2. Review of Literature

Digital image compression has become a very vigorous area of research due to the growing demand in diverse fields. Compression techniques reduce the amount of data needed to represent an image so that images can be economically transmitted and stored. Vector quantization (VQ) is an efficient technique for data compression and has been successfully used in various applications involving VQ-based encoding and VQ-based recognition [13-15]. Many types of VQ, such as classified VQ [16], [17],[24] address VQ [16], [18], finite state VQ [16], [19], side match VQ [16], [20], mean-removed classified VQ [16], [21], and predictive classified VQ [16], [22], have been used for various purposes. VQ has been applied to many applications, such as index compression [16], [23], and inverse halftoning [16], [25], [28]. VQ has been very popular in a variety of research fields such as speech recognition and face detection [26], [27], pattern recognition [30]. VQ is also used in real-time applications such as real-time video-based event detection [26], [28] and anomaly intrusion detection systems [26], [29], speech data compression [71], image segmentation [72-76], [212], CBIR [77], [78], [210] iris recognition [186-188], face recognition [213], signature recognition [209] and colorization [214].

A VQ is nothing more than an approximation. The idea is similar to that of “rounding-off” (say to the nearest integer). An example of scalar quantization is shown below in Figure 2.1.

Figure 2.1 Scalar Quantization

Here, every number less than -2 is approximated by -3. Every number between -2 and 0 is approximated by -1. Every number between 0 and 2 is approximated by +1. Every number greater
than 2 is approximated by +3. Note that the approximate values are uniquely represented by 2 bits.

An example of a 2-dimensional VQ is shown below in Figure 2.2.

![Voronoi Partitions](image)

Here, every pair of numbers falling in a particular region are approximated by a red star associated with that region. Note that there are 32 regions and 32 red stars each of which can be uniquely represented by 5 bits. Thus, this is a 2-dimensional, 5-bit VQ.

In the above two examples, the red stars are called codevectors and the regions defined by the blue borders are called encoding regions. The set of all codevectors is called the codebook and the set of all encoding regions is called the partition of the space. Voronoi partitioned space is shown in Figure 2.2.

There are several known methods for generating a codebook [14]. The most cited and widely used is the generalized Lloyd algorithm.
(GLA) [15]. It starts with an initial solution, which is iteratively improved using two optimality criteria in turn until a local minimum is reached. A different approach is to build the codebook hierarchically is iterative splitting algorithm [63, 64] starts with a codebook of size 1 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 neighbor (PNN) [65], uses an opposite, bottom-up approach to the codebook generation. 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. The hierarchical methods have two advantages over the GLA, they can produce codebooks of higher quality [66], and they implicitly generate codebooks of several different sizes during the same run. From the two approaches, the PNN has higher potential because it gives better results with a simpler implementation. The greatest deficiency of the PNN, however, is its slow speed. The method (referred as the exact PNN) uses local optimization for finding the code vectors to be combined. A straightforward implementation of this requires an $O(M^3)$ time [67] (where $Mxk$ is the size of the training set), which is rather slow for large training sets. Several suboptimal modifications have been proposed in the literature for speeding up the PNN algorithm. Equitz proposed an $O(M\log M)$ time variant of the PNN, referred as the fast PNN [65]. It uses K-d tree for localizing the search for the code vectors, and it merges several vector pairs at the same time. The method, however, has not gained as much popularity as the exact PNN, probably because of its more complex implementation and suboptimal results [70]. Another possibility is to generate a preliminary codebook of size $M_0$ ($N>M_0>M$) using the GLA and then apply the exact PNN until the codebook reaches its
final size $M$ [68]. In the exact PNN, most of the computation originates from the calculation of the pairwise distances. Since only two code vectors are changed in each step of the PNN, most of the distance calculations are unnecessary. Kurita proposed to store all pairwise distances into a heap structure for reducing the unnecessary calculations [69]. Only $O(M)$ updates are needed after each step of the PNN, each taking $O(\log M)$ time. The method thus performs the exact PNN in $O(M^2 \log M)$ time but it requires $O(M^3)$ memory, which is impractical for large training sets.

### 2.1 Codebook Design algorithms

Codebook can be designed in spatial domain using clustering algorithms [13-15], [58], [59], [82-85] or in transform domain [31-33]. Following existing codebook generation algorithms are given in brief.

- Linde Buzo Gray (LBG)[14], [15]
- Kekre’s Propoertionate Error algorithm (KPE) [84]
- Kekre’s Median Codebook Generation algorithm (KMCG) [84]
- Kekre’s Fast Codebook Generation algorithm (KFCG) [83]
- K-Means [58]

#### 2.1.1 LBG Algorithm [14], [15]

In this algorithm centroid is computed as the first codevector for the training set. Figure 2.3. shows the pictorial representation of LBG algorithm for 2 dimensional case. Two vectors $v_1$ & $v_2$ are generated by adding constant error to the codevector. Euclidean distances of all the training vectors are computed with vectors $v_1$ & $v_2$ and two clusters are formed based on nearest of $v_1$ or $v_2$. This procedure is repeated for every cluster. The drawback of this algorithm is that the cluster elongation is $+135^0$ to horizontal axis in two dimensional cases. This results in inefficient clustering.
2.1.2 Kekre’s Proportional Error algorithm (KPE) [84]

In this algorithm a proportionate error is added to the centroid to generate two vectors $v_1$ & $v_2$. The error ratio is decided by magnitude of coordinates of the centroid. Hereafter the procedure is same as that of LBG. In this algorithm while adding proportionate error a safe guard is introduced so that neither $v_1$ nor $v_2$ go beyond the training vector space. This overcomes the disadvantage of the LBG of inefficient clustering.

2.1.3 Kekre’s Median Codebook Generation algorithm (KMCG) [84]

The algorithm uses sorting and median technique for codebook generation but avoids the Euclidean distance computation therefore it is faster than other codebook generation algorithms. In this algorithm image is divided in to blocks and blocks are converted to the vectors of size $k$. The equation 2.1 given below represents
matrix $T$ of size $M \times k$ consisting of $M$ number of image training vectors of dimension $k$.

$$T = \begin{bmatrix}
  x_{1,1} & x_{1,2} & \cdots & x_{1,k} \\
  x_{2,1} & x_{2,2} & \cdots & x_{2,k} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_{M,1} & x_{M,2} & \cdots & x_{M,k}
\end{bmatrix} \quad (2.1)$$

Each row of the matrix is the image training vector of dimension $k$.

The training vectors are sorted with respect to the first element of all the vectors i.e 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 (codevector) and is put into the codebook, and the size of the codebook is set to one. The matrix is then divided into two equal parts and each of the part is then again sorted with respect to the second element of all the training vectors i.e. with respect to the second column of the matrix $T$ and two clusters both consisting of equal number of training vectors are obtained. The median of both the parts is picked up and written to the codebook, now the size of the codebook is increased to two, consisting of two codevectors and again each part is further divided to half. Each of the above four parts obtained are sorted with respect to the third column of the matrix $T$ and four clusters are obtained and accordingly four codevectors are obtained. The above process is repeated till the codebook of desired size is obtained. Here quick sort algorithm is used. This algorithm takes least time to generate codebook, since Euclidean distance computation is not required.

### 2.1.4 Kekre’s Fast Codebook Generation algorithm (KFCG) [83]

The algorithm reduces the codebook generation time since it avoids the Euclidean distance computations. Initially there is one cluster with the entire training vectors and the codevector $C_1$ which is
centroid. In the first iteration of the algorithm, the clusters are formed by comparing first element of training vector with first element of code vector $C_1$. The vector $X_i$ is grouped into the cluster 1 if $x_{i1} < c_{11}$ otherwise vector $X_i$ is grouped into cluster 2. In second iteration, the cluster 1 is split into two by comparing second element $x_{i2}$ of vector $X_i$ belonging to cluster 1 with that of the element $c_{12}$ of the codevector $C_1$. Cluster 2 is split into two by comparing the element $x_{i2}$ of vector $X_i$ belonging to cluster 2 with that of the element $c_{22}$ of the codevector $C_2$.

This procedure is repeated till the codebook size is reached to the size specified by user. It is observed that this algorithm gives minimum error and requires least time to generate codebook as compared to other algorithms [87]. The pictorial representation of this algorithm for two dimensional case is shown in Figure 2.4. It is easily extended to higher dimensions.
2.1.5 K-Means Algorithms [58]

Following are the steps for K-Means algorithm

1. Select k random vectors from the training set and call it as codevectors.
2. Find the squared Euclidean distance of all the training vectors with the selected k vectors and k clusters are formed.
3. A training vector $X_j$ is put in $i^{th}$ cluster if the squared Euclidean distance of the $X_j$ with $i^{th}$ codevector is minimum. In case the squared Euclidean distance of $X_j$ with codevectors happens to be minimum for more than one codevector then $X_j$ is put in any one of them.
4. Compute centroid for each cluster. Compute Mean Squared Error (MSE) for each of k clusters. Compute net MSE.
5. Centroid obtained for each cluster replaces the initial k vectors and are given as input to next iteration.
6. Repeat step 2 to 5 till the net MSE converges. This algorithm takes very long time to converge and gives minimum net MSE if started from random k vectors selection.
2.2. Encoding Methods

The next phase of VQ is encoding in which codebook is searched; here image is first converted into blocks of vectors of dimension k. Each of the image vectors are searched for the nearest codevector in the codebook CB. Once the nearest codevector is obtained it’s corresponding index is sent to the receiver.

Let the image training vector \( X_i = (x_{i1}, x_{i2}, \ldots, x_{ik}) \). The codebook is then searched for the nearest codevector \( C_{\text{min}} \) by computing squared Euclidean distance as presented in equation (2) with vector \( X_i \) and all the codevectors of the codebook \( \text{CB} \). This method is called exhaustive search (ES).

\[
\text{d}(X_i, C_{\text{min}}) = \min_{1 \leq j \leq N} \{\text{d}(X_i, C_j)\} \quad \text{where} \quad \text{d}(X_i, C_j) = \sum_{p=1}^{k} (x_{ip} - c_{jp})^2
\]

Although the Exhaustive Search (ES) method gives the optimal result at the end, it involves heavy computational complexity. It is observed that from equation (2.2) to obtain one nearest codevector for a training vector requires \( N \) Euclidean distance computation where \( N \) is the size of the codebook. So for \( M \) image training vectors, will require \( M*N \) number of Euclidean distances computations. It is obvious that if the codebook size is increased to reduce the distortion the search time will also increase.

In order to reduce the search time there are various search algorithms available in literature. So far, Partial Distortion Elimination (PDE) [36], Triangular Inequality Elimination (TIE) [37-39], Mean distance ordered Partial codebook Search (MPS) algorithm [40], Double Test Algorithm (DTA) [41], fast codebook search algorithm based on the Cauchy-Schwarz Inequality (CSI) [42], fast codebook search based on Sub Vector Technique (SVT) [43], the image encoding based on \( L_2 \)-norm pyramid of codewords
[44] and the fast algorithms using the Modified L2-norm Pyramid (MLP) [45], fast codeword search algorithm based on MPS+TIE+PDE proposed by Yu-Chen, Bing-Hwang and Chih-Chiang (YBC) in 2008 [46], Eigen Vector Method (EVM) [47], and others [35], [48], [49], [50], [41], [51] are classified as partial search methods. Some of the partial techniques use data structure to organize the codebook for example tree-based [26], [52], [53], [54], [55], Kekre’s search methods [56], [79-81] and projection based structure [26], [51], [57]. All these algorithms reduce the computational cost needed for VQ encoding keeping the image quality close to Exhaustive search algorithm.

2.2.1 Existing Search Algorithms
Some fast search algorithms are reviewed in this section.

2.2.1.1 Partial distortion Elimination (PDE) [36]
In 1985 Bei and Gray proposed the partial distance elimination (PDE) [36] to accelerate the closest codeword search for image training vector. The algorithm uses the following premature-exit condition to reject the codevectors for image training vector \( X_i \) and the \( j^{th} \) codevector \( C_j \).

\[
\sum_{q=1}^{s} (x_{iq} - c_{jq})^2 \geq d_{\text{min}} \quad \text{for some } s < k \tag{2.3}
\]

Here \( d_{\text{min}} \) is the minimum squared Euclidean distance value searched so far.

2.2.1.2 Partial search partial distortion algorithm (PSPD)[60]
PSPD algorithm builds up a partial codebook based on the mean value \( m_x \) of a k-dimensional training vector \( x = \{x_1, x_2, \ldots, x_k\} \) in which \( m_x \) is defined as \( m_x = \text{integer part of} \left[ \frac{1}{k} \sum x_j + 0.5 \right] \) The algorithm then uses the PDE method to search the partial codebook for the closest codevector.
The PSPD algorithm first calculates the mean values of all codevector and sorts the codebook according to increasing order of the codevector means. For each training vector, it then finds the codeword \( y_p \) with minimum mean difference to the training vector. The PDE method is then employed to find the closest codevector in this partial codebook from the codevector \( y_{p+1} \) to the last codevector in the codebook.

Sometimes the closest codevector may not be located in the partial codebook, this will introduce more distortion than sequential search.

### 2.2.1.3 Fast Nearest Neighbour Search Algorithm (FNNS)[37-39]

FNNS algorithm uses triangle inequality to reject a great many unlikely codewords. For a vector \( x \), it first finds a probably nearby codevector \( y_i \) with distortion \( d^2(x, y_i) \).

This algorithm then eliminates those codevector, which are impossible to be closest codevector, based on the triangle inequality and a pre computed table, which contains the distances of all pairs of codevector.

That is, for each codevector \( y_j \), if 
\[
d(y_j, y_i) > 2d(x, y_i)
\]
Through the triangle inequality, following equation is obtained 
\[
d(x, y_j) + d(x, y_i) \geq d(y_j, y_i) > 2d(x, y_i)
\]
The above inequality can be reduced to be
\[
d(x, y_j) > d(x, y_i)
\]
Therefore those codevector with distances to \( y_i \) larger than \( 2d(x, y_i) \) will be eliminated from consideration to be a candidate of the closest codeword.
Let $N$ be the size of codebook.

Let $\text{map}(p,j)$ be the Euclidean distance between $p^{th}$ codevector and $j^{th}$ codevector.

Let $d_{\text{min}}$ be the Euclidean distance between 1$^{st}$ codevector and the training vector $x$.

Let $p=1$

For $j=2$ to $N$

\begin{verbatim}
  begin
    If $2d_{\text{min}} \leq \text{map}(p,j)$
    Then compute $d$ = Euclidean distance between $j^{th}$
    codevector and the training vector $x$.
    If $d \leq d_{\text{min}}$ then
      Assign $d_{\text{min}} = d$ and $p=j$
  End.
\end{verbatim}

Repeat the above for loop for all training vectors.

This algorithm requires a table of size $N^2/2$ to store the distances of all pairs of all pairs of codevectors. When $N$ is large, the memory requirement is a serious problem.

### 2.2.1.4 Fast codebook search algorithm based on Cauchy-Schwarz inequality (CSI) [42]

In this algorithm $||C_i||$ for $i = 1, 2, \ldots, N$ is pre-calculated and codebook is sorted so that $||y_1|| \leq ||y_2|| \leq \ldots \leq ||y_N||$. For the $i^{th}$ input vector $||X_i||$ is calculated and closest codevector is searched based on the closest norm value. Then compute $d_1$-distortion using equation (4)

$$d_1(X_i, C_j) = ||C_j||^2 - 2 \sum_{q=1}^{k} (x_{iq} \ast c_{jq})$$

Based on the Cauchy-Schwarz inequality

$$d_1(X_i, C_j) \geq ||C_j||^2 - 2||C_j|| \cdot ||X_i||$$
Codevector $C_j$ can be safely rejected if $d_1(X_i, C_j) \geq d_{1\text{min}}$, where $d_{1\text{min}}$ denotes the minimal $d_1$-distortion calculated so far.

### 2.2.1.5 Codebook search based on subvector technique (SVT) [43]

Three inequalities are used in this algorithm based on sum and variance of each image vector. In this method each image vector $X$ of dimension $k$ is divided into equal-sized subvectors $X_s$ and $X_f$ of dimension $k/2$. Following inequalities hold for image vector $X$ and the $i^{th}$ codevector $C_i$ in the codebook.

\[(\text{Sum}(X) - \text{Sum}(C_i))^2 + k \times (\text{Var}(X) - \text{Var}(C_i))^2 \leq k \times d(X, C_i)\] (2.6)

Here $\text{Sum}(X)$ and $\text{Var}(X)$ denote sum and variance of vector $X$.

Let $d_{\text{min}}$ be the minimal squared Euclidean distance between image vector $X$ and the searched closest codevector $C_{\text{min}}$ found so far. The derived test condition based on the above inequality is given below.

\[k \times d_{\text{min}} \leq (\text{Sum}(X) - \text{Sum}(C_i))^2 + k \times (\text{Var}(X) - \text{Var}(C_i))^2\] (2.7)

If the above inequality satisfies then Codevector $C_i$ can be safely rejected since $d_{\text{min}} < d(X, C_i)$.

The authors proved that inequality in equation (6) is true for $k$-dimensional vectors as well as for $k/2$ dimensional vectors. The $i^{th}$ codevector is also divided into two $k/2$ dimensional vectors $C_{is}$ and $C_{if}$ and two additional test conditions are derived.

\[k \times d_{\text{min}} \leq (\text{Sum}(X_f) - \text{Sum}(C_{if}))^2 + k \times (\text{Var}(X_f) - \text{Var}(C_{if}))^2\] (2.8)

\[k \times d_{\text{min}} \leq (\text{Sum}(X_s) - \text{Sum}(C_{is}))^2 + k \times (\text{Var}(X_s) - \text{Var}(C_{is}))^2\] (2.9)

This algorithm uses three test condition to reject the codevectors.

### 2.2.1.6 Codebook search using modified $L_2$-norm pyramid (MLP) [45]

The inequality describing the squared Euclidean distance and $L_2$-norm is as follows:

\[d(X, C_j) \geq (||X|| - ||C_j||)^2\] (2.10)
if $(||X|| - ||C_i||)^2 \geq d(X, C_i)$, it guarantees that $d(X, C_i)$ is greater than $d_{\text{min}}$.

In this method codevectors of nxn pixels are arranged in $L_2$-norm pyramid structure. The top level $L_0$ stores the real $L_2$-norm of $X$ and the bottom level $L_n$ stores the original $X$. Each pixel at a higher level $m$ can be computed from the corresponding four pixels at its lower level $(m+1)$. Let $d_m(X, C_i)$ denotes the squared Euclidean distance between $X$ and $C_i$ in the $m^{th}$ level of the pyramid. The modified distance is given as below:

$$
\overline{d}_m(X, C_i) = d_m(X, C_i) - (||X||)^2
$$

(2.11)

The relationship between the distances in different levels of the pyramid is given as follows:

$$
\overline{d}(X, C_i) = \overline{d}_n(X, C_i) \geq \overline{d}_{n-1}(X, C_i) \geq \ldots \geq \overline{d}_1(X, C_i) \geq \overline{d}_0(X, C_i)
$$

(2.12)

The test condition given in equation (2.11) is applied for the remaining levels of the pyramid.

### 2.2.1.7 Fast codebook search algorithm based on MPS+TIE+PDE (YBC) [46]

This algorithm is proposed by Yu-Chen, Bing-Hwang and Chih-Chiang (YBC) in 2008, the steps of this algorithm are given below.

Output: indices of closest codevectors in the codebook.

Step 1: Sort the codevectors by their mean values. Partition image into a set of non-overlapped image blocks of $k$ pixels.

Step 2: For each image blocks $X$, compute the mean value $X_{\text{mean}}$ of $X$ and then determine the initial searched codeword $C_p$, where $p = \text{Index}(X_{\text{mean}})$.

Step 3: Calculate the squared Euclidean distance between $X$ and $C_p$ and
store the result in $d_{\text{min}}$ and, $I_{\text{min}} = p$.  

Step 4: For each codeword $C_i$ to be searched 

Step 4.1: Apply the MPS[41] to check whether $C_i$ can safely be rejected. If $C_i$ can be safely rejected, go to Step 6. 

Step 4.2: Perform the TIE[38-40] to check whether $C_i$ can safely be rejected. If $C_i$ can be safely rejected, go to Step 6. 

Step 4.3: Compute the squared Euclidean distance $d(X, C_i)$ between $X$ and $C_i$. The PDE technique is used here to check whether the calculation of $d(X, C_i)$ can be early terminated. 

Step 4.4: If $d(X, C_i) < d_{\text{min}}$, then $d_{\text{min}} = d(X, C_i)$ and $I_{\text{min}} = i$. 

Step 5: Output $I_{\text{min}}$. 

Step 6: If there is any image block to be processed, go to Step 2. Otherwise, the image encoding process stops. 

In this algorithm MPS and TIE are used to reject the impossible codevectors. In addition the look up table of 256 entries storing the squared values ranging from 0 to $255^2$ is used and further PDE technique is used for computing squared Euclidean distance. 

2.2.1.8. Equal average nearest neighbor search algorithm (ENNS)[61] 

The ENNS algorithm uses the fact that mean of the nearest codevector is usually close to the mean of the input vector. Let $m_p$ and $m_i$ be the mean values of training vector $X_p$ and codevector $C_i$ respectively. If the mean of the codevector $C_i$ satisfies 

$$m_i \geq m_p + \frac{\sqrt{d_{\text{min}}}}{k}\text{ or } m_i \leq m_p - \frac{\sqrt{d_{\text{min}}}}{k}$$ 

(2.13) 

then $C_i$ will not be the nearest codevector to $X_p$. To perform ENNS algorithm mean of all the codevectors should be computed off-line first and stored.
2.2.1.9. Equal average equal variance nearest neighbor search (EENNS)[62]

EENNS algorithm introduces another significant feature of vector, the deviation, to reject codevectors. Let \( v_p \) and \( v_i \) are the deviations of \( X_p \) and \( C_i \) respectively, then

\[
(v_p - v_i)^2 \leq d(X_p, C_i)
\]

If the deviation of the codevector \( C_i \) satisfies

\[
v_i \geq v_p + \sqrt{\frac{d_{\min}}{k}} \quad \text{or} \quad v_i \leq v_p - \sqrt{\frac{d_{\min}}{k}}
\]

then \( C_i \) will not be the nearest codevector to \( X_p \).

EENNS algorithm performs in two steps. In the first step, if

\[
m_i \geq m_p + \frac{\sqrt{d_{\min}}}{k} \quad \text{or} \quad m_i \leq m_p - \frac{\sqrt{d_{\min}}}{k}
\]

then codevector \( C_i \) can be rejected. Otherwise, in the second, if

\[
v_i \geq v_p + \sqrt{\frac{d_{\min}}{k}} \quad \text{or} \quad v_i \leq v_p - \sqrt{\frac{d_{\min}}{k}}
\]

then codevector \( C_i \) can also be rejected. Hence this algorithms requires \( N \) mean values and \( N \) deviations of all codevectors.

2.2.1.10. Nearest Neighbor search algorithm based on orthonormal transform (OTNNS)[11]

Here orthonormal base vectors \( V=(v_1, v_2, ..., v_k) \) for the Euclidean vector space \( \mathbb{R}^k \) are considered. For any \( k \)-dimensional vector \( x=(x_1, x_2, ..., x_k) \) it can be transformed to another Euclidean space defined by the \( k \) orthonormal base vectors, i.e. \( x= \sum_{j=1}^{k} X_j v_j \) where \( X=(X_1, X_2, ..., X_k) \) is the coefficient vector in the transformed space.

In this algorithm each input vector is a 3-D residual vector and the orthonormal base vectors are

\[
v_1 = (1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3}), \quad v_2 = (1/\sqrt{6}, 1/\sqrt{6}, -2/\sqrt{6}), \quad v_3 = (1/\sqrt{2}, -1/\sqrt{2}, 0)
\]

The conditions for judging possible nearest codevectors are

\[
X_{i,\min} \leq Y_{ji} \leq X_{i,\max} \quad 1 \leq i \leq 3
\]
Where \( Y_j = (Y_{j1}, Y_{j2}, Y_{j3}) \) is the coefficient vector of \( C_j \) in the transformed space, and

\[
X_{i,\text{min}} = X_i - d_{\text{min}} \tag{2.17}
\]
\[
X_{i,\text{max}} = X_i - d_{\text{min}} \tag{2.18}
\]

**Preprocessing:**
Transform each codevector of the codebook into the space with base vector \( V=(v_1, v_2, v_3) \) and then sort codevectors in ascending order with respect to the first elements, i.e. the coefficients along the base vector \( v_1 \).

**Online step:**
For searching each input vector \( X_i \) is transformed to obtained \( \hat{X}_i \).
The probable nearby codevector \( Y_j \) is guessed based on the minimum first element difference criterion. \( d_{\text{min}}, X_{i,\text{min}}, X_{i,\text{max}} \) are calculated. For each codevector \( Y_j \) first check if (15) is satisfied. If not then \( Y_j \) is rejected else \( d(\hat{X}_i, Y_j) \) is calculated. If \( d(\hat{X}_i, Y_j) < d_{\text{min}} \), then the current closest codevector to \( \hat{X}_i \) is taken as \( Y_j \) with \( d_{\text{min}} \) set to \( d(\hat{X}_i, Y_j) \) and \( X_{i,\text{min}} \) and \( X_{i,\text{max}} \) are updated accordingly. The procedure is repeated until best match is found.

**2.2.1.11. Kekre’s Centroid Search algorithm (KCS) [80], [81]**
Let CB be the codebook consisting of \( k \)-dimensional vector. First sort the codebook with respect to the first element of the codevector and then compute the centroid \( c_0 \) of the first element of all the codevectors. The codebook is then divided into two parts based on the centroid of the first element. The upper part of the codebook is again sorted with respect to the second element of the codevector and again centroid \( c_{00} \) is computed for the second element for the upper part. The process is repeated for the lower part too i.e. lower part of codebook is also sorted with respect to
the second element of the codevectors corresponding the lower part of the codebook and centroid $c_{01}$ is computed for the lower part. Based on the centroid the upper part of the codebook, it is further divided into two parts and the above process is repeated, similarly lower part of the codebook is also divided based on to centroid and above process is repeated. For codebook of size $N$ the above process is repeated for $(\log_2 N - 4)$ times so get $2^{(\log_2 N - 4)}$ parts of the codebook. The formation of the codebook into subparts is a preprocessing step for encoding is depicted in Figure 2.5. as shown below.

**Encoding process:**

In Encoding step the first element $x_{i1}$ of image training vector $X_i$ is compared with the $c_0$ if $x_{i1} < c_0$ then $x_{i2}$ is compared with $c_{00}$ else then $x_{i2}$ is compared with $c_{01}$ and so on. Once the training vector reaches the last level of tree the nearest codevector is searched from the group of codevectors using Euclidean distance computation. Instead of full search codebook is divided into subparts, nearest subpart for the training vector is found out and then closest codevector is searched using exhaustive search applied only to the subpart that is obtained. To locate the nearest subpart $(\log_2 N - 4)$ comparisons are required. The algorithm requires $2^{(\log_2 N - 4)} - 1$ extra memory space to store the centroids.
2.2.1.12. Kekre’s Median Search algorithm (KMS) [79]

Let $\mathbf{CB} = \{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_N\}$ represents the codebook. Each codevector is $k$ dimensional, e.g., $\mathbf{C}_j = (c_{j,1}, c_{j,2}, \ldots, c_{j,k})$, $j = 1, 2, \ldots, N$. In this algorithm the codebook of size $N$ is sorted with respect to the first element of all the codevectors i.e. $c_{i,1}$ for $i = 1, 2, \ldots, N$. The code book is divided in two parts of size $N/2$. The first element i.e. $c_{N/2,1}$ of the centre codevector $\mathbf{C}_N$ of the sorted codebook is selected. The upper $N/2$ block is again sorted with respect to the next element of all the codevectors i.e. $c_{i,2}$ for $i = 1, 2, \ldots, N/2$ in the block and the block of size $N/2$ is again divided in to two parts of size $N/4$ each. Second element of the center code vector i.e. $c_{N/4,2}$ is selected and is attached as the left child to the root $c_{N/2,1}$ of the binary tree. Similarly the lower $N/2$ block is again sorted with respect to the next element of all the codevectors i.e. $c_{i,2}$ for $i = (N/2)+1, (N/2)+2, \ldots, N$ in the block and the block of size $N/2$ is again divided in to two parts of size $N/4$ each, and second element of the center codevector i.e. $c_{3N/4,2}$ is selected and is attached as the right child to the root $\mathbf{C}_N/2,1$ of the binary tree. Each of the $N/4$ blocks is again sorted with respect to the third element of all the codevector in respective blocks. The above process is repeated till block of size 2 is obtained. The process is depicted in the Figure 2.6. The leaves of the binary tree consist of the indexes of the codevector.
Figure 2.6. Dividing codebook for indexing

**Encoding process:** Let $X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})$ be the image training vector the first element $x_{i,1}$ is compared with the $c_{N/2,1}$ if $x_{i,1} \leq c_{N/2,1}$ then second element of the training vector $x_{i,2}$ is compared with the $c_{N/4,2}$ (i.e. the left branch of the tree is taken) else $x_{i,2}$ is compared with $c_{3N/4,2}$. The above process is repeated till the index of the codevectors is obtained.