A Fuzzy-Soft Competitive Learning Approach for Grayscale Image Compression

Dimitrios M. Tsolakis and George E. Tsekouras

Abstract In this chapter we develop a fuzzy-set-based vector quantization algorithm for the efficient compression of grayscale still images. In general, vector quantization can be carried out by using crisp-based and fuzzy-based methods. The motivation of the current work is to provide a systematic framework upon which the above two general methodologies can effectively cooperate. The proposed algorithm accomplishes this task through the utilization of two main steps. First, it introduces a specially designed fuzzy neighborhood function to quantify the lateral neuron interaction phenomenon and the degree of the neuron excitation of the standard self-organizing map. Second, it involves a codeword migration strategy, according to which codewords that correspond to small and underutilized clusters are moved to areas that appear high probability to contain large number of training vectors. The proposed methodology is rigorously compared to other relative approaches that exist in the literature. An interesting outcome of the simulation study is that although the proposed algorithm constitutes a fuzzy-based learning mechanism, it finally obtains computational costs that are comparable to crisp-based vector quantization schemes, an issue that can hardly be maintained by the standard fuzzy vector quantizers.

Keywords Learning vector quantization • Soft learning vector quantization • Self-organizing map • Neighborhood function • Codeword migration

1 Introduction

Image compression [1–7] is one of the most widely used and commercially successful technologies in the field of digital image processing. Web page images and high-resolution digital camera photos also are compressed routinely to save storage space and reduce transmission time. The methods developed so far to perform image compression can be classified in two categories: (a) lossless compression and

D.M. Tsolakis (✉) • G.E. Tsekouras
Laboratory of Intelligent Multimedia, Department of Cultural Technology and Communication, University of the Aegean, 81100 Mytelene, Lesvos Island, Greece
e-mail: ctmb05018@ct.aegean.gr; gtsek@ct.aegean.gr

M.E. Celebi, K. Aydin (eds.), *Unsupervised Learning Algorithms*,
DOI 10.1007/978-3-319-24211-8_14

(b) lossy compression. Lossless compression is an error-free procedure according to which the pixel intensities of the original image are fully recovered in the compressed image representation. Although the quality of the resulting image is excellent, this approach is computationally complex and yields low compression rates. Contrary, lossy compression attempts to compromise the quality of the recovered image in exchange for higher compression rates. Thus, although the original image cannot be perfectly recovered, the computational demands are significantly reduced.

A common procedure to perform image compression is vector quantization (VQ) [4, 5, 8–14]. Vector quantization (VQ) refers to the process according to which a set of data vectors is partitioned into a number of disjoint clusters. The center elements of these clusters are referred to as the codewords, and the aggregate of all codewords as the codebook. The image is reconstructed by replacing each training vector by its closest codeword.

In general, vector quantization methods can be classified into two categories, namely crisp and fuzzy. Crisp vector quantization is usually performed in terms of the c-means algorithm and its variants [1, 5, 9, 15, 16], the self-organizing map (SOM) [17–21], and the learning vector quantization (LVQ) [9, 12, 17, 18]. On the other hand, the backbone of fuzzy techniques in designing vector quantizers is the fuzzy c-means algorithm and its variants [22–31]. The main difference between these two categories is that the former performs strong competition between the neurons by means of the winner-takes-all principle, while the latter embeds into the learning process soft conditions. The main advantages and disadvantages of the resulting behaviors can be identified as follows: A crisp-based algorithm obtains a fast learning process, yet it is very sensitive to the initialization [36, 37], and therefore it might not yield an efficient partition of the feature space. On the contrary, a fuzzy-based algorithm is less sensitive to the initialization generating effective partitions of the data set, yet it constitutes a slow process with high computational cost.

This chapter describes an algorithm that attempts to combine the merits of the above two general methodologies into a unique framework. The key idea is to alleviate the strong competitive nature of a crisp-based vector quantizer by defining a fuzzy-based novel neighborhood function able to measure the lateral neuron interaction phenomenon and the degree of the neuron excitations, as well. As a later step, a specialized codeword relocation procedure is developed in order to increase the individual contributions of the resulting codewords to the final partition.

This chapter is synthesized as follows. Section 2 reports two general methodologies to perform VQ. Section 3 presents the proposed methodology. The simulation experiments are presented in Sect. 4. Finally, the chapter's conclusions are discussed in Sect. 5.

2 Related Work

Vector quantization (VQ) is concerned with the representation of a set of training vectors by a set of codewords. The feature space is considered the p-dimensional Euclidean space. The set of training vectors is denoted as $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ with $\boldsymbol{x}_i \in \Re^p$, while the set of codewords as $\boldsymbol{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_c\}$ with $\boldsymbol{v}_i \in \Re^p$. The corresponding set cardinalities (i.e. set sizes) are $card\,(\boldsymbol{X}) = n$ and $card\,(\boldsymbol{V}) = c$. Using the above nomenclature the VQ quantization is defined as a function $f : \boldsymbol{X} \rightarrow \boldsymbol{V}$, which yields a partition of the set $\boldsymbol{X}$ into a number of c disjoint subsets (commonly called clusters):

$$\boldsymbol{C}_i = \{\boldsymbol{x}_k \in \boldsymbol{X} : f\,(\boldsymbol{x}_k) = \boldsymbol{v}_i, \quad 1 \leq k \leq n\} \tag{1}$$

with $i \in \{1, 2, \ldots, c\}$. The equation $f\,(\boldsymbol{x}_k) = \boldsymbol{v}_i$ in relation (1) is realized in terms of the following nearest neighbor condition:

$$\|\boldsymbol{x}_k - \boldsymbol{v}_i\| = \min_{1 \leq j \leq c} \left\{\left\|\boldsymbol{x}_k - \boldsymbol{v}_j\right\|\right\} \tag{2}$$

Note that any partition of the set $\mathbf{X}$ will also yield a partition of the feature space. Within the next subsections we shall describe two different approaches to VQ.

2.1 The Batch Learning Vector Quantization

The SOM constitutes a class of vector quantization algorithms based on unsupervised competitive learning. The SOM produces a similarity graph of the input data and converts the nonlinear statistical relationships between high-dimensional data into simple geometric relationships in a low-dimensional display grid, usually a regular two-dimensional grid of nodes. Therefore it can be viewed as a similarity graph and a clustering diagram, as well [17, 19, 21].

In this sense its computation procedure relies on nonparametric, recursive regression process [18]. Figure 1 shows a 16×16 neuron display grid. Each node in the grid is associated with a specific neuron weight. The neuron weights coincide with the codewords. Therefore, from now on these two concepts will be used to mean the same thing. Given the training set $\boldsymbol{X} = \{\boldsymbol{x}_k : 1 \leq k \leq n\}$, and providing initial values for the set of the codewords $\boldsymbol{V} = \{\boldsymbol{v}_i : 1 \leq i \leq c\}$, the SOM is implemented as follows. In the iteration t, for the training vector $\boldsymbol{x}_k$ we locate the nearest codeword according to the relation:

$$\left\|\boldsymbol{x}_k - \boldsymbol{v}_{i(\boldsymbol{x}_k)}\,(t-1)\right\| = \min_{1 \leq j \leq c} \left\{\left\|\boldsymbol{x}_k - \boldsymbol{v}_j\,(t-1)\right\|\right\} \tag{3}$$

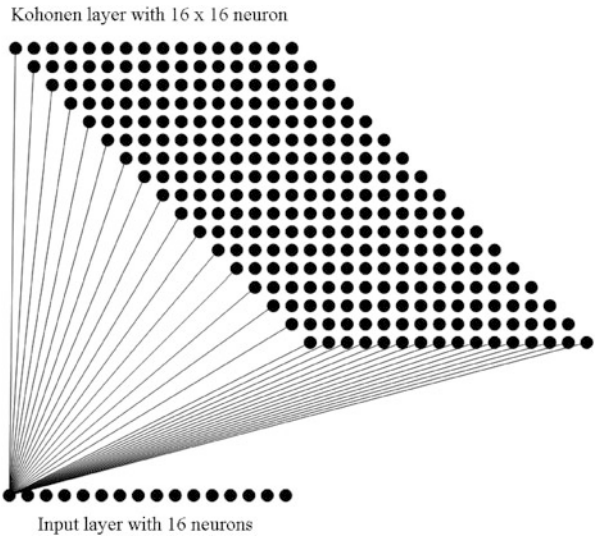

Fig. 1 SOM neural network with 16 inputs

Then, the updating rule for the codewords is given in the next formula:

$$\boldsymbol{v}_j(t) = \boldsymbol{v}_j\,(t-1) + \eta(t)\; h_{i(\boldsymbol{x}_k),\,j}\;\left(\boldsymbol{x}_k - \boldsymbol{v}_j\,(t-1)\right) \tag{4}$$

where $j = 1,\ 2, \ldots,\ c$, $\eta(t)$ is the learning parameter which is required to decrease with the number of iterations, and $h_{i(\boldsymbol{x}),j}$ is the neighborhood function. The $h_{i(\boldsymbol{x}),j}$ quantifies the lateral neural interaction phenomenon and the degree of excitation of the neurons. Note that the index $i\,(\boldsymbol{x}_k) \in \{1,\ 2, \ldots,\ c\}$ denotes the codeword $\boldsymbol{v}_{i(\boldsymbol{x}_k)}$ that is closer to $\boldsymbol{x}_k$, while the index $j = 1,\ 2, \ldots, c$ corresponds to anyone of the codewords (including $i(\boldsymbol{x}_k)$). The typical form of the neighborhood function is calculated considering the corresponding nodes in the grid and reads as [17, 18, 31]:

$$h_{i(\boldsymbol{x}_k),\,j} = \exp\left(-\left(\frac{\left\|\boldsymbol{r}_{i(\boldsymbol{x}_k)} - \boldsymbol{r}_j\right\|}{\sqrt{2}\,\sigma(t)}\right)^2\right) \tag{5}$$

where $\boldsymbol{r}_{i(\boldsymbol{x})} \in \Re^2$ and $\boldsymbol{r}_j \in \Re^2$ are vectorial locations on the grid that correspond to the codewords $\boldsymbol{v}_{i(\boldsymbol{x}_k)}$ and $\boldsymbol{v}_j$, and $\sigma(t)$ stands for the width, which is also required to reduce with time. A simpler way to define the neighborhood function is:

$$h_{i(\boldsymbol{x}_k),j} = \begin{cases} 1, \text{ if } j = i\,(\boldsymbol{x}_k) \\ 0, \quad \text{otherwise} \end{cases} \tag{6}$$

Equation (6) leads to the direct definition of the LVQ. Note that in the standard SOM the use of Eq. (5) forces all the codewords close to the winning one to move, while the LVQ updates only the position of the winning node. As a result, the very

nature of the LVQ is dominated by the winner-takes-all principle. Although the LVQ is sensitive to random initialization it constitutes quite a fast procedure. The application of LVQ is accomplished in a sequential manner, meaning that the final result is an on-line procedure. In [18] Kohonen modified the on-line implementation of the LVQ and introduced a batch iterative procedure called batch LVQ, where in each iteration all the training vectors were taken into account. The basic assumption to carry out this modification relies on the observation that, for a specific training vector $\boldsymbol{x}_k$, the on-line LVQ will converge to a stationary point, say $\boldsymbol{v}_i^{\oplus}$. Therefore, the expectation values of $\boldsymbol{v}_j\,(t-1)$ and $\boldsymbol{v}_j(t)$ must be equal to $\boldsymbol{v}_j^{\oplus}$ as $t \to \infty$, which implies that

$$E\left[h_{i(\boldsymbol{x}_k),\,j}\left(\boldsymbol{x}_k - \boldsymbol{v}_j(t)\right)\right] = 0 \quad \text{as } t \to \infty \tag{7}$$

Applying an empirical distribution the above equation yields [18]:

$$\boldsymbol{v}_j = \frac{\sum_{k=1}^{n} h_{i(\boldsymbol{x}_k),\,j}\,\boldsymbol{x}_k}{\sum_{k=1}^{n} h_{i(\boldsymbol{x}_k),\,j}} \tag{8}$$

To this end, the batch LVQ algorithm is given by the next steps.

The Batch LVQ Algorithm

Step 1. Choose the number of neurons c, the maximum number of iterations $t_{\max}$, a small value for the parameter $\varepsilon \in (0, 1)$, and randomly initialize the matrix $\boldsymbol{V} = \{[\boldsymbol{v}_j] :\ 1 \le j \le c\}$. Set $t = 0$.

Step 2. For each $\boldsymbol{x}_k$ $(1 \le k \le n)$ calculate the nearest codeword $\boldsymbol{v}_{i(\boldsymbol{x}_k)}$ using the condition in Eq. (3) and calculate the quantities $h_{i(\boldsymbol{x}),j}$.

Step 3. Using Eq. (8) update the codewords.

Step 4. If $\sum_{j=1}^{c} \left\|\boldsymbol{v}_j(t) - \boldsymbol{v}_j\,(t-1)\right\| < \varepsilon$ then stop. Else set $t = t+1$ and go to step 2.

If we choose Eq. (5) to calculate the quantities $h_{i(\boldsymbol{x}),j}$ at the step 2, then the whole approach yields a batch version of the standard SOM.

2.2 The Fuzzy Learning Vector Quantization Algorithm

The fuzzy learning vector quantization (FLVQ) algorithm was introduced in [32] as an alternative to the standard fuzzy c-means model. In later works [22, 33] certain

properties of FLVQ were studied. The task of FLVQ is to design an optimal VQ scheme by minimizing the subsequent objective function:

$$J(\boldsymbol{U}, \boldsymbol{V}) = \sum_{k=1}^{n} \sum_{i=1}^{c} (u_{ik})^m \|\boldsymbol{x}_k - \boldsymbol{v}_i\|^2 \tag{9}$$

where $\boldsymbol{U} = \{[u_{ik}] : 1 \leq i \leq c,\ 1 \leq k \leq n\}$ is the partition matrix, $\boldsymbol{V} = \{[\boldsymbol{v}_i] : 1 \leq i \leq c\}$ the cluster centers' matrix, and $m \in (1, \infty)$ the fuzziness parameter. The above minimization task is subjected to the constraint:

$$\sum_{i=1}^{c} u_{ik} = 1 \quad \forall k \tag{10}$$

To this end, the algorithm is summarized in the following steps.

The FLVQ Algorithm

Choose the number of clusters c, a small value for the parameter $\varepsilon \in (0, 1)$, and randomly initialize the matrix $\boldsymbol{V} = \{[\boldsymbol{v}_i] : 1 \leq i \leq c\}$. Also set the maximum number of iterations $t_{\max}$, and the initial (m_0) and final (m_f) value of the fuzziness parameter.

For $t = 0,\ 1,\ 2, \ldots,\ t_{\max}$

(a) Calculate the fuzziness parameter as:

$$m(t) = m_0 - \frac{t\left(m_0 - m_f\right)}{t_{\max}} \tag{11}$$

(b) Set:

$$\eta_{ik}(t) = \left[\sum_{j=1}^{c} \left(\frac{\|\boldsymbol{x}_k - \boldsymbol{v}_i\|}{\|\boldsymbol{x}_k - \boldsymbol{v}_j\|} \right)^{\frac{2}{m(t)-1}} \right]^{-m(t)} \tag{12}$$

(c) Update the cluster centers as:

$$\boldsymbol{v}_i(t) = \frac{\sum_{k=1}^{n} \eta_{ik}(t) \boldsymbol{x}_k}{\sum_{k=1}^{n} \eta_{ik}(t)} \tag{13}$$

(d) If error $(t) = \sum_{i=1}^{c} \|\boldsymbol{v}_i(t) - \boldsymbol{v}_i\,(t-1)\| < \varepsilon$ then stop. Else continue.

The FLVQ consists a soft learning scheme that maintains the transition from fuzzy to crisp conditions by gradually reducing the fuzziness parameter $m(t)$ from the large value m_0 (fuzzy conditions) to the value m_f which is chosen close to unity (nearly crisp conditions). Note that in the initial stages of the learning process, all of the codewords are assigned training patterns due to the existence of fuzzy conditions. Therefore, all codewords are forced to move and to become more competitive with each another. This fact makes the FLVQ less sensitive to random initialization when compared to the batch SOM and the batch LVQ. However, the FLVQ is related to two difficulties. The first one concerns the selection of appropriate values for parameters m_0 and m_f, since different values will obtain quite different partitions of the feature space. Secondly, it requires high computational efforts due to the number of distance calculations involved in Eq. (12).

3 The Proposed Vector Quantization Approach

The motivation of the proposed vector quantization algorithm is twofold. First to alleviate the strong dependence on initialization of crisp vector quantization. Second to produce a fast fuzzy learning mechanism. To do so we propose the two-level iteration depicted in Fig. 2.

The first level concerns the implementation of a novel competitive learning scheme. This level is put into practice by developing a specialized fuzzy-set-

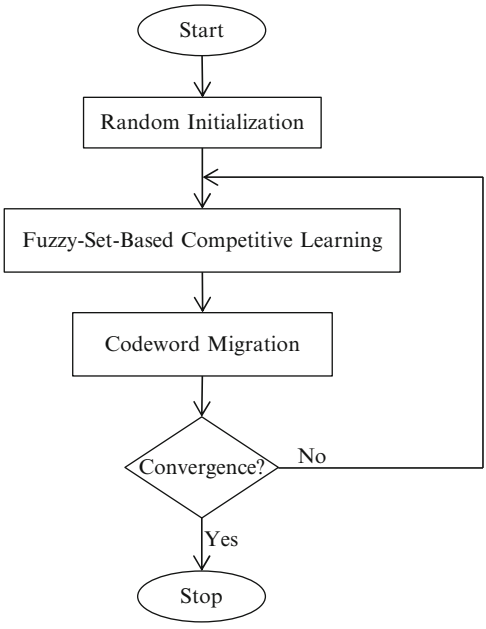

Fig. 2 The proposed fuzzy-soft learning vector quantizer

based neighborhood function able to effectively control the competition between codewords in such a way that every codeword is enabled to win as many training vectors as possible. The second level attempts to improve the partition generated by the previous level by detecting codewords that correspond to underutilized (i.e. small) clusters and relocating them to the neighborhood of large clusters. Given the nomenclature so far, the objective of the proposed vector quantizer is to minimize the distortion function:

$$D(\boldsymbol{X},\ \boldsymbol{V}) = \frac{1}{n}\sum_{k=1}^{n} \min_{1\le i\le c} \left\{ \|\boldsymbol{x}_k - \boldsymbol{v}_i\|^2 \right\} \tag{14}$$

3.1 Fuzzy-Set-Based Competitive Learning

The problem discussed in this section concerns the way the codewords must be updated so that the final learning strategy will be fast and less sensitive to initialization. Before proceeding with the mathematical formulation of our approach, let us identify how the methods described previously cope with this problem. The SOM utilizes Eq. (5) to decide which codewords should be updated, given the winner node for a specific training vector. Roughly speaking, Eq. (5) transfigures the information coded in the grid into meaningful information in the feature space. Its nature is stochastic in the sense that the existence of the Gaussian function mostly favors those codewords that correspond to nodes that are close to the winning node in the grid. However, it is highly possible for nodes that correspond to large clusters to be in the neighborhood of many winning nodes and therefore they would be continuously updated; leaving out of competition the codewords that correspond to small clusters, rendering them underutilized. Therefore, the resulting partition might not reveal the underlying structure of the data and eventually might not be efficient. This phenomenon can partially be resolved by providing the learning process with good initialization as far as the codewords are concerned. As stated in [18] such a case would be the selection of the codeword initial values from the training data set itself. Yet, even in this case, the danger of bad initial conditions is still high. The LVQ and the batch LVQ employ the condition in Eq. (6) to come up with a codeword updating decision strategy. This condition imposes an aggressive competition between the codewords. Thus, although the resulting learning is quite fast, the quality of the obtained partition is inferior. On the contrary, the FLVQ adopts Eq. (13) to update the codewords. The implementation of this equation is a direct result of the learning rate appearing in Eq. (12). Actually, this learning rate plays the role of the neighborhood function in accordance with Eq. (5) used by the SOM. As mentioned previously, we can easily observe that Eq. (12) will, indeed, force a large number of codewords to become more competitive [22, 30, 32, 33]. It thus appears that the fuzzy conditions encapsulated in this equation initiate the

generation of better partitions than the SOM and the LVQ as far as the discovery of the real data structure is concerned. However, Eq. (12) constitutes a computationally complex approach.

In our opinion, the codeword updating process should take into consideration their relative locations in the feature space. To justify this claim we concentrate on the following analysis. The codeword locations are directly affected by the distribution of the clusters across this space. With its turn, the cluster distribution across the feature space should possess two main properties [22, 33, 34]: (a) clusters should be as compact as possible, and (b) clusters should be as separated as possible with each another. The former states that data belonging to the same cluster should be as similar as possible, while the latter delineates the dissimilarity between data belonging to different clusters. Both of these properties are affected by the relative locations of the codewords (i.e. cluster centers). Therefore, the first requirement the neighborhood function must fulfill is the involvement of the Euclidean distances between pairs of codewords. In a similar fashion to the SOM and the FLVQ, the second requirement states that the codewords located closer to the winning neuron must be updated with larger rates than the ones located far away. The solution to this problem comes from the area of the optimal fuzzy clustering. Specifically, the two properties mentioned previously are the basic tools to perform optimal fuzzy cluster analysis. The neighborhood function introduced here is inspired by the separation index between fuzzy clusters developed in [34] and is mathematically described next.

Assume that we have a partition of the data set $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ into c clusters with codewords $\boldsymbol{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_c\}$. Define in sequence the arithmetic mean of the codewords:

$$\overline{\boldsymbol{v}} = \frac{1}{c}\sum_{i=1}^{c} \boldsymbol{v}_i \tag{15}$$

and the $p \times (c+1)$ matrix:

$$\boldsymbol{Z} = \left[\boldsymbol{z}_1\ \boldsymbol{z}_2 \cdots \boldsymbol{z}_c\ \boldsymbol{z}_{c+1}\right] = \left[\boldsymbol{v}_1\ \boldsymbol{v}_2 \cdots \boldsymbol{v}_c\ \overline{\boldsymbol{v}}\right] \tag{16}$$

Note that for $1 \leq i \leq c$ it holds that $\boldsymbol{z}_i = \boldsymbol{v}_i$ while $\boldsymbol{z}_{c+1} = \overline{\boldsymbol{v}}$. The need for the appearance of $\overline{\boldsymbol{v}}$ in Eq. (16) will be shortly explained. Then, we view each codeword as the center of a fuzzy set, the elements of which are the rest of the codewords. Considering the codeword $\boldsymbol{z}_i$ $(1 \leq i \leq c)$ the membership function of the associated fuzzy set is define as:

$$\mu_{ij} = \frac{1}{\sum\limits_{\substack{l=1 \\ l \neq j}}^{c+1} \left(\frac{\left\| \boldsymbol{z}_j - \boldsymbol{z}_i \right\|}{\left\| \boldsymbol{z}_j - \boldsymbol{z}_l \right\|} \right)^{\frac{2}{m-1}}} \quad (1 \leq i \leq c; 1 \leq j \leq c) \tag{17}$$

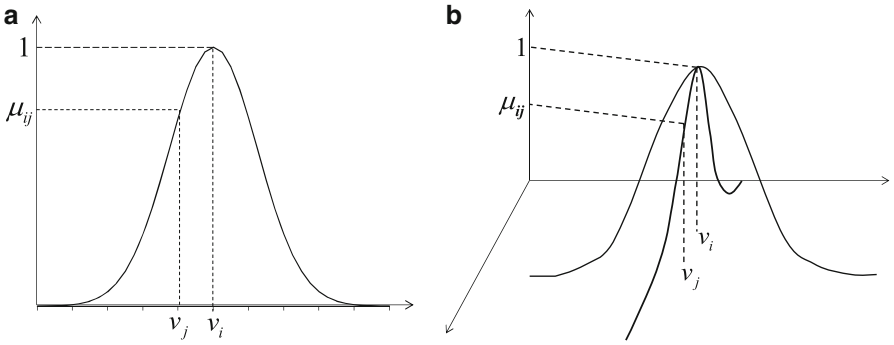

Fig. 3 The structure of the membership function μ_{ij}: (**a**) one-dimensional and (**b**) two-dimensional feature space

where $m \in (1, \infty)$ is the fuzziness parameter. Note that the indices i and j are not assigned the value $c + 1$, which corresponds to the mean $\overline{\boldsymbol{v}}$. Figure 3 illustrates what the function μ_{ij} looks like in the one-dimensional and two-dimensional Euclidean space.

Now, the need for using $\overline{\boldsymbol{v}}$ is obvious because if it's not going to be taken into account, when there are only two codewords (i.e. $c = 2$) Eq. (17) will give one in all cases and thus, it does not work. In addition, $\overline{\boldsymbol{v}}$ provides useful information for the distribution of the codewords and its presence is necessary. For each $\boldsymbol{x}_k$ we detect the nearest codeword $\boldsymbol{v}_{\ell(\boldsymbol{x}_k)}$ according to the condition:

$$\left\|\boldsymbol{x}_k - \boldsymbol{v}_{\ell(\boldsymbol{x}_k)}\right\| = \min_{1 \le i \le c} \{\|\boldsymbol{x}_k - \boldsymbol{v}_i\|\} \tag{18}$$

To this end, the neighborhood function is defined as:

$$h_{\ell(\boldsymbol{x}_k),\, i} = \begin{cases} 1, & \text{if } i = \ell\,(\boldsymbol{x}_k) \\ \mu_{\ell(\boldsymbol{x}_k),\, i}, & \text{otherwise} \end{cases} \tag{19}$$

where $\ell\,(\boldsymbol{x}_k) \in \{1,\ 2, \ldots,\ c\}$, $1 \le i \le c$, and $\mu_{\ell(\boldsymbol{x}_k),\, i}$ is calculated in (17) by changing the index i with the $\ell(\boldsymbol{x}_k)$ and the index j with i.

The main properties of the function $h_{\ell(\boldsymbol{x}_k),\, i}$ can be enumerated as follows. First, it provides the excitation degree of each codeword (with respect to the winning one) by considering the locations of the rest of codewords. Note that the decision how much each codeword is going to move is not affected only by distances between pairs but also by the relative locations of the codewords. That is, before updating a codeword, the algorithm "looks" at the overall current partition and then makes the move. Second, although the $\mu_{\ell(\boldsymbol{x}_k),\, i}$ is a fuzzy membership degree, its calculation is fast because it includes only the codewords and not the training vectors. Specifically, the number of distance calculations performed by the FLVQ is dominated by the learning rate's estimation in Eq. (12), which calculates $(c + 1)\ \ c\ n$ distances per

iteration. Since $(c+1)\ c\ n \leq n(c+1)^2$ the computational complexity of the FLVQ is $O(nc^2)$. In our case the dominant effect is provided by Eq. (17), where we can easily see that the number of distance calculations is $(c+1)^2\ c \leq (c+1)^3$ meaning that the computational complexity of the proposed algorithm is $O(c^3)$. Since $c \ll n \Rightarrow c^3 \ll nc^2$, we conclude that the proposed vector quantizer is much faster that the FLVQ, something that is supported by the simulations reported later on in this chapter.

Having introduced the neighborhood function the learning rule to update the codewords is given as:

$$\boldsymbol{v}_i(t) = \boldsymbol{v}_i\,(t-1) + \eta(t)\, h_{\ell(\boldsymbol{x}_k),\,i}\ \ (\boldsymbol{x}_k - \boldsymbol{v}_i\,(t-1)) \tag{20}$$

The above rule is an on-line LVQ scheme. As we are interested to perform a codeword migration process in the second step of the algorithm we intend to modify the above mechanism in order to produce a batch vector quantization process. In a similar way to the standard SOM model, we employ the next condition:

$$E\left[h_{\ell(\boldsymbol{x}_k),\,i}\,(\boldsymbol{x}_k - \boldsymbol{v}_i(t))\right] = 0 \quad \text{as } t \to \infty \tag{21}$$

Thus, the updating rule is modified as follows:

$$\boldsymbol{v}_i(t) = \frac{\sum_{k=1}^{n} h_{i(\boldsymbol{x}_k),\,i}^{(t-1)}\,\boldsymbol{x}_k}{\sum_{k=1}^{n} h_{i(\boldsymbol{x}_k),\,i}^{(t-1)}} \tag{22}$$

where t stands for iteration number.

3.2 Codeword Migration Process

Although the fuzzy conditions of the above competitive learning are able to effectively deal with a bad initialization, it is highly probable the resulting partition to include small and underutilized clusters. In this step of the algorithm we intend to improve the partition by incorporating into the learning process a codeword migration mechanism. As we want all clusters to contribute as much as possible, we expect that this process will yield an optimal vector quantizer [35]. The key idea is to detect small and underutilized clusters and relocate the respective codeword to an area close to a large cluster. To perform this task, we employ the distortion utility measure developed in [16],

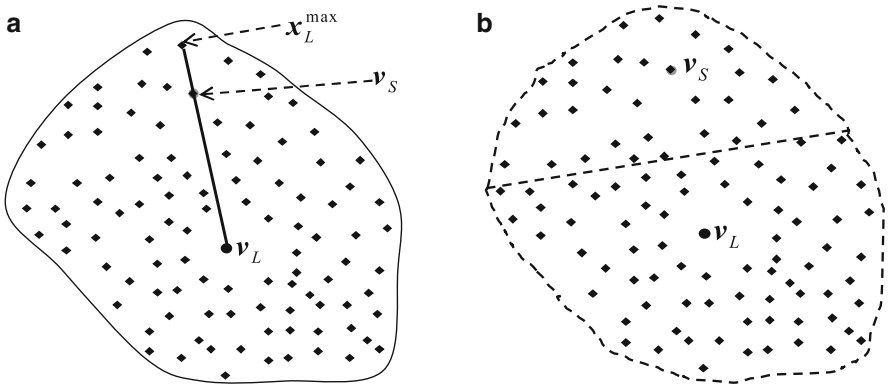

Fig. 4 (**a**) Relocation of the codeword of a small cluster to the area of a large one. (**b**) Division of the large cluster into two areas

$$\Lambda_i = \frac{D_i}{(1/c)\sum_{j=1}^{c} D_j} \tag{23}$$

where D_i is the individual distortion of the ith cluster,

$$D_i = \sum_{k=1}^{n} \psi_{ik} \, \|\boldsymbol{x}_k - \boldsymbol{v}_i\|^2 \tag{24}$$

The quantity ψ_{ik} in Eq. (24) is calculated as indicated next:

$$\psi_{ik} = \begin{cases} 1, \text{ if } \|\boldsymbol{x}_k - \boldsymbol{v}_i\| = \min\limits_{1 \le j \le c} \left\{ \|\boldsymbol{x}_k - \boldsymbol{v}_j\| \right\} \\ 0, \text{ otherwise} \end{cases} \tag{25}$$

The criterion to select a small cluster is:

$$\Lambda_i \le \gamma \tag{26}$$

with $\gamma \in (0,\ 1)$. Intuitively, in order to detect a big cluster (Fig. 4),

$$\Lambda_i > 1 \tag{27}$$

In each iteration we migrate the codeword of all small clusters detected by the condition (26) to the neighborhood of large clusters detected by the condition (27). In what follows, we shall concentrate to one small cluster denoted as C_s and one large denoted as C_L. The respective codewords are symbolized as $\boldsymbol{v}_S$ and $\boldsymbol{v}_L$. As far as the cluster C_L is concerned, we detect the training vector $\boldsymbol{x}_L^{\max} \in C_L$ such that

$\left\|\boldsymbol{x}_L^{\max} - \boldsymbol{v}_L\right\| = \max_{\boldsymbol{x} \in C_L} \{\|\boldsymbol{x} - \boldsymbol{v}_L\|\}$. The line segment that connects $\boldsymbol{x}_L^{\max}$ and $\boldsymbol{v}_L$ is:

$$\left[\boldsymbol{v}_L,\ \boldsymbol{x}_L^{\max}\right] = \boldsymbol{v}_L + \lambda\ \left(\boldsymbol{x}_L^{\max} - \boldsymbol{v}_L\right) = (1-\lambda)\,\boldsymbol{v}_L + \lambda\,\boldsymbol{x}_L^{\max} \quad \forall \lambda \in [0,\ 1] \tag{28}$$

We select a $\lambda \in (0.5,\ 1)$ and remove the $\boldsymbol{v}_S$, the location on the segment defined by this value of λ. Note that in this case the codeword $\boldsymbol{v}_S$ will be closer to the $\boldsymbol{x}_L^{\max}$ than the $\boldsymbol{v}_L$. After the migration, the training vectors of the small cluster C_S will be reassigned to their closest codewords in terms of the condition (25).

The final step of the algorithm concerns the acceptance or the rejection of the above relocation process. The relocation of a codeword is accepted if the sum of the individual distortions of the clusters C_S and C_L has been decreased [16]. To verify this, we run two iterations of the algorithm using as codebook only the codewords $\boldsymbol{v}_S$ and $\boldsymbol{v}_L$ and as training data the data belonging to the C_L [35]. Finally, all the small clusters detected in each iteration are processed in parallel.

4 Experimental Study

To test the efficiency of the proposed method we compare it with the batch LVQ and the FLVQ. The experimental data consisted of the well-known Lena, Girlface, Peppers, and Airplane grayscale images of size 512×512 pixels, which are showed in Fig. 5.

For each simulation we run all algorithms using the same initialization for each codebook size and for each image. We used ten different initial conditions for each codebook size and for each image. To carry out the experiments, each image was divided in 4×4 blocks resulting in 16,384 training vectors in the 16-dimensional feature space. The performances of the algorithms were evaluated in terms of the distortion measure given in Eq. (14) and the peak signal-to-noise ratio (PSNR):

$$\text{PSNR} = 10 \log_{10} \left(\frac{512^2\, 255^2}{\sum_{i=1}^{512} \sum_{j=1}^{512} \left(I_{ij} - \widehat{I}_{ij}\right)^2} \right) \tag{29}$$

where 255 is the peak grayscale signal value, I_{ij} denotes the pixels of the original image, and $\hat{I}_{ij}$ the pixels of the reconstructed image. The neighborhood function used by batch LVQ is the one calculated in Eq. (6). For the FLVQ the fuzziness parameter was gradually decreasing from $m_0 = 2$ to $m_f = 1.1$. In all simulations, the threshold in (26) was fixed to $\gamma = 0.5$ and the value λ in (28) set equal to $\lambda = 0.75$. The following five subsections present a thorough experimental investigation of the overall behavior of the algorithm.

Fig. 5 Testing images. (**a**) Airplane; (**b**) Girlface; (**c**) Peppers; (**d**) Lena

4.1 Study of the Behavior of the Distortion Measure and the PSNR

Herein, we study the distortion measure in Eq. (14) and the PSNR in (29). We choose to run the three algorithms for codebooks of sizes $c = 2^{qb}$ ($qb = 5,\ 6, \ldots,\ 10$). Since each feature vector represented a block of 16 pixels, the resulting compression rate was equal to $qb/16$ per pixel (bpp).

Tables 1, 2, 3, and 4 report the resulting mean values of the distortion measure for the four images and various codebook sizes. In all cases, our algorithm obtains the smallest distortion value.

Table 1 Distortion mean values for the Lena image

Method	$c = 32$	$c = 64$	$c = 128$	$c = 256$	$c = 512$	$c = 1024$
Batch LVQ	1548.68	1380.63	1161.19	975.22	945.53	862.75
FLVQ	1307.60	1039.02	864.71	662.24	566.09	486.51
Proposed	1260.10	977.66	770.53	602.67	479.56	415.32

Table 2 Distortion mean values for the Airplane image

Method	$c = 32$	$c = 64$	$c = 128$	$c = 256$	$c = 512$	$c = 1024$
Batch LVQ	2478.09	1888.24	1503.29	1361.53	1222.03	1098.98
FLVQ	2130.63	1654.86	1263.81	1046.38	796.68	592.37
Proposed	1968.54	1466.41	1152.49	903.61	684.63	502.49

Table 3 Distortion mean values for the Girlface image

Method	$c = 32$	$c = 64$	$c = 128$	$c = 256$	$c = 512$	$c = 1024$
Batch LVQ	1335.62	986.85	831.09	790.16	734.80	737.29
FLVQ	1224.27	971.99	819.16	662.19	543.95	470.33
Proposed	1215.94	936.52	761.67	620.03	485.47	393.55

Table 4 Distortion mean values for the Peppers image

Method	$c = 32$	$c = 64$	$c = 128$	$c = 256$	$c = 512$	$c = 1024$
Batch LVQ	1561.34	1449.77	1300.15	1170.87	1080.21	1021.44
FLVQ	1641.61	1251.23	1011.29	812.79	714.38	625.69
Proposed	1556.65	1184.17	923.01	739.87	599.65	479.78

Table 5 PSNR mean values (in dB) using various codebook sizes for the Lena image

Method	$c = 32$	$c = 64$	$c = 128$	$c = 256$	$c = 512$	$c = 1024$
Batch LVQ	28.2685	28.7669	29.5177	30.2749	30.4087	30.8068
FLVQ	29.0023	30.0011	30.7963	31.9534	32.6327	33.2892
Proposed	29.1658	30.2688	31.3061	32.3720	33.3953	34.0400

The quality of the reconstructed image was evaluated using the PSNR. Tables 5, 6, 7, and 8 summarize the mean PSNR values. The results reported in these tables are highly convincing, since in all cases the proposed algorithm outperformed the others.

Table 6 PSNR mean values (in dB) using various codebook sizes for the Airplane image

Method	$c = 32$	$c = 64$	$c = 128$	$c = 256$	$c = 512$	$c = 1024$
Batch LVQ	26.2282	27.4084	28.3985	28.8280	29.2963	29.7567
FLVQ	26.8844	27.9811	29.1497	29.9695	31.1513	32.4366
Proposed	27.2284	28.5061	29.5516	30.6073	31.8102	33.1505

Table 7 PSNR mean values (in dB) using various codebook sizes for the Girlface image

Method	$c = 32$	$c = 64$	$c = 128$	$c = 256$	$c = 512$	$c = 1024$
Batch LVQ	28.9102	30.2244	30.9678	31.1870	31.5028	31.4873
FLVQ	29.2930	30.2899	31.0299	31.9536	32.8061	33.4358
Proposed	29.3181	30.4528	31.3540	32.2484	33.3219	34.2805

Table 8 PSNR mean values (in dB) using various codebook sizes for the Peppers image

Method	$c = 32$	$c = 64$	$c = 128$	$c = 256$	$c = 512$	$c = 1024$
Batch LVQ	28.2337	28.5550	29.0272	29.4821	29.8316	30.0748
FLVQ	28.0167	29.1943	30.1177	31.0652	31.6248	32.1986
Proposed	28.2515	29.4383	30.5170	31.4821	32.3985	33.3924

4.2 Computational Demands

In this experiment we employ the Lena image. To perform the simulation, we quantify the computational demands of the batch LVQ, the FLVQ, and the proposed algorithm. We use the Lena image to generate codebooks of sizes $c = 2^{qb}$ ($qb = 5, 6, \ldots, 11$) and measure the time needed by the CPU in seconds per iteration.

All algorithms are implemented on a computer machine with dual-core 2.13GHz CPU, 4GB RAM, and the Matlab software under MS Windows 7. The results are illustrated in Fig. 6. Notice that the FLVQ exhibits an exponential growth in computational time while, as expected, the batch LVQ obtains the faster response. On the other hand the proposed algorithm maintains quite low computational efforts, which can be seen to be much better than the respective efforts of the FLVQ.

4.3 Study of the Migration Strategy

In this simulation experiment the distribution of the utility measures achieved by the three algorithms are compared by considering again the Lena image with codebook of size $c = 256$.

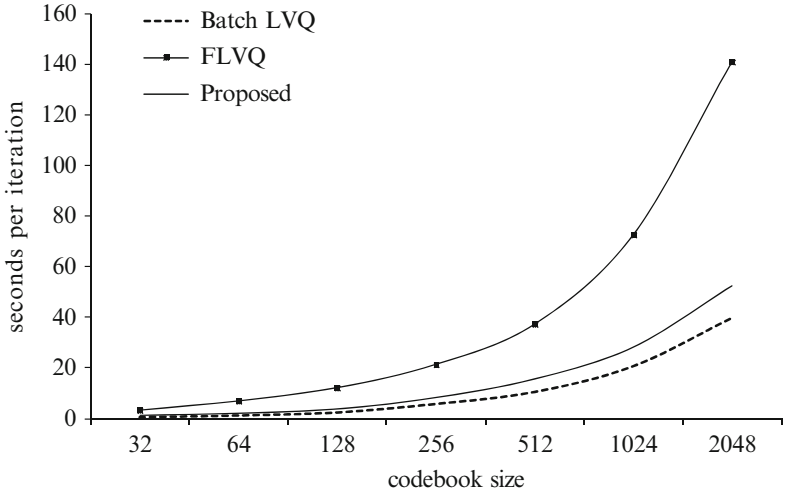

Fig. 6 Computational time in seconds per iteration as a function of the codebook size for the Lena image

Table 9 PSNR values (in dB) for the results of Fig. 7

Method	Batch LVQ	FLVQ	Proposed
PSNR	30.1920	31.6743	32.3522

Figure 7 shows the obtained distributions. The corresponding PSNR values are reported in Table 9. From the figure we can easily verify that our method provides a large percentage amount of utilities close to 1. Moreover, the distributions produced by the batch LVQ appear to have the largest deviations allowing clusters with utilities greater than 3, while the largest amount of clusters are assigned utilities much less than 1 (close to zero). On the other hand, although the FLVQ managed to reduce this variance, it creates a large number of clusters that are assigned very small utilities, something that is not desirable. The obvious conclusion of the figure is that the proposed method obtained the best distribution. To conclude, the codeword relocation approach manages to substantially increase the size of small clusters yielding an effective topology of the final partition.

4.4 Literature Comparison

In this section, we present a comparison between the best performances obtained here and the respective performances of other methods as they are reported in the corresponding references.

The comparison is based on the Lena image because it is the most used testing image in the literature.

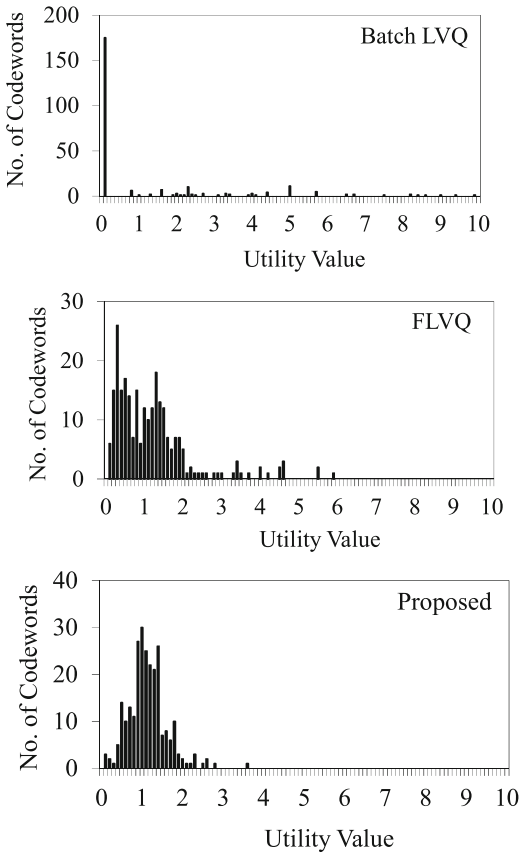

Fig. 7 Distributions of the distortion utility measures for $c = 256$

Table 10 Literature comparison results for different codebook sizes in terms of the PSNR for the Lena image

Method	$c = 32$	$c = 128$	$c = 256$	$c = 512$
Chen's model [2]	–	28.530	29.320	30.040
Hu et al. [5]	–	30.802	31.983	32.956
IFLVQ [30]	28.752	–	31.915	32.890
Proposed algorithm	29.208	31.342	32.493	33.418

Table 10 depicts the comparative results. As far as this table indicates, the proposed algorithm obtains the best results in all cases.

5 Conclusions

We have proposed a fuzzy-soft competitive LVQ for grayscale image compression. The proposed algorithm scheme combines the merits of the crisp-based and fuzzy-based methodologies into a unique framework, by utilizing two sequential steps. The first step defines a fuzzy-based neighborhood function which it is able to measure

the lateral neuron interaction phenomenon and the degree of the neuron excitations. The second step involves a codeword migration strategy that is based on a cluster utility measure. Specifically, we detect clusters with small utilities and relocate the corresponding codewords to areas that appear high probability to contain large number of training vectors. This strategy increases the individual contributions of the resulting codewords to the final partition. The performance of the algorithm was evaluated in terms of the computational demands and the quality of the reconstructed image. Several simulation experiments took place, where the algorithm proved to be very fast and competitive to other related approaches.

References

1. Chang, C.C., Lin, I.C.: Fast search algorithm for vector quantization without extra look-up table using declustered subcodebooks. IEE Proc. Vis. Image Sign. Process. Lett. **7**(11), 304–306 (2005)
2. Chen, P.Y.: An efficient prediction algorithm for image quantization. IEEE Trans. Syst. Man Cybern. Part B Cybern. **34**(1), 740–746 (2004)
3. Chen, T.-H., Wu, C.-S.: Compression-unimpaired batch-image encryption combining vector quantization and index compression. Inform. Sci. **180**, 1690–1701 (2010)
4. Horng, M.-H.: Vector quantization using the firefly algorithm for image compression. Expert Syst. Appl. **39**, 1078–1091 (2012)
5. Hu, Y.-C., Su, B.-H., Tsou, C.-C.: Fast VQ codebook search algorithm for grayscale image coding. Image Vis. Comput. **26**, 657–666 (2008)
6. Poursistani, P., Nezamabadi-pour, H., Askari Moghadam, R., Saeed, M.: Image indexing and retrieval in JPEG compressed domain based on vector quantization. Math. Comput. Modell. **57**, 1005–1017 (2013)
7. Qian, S.E.: Hyper spectral data compression using a fast vector quantization algorithm. IEEE Trans. Geosci. Remote Sens. **42**(8), 1791–1798 (2004)
8. De, A., Guo, C.: An adaptive vector quantization approach for image segmentation based on SOM network. Neurocomputing **149**, 48–58 (2015)
9. Kirstein, S., Wersing, H., Gross, H.-M., Körner, E.: A life-long learning vector quantization approach for interactive learning of multiple categories. Neural Netw. **28**, 90–105 (2012)
10. Pham, T.D., Brandl, M., Beck, D.: Fuzzy declustering-based vector quantization. Pattern Recognit. **42**, 2570–2577 (2009)
11. Rizzo, F., Storer, J.A., Carpentieri, B.: Overlap and channel errors in adaptive vector quantization for image coding. Inform. Sci. **171**, 125–140 (2005)
12. Villmann, T., Haase, S., Kaden, M.: Kernelized vector quantization in gradient-descent learning. Neurocomputing **147**, 83–95 (2015)
13. Yan, S.B.: Constrained-storage multistage vector quantization based on genetic algorithms. Pattern Recognit. **41**, 689–700 (2008)
14. Zhou, S.S., Wang, W.W., Zhou, L.H.: A new technique for generalized learning vector quantization algorithm. Image Vis. Comput. **24**, 649–655 (2006)
15. Linde, Y., Buzo, A., Gray, R.M.: An algorithm for vector quantizer design. IEEE Trans. Commun. **28**(1), 84–95 (1980)
16. Patane, G., Russo, M.: The enhanced LBG. Neural Netw. **14**, 1219–1237 (2001)
17. Kohonen, T.: Self-Organizing Maps, 3rd edn. Springer, Berlin (2001)
18. Kohonen, T.: The self-organizing map. Neurocomputing **2**, 1–6 (2003)
19. Uchino, E., Yano, K., Azetsu, T.: A self-organizing map with twin units capable of describing a nonlinear input-output relation applied to speech code vector mapping. Inform. Sci. **177**, 4634–4644 (2007)

20. D'Ursoa, P., De Giovanni, L., Massari, R.: Self-organizing maps for imprecise data. Fuzzy Sets Syst. **237**, 63–89 (2014)
21. Wang, C.-H., Lee, C.-N., Hsieh, C.-H.: Classified self-organizing map with adaptive subcodebook for edge preserving vector quantization. Neurocomputing **72**, 3760–3770 (2009)
22. Bezdek, J.C., Pal, N.R.: Two soft relatives of learning vector quantization. Neural Netw. **8**, 729–743 (1995)
23. Filippi, A.M., Jensen, J.R.: Fuzzy learning vector quantization for hyperspectral coastal vegetation classification. Remote Sens. Environ. **100**, 512–530 (2006)
24. Feng, H.-M., Chen, C.-Y., Ye, F.: Evolutionary fuzzy particle swarm optimization vector quantization learning scheme in image compression. Expert Syst. Appl. **32**, 213–222 (2007)
25. Hung, W.-L., Chen, D.-H., Yang, M.-S.: Suppressed fuzzy-soft learning vector quantization for MRI segmentation. Artif. Intell. Med. **52**, 33–43 (2011)
26. Karayiannis, N.B., Randolph-Gips, M.M.: Soft learning vector quantization and clustering algorithms based on non-Euclidean norms: single-norm algorithms. IEEE Trans. Neural Netw. **16**(2), 423–435 (2005)
27. Tsekouras, G.E., Dartzentas, D., Drakoulaki, I., Niros, A.D.: Fast fuzzy vector quantization. In: Proceedings of IEEE International Conference on Fuzzy Systems, Barcelona, Spain (2010)
28. Tsolakis, D., Tsekouras, G.E., Tsimikas, J.: Fuzzy vector quantization for image compression based on competitive agglomeration and a novel codeword migration strategy. Eng. Appl. Artif. Intell. **25**, 1212–1225 (2011)
29. Tsekouras, G.E.: A fuzzy vector quantization approach to image compression. Appl. Math. Comput. **167**, 539–560 (2005)
30. Tsekouras, G.E., Mamalis, A., Anagnostopoulos, C., Gavalas, D., Economou, D.: Improved batch fuzzy learning vector quantization for image compression. Inform. Sci. **178**, 3895–3907 (2008)
31. Wu, K.-L., Yang, M.-S.: A fuzzy-soft learning vector quantization. Neurocomputing **55**, 681–697 (2003)
32. Tsao, E.C.K., Bezdek, J.C., Pal, N.R.: Fuzzy Kohonen clustering networks. Pattern Recognit. **27**(5), 757–764 (1994)
33. Pal, N.R., Bezdek, J.C., Hathaway, R.J.: Sequential competitive learning and the fuzzy c-means clustering algorithms. Neural Netw. **9**(5), 787–796 (1996)
34. Tsekouras, G.E., Sarimveis, H.: A new approach for measuring the validity of the fuzzy c-means algorithm. Adv. Eng. Software **35**, 567–575 (2004)
35. Tsolakis, D., Tsekouras, G.E., Niros, A.D., Rigos, A.: On the systematic development of fast fuzzy vector quantization for grayscale image compression. Neural Netw. **36**, 83–96 (2012)
36. Celebi, M.E., Kingravi, H., Vela, P.A.: A comparative study of efficient initialization methods for the K-means clustering algorithm. Expert Syst. Appl. **40**(1), 200–210 (2013)
37. Celebi, M.E., Kingravi, H.: Deterministic initialization of the K-means algorithm using hierarchical clustering. Int. J. Pattern Recognit. Artif. Intell. **26**(7), 1250018 (2012)