Convergence Analysis of Codebook Generation Techniques for Vector Quantization using K-Means Clustering Technique

S.Vimala
Department of Computer Science
Mother Teresa Women's University
Kodaikanal – 624 102
Tamil Nadu, India

ABSTRACT
Vector Quantization (VQ) is one of the lossy image compression techniques. VQ comprises of three different phases: Codebook Generation, Image Encoding and Image Decoding. The performance of VQ is mainly based on the codebook generation phase. In this paper, five different codebook generation techniques namely the Simple Codebook Generation (SCG), Ordered Codebook Generation (OCG), Codebook Generation by Sorting the Sum of Sib Vectors (CBSSSV), Codebook Generation with Edge Features (CBEF) and Codebook Generation with Cluster Density (CBCD) for Vector Quantization have been discussed and their performance in terms of number of iterations required to converge with respect to Peak Signal to Noise Ratio (PSNR) is compared when k-Means Clustering technique is used to optimize the initial codebook that is created by any of the above techniques. Of these discussed techniques, the CBEF technique performs better.

General Terms
Vector Quantization, Image Compression

Keywords
Compression, codevector, training vector, clustering, MSE.

1. INTRODUCTION
Image compression is essential for applications such as TV transmission, video conferencing, facsimile transmission of printed material, graphics images, or transmission of sensing images obtained from satellites and reconnaissance aircraft [1]. Image compression techniques deal with the reduction of data required to represent images. Compression of images also reduces the time required for images to be sent over the Internet. The image compression techniques are generally classified into two major types namely, the lossy compression techniques and lossless compression techniques.

Vector Quantization is one of the lossy image compression techniques. It is theoretically proved that VQ is more efficient than scalar quantization [2]. The VQ algorithms for reducing the transmission bit rate or the storage have recently been extensively investigated for speech and image signals [3] and [4].

The design of an efficient VQ encoder involves global codebook generation by selecting a good clustering algorithm and using appropriate features extracted from the training data set [5]. VQ has been successfully used in various applications involving VQ-based encoding and recognition [6]. 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. VQ comprises of three stages: 1. Codebook Generation, 2. Image Encoding and 3. Image Decoding. Codebook Generation is the key component of VQ. The performance of the VQ mainly depends on the quality of the codebook. LBG (Linde, Buzo, Gray) [7] is the most widely referred VQ method for designing a codebook. There are several known methods for generating a codebook.

In normal VQ, the input image is divided in to small blocks of size 4 x 4 pixels. These blocks are converted into vectors of size k-dimension (k = 4 x 4). These vectors are called training vectors and the set of training vectors is called training set of size N vectors. N is computed using the equation (1).

$$N = (m \times m)/16 \quad (1)$$

where 16 is the size of the vector. A codebook of size M (M<N) is generated by selecting M codevectors from the training set. These codevectors act as the representative vector for the whole training set. In image encoding, all the input blocks are compared against the codevectors and whichever codevector is closest to the input block, the corresponding index is stored or transmitted. In the decoding stage, the corresponding codevector of the index is coded to get the reconstructed image. The difference between the input image and the reconstructed image is the Mean Square Error (MSE) and is computed using the equation (2). The Peak Signal to Noise Ratio is the inverse of MSE and gives the quality of the reconstructed image. PSNR is computed using the equation (3).

$$MSE = \sum_{y=1}^{M}\sum_{x=1}^{N}\left[I(x,y) - I'(x,y)\right]^2 \quad (2)$$

$$PSNR = 20 * \log_{10}\left(\frac{255}{sqrt(MSE)}\right) \quad (3)$$

The efficiency of VQ mainly depends on the quality of the codebook. There are various techniques existing to generate initial codebooks for VQ. The performance of the codebook generation techniques is measured in terms of time taken to generate the codebook and the quality of the reconstructed images using the codebooks. When MSE is less, the quality is high. The initial codebook thus generated using any of the existing techniques is optimized using Generalized Lloyd Algorithm (GLA) otherwise called k-Means Clustering algorithm. When optimized, the MSE obtained with the initial codebook will further be reduced. The initial MSE and the improved MSE of all the discussed methods are compared.

The remaining paper is organized as follows: In section 2, various codebook generation methods are discussed. In section 4, the k-Means clustering is explained and the initial codebooks that are generated using the methods discussed in section 3 are improved using k-Means clustering method and the results obtained are discussed in Section 4. The conclusion is given in Section 5 and the references are given in section 6.

2. THE CODEBOOK GENERATION METHODS

2.1. Simple Codebook Generation

In simple codebook generation, the training vectors at every nth position are selected to form the codebook. The value of ***n*** is computed using the equation (4).

$$n = N/M \quad (4)$$

where, M is the size of the desired codebook to be generated. In SCG, the codevectors are selected randomly. Hence there are chances for more than one codevector to be closer.

2.2. Ordered Codebook Generation (OCG)

An enhanced form of SCG is the ordered codebook generation technique [8]. In this method, the training vectors are sorted in ascending order based on the sum of the components (magnitude) of the vectors. The magnitude of the vector is computed using the equation (5).

$$S_i = \sum_{j=1}^{k} x_j \quad (5)$$

where 1 <= i <= N, and x_j is the j^{th} component of the vector. After sorting, the training vectors at every nth position are selected to form the codebook. In this method, the training vectors are uniformly distributed and there are no chances for the closest codevectors to occur in the codebook and improves the quality of the codebook.

2.3. Codebook Generation by Sorting the Sum of Sub Vectors (CBSSSV)

CBSSSV method [9] is an enhancement done to the OCG method. When the training vectors are sorted based on their magnitudes as in OCG, there are chances for different training vectors to give same magnitude.

For example, we have taken two training 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} from the image cameraman. If we compute the magnitude for both the vectors using the equation (4), 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. As in OCG, if we select the training vectors for codebook from every nth 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. Now the sum values of the above two vectors will be

539-606+662-663= -68

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. Hence there is less number of chances for missing one of the vectors while generating the codebook.

2.4. Codebook Generation with Cluster Density

In this method [10], the number of closest codevectors for each training vector is identified and is stored as the corresponding cluster density. The cluster densities for all training vectors are computed and are sorted in descending order. From the sorted list, the top M training vectors with higher cluster densities are identified and grouped as codebook. The closest codevector is identified by calculating the distance between the codevector and the training vector using the equation (6).

$$d(X, Y_i) = \sum_{j=1}^{k}\left(X_j - Y_{ij}\right), \quad 1<=i<=N \quad (6)$$

where d(X, Yi) is the minimum, X being the current vector and Y_i refers to all other vectors.

2.5. Codebook Generation with Edge Features (CBEF)

With the above methods, the reconstructed images have ragged edges. To overcome this problem, in CBEF method [11], the training vectors are classified into two categories namely the edge blocks and shade blocks. The high detail blocks are called edge blocks and the low detail blocks are called the shade blocks. To identify whether a block is a high detail one or low detail one, the mean $\bar{x}$ is computed using the equation (8).

$$\bar{x} = \frac{1}{k}\sum_{i=1}^{k} x_i \qquad (7)$$

The sum of the difference between the individual components and the mean is computed using the equation (9).

$$SD = \sum_{i=1}^{k}\left(x_i - \bar{x}\right) \qquad (8)$$

The value SD is compared against a threshold value and if greater, the block is a high detail block or an edge block. If less, it is a low detail block or shade block. After classifying all the blocks into edge or low detail blocks, the codebook is first populated with the edge blocks, and then to fill the gap, the shade blocks are selected in random. This method enables to have smooth edges. The codebook thus generated using the above techniques can be optimized to give better performance in terms of MSE using the k-Means Clustering technique.

3. K-MEANS CLUSTERING TECHNIQUE

This technique is used to optimize the codebook that is initially created by any one of the above said techniques. The N training vectors are grouped into M clusters with the initial codevectors as the centroids of all the clusters. Pick up the centroid Yi(k), of any cluster. Find all the image blocks Xi that are closer to Yi than to any other Yj. i.e. find the set of all Xi (training vectors) that satisfy:

$$d(X_i, Y_i) < d(X_i, Y_j) \text{ for all } j \neq i, \qquad (9)$$

where the distance between the training vector Xi and the codevector Yi is computed as

$$dy = \sum_{j=1}^{k}\left|X_j - Y_{ij}\right| \qquad (10)$$

Calculate the sum vector by adding all the training vectors Xi that are closer to Yi. The individual component of the sum vector is calculated by adding the corresponding components of all the training vectors of the same cluster as:

$$Sum_{ij} = \sum_{i=1}^{c} X_{ij} \qquad (11)$$

where, c is the cluster strength. Now every component is divided by the cluster strength c to get the new centroid of the cluster.

$$\text{Centroid} = Sum_{ij} / ni, \quad \text{where } i=1,2,\ldots.n \qquad (12)$$

The codevector Yi is replaced with the newly generated centroid to form the refined codebook. These steps are repeated till the codebooks of consecutive iterations converge.

3.1. k-Means Clustering Algorithm

Step1: The training vectors are grouped into M clusters based on the distance between the codevectors and the training vectors using the equation (10) and (11).

Step2: Compute the sum vector for every cluster by adding the corresponding components of all the training vectors that belong to the same cluster using the equation (12).

Step3: Compute the centroid for each cluster by dividing the individual components of the sum vector by the cluster strength n_i using the equation (12).

Step4: Replace the existing codevector with the new centroid to form the revised codebook.

Step5: Repeat the steps 1 through 4 till the codebooks of the consecutive iterations converge.

4. RESULTS AND DISCUSSION

The algorithms are implemented using Matlab 7.0 on Windows Operating System. The hardware used is the Intel Core 2 Duo E7400@ 2.8 GHz Processor with 2 GB RAM. The experiments were conducted with standard images Baboon, Barbara, Boats, Bridge, Camera and Lena of size 256 x 256 pixels.

Tables 1, 2, 3, 4 and 5 show the comparison of PSNR, the time taken for generating the initial codebook, the time taken for optimizing the codebook, the improved PSNR after optimization and the number of iterations required for the convergence of codebook during optimization for different images with different codebook sizes 128, 256 and 512.

The terms used in the following tables refer to

1. initTime - Time taken to generate the initial codebook
2. initPSNR - Quality of the image reconstructed with the initial codebook
3. impPSNR - Improved PSNR with the optimized codebook

Table 1: Comparison of PSNR and the number of iterations required for optimization by SCG method for different codebook sizes 128, 256 and 512.

CB Size	Image	Baboon	Barbara	Boats	Bridge	Camera	Lena	Average
128	**initTime**	0.00	0.00	0.00	0.015	0.015	0.016	0.01
	initPSNR	33.12	32.59	26.4	26.02	26.15	29.24	28.92
	Time	21.42	51.86	14.89	11.66	27.16	59.00	31.00
	ImpPSNR	36.11	35.97	30.35	28.12	29.19	32.94	32.11
	Iterations	103	248	72	56	131	284	149

256	**initTime**	0.02	0.00	0.02	0.00	0.00	0.02	0.01
	initPSNR	34.36	34.31	28.33	26.90	26.80	30.66	30.23
	Time	83.56	103.05	60.16	24.14	75.59	203.25	91.63
	ImpPSNR	37.88	37.76	31.69	29.48	31.79	34.68	33.88
	Iterations	201	248	145	58	182	489	220
512	**initTime**	0.00	0.02	0.02	0.02	0.02	0.02	0.01
	initPSNR	36.26	35.96	29.56	27.98	28.00	32.23	31.67
	Time	77.44	349.94	88.00	40.86	328.69	389.23	212.36
	ImpPSNR	39.68	40.02	33.55	31.17	33.15	36.86	35.74
	Iterations	92	417	105	48	392	464	253
Avg.	**initTime**	0.01	0.01	0.01	0.01	0.01	0.02	0.01
	initPSNR	34.58	34.29	28.10	26.97	26.98	30.71	30.27
	Time	60.81	168.28	54.35	25.55	143.81	217.16	111.66
	ImpPSNR	37.89	37.92	31.86	29.59	31.38	34.83	33.91
	Iterations	132	304	107	54	235	412	208

Table 2: Comparison of PSNR and the number of iterations required for optimization by OCG method for different codebook sizes 128, 256 and 512.

CB Size	Image	Baboon	Barbara	Boats	Bridge	Camera	Lena	Average
128	**initTime**	0.27	0.31	0.33	0.31	0.34	0.31	0.31
	initPSNR	33.88	33.51	28.39	25.96	26.15	30.43	29.72
	Time	15.42	62.2	11.25	15.42	24.36	43.50	28.69
	ImpPSNR	35.94	36.07	30.72	28.33	29.94	33.05	32.34
	Iterations	74	299	54	74	117	209	138
256	**initTime**	0.28	0.31	0.33	0.31	0.34	0.31	0.31
	initPSNR	35.01	34.60	29.21	27.01	26.80	31.37	30.67
	Time	65.61	150.33	46.81	43.84	51.56	133.66	81.97
	ImpPSNR	37.44	37.97	32.06	29.56	31.52	34.73	33.88
	Iterations	157	360	112	105	124	321	197
512	**initTime**	0.27	0.31	0.33	0.31	0.34	0.31	0.31
	initPSNR	36.09	36.20	30.08	27.75	28.06	32.35	31.76
	Time	79.77	559.64	147.11	27.75	204.25	855.27	312.30
	ImpPSNR	39.50	39.96	33.88	31.16	33.05	36.25	35.63
	Iterations	95	666	175	33	244	1020	372
Average	**initTime**	0.27	0.31	0.33	0.31	0.34	0.31	0.31
	initPSNR	34.99	34.77	29.23	26.91	27.00	31.38	30.71
	Time	53.60	257.39	68.39	29.00	93.39	344.14	140.99
	ImpPSNR	37.63	38.00	32.22	29.68	31.50	34.68	33.95
	Iterations	109	442	114	71	162	517	236

Table 3: Comparison of PSNR and the number of iterations required for optimization by CBSSSV method for different codebook sizes 128, 256 and 512.

CB Size	Image	Baboon	Barbara	Boats	Bridge	Camera	Lena	Average
128	initTime	0.30	0.30	0.30	0.30	0.31	0.31	0.30
	initPSNR	33.74	33.24	28.26	26.09	26.27	30.19	29.63
	Time	15.53	50.73	23.08	7.50	28.64	25.92	25.23
	ImpPSNR	35.90	36.12	30.63	28.29	29.95	32.93	32.30
	Iterations	75	244	111	36	138	125	122
256	initTime	0.31	0.30	0.30	0.31	0.31	0.31	0.31
	initPSNR	34.95	34.66	29.18	27.03	27.23	31.33	30.73
	Time	50.84	134.95	74.03	11.69	98.45	180.11	91.68
	ImpPSNR	37.74	37.89	31.97	29.55	31.29	34.63	33.85
	Iterations	122	324	178	28	237	433	220
512	initTime	0.30	0.30	0.31	0.31	0.31	0.31	0.31
	initPSNR	36.29	36.32	30.02	28.01	28.32	32.45	31.90
	Time	37.13	717.59	100.63	50.42	20.8.47	771.44	279.54
	ImpPSNR	39.62	39.86	33.78	31.18	33.15	36.20	35.63
	Iterations	44	856	120	60	248	921	375
Average	initTime	0.30	0.30	0.30	0.31	0.31	0.31	0.31
	initPSNR	34.99	34.74	29.15	27.04	27.27	31.32	30.75
	Time	34.50	301.09	65.91	23.20	42.36	325.82	132.15
	ImpPSNR	37.75	37.96	32.13	29.67	31.46	34.59	33.93
	Iterations	80	474	136	41	207	493	239

Table 4: Comparison of PSNR and the number of iterations required for optimization by CBCD method for different codebook sizes 128, 256 and 512.

CB Size	Image	Baboon	Barbara	Boats	Bridge	Camera	Lena	Average
128	initTime	0.63	0.63	0.63	0.63	0.63	0.64	0.63
	initPSNR	33.12	32.59	27.69	26.08	25.51	30.27	29.21
	Time	21.39	51.52	21.80	13.33	18.70	21.34	24.68
	impPSNR	36.11	35.97	30.47	28.38	28.58	32.91	32.07
	Iterations	103	248	105	64	91	103	119
256	initTime	0.63	0.61	0.63	0.63	0.63	0.66	0.63
	initPSNR	34.36	34.31	28.54	26.75	26.16	30.88	30.17
	Time	83.74	103.08	44.86	22.84	51.94	148.47	75.82
	ImpPSNR	37.88	37.76	31.89	29.33	30.49	34.51	33.64
	Iterations	201	248	108	55	126	358	183
512	initTime	0.61	0.63	0.63	0.61	0.63	0.63	0.62
	initPSNR	36.26	35.96	29.62	27.28	26.96	31.60	31.28
	Time	77.27	349.47	191.00	30.92	244.64	363.41	209.45
	ImpPSNR	39.68	40.02	33.13	30.65	32.25	36.10	35.31
	Iterations	92	417	228.00	37	292	433	250

Average	**initTime**	0.62	0.62	0.62	0.63	0.63	0.64	0.63
	initPSNR	34.58	34.29	27.84	26.60	26.21	30.92	30.07
	Time	60.80	168.02	32.53	93.60	105.09	177.74	106.30
	impPSNR	37.89	37.92	31.00	29.99	30.44	34.51	33.62
	Iterations	132	304	83	137	170	298	187

Table 5: Comparison of PSNR and the number of iterations required for optimization by CBEF method for different codebook sizes 128, 256 and 512.

CB Size	Image	Baboon	Barbara	Boats	Bridge	Camera	Lena	Average
128	**initTime**	0.63	0.63	0.63	0.63	0.63	0.64	0.63
	initPSNR	33.12	32.59	27.69	26.08	25.51	30.27	29.21
	Time	21.39	51.52	21.80	13.33	18.70	21.34	24.68
	impPSNR	36.11	35.97	30.47	28.38	28.58	32.91	32.07
	Iterations	103	248	105	64	91	103	119
256	**initTime**	0.63	0.61	0.63	0.63	0.63	0.66	0.63
	initPSNR	34.36	34.31	28.54	26.75	26.16	30.88	30.17
	Time	83.74	103.08	44.86	22.84	51.94	148.47	75.82
	ImpPSNR	37.88	37.76	31.89	29.33	30.49	34.51	33.64
	Iterations	201	248	108	55	126	358	183
512	**initTime**	0.61	0.63	0.63	0.61	0.63	0.63	0.62
	initPSNR	36.26	35.96	29.62	27.28	26.96	31.60	31.28
	Time	77.27	349.47	191.00	30.92	244.64	363.41	209.45
	ImpPSNR	39.68	40.02	33.13	30.65	32.25	36.10	35.31
	Iterations	92	417	228.00	37	292	433	250
Average	**initTime**	0.62	0.62	0.62	0.63	0.63	0.64	0.63
	initPSNR	34.58	34.29	27.84	26.60	26.21	30.92	30.07
	Time	60.80	168.02	32.53	93.60	105.09	177.74	106.30
	impPSNR	37.89	37.92	31.00	29.99	30.44	34.51	33.62
	Iterations	132	304	83	137	170	298	187

Table 6: Comparison of the average time, iterations and the PSNR with respect to the methods SCG, OCG, CBSSSV, CBEF and CBCD

Techniques	SCG	OCG	CBSSSV	CBEF	CBCD
initTime	0.01	0.31	0.31	0.63	16.65
initPSNR	30.27	30.71	30.75	30.07	30.61
Time	111.66	140.99	132.15	106.30	125.94
impPSNR	33.91	33.95	33.93	33.62	33.64
Iterations	208	236	239	187	221

In Tables 1 thru. 5, the results obtained with all the techniques are presented in separate tables. The average values are presented in Table 6. As far as the initial codebook is concerned, the OCG method performs better in terms of time taken to generate the codebook. It takes on an average of 0.01 seconds.

The OCG and CBSSSV methods take o.31 seconds on an average. The CBEF method takes 0.63 seconds and the CBCD method takes more time, i.e. 16.65 seconds. When compared to CBCD method, the time taken by SCG method is negligible.

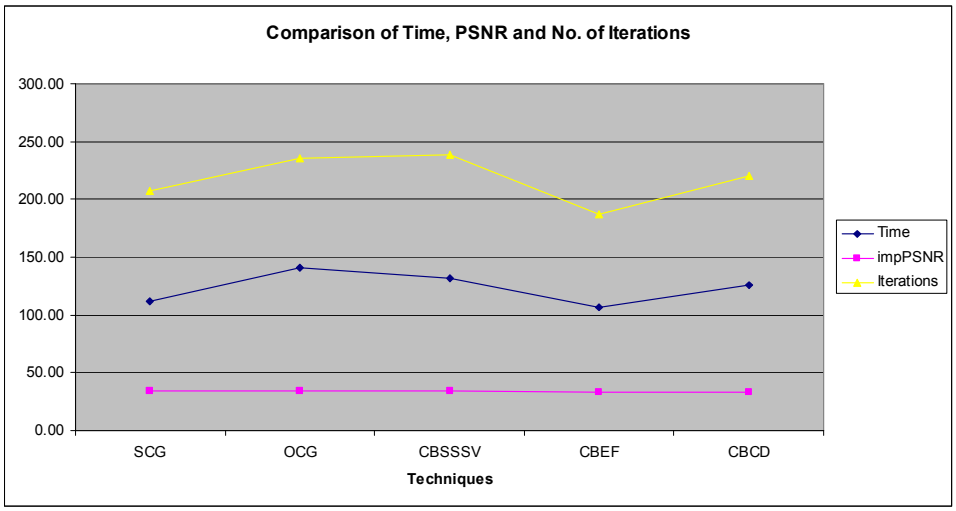

Fig.1. Average variation in PSNR, Time and number of iterations required towards convergence of codebook.

The initial PSNR is almost the same for all methods. The average difference n PSNR is 0.27. The average times taken by k-Means Clustering method to optimize the initial codebooks that are generated by all techniques are given in Table 6. The time taken by CBEF method is the minimum which is 106.30. The highest value is 140.99 and is taken by OCG method. The number of iterations is minimum (187) in case of CBEF and maximum (239) in case of CBSSSV.

When we look at the graph given in Fig.1, the average PSNR obtained with the entire techniques lie on same line. But there is a significant variation in the time taken to optimize the codebook and the number of iterations required by all the methods. CBEF gives better results in terms of time to optimize the codebook and number of iterations required for convergence.

The input images of size 256 x 256 pixels and of varying gray shades taken for the study are given in Fig. 2.

(a) Baboon

(b) Barbara

(c) Boats

(d) Bridge

(e) Cameraman

(f) Lena

Fig. 2. Input images taken for the study.

5. CONCLUSION

We have discussed five different techniques for generating the initial codebook that form the initial seeds for the k-Means clustering algorithm. k-Means clustering technique is an optimizing tool for improving the quality of the codebooks that are generated for VQ. We experimented the performance of all the methods using the k-Means clustering in terms of i. time taken for optimization, ii. the improved PSNR (quality of the reconstructed images) and iii. the number of iterations needed for the convergence of codebooks of consecutive iterations. The Codebook Generation with Edge Features (CBEF) gives better results both in terms of time and the number of iterations. But all methods give more or less same PSNR values. Hence CBEF is considered to be the best of all methods. The experiments are conducted with gray scale images. As the gray scale images form the base for color images, the above techniques can also be used for color images. These techniques can be used imaging applications in hand-held devices.

6. REFERENCES

[1] Nasser M.Nasrabadi, "Image Coding using Vector Quantization: A Review", IEEE Transactions on Communications, Vol. 36, No. 8, August 1988.

[2] Berger T, "Rate Distortion Theory", Englewood Cliffs, Prentice-Hall,NJ, 1971.

[3] A.Gersho and V.Cuperman, "Vector Quantization: A Pattern Matching Technique for Speech Coding", IEEE Communications, Mag., pp 15-21, 1983.

[4] R.M.Gray, "Vector Quantization", IEEE ASSP Mag., pp. 4-29, Apr., 1984.

[5] Shuyu Yang, Sunanda Mitra, "Content Based Vector Coder for Efficient Information Retrieval", Dept. of Electrical and Computer Engineering, Texas Tech University, USA.

[6] H.B.Kekre, Tanuja K.Sarode, "Vector Quantized Codebook Optimization using k-Means", International Journal on Computer Science and Engineering, Vol. 1, No. 3, pp. 283-290, 2009.

[7] Gersho and R.M.Gray, Vector Quantization and Signal Compression, Dordrecht, The Netherlands: Kluwer, 1992.

[8] K.Somasundaram and S.Vimala, "Simple and Fast Ordered Codebook Generation for Vector Quantization," Proceedings of the National Conference on Image Processing, Gandhigram Rural Institute, Allied Publishers, India. ISBN 978-81-8424-574-5, Mar 2010.

[9] K.Somasundaram, S.Vimala, "Codebook Generation by Sorting the Sum of Sub Vectors", CiiT, International Journal of Digital Image Processing, August 2010.

[10] K.Somasundaram, S.Vimala, "A Novel Codebook Initialization Technique for Generalized Lloyd Algorithm using Cluster Density", International Journal on Computer Science and Engineering, Vol. 2, No. 5, pp. 1807-1809, 2010.

[11] K.Somasundaram, S.Vimala, "Codebook Generation for Vector Quantization with Edge Features", CiiT International Journal of Digital Image Processing, Vol. 2, No.7, pp. 194-198, 2010.