CHAPTER 5

Enhancement of Codebook Generation Algorithms

5.1. Introduction

Codebook generation is an important component of VQ method. In this chapter, we propose three simple adaptive techniques, to improve the quality of the codebook that is generated using Ordered Codebook Generation (OCG) that has been introduced in the previous chapter.

Three different methods have been introduced to generate a codebook. In one method, edge blocks are preserved to avoid ragged edges in the reconstructed images. In the second method, for each training vector, the other training vectors that are closer to it are identified. The nearest training vectors are grouped into clusters. The number of training vectors that belong to one cluster is referred to as Cluster Density. The training vectors are sorted based on the cluster densities and the top $M$ training vectors with higher cluster densities are selected as codevectors. In the third method, being the variant of ordered codebook generation (OCG) technique, the training vectors are sorted based on the sum of the sub vectors. This enables to identify the representative codevectors to be, more closer to the training vectors. The three methods are explained in detail in this chapter.
5.2. Codebook Generation for Vector Quantization with Edge Features (CGEF)

The most noticeable artefact with the techniques discussed so far is a staircase effect which causes the diagonal edges in the reconstructed images to be degraded to jagged ones. The staircase effect shows that there is a need for a larger set of training vectors containing portions of edges to allow greater flexibility to match blocks that contain edges in the original image. Edge is a very significant feature in an image. A truthful coding that preserves the edge information is of importance. Hence this method is proposed to retain the edges blocks that results in better quality of the reconstructed images.

The input image of size \( m \times m \) pixels is divided into \( N \) sub blocks of size \( 4 \times 4 \) pixels using the equation (5.1):

\[
N = \frac{m \times m}{16}
\]  

(5.1)

The \( N \) image blocks thus formed are categorized into two classes namely the \textit{edge} blocks and the \textit{shade} blocks. The high detailed blocks are classified as edge vectors and the low detailed blocks are classified into shade vectors. The set of edge vectors is treated as a separate codebook \( T_1 \) and the set of shade vectors is \( T_2 \). The final codebook is the concatenation of \( T_1 \) and \( T_2 \).

The categorization of blocks is done as follows:
Mean of the components of each vector is computed using the equation (5.2) as

$$m = \frac{\sum_{i=1}^{16} X_{ij}}{16} \quad \text{where} \ 1 \leq i \leq N \ \text{and} \ 1 \leq j \leq 16$$

(5.2)

Sum of the difference between the individual components and the mean value is computed as:

$$s = \sum_{j=1}^{16} \left|m - X_{ij}\right|$$

(5.3)

If the value of $s$ is greater than a threshold value, the block is categorized as an edge block that belongs to the set T1; otherwise it is categorized as a shade block that belongs to T2. The categorization is based on the assumption that, if it is an edge block, the gray values of the adjacent pixels will be of different values. If it is a shade block, the gray levels of the adjacent pixels will be more or less the same. Hence the value of $s$ will be less for a shade block. The threshold value differs depending on the size of the codebook to be generated. For an image of size 256 x 256 pixels, the initial training set would contain 4096 training vectors of size 16. If the size of T1 is 100, then the size of T2 would be 3996 i.e. 4096 - sizeof(T1). The final codebook that is to be generated includes all the edge blocks and for the remaining number, the shade blocks are selected from every $n^{th}$ position. i.e. for a codebook of size 256 to be created, the whole T1 (size : 100) is taken. For the remaining 156 code vectors, the codevectors at every $n^{th}$ $(n=3996/156)$ position are selected from the training set T2. Now a codebook of desired size covering all the edge blocks is created.
5.2.1. CGEF algorithm

**Step1**: Input the given image of size \( n \times n \) pixels. Divide the image into blocks of size \( 4 \times 4 \) pixels.

**Step2**: Generate \( N \) training vectors using equation (5.1).

**Step3**: For every training vector \( X_i \), calculate the mean using the equation (5.2).

**Step4**: Calculate the sum using the equation (5.3).

**Step5**: If \( s > \) threshold value, then \( X_i \) belongs to \( T_1 \) else \( X_i \) belongs to \( T_2 \).

**Step6**: Repeat the steps 3 thru. 5 for all training vectors.

**Step7**: Add the whole set \( T_1 \) to the final codebook.

**Step8**: Compute \( v_1 = \text{sizeof}(TS) - \text{sizeof}(T_1) \)

**Step9**: Compute \( v_2 = \text{sizeof}(CB) - \text{sizeof}(T_1) \)

**Step10**: Compute \( n = v_1/v_2 \)

**Step11**: Select every \( n \) shade block from \( T_2 \) and add it to the final codebook.

5.3. Codebook Generation using Cluster Density (CGCD)

In this method, initially all the training vectors are treated as Cluster centers. The distance between a training vector \( X_i \) and all the other training vectors \( Y_j \) where \( i \neq j \) is computed using the equation (5.4) as

\[
D = \sum_{j=1}^{N} (X_i - Y_j)^2
\]

\[X \neq Y\]

where \( i \) and \( 1 \leq i \leq N \) (5.4)

The minimum of all the distances is identified and the Cluster Density of the corresponding training vector is incremented by 1. For example, if the distance between the 1st and the 250th vectors is the minimum, then the cluster density of the 250th vector is incremented by 1. Similarly the distance between the second training
vector and all the other vectors are computed and so on. Whichever distance is minimum the cluster density of the corresponding vector is incremented by one. The above steps are repeated for all the training vectors one by one. Hence if 25 training vectors are closer to any training vector \( i \), then the cluster density of the \( i \)th vector is 25. The training vectors are sorted in the descending order based on their cluster densities. Finally the top \( M \) training vectors are selected as the seeds for the initial codebook that is to be used as the input for the codebook optimization technique.

5.3.1. Algorithm for the proposed method

**Step1:** Input the image

**Step2:** Split the image into small blocks of size 4 x 4.

**Step3:** Generate the training set of size \( N \).

**Step4:** Initialize the Density array of size \( N \) with zero.

**Step5:** for \( i = 1 \) to \( N \)

**Step6:** for \( j = 1 \) to \( N \)

**Step7:** if \( i \) not equal to \( j \)

**Step8:** compute the distance between the vectors \( i \) and \( j \) using the equation (1).

**Step9:** next \( j \)

**Step10:** Find the minimum of all the \( N-1 \) distances computed and increment the corresponding density by 1.

**Step11:** next \( i \).

**Step12:** Sort the training vectors in ascending order based on their density values.

**Step13:** Select the top \( M \) training vectors to form the initial codebook.
5.4. Codebook Generation by Sorting the Sum of Sub Vectors

In this method, (Codebook Generation by Sorting the Sum of the Sub Vectors - CGSSSV), each training vector TV is sub divided into 4 sub vectors as:

\[
SV1 = \{TVi | 1 \leq i \leq 4\}, \quad (5.5)
\]
\[
SV2 = \{TVi | 5 \leq i \leq 8\}, \quad (5.6)
\]
\[
SV3 = \{TVi | 9 \leq i \leq 12\} \text{ and } \quad (5.7)
\]
\[
SV4 = \{TVi | 13 \leq i \leq 16\}. \quad (5.8)
\]

The sum values of the components of the individual sub vectors are calculated as:

\[
S1 = \sum_{i=1}^{4} TV_i \quad (5.9)
\]
\[
S2 = \sum_{i=5}^{8} TV_i \quad (5.10)
\]
\[
S3 = \sum_{i=9}^{12} TV_i \quad \text{and} \quad (5.11)
\]
\[
S4 = \sum_{i=13}^{16} TV_i \quad (5.12)
\]

The final sum \( S \) for each training vector \( X \) is calculated using the equation (5.15)

\[
S = S1 - S2 + S3 - S4 \quad (5.13)
\]

The training vectors are then sorted in ascending order according to the sum of the vectors. Sorting the training vectors based on the sum values enables us to have uniform distribution of training vectors. From the sorted list, the training vectors at
every \( n^{th} \) position are selected to form the codebook. The value \( n \) is computed using the equation (5.1)

### 5.4.1. The steps in CGSSSV algorithm

**Step1:** Input the image of size \( m \times m \) pixels.

**Step2:** Generate the training set of size \( N \) by dividing the image into small blocks of size \( 4 \times 4 \).

**Step3:** Set the value of \( M \), the desired size of the codebook.

**Step4:** Compute the sum values of sub vectors using the equations (5.9) to (5.12).

**Step5:** Compute the final sum \( S \) using equation (5.13).

**Step6:** Repeat the steps 4

**Step7:** Sort the training set in ascending order based on the sum values.

**Step8:** Compute the position \( n \) using equation (5.1).

**Step9:** Select the training vectors at every \( n^{th} \) position to form the codebook of desired size \( M \).

### 5.4.2. Justification for taking the sum of sub vectors

\{165, 170, 169, 35, 166, 166, 173, 101, 162, 164, 170, 166, 161, 165, 165, 172\} and \{126, 21, 46, 104, 244, 228, 150, 92, 197, 199, 153, 107, 248, 250, 163, 142\} are two sample training vectors taken from the image cameraman. If we calculate the sum of all the components for both the vectors, the values will be 2470 and 2470 respectively. The difference between the vectors is 0. When we sort the training vectors in ascending order based on the sum values, both the training vectors
will be in adjacent locations. By using the above algorithm, if we select the training vectors for codebook from every $n^{th}$ position, there are chances for missing one of the vectors. But when we look at the intensity values of both the vectors, the corresponding elements are entirely different. Hence the above proposed method is adopted. In this method, the sum values of the sub vectors are taken rather than the sum of the whole vector. As per the proposed method, the sum values of the above two vectors will be

1. $539-606+662-663 = -68$
2. $297-714+656-803 = -564$

The absolute difference between both the vectors now is 496 ($-68-(564)$). Hence these two vectors will be far away to each other in the sorted list. There is less number of chances for missing one of the vectors while generating the codebook. The codebook thus generated using the above technique can further be improved to give better performance in terms of PSNR using the K-Means Clustering (code optimization) technique.

5.5. Result and Discussion

Experiments were conducted on standard images Kush, Lena, Baboon, Pepper, Goldhill, and Boats. The images that are taken for the study are given in Figure 5.1.
Figure 5.1. The training sets taken to generate the codebook.

The PSNR (quality) of the reconstructed images and the time taken to generate the codebook using the CGEF (proposed) are compared with the results of LBG, KPE, KEFA, SCG and CGEF techniques in Table 5.1 and 5.2 for codebooks of sizes 128 and 256.
Table 5.1. Time (in seconds) to generate the codebook and the PSNR values of the reconstructed Images with the codebook of size 128.

<table>
<thead>
<tr>
<th>Image</th>
<th>Codebook Size : 128</th>
<th>LBG</th>
<th>KPE</th>
<th>KEFA</th>
<th>SCG</th>
<th>CGEF</th>
<th>OCGEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kush</td>
<td>Time</td>
<td>85.34</td>
<td>85.93</td>
<td>0.02</td>
<td>0.016</td>
<td>0.91</td>
<td>0.91+5.23</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>28.79</td>
<td>28.80</td>
<td>29.46</td>
<td>30.41</td>
<td>30.84</td>
<td>33.31</td>
</tr>
<tr>
<td>Lena</td>
<td>Time</td>
<td>87.37</td>
<td>88.96</td>
<td>0.02</td>
<td>0.016</td>
<td>0.91</td>
<td>0.91+5.27</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>27.21</td>
<td>27.27</td>
<td>28.68</td>
<td>29.24</td>
<td>30.09</td>
<td>32.38</td>
</tr>
<tr>
<td>Baboon</td>
<td>Time</td>
<td>84.87</td>
<td>83.37</td>
<td>0.02</td>
<td>0.015</td>
<td>0.81</td>
<td>0.81+5.00</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>21.07</td>
<td>21.26</td>
<td>22.00</td>
<td>33.12</td>
<td>33.42</td>
<td>36.06</td>
</tr>
<tr>
<td>Pepper</td>
<td>Time</td>
<td>85.10</td>
<td>85.04</td>
<td>0.02</td>
<td>0.016</td>
<td>0.89</td>
<td>0.89+5.30</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>26.36</td>
<td>26.38</td>
<td>27.65</td>
<td>30.78</td>
<td>31.06</td>
<td>35.03</td>
</tr>
<tr>
<td>GoldHill</td>
<td>Time</td>
<td>87.81</td>
<td>85.96</td>
<td>0.02</td>
<td>0.015</td>
<td>0.91</td>
<td>0.91+5.34</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>26.97</td>
<td>26.98</td>
<td>28.09</td>
<td>32.45</td>
<td>32.51</td>
<td>36.04</td>
</tr>
<tr>
<td>Boats</td>
<td>Time</td>
<td>98.31</td>
<td>96.75</td>
<td>0.02</td>
<td>0.015</td>
<td>0.83</td>
<td>0.83+5.23</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>24.99</td>
<td>25.17</td>
<td>25.80</td>
<td>26.40</td>
<td>27.03</td>
<td>29.45</td>
</tr>
<tr>
<td>Average</td>
<td>Time</td>
<td>88.13</td>
<td>87.67</td>
<td><strong>0.02</strong></td>
<td><strong>0.016</strong></td>
<td>0.88</td>
<td>0.88+5.23</td>
</tr>
<tr>
<td></td>
<td>PSNR</td>
<td>25.90</td>
<td>25.98</td>
<td>26.95</td>
<td>30.40</td>
<td><strong>30.83</strong></td>
<td><strong>33.71</strong></td>
</tr>
</tbody>
</table>

Table 5.2. Time (in seconds) to generate the codebook and the PSNR values of the reconstructed Images with the codebook of size 256.

<table>
<thead>
<tr>
<th>Image</th>
<th>Codebook Size : 256</th>
<th>LBG</th>
<th>KPE</th>
<th>KEFA</th>
<th>SCG</th>
<th>CGEF</th>
<th>OCGEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kush</td>
<td>102.59</td>
<td>104.95</td>
<td>0.02</td>
<td>0.016</td>
<td>0.80</td>
<td>0.80+</td>
<td>10.53</td>
</tr>
<tr>
<td></td>
<td>29.00</td>
<td>29.13</td>
<td>30.42</td>
<td>31.71</td>
<td>31.89</td>
<td>34.97</td>
<td></td>
</tr>
<tr>
<td>Lena</td>
<td>105.45</td>
<td>101.34</td>
<td>0.02</td>
<td>0.016</td>
<td>0.78</td>
<td>0.78+10.02</td>
<td>34.69</td>
</tr>
<tr>
<td></td>
<td>27.47</td>
<td>27.84</td>
<td>29.32</td>
<td>30.66</td>
<td>30.83</td>
<td>34.69</td>
<td></td>
</tr>
<tr>
<td>Baboon</td>
<td>100.25</td>
<td>101.92</td>
<td>0.02</td>
<td>0.015</td>
<td>0.84</td>
<td>0.84+10.72</td>
<td>37.69</td>
</tr>
<tr>
<td></td>
<td>21.33</td>
<td>21.83</td>
<td>22.65</td>
<td>34.36</td>
<td>34.66</td>
<td>37.69</td>
<td></td>
</tr>
<tr>
<td>Pepper</td>
<td>102.02</td>
<td>104.95</td>
<td>0.02</td>
<td>0.016</td>
<td>0.91</td>
<td>0.91+10.63</td>
<td>37.00</td>
</tr>
<tr>
<td></td>
<td>26.59</td>
<td>26.77</td>
<td>29.22</td>
<td>32.27</td>
<td>32.91</td>
<td>37.00</td>
<td></td>
</tr>
<tr>
<td>GoldHill</td>
<td>105.46</td>
<td>103.39</td>
<td>0.02</td>
<td>0.015</td>
<td>0.84</td>
<td>0.84+9.99</td>
<td>37.84</td>
</tr>
<tr>
<td></td>
<td>27.21</td>
<td>27.29</td>
<td>28.92</td>
<td>34.04</td>
<td>34.32</td>
<td>37.84</td>
<td></td>
</tr>
<tr>
<td>Boats</td>
<td>118.70</td>
<td>119.67</td>
<td>0.02</td>
<td>0.015</td>
<td>0.91</td>
<td>0.91+8.04</td>
<td>31.23</td>
</tr>
<tr>
<td></td>
<td>25.28</td>
<td>25.94</td>
<td>26.92</td>
<td>28.33</td>
<td>28.68</td>
<td>31.23</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>105.75</td>
<td>106.04</td>
<td><strong>0.02</strong></td>
<td><strong>0.016</strong></td>
<td>0.85</td>
<td>0.85+9.99</td>
<td><strong>35.57</strong></td>
</tr>
<tr>
<td></td>
<td>26.15</td>
<td>26.47</td>
<td>27.91</td>
<td>31.90</td>
<td><strong>32.22</strong></td>
<td><strong>35.57</strong></td>
<td></td>
</tr>
</tbody>
</table>
The column OCGEF in Table 5.1 represents the results obtained using the optimized codebook with edge features. The results are taken after 20 iterations for all images for codebook of sizes 128 and 256. From the results, it is observed that the time taken by KEFA and SCG are less (just 0.02 seconds) when compared to that of the other methods. The time taken by CGEF is also less than a second. But there is a significant improvement in PSNR values of the reconstructed images with respect to CGEF and OCGEF. The time taken by LBG, KPE, CGEF and OCGEF are 85.34, 85.94, 0.91 and 6.14 seconds respectively. The time taken by LBG and KPE methods is more or less 94 times more than that of CGEF (before optimization) method. After optimization, the time taken by LBG and KPE is 14 times more than that of OCGEF (optimized codebook with edge features). The PSNR of reconstructed images by LBG, KPE, CGEF and OCGE are 28.79, 28.80, 30.84 and 33.31 respectively for Kush image. There is a significant improvement in the quality of the reconstructed image and also in the time taken to generate the codebook. Hence this method outperforms LBG and KPE both in terms of time and PSNR values.

When compared to KEFA and SCG, the CGEF method has taken 0.88 seconds more and the OCGEF method has taken 5 seconds more. But the raise in PSNR value is 1 dB with CGEF and 3 dB with OCGEF. The PSNR is raised by a minimum of 3 decibels on an average for a codebook of size 128, which is a significant improvement. For a codebook of size 256, the PSNR of OCGEF is raised
by at least 3.5 decibels. The visual comparison of the reconstructed images with the above methods along with the PSNR values is given in Figure 5.2.

Figure 5.2. The PSNR values of the reconstructed images with respect to the LBG, KPE, KEFA, SCG and CGEF techniques for a codebook of size 256.

We carried out the experiments using the SCG and Codebook Generation using the Cluster Density (CGCD) techniques on standard images Lena, Boats and Baboon, each of size 256 x 256 pixels. The input images taken to generate the codebook using Cluster Density method are given in Figure 5.3.
Figure 5.3. The input images taken for the study

The PSNR values of the reconstructed images and the time taken for generating the codebook using the CGCD method are compared with the results of SCG method in Table 5.3.

<table>
<thead>
<tr>
<th>CB Size</th>
<th>Lena</th>
<th>Boats</th>
<th>Baboon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SCG</td>
<td>CGCD</td>
<td>SCG</td>
</tr>
<tr>
<td>128</td>
<td>PSNR</td>
<td>29.24</td>
<td>33.80</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>0.02</td>
<td>55.80</td>
</tr>
<tr>
<td>256</td>
<td>PSNR</td>
<td>30.66</td>
<td>34.82</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>0.02</td>
<td>54.97</td>
</tr>
<tr>
<td>512</td>
<td>PSNR</td>
<td>32.23</td>
<td>36.30</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>0.02</td>
<td>54.98</td>
</tr>
<tr>
<td>1024</td>
<td>PSNR</td>
<td>33.83</td>
<td>37.57</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>0.02</td>
<td>54.99</td>
</tr>
<tr>
<td>2048</td>
<td>PSNR</td>
<td>36.35</td>
<td>39.34</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>0.02</td>
<td>55.25</td>
</tr>
</tbody>
</table>

Codebooks of sizes 128, 256, 512, 1024 and 2048 are created using both the methods.

It is inferred clearly from Table 5.3, that the PSNR values of the reconstructed images
obtained using the CGCD method are improved significantly, i.e. in all the cases (with different codebook sizes), the PSNR is uniformly increased by 4 dB.

From Table 5.3, it is observed that the time taken by the CGCD method is more when compared with that of the SCG method. But the quality of the reconstructed images obtained with the proposed method is significantly better than that of SCG. On an average, the PSNR of the SCG method is increased by 4.49 with the proposed method. The average PSNR values obtained with both the techniques (SCG and CGCD) are 32.87 and 37.37 respectively.

**Figure-5.4.** Average PSNR with respect to SCG and the CGCD methods for different codebook sizes.
The performance of CGCD is compared with that of the SCG method in terms of PSNR values for different codebook sizes in Figure 5.4. It is further observed that our proposed method, on an average yields 12.04% increase in the PSNR value obtained by the existing SCG method. The proposed algorithm gives 16.87%, 14.98%, 14.39%, 13.20%, 9.80% and 13.67% increase in PSNR values for the codebooks of sizes 124, 256, 512, 1024 and 2048 respectively.

Experiments were conducted with Codebook Generation by Sorting the Sum of Sub Vectors (CGSSSV) method over standard images: Lena, Bridge and Cameraman of 256 x 256 pixels. The performance of the SCG, PNN and CGSSSV (proposed) are compared in terms of time taken to generate the codebook and the PSNR values (quality of the reconstructed image) in Table 5.4.
From the Table 5.4, it is observed that the SCG and the CGSSSV methods take less time when compared to that of PNN method. The time taken by SCG and CGSSV to generate is almost a fraction of second, which is negligible when compared with that of the PNN. For Lena image, for a codebook of size 1024, PNN takes 9551 seconds, where as SCG and CGSSV take only 0.02 and 0.34 seconds respectively. This is a significant reduction in time. Optimizing the codebook generated by the proposed method takes only a maximum of 48.31 seconds to generate the codebook but with significant improvement in the PSNR value.
The time given in the last column of Table 5.4 is the sum of the time taken for generating the initial codebook by the proposed method and the time taken to optimize the codebook. The time taken by the proposed method to generate the initial codebook is almost same (0.33 seconds) and the time taken to optimize the codebook increases as the size of the codebook increases. As far as the PSNR values are concerned, the proposed method along with the codebook optimization gives better results. Table-1 reveals that the PSNR values obtained with the codebook generated by the proposed method is better in almost all cases (boldfaced). For the image Camera with the codebook of size 1024, the PSNR value obtained is 35.44 which is better when compared to other methods. This is a rare case. The reconstructed images obtained out of the three methods are shown in Figure 5.6.

(a) Cameraman  (b) Bridge  (c) Lena

Figure 5.5. Training sets taken for the study
(a) PNN  
PSNR : 26.80

(b) SCG  
PSNR: 30.25

(c) CGSSV  
PSNR: 31.26

Figure-5.6. Visual comparison of images reconstructed using the three methods

5.6. Conclusion

In this chapter, we present an adaptive codebook generation technique (CGEF) in which preference is given to edge details as the clear edges give smooth appearance to the images. The training set includes the images Kush, Lena, Baboon, Pepper, GoldHill and Boats of size 256 x 256. The existing techniques LBG, KPE, KEFA and the proposed method CGEF are tried with codebooks of sizes 128 and 256. With the codebooks of sizes 128 and 256, the method CGEF gives better results. The codebook generated using CGEF, when optimized (OCGEF) gives better results in terms of the quality of the reconstructed image. That is the PSNR value of the reconstructed images is significantly improved by at least 3 decibels. The computational complexity involved in this method is less and the method is simple to implement. We felt difficult to set the threshold value for identifying the edge blocks. We tried with different values and whichever value gave better result, we set that value as a threshold. If the same may be generalized, it would be a better approach.
A new codebook initialization algorithm CGCD based on the cluster density is proposed. It is observed from the results that the CGCD method gives better PSNR values. Generalized Lloyd Algorithm is the most widely used method for codebook generation for vector quantization. GLA needs initial codebook which is to be optimized by it. The CGCD method is one such technique to generate the initial codebook for GLA by considering the density of the clusters. From the results generated, it is observed that the quality of the reconstructed images obtained with the CGCD method is better when compared to that of one other codebook initialization method SCG. The computational complexity involved is also less.

The third proposed method called the CGSSSV generates the codebook efficiently in terms of the time taken to generate the codebook and the quality (PSNR) of the reconstructed images. It takes less time to generate the codebook when compared to that of the existing hierarchical method (PNN) for generating the codebook. Codebooks of sizes 128, 256, 512 and 1024 are generated to measure the performance of the proposed method in terms of time and quality of the reconstructed images. In all runs, the experimental results confirm that the proposed method gives better performance. The PSNR obtained by the proposed method is further improved by optimizing the codebook generated using the iterative clustering method. The computational complexity involved in this method is less and hence it is easy to implement this algorithm. We have implemented this method for gray level images. This can also be tried for color images.