Journal of Information Hiding and Multimedia Signal Processing ©2018 ISSN 2073-4212
Ubiquitous International Volume **9**, Number **1**, January **2018**

Optimal Codebook Design for Image Vector Quantization Based on Edge-classified Norm-ordered K-means Algorithm

Lang Wang[1], Zhe-Ming Lu[2*], Long-Hua Ma[1]

[1]School of Information Science and Engineering
Ningbo Institute of Technology, Zhejiang University
Ningbo, 315100, China
langwang1980@aliyun.com

[2]School of Aeronautics and Astronautics
Zhejiang University
Hangzhou, 310027, P. R. China
zheminglu@zju.edu.cn

Received January, 2017; revised July, 2017

Abstract. *Vector Quantization (VQ) is a widely used data compression technique for its simple decoding structure. Codebook design is the key step before using VQ in data compression. The K-means algorithm is a popular codebook design scheme for its relative simplicity. However, the initial codebook greatly affects the convergence speed and the quality of the generated codebook. Many algorithms have been proposed to improve the initial codebook, but most of them do not make full use of the features of the training vectors and some of them are with high extra computation load. This Letter presents a simple and efficient initialization technique based on edge classification and norm sorted grouping. First, we classify the training vectors into 8 subsets based on the edge classifier, and the number of initial codewords generated from each subset is proportional to the number of training vectors in this subset. Then, we calculate and sort the norms of the training vectors in each subset, and divide the training vectors in this subset into several equal-sized groups with the number of groups equaling the number of codewords to be generated from the subset. Each group generates one initial codeword which is just the centroid of the training vectors in the group. Experimental results show that, compared with the random selection and the existing norm-sorted grouping strategies, our initialization technique generates a better codebook with a faster speed.*
Keywords: Vector quantization, Codebook design, K-means algorithm, Initial codebook generation, Optimization.

1. **Introduction.** Vector quantization [1] has been successfully used in data compression [2] and data clustering [3]. A vector quantizer Q can be defined as a mapping from the n-dimensional Euclidean space R^n into a codebook C with K n-dimensional codewords, i.e.,

$$Q : R^n \to C = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_K\} \tag{1}$$

where $\boldsymbol{y}_i = \{y_{i1}, y_{i2}, ..., y_{in}\}^{\mathrm{T}}$. This quantization should be based on certain distortion metric $d(\boldsymbol{a}, \boldsymbol{b})$ that measures the dissimilarity between two n-dimensional vectors. The squared error $d(\boldsymbol{a}, \boldsymbol{b}) = \|\boldsymbol{a} - \boldsymbol{b}\|^2$ is often adopted as a distortion measure. Based on the

quantizer Q, the space R^n can be divided into K non-overlapping cells $V_i, i = 1, 2, ..., K$, satisfying

$$V_i = \{\boldsymbol{v} \in R^n | d(\boldsymbol{v}, \boldsymbol{y}_i) \leq d(\boldsymbol{v}, \boldsymbol{y}_j), j = 1, 2, ..., K\}$$
$$R^n = \bigcup_{i=1}^{K} V_i, V_i \cap V_j = \Phi(i \neq j) \tag{2}$$

In general, the codebook of Q is designed from a training set $X = \{x_1, x_2, \ldots, x_M\}$ rather than R^n, then we can quantify the performance of Q by the following average distortion

$$D = E[d(\cdot, Q(\cdot))] = \frac{1}{M} \sum_{i=1}^{M} \|\boldsymbol{x}_i - Q(\boldsymbol{x}_i)\|^2 \tag{3}$$

Obviously, the compression performance of VQ is mainly dependent on the codebook quality. In fact, the aim of optimal vector quantizer design is to find the codebook that minimizes the average distortion over all possible codebooks. It is well-known that an optimal vector quantizer should satisfy two optimal conditions, i.e., the nearest neighbor condition and the best codebook condition. On the one hand, each training vector should be mapped to the codeword that is closest to it. On the other hand, each codeword should be the centroid of the training vectors that are mapped to it. The traditional K-means algorithm just uses these two conditions iteratively to design a locally optimal codebook. Sometimes, we also call it the generalized Lloyd algorithm (GLA) or the Linde-Buzo-Gray (LBG) algorithm [4].

The traditional K-means algorithm often converges to a locally optimal codebook, and both the convergence speed and the performance of the generated codebook depend on the initial codebook. Thus, some researchers [5] have been focusing their attention on enhancing the performance of GLA so as to improve either the quality based on population-based meta-heuristics schemes [6, 7] or the speed of VQ based on more efficient codebook structures [8] or codeword reduction methods [9, 10, 11]. Other researchers aim to generating a better initial codebook, and in fact a good initial codebook generation method can be combined with any above-mentioned technique to further improve the performance of VQ. In the past, many algorithms have been proposed to obtain a good initial codebook, including the well known random initialization, splitting [4], pairwise nearest neighbor (PNN) design [12], maximum distance initialization [13], and the norm-ordered grouping strategy [14]. However, most conventional initialization techniques do not make full use of the features of each training vector, and some initialization techniques need high extra computation load, such as splitting [4], PNN[12] and maximum distance initialization [13]. Therefore, in this Letter, we propose a simple and efficient initialization scheme for the K-means algorithm used in image vector quantization by first classifying the training vectors into 8 classes by a edge classifier and then sort the training vectors in each class according to their norm values. Afterwards, the sorted training vectors in each class are grouped evenly. Finally, the centroid of each group is adopted as the initial codeword. Experimental results demonstrate that, our scheme can get a better codebook quality than random initialization and the existing norm-sorted grouping strategy with a faster speed.

2. **Related Work.**

2.1. **Conventional and Modified K-means Algorithms.**

Given a training set $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_M\}$ of size M, to design a codebook $C = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_K\}$ of size K, the conventional K-means algorithm [4] can be described as follows.

Step 1. Set the iteration number $m = 0$, initialize the codebook at iteration $m = 0, C^{(0)} = \{\boldsymbol{y}_1^{(0)}, \boldsymbol{y}_2^{(0)}, \ldots, \boldsymbol{y}_K^{(0)}\}$, and set the convergence threshold e.

Step 2. Based on the nearest-neighbor condition, obtain the partitions $V_j^{(m)} = \{\boldsymbol{v} \in X | Q^{(m)}(\boldsymbol{v}) = \boldsymbol{y}_j^{(m)}\}, j = 1, 2, \ldots, K$. Here, $Q^{(m)}$ denotes the vector quantizer defined as: $Q^{(m)}(\boldsymbol{v}) = \boldsymbol{y}_j^{(m)}$ if $d(\boldsymbol{v}, \boldsymbol{y}_j^{(m)}) \leq d(\boldsymbol{v}, \boldsymbol{y}_i^{(m)}), i = 1, 2, \ldots, K$.

Step 3. Based on the centroid condition, update the codebook $C^{(m)} = \{\boldsymbol{y}_j^{(m)}, j = 1, 2, \ldots, K\}$ to $C^{(m+1)} = \{\boldsymbol{y}_j^{(m+1)}, j = 1, 2, \ldots, K\}$ by

$$\boldsymbol{y}_j^{(m+1)} = \frac{1}{|V_j^{(m)}|} \sum_{\boldsymbol{v} \in V_j^{(m)}} \boldsymbol{v} \tag{4}$$

where $|V_j^{(m)}|$ stands for the operation to get the number of elements in $V_j^{(m)}$.

Step 4. Terminate the algorithm if the relative improvement $|d_{m+1} - d_m|/d_{m+1} \leq e$, where d_{m+1} is the average distortion after $m + 1$ iterations as follows

$$d_{(m+1)} = \frac{1}{M} \sum_{i=1}^{M} \|\boldsymbol{x}_i - Q^{(m+1)}(\boldsymbol{x}_i)\|^2 \tag{5}$$

Otherwise, replace m by $m + 1$ and go to Step 2.

To speed up the K-means algorithm, Lee et al. [15] proposed a modified K-means algorithm which is almost the same as the conventional K-means algorithm except for a modification at the codebook updating step. They update the current codeword $\boldsymbol{y}_j^{(m)}$ at iteration m to the new codeword $\boldsymbol{y}_j^{(m+1)}$ at iteration $m + 1$ as

$$\boldsymbol{y}_j^{(m+1)} = \boldsymbol{y}_j^{(m)} + s \times ((\frac{1}{|V_j^{(m)}|} \sum_{\boldsymbol{v} \in V_j^{(m)}} \boldsymbol{v}) - \boldsymbol{y}_j^{(m)}) \tag{6}$$

where $1 < s < 2$ is a scale factor. Based on the squared-error distance measure, Lee et al. [15] have shown experimentally that the best results are found when the scale factor is set to a fixed value of $s = 1.8$. Paliwal and Ramasubramanian [16] found that the use of a fixed scale factor for the entire algorithm results in the use of step sizes larger than the corresponding centroid-update at iterations closer to convergence and causes undesirably high perturbations of the codewords. Thus the number of iterations that are required to converge increases as well as the codebook falling to a poorer local optimum. Thus, they proposed the use of a variable scale factor s inversely proportional to m as follows [16]:

$$s = 1 + \frac{x}{x + m} \tag{7}$$

where $x > 0$. Paliwal and Ramasubramanian have studied the algorithm with various values of x. According to their results, they finally adopt $x = 9$.

2.2. **Some Existing Initialization Techniques.** The conventional famous algorithms for selecting an initial codebook are random initialization, splitting [4], PNN [12], maximum distance initialization [13], and the norm-ordered grouping strategy [14].

The random selection algorithm is simply to select K codewords from the training vectors of size M using a random selection strategy as an initial codebook, where $M \gg K$. This algorithm is very simple and has a low complexity in initial codebook design. However, the performance of randomly selected initial codebook is rather poor on average.

The splitting algorithm proposed by Linde et al.[4] splits the small codebook into a larger codebook. At first, the size of codebook is set to be one and the codeword in the codebook is the centroid of the whole training set. In each splitting phase, each codeword

$\boldsymbol{y}_i$ in the codebook $C = \{\boldsymbol{y}_i, i = 1, 2, \ldots, N\}$ is split into two close codewords $\boldsymbol{y}_i + \varepsilon$ and $\boldsymbol{y}_i - \varepsilon$, where ε is a fixed perturbation vector. Then the set $\{\boldsymbol{y}_i + \varepsilon, \boldsymbol{y}_i - \varepsilon; i = 1, 2, \ldots, N\}$ becomes the new codebook of size $2N$ in the next splitting phase. The splitting algorithm is complete until the desired codebook of size K is reached.

The PNN algorithm proposed by Equite [12] performs a set of inverse operations. First, each cluster has only one vector and the vector is the codeword of the cluster in the PNN algorithm. Then the PNN algorithm merges two closest clusters together at a time and the codeword of the new cluster is the centroid of the two closest clusters. The PNN algorithm is complete until the desired codebook size K is reached and the codewords in the codebook are the centroids of each cluster. Both the PNN algorithm and the splitting algorithm have a very high computational complexity.

The maximum distance initialization technique is proposed by Katsavounidis et al.[13]. They pay attention to the training vectors that are most far apart from each other because they are more likely to belong to different classes. They first calculate the norms of all vectors in the training set. Choose the vector with the maximum norm as the first codeword. Then, calculate the distance of all training vectors from the first codeword, and choose the vector with the largest distance as the second codeword. Thus, with a codebook of size $i, i = 2, 3, \ldots$, they compute the distance between any remaining training vector v and all existing codewords and call the smallest one the distance between v and the codebook. Then, the training vector with the largest distance from the codebook is chosen to be the $(i + 1)$-th codeword. The procedure stops when we obtain a codebook of size K. This algorithm needs high extra computational load.

Chen et al. presented a norm-sorted grouping strategy [14], where the training vectors are sorted according to the norm of each vector component. The ordered vectors are then partitioned into K groups where each group has the same number of vectors. The initial codebook is composed of the centroid of each group. This method has worse performance than the random method in the case of large codebook size.

3. Proposed Edge-classified Norm-ordered K-means Algorithm.

In general, the feature distribution of the training set has a great effect on the initial codebook generation. To make full use of the features, our algorithm considers classifying the training vectors into eight classes according to their edge orientations and then sort the training vectors in each class based on their norm values. Assume the training set is $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_M\}$, the detailed scheme can be described as follows.

Step 1: Each training vector $\boldsymbol{x}_i (i = 1, 2, ..., M)$ is input into the edge classifier, and an index $t_i \in 1, 2, ..., 8$ is output to denote which class the training vector belongs to.

Step 2: We collect all the training vectors belonging to the same class to generate a subset, and thus we have 8 subsets P_j of sizes $s_j (j = 1, 2, \ldots, 8)$ respectively, where $s_1 + s_2 + \ldots + s_8 = M$.

Step 3: We initialize $K \times s_j/M$ initial codewords from each subset P_j based on norm-sorted grouping strategy as follows:

Step 3.1: Calculate the norm value of each training vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)^{\mathtt{T}}$ in subset P_j as follows:

$$norm_v = \sqrt{\sum_{i=1}^{n} v_i^2} \tag{8}$$

Step 3.2: Sort the training vectors in P_j in the descending order of their norms.

Step 3.3: Divide the sorted subset P_j into $K \times s_j/M$ groups evenly.

Step 3.4: Calculate the centroid of each group as the initial codeword.

$$\mathbf{E}_1=\begin{bmatrix}-4&2&2&2\\-4&0&0&2\\-4&0&0&2\\-4&2&2&2\end{bmatrix},\mathbf{E}_2=\begin{bmatrix}2&2&2&2\\0&0&2&2\\-4&-4&0&2\\-4&-4&0&2\end{bmatrix},\mathbf{E}_3=\begin{bmatrix}2&2&2&2\\2&0&0&2\\2&0&0&2\\-4&-4&-4&-4\end{bmatrix}$$

$$\mathbf{E}_4=\begin{bmatrix}2&2&2&2\\2&2&0&0\\2&0&-4&-4\\2&0&-4&-4\end{bmatrix},\mathbf{E}_5=\begin{bmatrix}2&2&2&-4\\2&0&0&-4\\2&0&0&-4\\2&2&2&-4\end{bmatrix},\mathbf{E}_6=\begin{bmatrix}2&0&-4&-4\\2&0&-4&-4\\2&2&0&0\\2&2&2&2\end{bmatrix}$$

$$\mathbf{E}_7=\begin{bmatrix}-4&-4&-4&-4\\2&0&0&2\\2&0&0&2\\2&2&2&2\end{bmatrix},\mathbf{E}_8=\begin{bmatrix}-4&-4&0&2\\-4&-4&0&2\\0&0&2&2\\2&2&2&2\end{bmatrix}$$

FIGURE 1. eight 4×4 templates for edge classification.

Step 4: Collect all calculated centroids from different groups, we can in total obtain K initial codewords. Then the modified K-means algorithm in [16] is performed on the generated K initial codewords to generate the final codebook. During the iteration steps, if an empty cell occurs, we just judge the classes that all current codewords belong to and find the class with the least number of codewords, and randomly select a training vector from this class as a new centroid.

Now, we explain our edge classifier. Inspired by the Structured Local Binary Kirsch Pattern (SLBKP) in [17] that adopts eight 3×3 Kirsch templates to denote eight edge directions, we propose following eight 4×4 templates for edge classification shown in Fig.1.

For a 4×4 block $x(p,q), 0 \le p < 4, 0 \le q < 4$, an edge orientation vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_8)$ can be calculated as follows:

$$v_i = |\sum_{p=0}^{3}\sum_{q=0}^{3}[x(p,q) \cdot e_i(p,q)]| 1 \le i \le 8 \tag{9}$$

Where $e_i(p,q)$ denotes the element at the position (p,q) of E_i. Thus, an input block $x(p,q)$ is classified into the j-th category if

$$j = \arg\max_{1 \le i \le 8} v_i \tag{10}$$

Thus, we can classify each training vector into one of eight categories according to its edge orientation.

4. **Experimental Results.** To show the performance of the proposed edge-classified norm-ordered K-means algorithm, we compared it with the traditional K-means algorithm (KMeans) and the norm-ordered grouping based algorithm [14] (NOKMeans). For KMeans, we use the random selection technique, the performance is averaged over ten runs. In our experiments, we used three 512×512 monochrome images with 256 gray levels, Lena, Peppers and Baboon. We segmented each image into 16384 blocks, and each block is of size 4×4. We tested the performance for different codebook sizes of 256, 512, and 1024. The quality of the compressed images is evaluated by mean squared error

TABLE 1. Performance comparison for Lena image. (itr: number of iterations).

Codebook size	**256**		**512**		**1024**	
Performance	**MSE**	**itr**	**MSE**	**itr**	**MSE**	**itr**
KMeans	60.375	39	49.201	38	40.003	34
NOKMeans[14]	59.344	42	49.104	45	41.790	41
Our	**59.155**	**35**	**48.228**	**30**	**39.101**	**25**

TABLE 2. Performance comparison for Peppers image. (itr: number of iterations).

Codebook size	**256**		**512**		**1024**	
Performance	**MSE**	**itr**	**MSE**	**itr**	**MSE**	**itr**
KMeans	68.368	45	58.035	39	48.195	29
NOKMeans[14]	68.057	45	58.234	49	49.564	35
Our	**67.951**	**24**	**57.094**	**20**	**46.974**	**18**

TABLE 3. Performance comparison for Baboon image. (itr: number of iterations).

Codebook size	**256**		**512**		**1024**	
Performance	**MSE**	**itr**	**MSE**	**itr**	**MSE**	**itr**
KMeans	313.68	46	266.97	39	225.39	26
NOKMeans[14]	310.11	47	267.62	61	229.71	29
Our	**309.02**	**45**	**265.09**	**30**	**222.01**	**19**

(MSE). All the algorithms are terminated when the ratio of the MSE difference between two iterations to the MSE of the current iteration is within 0.0001 or 0.01%. In Tables 1-3, the PSNR values and the numbers of iterations with different codebook sizes are shown for Lena, Peppers and Baboon images respectively. From Tables 1-3, we can see that, our scheme requires the least average number of iterations than other algorithms and can also get better codebooks than other algorithms.

5. **Conclusions.** This paper presents an edge-classified norm-ordered K-means algorithm for image vector quantization. The main idea is to classify the training vectors into 8 categories based on the edge orientation of each vector, and then select initial codewords from each category based on norm sorting and grouping with the number of codewords proportional to the number of training vectors in each category. The experimental results based on three test images show that our algorithm can converge to a better locally optimal codebook with a faster convergence speed.

Acknowledgments. This work was supported partially by the financial support from the National Nature Science Foundation of China under grant No. 61272020 and the Zhejiang Provincial Natural Science Foundation of China under grant No. LZ15F030004 and Ningbo Science &Techology Plan Project(2014B82015). This work was also supported by Research Programs of Educational Commission Foundation of Zhejiang Province of ChinaNo.Y201636903.

REFERENCES

[1] A. Gersho, and R. M. Gray, Vector quantization and signal compression, Springer Science & Business Media, 2012

[2] F. D. V. R. Oliveira, H. L. Haas, J. G. R. C. Gomes, and A. Petraglia, CMOS imager with focal-plane analog image compression combining DPCM and VQ, *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no.5, pp. 1331–1344, 2013.

[3] F. X. Yu, H. Luo, and Z. M. Lu, Colour image retrieval using pattern co-occurrence matrices based on BTC and VQ, *Electronics Letters*, vol. 47, no.2, pp. 100–101, 2011.
[4] Y. Linde, A. Buzo, and R. M. Gray, An algorithm for vector quantizer design, *IEEE Transactions on Communications*, vol. 28, no.1, pp. 84–95, 1980.
[5] A. Vasuki, and P. Vanathi, A review of vector quantization technique, *IEEE Potentials*, vol. 25, no.4, pp. 39–47, 2006.
[6] H. A. S. Leitao, W. T. A. Lopes, and F. Madeiro, PSO algorithm applied to codebook design for channel-optimized vector Quantization, *IEEE Latin America Transactions*, vol. 13, no. 4, pp. 961-967, 2015.
[7] S. Alkhalaf, O. Alfarraj, and A. M. Hemeida, Fuzzy-VQ image compression based hybrid PSOGSA optimization algorithm, *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–6, 2015.
[8] C. C. Chang, Y. C. Li, and J. B. Yeh, Fast codebook search algorithms based on tree-structured vector quantization, *Pattern Recognition Letters*, vol. 27, no. 10, pp. 1077–1086, 2006.
[9] J. S. Pan, F. R. McInnes and M. A. Jack, Fast Clustering Algorithms for Vector Quantization, *Pattern Recognition*, vol. 29, no. 3, pp.511–518, 1996.
[10] J. S. Pan, F. R. McInnes and M. A. Jack, VQ Codebook Design Using Genetic Algorithms, *Electronics Letters*, vol. 31, no. 17, pp.1418–1419, 1995.
[11] J. Z. C. Lai, Y. C. Liaw, and J. Liu, A fast VQ codebook generation algorithm using codeword displacement, *Pattern Recognition*, vol. 41, no. 1, pp. 315–319, 2008.
[12] W. H. Equitz, A new vector quantization clustering algorithm, *IEEE Transactions on Acoustic Speech Signal Processing*, vol. 37, no.10, pp. 1568–1575, 1989.
[13] I. Katsavounidis, C. C. J. Kuo, and Z. Zhang, A new initialization technique for generalized Lloyd iteration, *IEEE Signal Processing Letters*, vol. 1, no.10, pp. 144–146, 1994.
[14] S. X. Chen, F. W. Li, W. L. Zhu, and T.Q. Zhang, Initial codebook algorithm of vector quantization, *IEICE Transactions on Information and Systems*, vol. E91-D, no.8, pp. 2189–2191, 2008.
[15] D. Lee, S. Baek, and K. Sung, Modified K-means algorithm for vector quantizer design, *IEEE Signal Processing Letters*, vol. 4, no.1, pp. 2–4, 1997.
[16] K. K. Paliwal, and V. Ramasubramanian, Comments on modified K-means algorithm for vector quantizer design, *IEEE Transactions on Image Processing*, vol. 9, no.11, pp. 1964–1967, 2000.
[17] G. Y. Kang, S. Z. Guo, D. C. Wang, L. H. Ma, and Z. M. Lu, Image retrieval based on structured local binary kirsch pattern, *IEICE Transactions on Information and Systems*, vol. 96, no.5, pp. 1230–1232, 2013.