Searching similar images - Vector Quantization with S-tree

Jan Platos, Pavel Kromer, Vaclav Snasel, Ajith Abraham
Department of Computer Science, FEECS
IT4 Innovations, Centre of Excellence
VSB-Technical University of Ostrava
17. listopadu 15, 708 33, Ostrava Poruba, Czech Republic
{*jan.platos,pavel.kromer,vaclav.snasel*}*@vsb.cz, ajith.abraham@ieee.org*

***Abstract*—Searching of similar pictures was in the past based mainly on searching of similar picture names. We try to find an effective method how to search pictures by searching of similar information in the picture (histograms, shapes, blocks,). There already are some methods but still not effective enough. In this paper we describe a method where we combine vector quantization (VQ) and fuzzy S-trees. Work contains testing of our approach and you can see results in a final chapter of this paper. The benefit of this work is not the final solution but we put a key-stone for further research and for optimizations. First tests show up the efficiency and usefulness of our approach, which is under laid by executed tests.**

***Keywords*-vector quantization, image similarity, s-tree, fuzzy sets**

I. Introduction

Nowadays we still don't have an effective method for searching of similar pictures. Basically, there are two ways how to search similar pictures. First, we may search a similar name of picture; second we try to exploit image information from the picture, eventually combination of these methods.

If we search just by picture name, we have to demand on the correct label of a picture which also clearly describes what is in the picture. This is never guaranteed.

That's why we focus on an alternative method. There already are some methods using information in an image instead of name but in some cases they fail.

As a first step we give a reference picture to the algorithm and algorithm tries to find the most similar picture(s). Algorithm may use color information (histogram) or image may be spread out into vectors and we compare those vectors. The method where we compare vectors is called vector quantization. Vector quantization was used in the past several times. Vector quantization was used for comparison of image thumbnails in [4]. A comparison of the vector quantization codebook was used for image retrieval in [9], Teng [11] uses vector quantization fro image indexing and retrieval with preservation of pixel position and Rahman et al. [8] uses vector quantization in fuzzy feature space for Biomedical image retrieval.

II. Vector Quantization (VQ)

Vector quantization has been used for image compression for many years. In this section, we will briefly review the basic concepts of VQ image compression.

In most image compression techniques, the actual quantization or coding is done on scalars (e.g. on individual real-value samples of waveforms or pixels of images). Transform coding does it by first taking the block transform for a block of pixels and then individually coding the transform coefficients. Predictive coding does it by quantizing an error term formed as the difference between the new sample and a prediction of the new sample based on pasted coded outputs [11].

A fundamental result of Shannon's rate-distortion theory, the branch of information theory devoted to data compression, is that better performance can always be achieved by coding vectors (a group of values) instead of scalar (individual value). Thus, vector quantization can successfully be used for image and audio compression. A vector quantizer can be defined as a mapping Q of K-dimensional Euclidean space R^K into a finite subset Y of R^K, that is

$$Q : R^K \rightarrow Y$$

where $Y = (x'i; i = 1, 2, \ldots, N)$, and $x'i$ is the i-th vector in Y . Y is the set of reproduction vectors and is called a VQ codebook or VQ table. N is the number of vectors in Y. At the encoder, each data vector x belonging to R^K is matched or approximated with a codeword in the codebook and the address or index of that codeword is transmitted instead of the data vector itself. To find the best match codeword for a data vector, we can use Euclidean or Manhattan distance.

III. Structure of S-Tree

Basic Concept of S-Tree: In the dynamic office environment retrieval and insertions are frequent operations. A self-organizing tree (S-Tree) has been proposed as a means for solving problems by imposing tree-structured constraint on the solution. Applying the indexed signature approach here, means to store the signatures in the order the associated objects are inserted. That is, new signatures are entered into the last partially filled node, and OR-ed to the covering signatures above [7]. Although suggests supporting the expectation of many insert operations by

978-1-4673-4792-1

partially filling the nodes at creation time, performance may decrease because the signature tree is designed as a static structure. Moreover, a frequent processing of a time consuming periodic reorganization should be avoided in the dynamic office environment. Often new inserted signatures would not be inserted into the appropriate leaf. Here, 'appropriate' means the leaf where similar signatures are stored. 'Similar signatures' are signatures with many set ones in same positions [2].

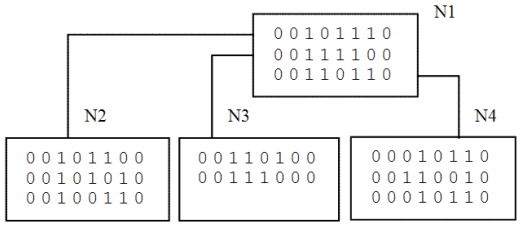

Figure 1. S-Tree - overlay signatures

Definition: Similar to a B+-tree, an S-tree is a height balanced multiway tree, whose index part is managed like a B tree [1]. Each node corresponds to a page. The leaf nodes contain either the objects or *Object identifiers* (Oid). The former case we call an immediate S-tree, the latter a mediate S-tree. The leaves of a mediate S-tree contain entries of the form $\langle s, Oid \rangle$ where the object is accessed by the *Oid*. The signature s is generated by applying an appropriate hash transformation on the object's attribute values, which maps them into a bit string $s = b1|b2|\ldots|bL$ of fixed length L with $bi \in 0, 1$. A signature in a non-leaf node is defined by superimposing the signatures contained in its son node (via the signature operator σ). Therefore, entries E in non-leaf nodes have the form $\langle s, p \rangle$ with the property

$$s = s(N(p)) := (\{E.s | E \in N(p)\}) := \sum_{E \in N(p)} E.s$$

where $N(p)$ refers to the node p and $E.s$ denotes the signature component of an entry E. Now we can define a mediate S-tree of the type (K, k, h), where $K, k, h \in N_0$, with the following properties:

1) Each path from the root to any leaf hat the same length h (height).
2) The root has at least 2 and at most K sons unless it is a leaf.
3) Every node except the root has at least k and at most K sons.
4) The signatures contained in each non-leaf node are minimal.

IV. Fuzzy sets

Fuzzy set A is defined in terms of a relevant universal set, X, by a function analogous to the characteristic function. This function called a *membership function*, assigns to each element $x \in X$ a number, $A(x)$, in the closed unit interval $[0, 1]$ that characterizes the degree of membership of x in A. Membership functions are thus functions of the form

$$A : X \to [0, 1]$$

In defining a membership function, the universal set X is always assumed to be a classical set [5].

A. Fuzzy S-tree

We can obtain a data structure for the storing of fuzzy signatures by a modification of the S-tree.

Fuzzy signatures of images will be stored in leaf pages rather than ordinary signatures. In the non-leaf pages there will be fuzzy signature in a non-leaf page will correspond to another page at the lover level. These signatures are created as disjunctions of all fuzzy signatures in the corresponding pages [10].

B. Fuzzy S-tree operations

In our fuzzy S-tree we use these following basic fuzzy sets operations:

Conjunction of fuzzy signatures: The conjunction of fuzzy signatures F_i and F_j is the fuzzy signature

$$F_i \wedge F_j = (f_{i1} \wedge f_{j1}, f_{i2} \wedge f_{j2}, \ldots, f_{in} \wedge f_{jn})$$

The operation is defined for all elements of the fuzzy signature as

$$f_{ir} \wedge f_{jr} = min\{f_{ir}, f_{jr}\}$$

Disjunction of fuzzy signatures: The disjunction of fuzzy signatures Fi and Fj is the fuzzy signature

$$F_i \vee F_j = (f_{i1} \vee f_{j1}, f_{i2} \vee f_{j2}, \ldots, f_{in} \vee f_{jn})$$

The operation is defined for all elements of the fuzzy signature as

$$f_{ir} \vee f_{jr} = min\{f_{ir}, f_{jr}\}$$

V. The proposed method

In this section we describe the approach how we search similar images in a collection. Motivation of this method is to improve current methods that use vector quantization.

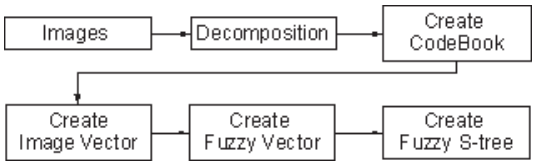

Figure 2. Proposed scheme

The very first step is to get the set of vectors representing a picture. We get these vectors by dividing an image into square blocks. Each block is a composite of explicit number of pixels. It's possible to get vectors not only from square blocks but also from another shape. Image may be divided into triangles, rhomboids, etc. Each block give us 3 different vectors: one vector represents red color component, one is for green color and one for blue color. It's possible to reduce the color information and obtain only one vector representing the block of picture in gray scale color.

Depending on a block size we get vectors of appropriate size. We typically use blocks 2x2, 4x4 and 8x8 which produce vector sizes 4, 16 and 64. This is a method how we get three sets of vectors which represents each image. We repeat this decomposition for each image which produces the multi-set for each color.

These sets are huge and that's why we choose only those vectors which are typical for our image. Selection of appropriate vectors is realized by LBG algorithm (Linde-Buzo-Grey) [6]. So far, several codebook design algorithms had been proposed to design the VQ codebooks. Among them, the LBG algorithm is the most commonly used method for codebook design.

It's indeed possible to use any other algorithm (Data clustering [3], K-means algorithm, Lloyd's algorithm, etc.). We implemented LBG with random initialization. This algorithm selects typical vectors and this leads to creating our codebook for each color (RGB). It is necessary to choose the right size of codebook. Larger codebook size leads to higher accuracy but it may take too much memory.

When we have a codebook we are able to create one unique vector for each image using algorithm for vector quantization (VQ). We implemented FSVQ algorithm which take one train vector and search the most similar vector in a codebook. Codeword index is put into mentioned vector. Then this vector exactly specifies each image. Vector size is given by number of blocks in an image.

Because these vectors are huge we decided to reduce them.

Example:: If we have an image about size 300x200 pixels and we have block size set up on 2x2, our codebook has size 512. We can count, how much space we need for image vector and fuzzy vector in memory.

Image:	300 x 200 = $60000 \div 4$ = 15000 blocks
Image vector:	15 000 x 4B = 60kB
Fuzzy vector:	512 x 4B = 2kB

We transformed each image vector to fuzzy vector. First we create en empty fuzzy vector of size of codebook. First item in vector represents the number of presence of first codeword in codebook (CW), the second item represents the number of presence of second codeword. We will repeat this technique for all items of fuzzy vector.

So, this fuzzy vector has a size of codebook and contains number of codewords' presence in an image vector.

Fuzzy vector contains fuzzy numbers. Precondition for this transformation is to have images of the same size because these will have the same count of train vectors. Whereas each image has the same count of train vectors we can consider codewords' presence as fuzzy numbers.

Fuzzy number is a real number range form 0 to 1.

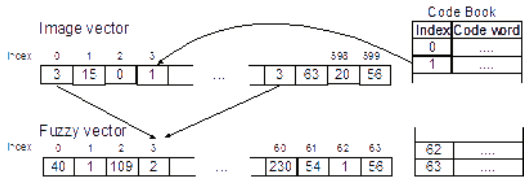

Figure 3. Image vector to fuzzy vector

Example in Fig. 3 shows the image vector of size 600 blocks and fuzzy vector of size 64. Transformation to fuzzy number is realized as

$$X = CW/CT : (X \subseteq R | X \langle 0, 1 \rangle)$$

where X is fuzzy number, CW is number of codewords in image vector and CT is number of train vectors.

So that, we may understand the first item of fuzzy vector as a fuzzy number, e.g. $40/600 = 0,067$. Because it is easier and faster to work with integers rather then real number, we decided to imagine fuzzy number X as CW/CT.

A. Fuzzy S-tree

Our method is designed for searching in very large collection of pictures so we had to choose some appropriate 'storage' for our vectors. As a very good structure we choose B+ trees first but these structures are not designed for storing vectors. The other and very similar structure is the S-Tree. We just had to redesign the algorithm for saving real numbers instead of just binary numbers which was the original implementation. Then we could easily store our fuzzy vectors into this modified S-Tree structure. Fuzzy S Tree features are very close to the original S-Tree. Differences are in these features:

1) We use fuzzy numbers $\langle 0, 1 \rangle$ in signatures instead of 0 or 1.
2) After inserting of fuzzy signatures we overlay signatures by disjunction of fuzzy signatures (4.2) instead of logical disjunction which was originally used in S-tree.
3) For searching of signatures we use conjunction of fuzzy signatures (4.2) instead of logical conjunction which was originally used in S-tree.
4) Original S-tree uses Hamming distance which is defined only for vectors $[0, 1]$. Because we use real numbers we had to use Euclidean distance for signature similarity detection

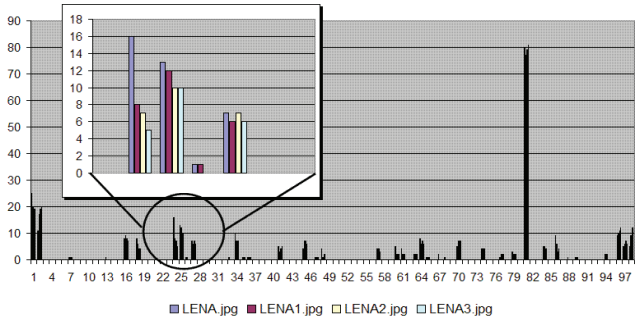

Figure 4. Vector of codewords in image

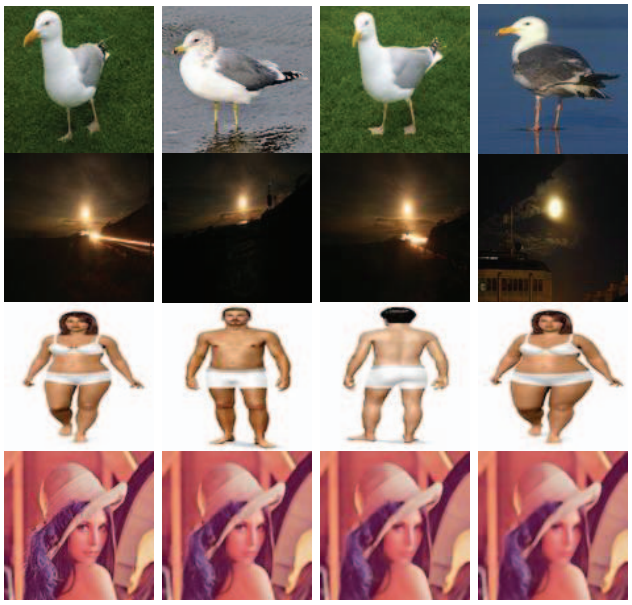

Figure 5. Image collection sample

$$Q = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

VI. Experimental results

We were testing two basic aspects of our method. Because we combined vector quantization and S-trees we tested each part independently. We tested many sets of pictures in collection of size of 100 pictures. In this paper we show four representative sets that have some typical results. The first set is a classic picture of Lena which is rustled if four steps. Then we show body collection, birds and moon collection. We have 4 set of test pictures shown in Fig 5.

Each picture in collection is spread out into vectors that are consequently compared among each other based on Euclidean's distance. Individual vectors look like vector in Fig. 4. Each column in graph represents the number of repetition of codeword in the picture. This graph shows four vectors for picture of Lena and we want to take a notice of similarities between them. From this graph is evident that some codewords haven't even occur in a picture (e.g. positions 25 and 29). This happened because the codebook is created for the whole collection of pictures. At the position 26, one of the pictures has much higher value of repetition than other pictures, so Euclidean distance will be longer than e.g. codeword on a position 28. We can easily see by naked eye that all values here are very similar so Euclidean distances will be similar too. That means that also pictures in this spot are similar.

If we compare all vectors, we get values representing distances between all individual pictures.

Tests are conceived to we could be able to compare a fruitfulness of searching of similar pictures using vector quantization with and without using fuzzy S-trees.

So let's first look at the quality of vector quantization.

A. Vector quantization testing

After some tests we found the best settings for codebook, block size and threshold as following:

- 8 - block size
- 512 - book size
- 0.01 - threshold

S-tree node was set to the size 1000 because we wanted to have all pictures in one node for VQ testing.

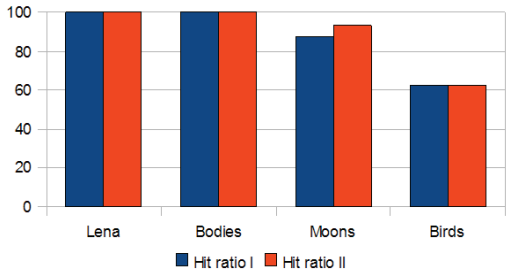

Figure 6. Vector quantization hit rate

As you can see in graphs and tables bellow, results of collections of Lena and Bodies were very good. In both collections we found all pictures from corresponding collection. Collections of moons and birds sometimes faulted, so we obtained pictures that were not in a collection, but we have to mention that there were sometimes big differences between pictures even in one collection. Fig.6 shows results of measurement where we measured hit ratio. Hit ratio I column is preciseness for pictures from test pictures. 100% means we found exactly mentioned 4 test pictures. Hit ratio II column show preciseness of searching when we found some picture that is not in test pictures but is still similar to those pictures (for example some other similar bird when we were searching birds).

B. Fuzzy S-Tree testing

For searching of similar pictures in fuzzy S-tree is necessary to chose the right size of the node in this tree. This size should be chosen according to size of the whole collection. If we would choose too small size of node, fuzzy S-tree will have too many nodes and cause its overlying. We choose the size of node to value 8 and other properties were set as following:

- 8 - block size
- 512 - book size
- 0.01 - threshold

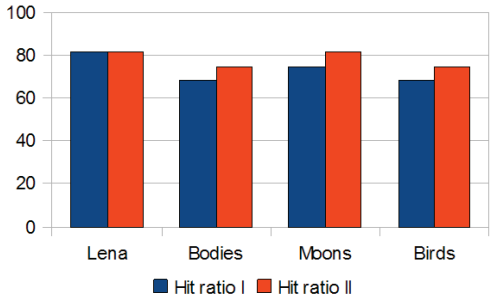

Figure 7. VQ with fuzzy S-tree hit rate

Now, let's look at results of second testing. As you can see in graphs and tables bellow, results of collections of Lena and Moon are still good enough but fruitfulness is not already excellent. In both collections we found almost all pictures from corresponding collection. Collections of Bodies and Birds faulted more frequently. With this method we achieved average fruitfulness about 75%. Hit ratio II column in Graph 7 show preciseness of searching when we found some picture that is not in test pictures but is still similar to those (Moons).

Now we can compare hit rate of searching similar images with and without using fuzzy S-Tree.

C. Comparation

Now let's compare methods 6.1 and 6.2. If we look at Graph 1. and Graph 2. we can deduce following conclusions. Test collections (Birds, Bodies, Moons and Lena) have worse hit rate of searching but fruitfulness of searching is still very good (75%). In comparison to method 6.1 is a fruitfulness just about 10 - 15% lower. It is caused by dividing of the node in S-tree and using of overlying. Even if using of S-tree is not as reliable as without S-tree, it is much faster in large collections.

VII. Conclusion

In our work we present a method for searching similar pictures using vector quantization and fuzzy S-trees. Fuzzy S-tree is actually modified data structure of S tree described in work of U. Depish [2]. Our approach for searching of similar pictures appears as a very effective method which can be used in a practice. This is based on executed experimental tests. Total fruitfulness of searching by our method is basically from range 75-100% which means that more than 75% of found pictures are exactly pictures form our small set of similar pictures. In the future it's possible to do some optimizations in both phases - phase of vector quantization and phase of saving fuzzy vectors into S-tree. This work confirmed our premise that using of S-trees maybe effective for searching in large collections and it does have a sense to deal with it. This work also gives a good base for further optimizations and further research.

References

[1] R. Bayer and E. McCreight. Organisation and maintenance of large ordered indexes. *Acta Informatica*, 1:173 – 189, 1972.

[2] U. Deppisch. S-tree: a dynamic balanced signature index for office retrieval. In *Proceedings of the 9th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '86, pages 77–87, New York, NY, USA, 1986. ACM.

[3] J. Hartigan. *Clustering algorithms*. Wiley series in probability and mathematical statistics: Applied probability and statistics. Wiley, 1975.

[4] V. R. Khapli and A. S. Bhalchandra. Compressed domain image retrieval using thumbnails of images. *Computational Intelligence, Communication Systems and Networks, International Conference on*, 0:392–396, 2009.

[5] G. Klir, U. Clair, and B. Yuan. *Fuzzy set theory: foundations and applications*. Prentice Hall, 1997.

[6] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *Communications, IEEE Transactions on*, 28(1):84 – 95, jan 1980.

[7] J. L. Pfaltz, W. J. Berman, and E. M. Cagley. Partial-match retrieval using indexed descriptor files. *Commun. ACM*, 23(9):522–528, Sept. 1980.

[8] M. M. Rahman, S. K. Antani, and G. R. Thoma. Biomedical image retrieval in a fuzzy feature space with affine region detection and vector quantization of a scale-invariant descriptor. In *Proceedings of the 6th international conference on Advances in visual computing - Volume Part III*, ISVC'10, pages 261–270, Berlin, Heidelberg, 2010. Springer-Verlag.

[9] G. Schaefer. Compressed domain image retrieval by comparing vector quantization codebooks. In C. C. J. Kuo, editor, *VCIP*, volume 4671 of *Proceedings of SPIE*, pages 959–966. SPIE, 2002.

[10] V. Snásel. Fuzzy signatures for multimedia databases. In *Proceedings of the First International Conference on Advances in Information Systems*, ADVIS '00, pages 257–264, London, UK, UK, 2000. Springer-Verlag.

[11] S. W. Teng and G. Lu. Image indexing and retrieval based on vector quantization. *Pattern Recognition*, 40(11):3299 – 3316, 2007.