired size is reached. This algorithm takes very less time compared to the above two algorithms, since Euclidean distance computation is not required.

III. PROPOSED METHODS

(a) Simple Codebook Generation (GLA Based)

An image of size \( M \times M \) is first divided into small non-overlapping blocks of size \( 4 \times 4 \) pixels. Usually \( M \) is a power of 2. The block is then converted to a one-dimensional array of 16 elements. An image of size \( M \times M \) will give \( N \) training vectors, where

\[
N = (M \times M)/16 \quad \text{… (1)}
\]

The aim of VQ is to design an optimal codebook (CB) of size \( n \) which comprises of \( n \) codevectors that will be the representative of the training vectors.

\[
\text{CB} = \{Ci|1 \leq i \leq n\}, n < N \quad \text{… (2)}
\]

To generate a codebook of size 256, training vectors at every 16th position are selected to form the codevectors. For a codebook of size 512, the training vectors at every 8th position and for the codebook of size 1024, the training vectors at every 4th position are selected. Similar to the KEFA method, the Euclidean distance computation is not needed in this method and hence it takes only less time. This technique has been named Simple Codebook Generation (SCG) technique. The position from which the codevectors to be selected is calculated as,

\[
p = N/n \quad \text{… (3)}
\]

where, \( n \) is the size of the codebook.

SCG Algorithm

Step 1: Input the given image of size \( M \times M \) pixels.
Step 2: Divide it into \( 4 \times 4 \) blocks.
Step 3: Generate \( N \) training vectors.
Step 4: Fix the codebook size to \( n \).
Step 5: Compute \( p = N/n \).
Step 6: Select every \( p^{th} \) training vector as a codevector till the codebook of desired size is obtained.

Compression of an image is done by taking every training vector and finding a closest match in the codebook by computing the minimum distortion \( d_{ij} \) such that,

\[
p = N/n \quad \text{… (3)}
\]

where, \( n \) is the size of the codebook.

SCG Algorithm

Step 1: Input the given image of size \( M \times M \) pixels.
Step 2: Divide it into \( 4 \times 4 \) blocks.
Step 3: Generate \( N \) training vectors.
Step 4: Fix the codebook size to \( n \).
Simple and Fast Ordered Codebook for Vector Quantization

Step 5: Compute \( p = \frac{N}{n} \).

Step 6: Select every \( p \)-th training vector as a codevector till the codebook of desired size is obtained.

Compression of an image is done by taking every training vector and finding a closest match in the codebook by computing the minimum distortion \( d_{ij} \) such that,

\[
d_{ij} = \min_{i,j=1}^{k} \left| \sum_{i=1}^{j} x_i - x_j \right|
\]

where \( i = 1, 2, 3, \ldots, N, j = 1, 2, 3, \ldots, M \) and \( i \neq j \)

where \( I \) is the \( i \)-th training vector and \( j \) is the \( j \)-th codevector.
The index of the codevector satisfying (4) is taken as the compressed data. A set of all indices generated for the image blocks along with the codebook are then written as the compressed data. The decompression is done by replacing the index in the compressed data by the corresponding codevector from the codebook. The encoded codevectors form the reconstructed image.

Experiments were carried out by applying SCG on standard images Lena, Boats, Baboon, Bridge, Kush, Pepper and Goldhill. The PSNR values and time taken to generate codebook are given in Table 1 for different codebook sizes 128 and 256. For comparison, results obtained by LBG, KPE and KEFA [3] are also given in Table 1.

From the Table 1, we note that the method SCG gives superior results in terms of PSNR and the time taken to generate codebook than that for LBG, KPE and KEFA.

For visual comparison, the reconstructed images by different algorithms are given in Fig. 1.

### Iterative Clustering

Iterative clustering is done to get the centroids to form the refined codebook. After every iteration, it is observed that the PSNR value of the reconstructed image has been increased. As mentioned above, the codebook thus generated using the above algorithm has undergone iterative clustering and averaging operations to improve the PSNR values. The observations of the above method are given in Table 2 for various images of size 256 × 256 pixels. Lena, Cameraman, Boats, Bridge and Rice of size 256 × 256 pixels are taken for the study.

### Iterative Clustering and Averaging Procedure

Iterative Clustering has been used to improve the optimality of the codebook. During the iterations, the training vectors are grouped into clusters and the average of the codevectors that belong to one cluster is computed to form the new codeword (Centroid). This is repeated until the codebooks of consecutive iterations converge.

---

**Table 1: Time (in Seconds) for Codebook Generation and PSNR Values Obtained for the Proposed Method**

<table>
<thead>
<tr>
<th>Codebook Size : 128</th>
<th>Codebook Size : 256</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LBG</td>
</tr>
<tr>
<td>Kush</td>
<td>85.34</td>
</tr>
<tr>
<td>Time</td>
<td>PSNR</td>
</tr>
<tr>
<td>Lena</td>
<td>87.37</td>
</tr>
<tr>
<td>Time</td>
<td>PSNR</td>
</tr>
<tr>
<td>Baboon</td>
<td>84.87</td>
</tr>
<tr>
<td>Time</td>
<td>PSNR</td>
</tr>
<tr>
<td>Pepper</td>
<td>85.10</td>
</tr>
<tr>
<td>Time</td>
<td>PSNR</td>
</tr>
<tr>
<td>GoldHill</td>
<td>87.81</td>
</tr>
<tr>
<td>Time</td>
<td>PSNR</td>
</tr>
<tr>
<td>Boats</td>
<td>98.31</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Fig. 1: Reconstructed images of Lena (a) at a Codebook of Size 256 by Different Methods: (b) LBG (30.22) (c) KPE (25.61) (d) KEFA (30.35) and (e) Proposed SCG (31.13). Numbers within Brackets Represent the PSNR Values
Table 2: Time (in Seconds) and PSNR Values of Images of Size 256 × 256 with and without Clustering Iteration

<table>
<thead>
<tr>
<th>CB Size</th>
<th>Simple Codebook</th>
<th>Simple Codebook with iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Image</td>
<td>Lena</td>
</tr>
<tr>
<td>128</td>
<td>Time</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>Iteration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>28.66</td>
</tr>
<tr>
<td>256</td>
<td>Time</td>
<td>0.016</td>
</tr>
<tr>
<td></td>
<td>Iteration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>30.35</td>
</tr>
<tr>
<td>512</td>
<td>Time</td>
<td>0.016</td>
</tr>
<tr>
<td></td>
<td>Iteration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>32.00</td>
</tr>
<tr>
<td>1024</td>
<td>Time</td>
<td>0.016</td>
</tr>
<tr>
<td></td>
<td>Iteration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>33.37</td>
</tr>
</tbody>
</table>

Table 3: Time (in Seconds) for Codebook Generation and PSNR Values for the Proposed Method

<table>
<thead>
<tr>
<th>Methods</th>
<th>Codebook Size : 128</th>
<th>Codebook Size : 256</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LBG</td>
<td>KPE</td>
</tr>
<tr>
<td>Kush</td>
<td>Time</td>
<td>85.34</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>28.79</td>
</tr>
<tr>
<td>Lena</td>
<td>Time</td>
<td>87.37</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>27.21</td>
</tr>
<tr>
<td>Baboon</td>
<td>Time</td>
<td>84.87</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>21.07</td>
</tr>
<tr>
<td>Pepper</td>
<td>Time</td>
<td>85.10</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>26.36</td>
</tr>
<tr>
<td>GoldHill</td>
<td>Time</td>
<td>87.81</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>26.97</td>
</tr>
<tr>
<td>Boats</td>
<td>Time</td>
<td>98.31</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>24.99</td>
</tr>
</tbody>
</table>

Fig. 2: Reconstructed image of Kush (a) at a codebook of size 256 by different methods (b) LBG (29.93) (c) KPE (31.36) (d) KEFA (31.85) and e) proposed OCD (34.54). Numbers within brackets represent the PSNR values.
(b) Ordered Codebook Generation

We further improved the codebook generation method SCG by sorting the training vectors and performing the following OCG algorithm.

**OCG Algorithm**

Step 1: Input the given image of size $M \times M$ pixels.
Step 2: Generate $N$ training vectors.
Step 3: Sort the training vectors in ascending order based on the magnitudes of the vectors.
Step 4: Fix the codebook size $n$.
Step 5: Compute $p = N/n$.
Step 6: Select every $p^{th}$ training vector as a from the sorted list.
Step 7: Repeat the Step 5 until all the training are processed.

Magnitude of a training vector is computed as,

$$Y = \sum_{i=1}^{k} C_i$$  \hspace{1cm} \ldots (6)

Using the equation (6), the magnitudes of all the training vectors are computed and the training vectors are sorted based on the magnitude values. From the sorted list, training vectors from every $p^{th}$ position are selected to form the codevectors. This leads to uniform distribution of codevectors.

The results obtained for standard images in terms of PSNR and time are given in Table 2.

**IV. RESULTS AND DISCUSSION**

From Table 3, we note that OCG gives better PSNR value than the other methods LBG, KPE and KEFA. Further our proposed OCG is also a fast algorithm. The time taken to generate the codebook is faster by 4000 times in most cases. The visual quality of the proposed SCG and OCG are also better than LBG, KPE and KEFA methods.

**V. CONCLUSION**

In this paper, we have proposed two codebook generation methods, simple codebook generation and ordered codebook generation. Both methods when applied to standard images compression give better PSNR values for the reconstructed image. Both are fast by 3 orders of magnitude when compared with the popular methods LBG, KPE and KEFA.

**REFERENCES**