Fast and Memory Efficient Implementation of the Exact PNN

Pasi Fränti, Timo Kaukoranta, Day-Fann Shen, and Kuo-Shu Chang

***Abstract*—Straightforward implementation of the exact pairwise nearest neighbor (PNN) algorithm takes $O(N^3)$ time, where N is the number of training vectors. This is rather slow in practical situations. Fortunately, much faster implementation can be obtained with rather simple modifications to the basic algorithm. In this paper, we propose a fast $O(\tau N^2)$ time implementation of the exact PNN, where τ is shown to be significantly smaller than N. We give all necessary data structures and implementation details, and give time complexity of the algorithm both in the best case and in the worst case. The proposed implementation achieves the results of the exact PNN with the same $O(N)$ memory requirement.**

***Index Terms*—Clustering, codebook generation, image coding, vector quantization.**

I. Introduction

We consider the codebook generation problem involved in the design of a *vector quantizer*. The aim is to find M *code vectors (codebook)* for a given set of N *training vectors (training set)* by minimizing the average pairwise distance between the training vectors and their representative code vectors. There are several known methods for generating a codebook [1]. The most cited and widely used is the *generalized Lloyd algorithm* (GLA) [2]. 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. The *iterative splitting algorithm* [3], [4] starts with a codebook of size one, 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) [5], 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.

From the two hierarchical approaches, the PNN has higher potential because it gives better results with a simpler implementation. It can also be used to produce an initial codebook for the GLA, or it can be embedded into hybrid methods such as a genetic algorithm [7] or the iterative split-and-merge method [6]. 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 $O(N^3)$ time [8], 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 $O(N \cdot \log N)$ time variant of the PNN, referred as the *fast PNN* [5]. 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. 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 [9].

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 [10]. Only $O(N)$ updates are needed after each step of the PNN, each taking $O(\log N)$ time. The method thus performs the exact PNN in $O(N^2 \cdot \log N)$ time but it requires $O(N^2)$ memory, which is impractical for large training sets.

In this paper, we propose a fast $O(\tau N^2)$ time implementation of the exact PNN, where τ is shown to be significantly smaller than N in practice. The main idea is to maintain a nearest neighbor table, which contains the index of the nearest cluster for each cluster. The optimal cluster pair to be merged can be found by a linear search from the nearest neighbor table. The proposed method achieves the result of the exact PNN with rather simple modifications to the basic algorithm. The method requires no complicated data structures and no distance matrix is needed for storing the pairwise distances. The proposed method thus performs the exact PNN using only $O(N)$ memory. The method has been presented independently in [11] and [12].

The rest of the paper is organized as follows. The problem formulation and the structure of the PNN are given in Section II. A fast implementation is introduced in Section III. Data structures and implementation details are discussed in the same section. Simulation results for various training sets appear in Section IV, and conclusions are drawn in Section V.

Manuscript received March 3, 1998; revised September 12, 1999. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Rashid Ansari.

P. Fränti is with the Department of Computer Science, University of Joensuu, FIN-80101 Joensuu, Finland.

T. Kaukoranta is with the Turku Centre for Computer Science, Department of Computer Science, University of Turku, FIN-20520 Turku, Finland.

D.-F. Shen and K.-S. Chang are with the Department of Electrical Engineering, Yunlin University of Science and Technology, Yunlin 640, Taiwan, R.O.C.

Publisher Item Identifier S 1057-7149(00)03563-6.

1057-7149/00$10.00 © 2000 IEEE

Set each training vector as a code vector (m=N).
Repeat
  Find two nearest clusters S_a and S_b to be merged.
  Merge the selected clusters; $m \leftarrow m$-1.
  Update data structures.
Until m=M.

Fig. 1. Structure of the exact PNN.

II. Pairwise Nearest Neighbor Algorithm

We consider a set of N training vectors (T_i) in a K-dimensional Euclidean space. The aim is to find a codebook C of M code vectors (C_i) by minimizing the average squared distance between the training vectors and their representative code vectors. The distance between two vectors is defined by their Euclidean distance. Let C be a codebook and P the partition of the training set. The distortion of the codebook C is then defined by

$$\text{distortion}(C) = \frac{1}{N}\sum_{i=1}^{N} ||T_i, C_{Pi}||^2 \tag{1}$$

where P_i is the partition index of training vector T_i. The basic structure of the PNN is shown in Fig. 1. The method starts by initializing each training vector T_i as its own code vector C_i. In each step of the algorithm, two nearest clusters (S_a and S_b) are then searched and merged. *Cluster* is defined as the set of training vectors that belong to the same partition a

$$S_a = \{T_i | P_i = a\}. \tag{2}$$

The distance (*cost function*) between two clusters is defined as the increase in the distortion of the codebook if the clusters are merged. It is calculated as the squared Euclidean distance of the cluster centroids (code vectors) weighted by the number of vectors in the two clusters [5]

$$d_{a,b} = \frac{n_a n_b}{n_a + n_b} \cdot ||C_a - C_b||^2. \tag{3}$$

The cost function is symmetric ($d_{a,b} = d_{b,a}$) and it can be calculated in $O(K)$ time, assuming that n_a, n_b, C_a, and C_b are known. In the following, we consider K as constant and thus, the cost function can be calculated in constant time.

The exact variant of the PNN applies local optimization strategy: all possible cluster pairs are considered and the one increasing the distortion least (smallest cost function value) is chosen. The clusters are then merged and the process is repeated until the codebook reaches the size M. Straightforward implementation of the method takes $O(N^3)$ time because there are $O(N)$ steps in total, and in each step there are $O(N^2)$ cluster pairs to be checked.

III. Fast Implementation of the PNN

The proposed algorithm follows the basic structure of the PNN method but the implementation of the individual steps is slightly different. The main idea is to maintain a *nearest neighbor* table (NN), which contains the index of the nearest cluster (NN_i) for each cluster i. The nearest neighbor for a cluster a is defined as cluster b that minimizes the cost function between a and any other cluster

$$NN_a = b \quad \text{if } d_{a,b} \le d_{a,j}\ \forall a \ne j,\ a \ne b. \tag{4}$$

Note that the nearest neighbor is not symmetrical, i.e., $NN_a = b$ does not imply $NN_b = a$. In the implementation, it is sufficient to maintain for each cluster the code vector (C_i), the cluster size (n_i), and the nearest neighbor pointer (NN_i) assigned with cost value (d_i) indicating the amount of increase in distortion if the clusters are merged. The memory requirement of the data structures is $O(N)$ in total.

A. Initialization

In the initialization phase, each training vector (T_i) is set as its own code vector (C_i), and the sizes of the clusters (n_i) are set to one. In order to generate the nearest neighbor table, we need to find nearest neighbor for every cluster. This is done by considering all other clusters as tentative neighbor and selecting the one that minimizes (3). The nearest neighbor pointer (NN_i) and the merge distortion (d_i) are stored in the nearest neighbor table. There are $O(N^2)$ pairs to be considered in total. Since the cost function can be calculated in $O(1)$ time the time complexity of the initialization phase is $O(N^2)$.

B. Finding the Two Nearest Clusters

The clusters to be merged are the cluster pair minimizing (3). In our method, this pair can be found by linear search from the nearest neighbor table. The clusters are the one with the minimum d_i-value and its nearest neighbor NN_i. This operation takes $O(N)$ time.

C. Merging the Clusters

Merging of the two clusters causes changes to all data structures. The merged clusters are denoted in the following by the indices a and b. The codebook can be updated straightforwardly using the information stored in the data structure. The code vector of the combined cluster is the centroid of the training vectors in the cluster and it can be calculated as the weighted average of C_a and C_b

$$C_{a+b} = \frac{n_a C_a + n_b C_b}{n_a + n_b}. \tag{5}$$

Here n_a and n_b are the number of training vectors in the two clusters. The size of the merged cluster is calculated as

$$n_{a+b} = n_a + n_b. \tag{6}$$

The above calculations can be performed in a constant time.

D. Updating the Nearest Neighbor Table

The key question of the implementation is the maintenance of the nearest neighbor table. It is obvious that the nearest neighbor for the merged cluster must be resolved by considering all other clusters. This can be performed in $O(N)$time. The rest of the clusters can be classified into two groups: 1) clusters whose nearest neighbor before merge was a or b, i.e., $NN_i = a$ or $NN_i = b$ and 2) all other clusters.

It was shown in [13] that the cost function (3) is monotonically increasing as a function of time. Therefore, only the clusters in the first group need to be updated. The update is performed for a cluster i by finding such cluster j that $d_{i,j}$ is minimized. This takes $O(N)$ time for a single cluster. An important question is therefore how many clusters belong to the first group. We denote this number by τ.

E. Amount of Necessary Updates

We are interested in the question of how many clusters can have the same cluster as their closest neighbor. This is closely related to the *kissing number problem*, which asks the highest number of K-dimensional spheres that can be packed so that they all touch one sphere [14]. Unfortunately, the cluster distance function (3) is not Euclidean and therefore the kissing number applies, for certain, only in the initial stage of PNN, and in cases when all cluster sizes are equal. It is thus possible that, in the worst case, the same cluster can be the nearest neighbor for all other clusters, and thus $\tau = O(N)$. This situation could appear when there are one small cluster and all the rest are large. Fortunately, this situation is not common in practice and the kissing number (even as an open problem in the general case) indicates that τ is a function of the vector dimension K.

In a favorable case, two merged clusters are chosen randomly. Since there is only one nearest neighbor pointer per each cluster, a randomly chosen cluster is the nearest neighbor for one cluster on average. The average number of the updated pointers is therefore $\tau = 2$ because there are two clusters involved in the merge operation. This shows that $\tau = O(1)$ in the best case. This approximation, however, is somewhat too optimistic for the average case because the merged clusters are not chosen randomly but they tend to be located in areas of high concentration of clusters.

To sum up, the time complexity of updating the nearest neighbor table is $O(\tau N)$, which is also the time complexity of the entire PNN step. In the best case, this is reduced to $O(N)$, but in the worst case it is still $O(N^2)$. The overall time complexity of our method is therefore $O(\tau N^2)$ as there are $O(N)$ steps of the algorithm in total (see Table I). This compares favorably with the original method since it is most likely that $\tau \ll N$. The time complexity of the original PNN method is $O(N^3)$ due to the time-consuming nearest neighbor search. Kurita's method requires $O(N^2 \cdot \log N)$ time originating from the update of the heap structure.

IV. Test Results

We generated training sets from six different images: *bridge*, *camera*, *Miss America*, *table tennis*, *airplane* and *house* (see Fig. 2). The vectors in the first two sets are 4×4 pixel blocks from the image. The third and fourth sets have been obtained by subtracting two subsequent image frames of the original video image sequences, and then constructing 4×4 spatial pixel blocks from the residuals. Only the first two frames have been used. The fifth and sixth data sets (*airplane, house*) consist of color values of the RGB images. Applications of this kind of data set are found in image and video image coding and in color image quantization.

TABLE I
TIME COMPLEXITIES OF THREE VARIANTS OF THE EXACT PNN: STRAIGHTFROWARD IMPLEMENTATION OF THE PNN WITHOUT ANY EXTRA DATA STRUCTURES (ORIGINAL M ETHOD), THE METHOD THAT STORES ALL PAIRWISE DISTANCES INTO A HEAP STRUCTURE (KURITA'S METHOD), AND THE NEW METHOD USING THE NEAREST NEIGHBOR TABLE (OUR METHOD)

	Original method:	**Kurita's method:**	**Our method:**
Initialization phase:	$O(N)$	$O(N^2)$	$O(N^2)$
Single merge phase:			
• Find two nearest	$O(N^2)$	$O(1)$	$O(N)$
• Merge the clusters	$O(1)$	$O(1)$	$O(1)$
• Recalculate distances	$O(1)$	$O(N)$	$O(\tau N)$
• Update data structures	$O(1)$	$O(N \cdot \log N)$	$O(N)$
		$O(N \cdot \log N)$	
Merge phases in total:	$O(N^3)$	$O(N^2 \cdot \log N)$	$O(\tau N^2)$
Algorithm in total:	$O(N^3)$	$O(N^2 \cdot \log N)$	$O(\tau N^2)$

For further testing of the parameters N and K, we generated subsets of size $N = (512, 1024, 2048, 4096)$ from *bridge* by random sampling of the original set. All sets were further processed by eliminating one dimension of the vectors at a time, in order to generate subsets with various length of vectors ($K = 2 \cdots 16$). In total, 60 different subsets were generated from *bridge*. The number of code vectors is fixed to $M = 256$ in the following tests.

The number of clusters whose nearest neighbor pointer must be updated (τ) may vary from 0 to $N-2$. We studied the number τ empirically by calculating its average and maximum values when generating codebooks from all 60 subsets of *bridge.* We observed that the average value of τ is small in all cases; it varies from 1 to 4 and is practically independent from the size of the training set (N). The average value of τ, on the other hand, is an increasing function of K but the growth is still rather slow (see Fig. 3). The average and maximum number of τ for all training sets are summarized in Table II.

Following the hypothesis that τ is independent of N, the computation time of our method should be multiplied by four when the size of training set is doubled, whereas the running time of the original PNN should be multiplied by eight. This is verified by the observed running times (see also Fig. 4), which are (120, 1082, 8913, 71 324) for the original PNN, and (3, 18, 78, 328) for our PNN when $N = (512, 1024, 2048, 4096)$. The corresponding multiplication factors for $(512 \rightarrow 1024 \rightarrow 2048 \rightarrow 4096)$ are (9.02, 8.32, 8.00) for the original PNN and (6.00, 4.33, 4.21) for our PNN.

Our method is compared with the Kurita's method in Fig. 5 for subsets of *bridge* with various number of K. The running times of the two methods are rather similar. Usually, τ is smaller than $\log N$ but the difference is compensated by the fact that the update of the heap ($\log N$) is faster because it does not depend on the dimension K. Our method is therefore slightly slower in cases when K is larger, and vice versa. Kurita's method was not applied for larger data sets because of its huge memory consumption.

Fig. 2. Sources of the training sets. *The training set *table tennis* is constructed by random sampling only every fourth block. **The images *airplane* and *house* are prequantized to 5 bits per each color component.

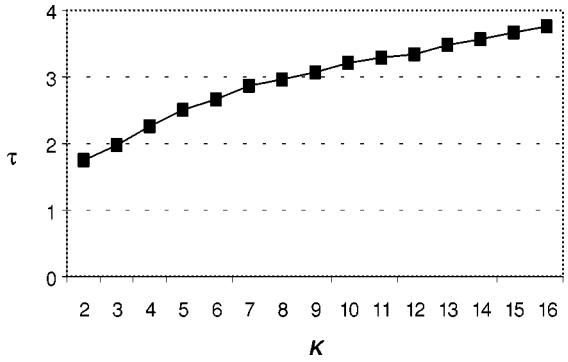


Fig. 3. Average number of τ as a function of K.

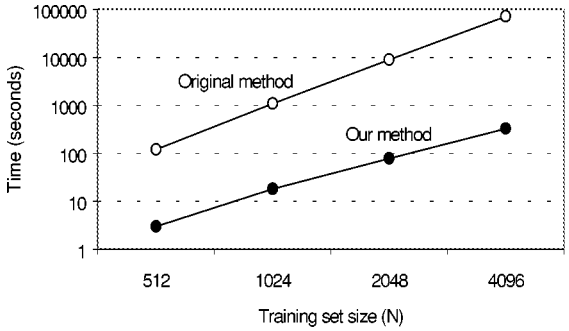


Fig. 4. Comparison of running times of the original and our method as a function of N (for *bridge*).

TABLE II
OBSERVED NUMBERS OF NECESSARY UPDATES (τ)

Training set:	Average:	Maximum:
Bridge	3.76	16
Camera	3.35	16
Miss America	2.44	8
Table tennis	4.22	23
Airplane	4.61	41
House	4.19	23

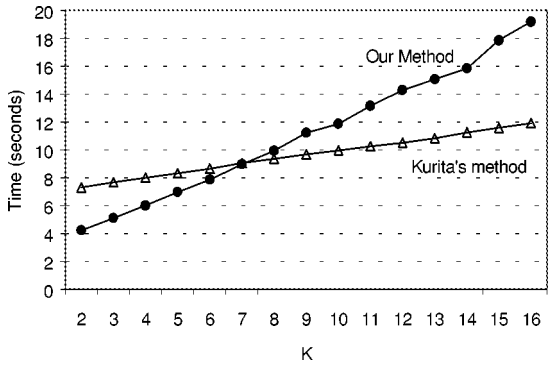


Fig. 5. Comparison of running times of Kurita's method and our method as a function of K (for a subset of *bridge* where $N = 1024$).

The observed running times for all training sets are summarized in Table III. A major speed-up is obtained in all cases in comparison to the original algorithm. The method is about 100–350 times faster than the original method for the tested training sets. Since the method evidently gives asymptotic improvement in the average case, the speed-up is greater for large training sets.

The exact PNN is compared in Fig. 6 with three suboptimal variants of PNN. The GLA-PNN method by deGarrido *et al.* [9] starts with an initial codebook of an intermediate size M_0 (generated by the GLA), which is then reduced to the final size M using the exact PNN. The method is a compromise between higher speed of the GLA and better quality of the PNN. The method reduces back to the exact PNN when $M_0 = N$, and to the GLA when $M_0 = M$. It is noted, that the result of the PNN can be improved further by the application of the GLA This method is referred as GLA-PNN- GLA in Fig. 6. We apply here the GLA implementation introduced in [15].

TABLE III
SUMMARY OF THE RUNNING TIME (IN SECONDS)

Training set	Original PNN	Our PNN	Speed-up factor
Bridge	73,006	331	221
Camera	73,040	300	243
Miss America	292,351	870	336
Table tennis	177,019	649	273
Airplane	4,751	48	99
House	2,341	27	87

V. CONCLUSIONS

A fast $O(\tau N^2)$ time implementation of the exact PNN was proposed. The main idea is to maintain a nearest neighbor table

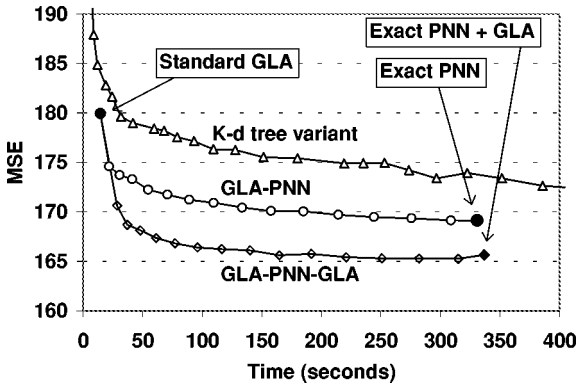


Fig. 6. Time-distortion performance of the suboptimal PNN variants for *bridge*. In the *GLA-PNN* method, we apply our $O(\tau N^2)$ implementation of the PNN instead of the original $O(N^3)$ time algorithm. The method is parametrized by changing the size of the initial codebook from $M_0 = 256$ (standard GLA) to $M_0 = 4096$ (exact PNN). The *K-d tree* variant is parametrized by varying the maximum bucket size from eight to 200. In all cases, the size of the final codebook is set to $M = 256$.

for avoiding unnecessary distance calculations. The number of necessary updates (τ) depends on the nature of the data and the dimension of the training vectors (K) but it is practically independent from the size of the training set (N). In practice, the number τ was observed to be significantly smaller than in all training sets, varying from 2.4 to 4.6 for the tested training sets. A speed-up of about 100–350 was thus obtained. To sum up, the method achieves the result of the exact PNN using only a fraction of time required by the original algorithm.

The proposed method is also practical and rather simple to implement. It requires no complicated data structures and no distance matrix is needed for storing the pairwise distances. The proposed method performs the exact PNN using only $O(N)$ memory, in contrary to Kurita's method that requires $O(N^2)$ amount of memory. The running times of the two methods were comparable but the proposed method is also applicable for large training sets.

REFERENCES

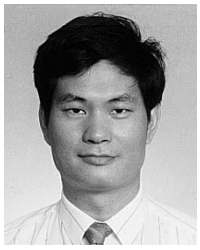
[1] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Dordrecht, The Netherlands: Kluwer, 1992.
[2] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. 28, pp. 84–95, Jan. 1980.
[3] X. Wu and K. Zhang, "A better tree-structured vector quantizer," in *IEEE Proc. Data Compression Conf.*, Snowbird, UT, 1991, pp. 392–401.
[4] P. Fränti, T. Kaukoranta, and O. Nevalainen, "On the splitting method for VQ codebook generation," *Opt. Eng.*, vol. 36, pp. 3043–3051, Nov. 1997.
[5] W. H. Equitz, "A new vector quantization clustering algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1568–1575, Oct. 1989.
[6] T. Kaukoranta, P. Fränti, and O. Nevalainen, "Iterative split-and-merge algorithm for VQ codebook generation," *Opt. Eng.*, vol. 37, pp. 2726–2732, Oct. 1998.
[7] P. Fränti, J. Kivijärvi, T. Kaukoranta, and O. Nevalainen, "Genetic algorithms for large scale clustering problem," *Comput. J.*, vol. 40, no. 9, pp. 547–554, 1997.
[8] J. Shanbehzadeh and P. O. Ogunbona, "On the computational complexity of the LBG and PNN algorithms," *IEEE Trans. Image Processing*, vol. 6, pp. 614–616, Apr. 1997.
[9] D. P. de Garrido, W. A. Pearlman, and W. A. Finamore, "A clustering algorithm for entropy-constrained vector quantizer design with applications in coding image pyramids," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 83–95, Apr. 1995.
[10] T. Kurita, "An efficient agglomerative clustering algorithm using a heap," *Pattern Recognit.*, vol. 24, pp. 205–209, Mar. 1991.
[11] D.-F. Shen and K.-S. Chang, "Fast PNN algorithm for design of VQ initial codebook," in *Proc. SPIE Visual Communications Image Processing (VCIP'98)*, San Jose, CA, 1998, pp. 842–850.
[12] P. Fränti and T. Kaukoranta, "Fast implementation of the optimal PNN method," in *IEEE Int. Conf. Image Processing (ICIP'98)*, vol. 3, Chicago, IL, 1998, pp. 104–108.
[13] T. Kaukoranta, P. Fränti, and O. Nevalainen, "Vector quantization by lazy pairwise nearest neighbor method," *Opt. Eng.*, vol. 38, Nov. 1999.
[14] J. H. Conway and N. J. A. Sloane, *Sphere Packing, Lattices and Groups*. New York: Springer-Verlag, 1988.
[15] T. Kaukoranta, P. Fränti, and O. Nevalainen, "Reduced comparison search for the exact GLA," in *IEEE Proc. Data Compression Conf.*, Snowbird, UT, 1999, pp. 33–41.

Pasi Fränti received the M.Sc. and Ph.D. degrees in computer science from the University of Turku, Turku, Finland, in 1991 and 1994, respectively.

From 1996 to 1999, he was a Postdoctoral Researcher with the University of Joensuu, funded by the Academy of Finland. Since 1999 he has been Professor with the University of Joensuu. His primary research interests are in image compression, vector quantization, and clustering algorithms.

Timo Kaukoranta received the M.Sc. and Ph.D. degrees in computer science from the University of Turku, Turku, Finland, in 1994 and 2000, respectively.

Since 1997, he has been a Researcher with the University of Turku. His primary research interests are in vector quantization and image compression.



Day-Fann Shen received the M.Sc. degree from the University of Cincinnati, Cincinnati, OH, in 1983 and the Ph.D. degree from North Carolina State University, Raleigh, in 1992, both in electrical and computer engineering.

From 1983 to 1986, he was with GTE Telnet Department, Reston, VA, and from 1989 to 1992, with IBM Communication Division, Research Triangle Park, NC. Since 1992, he has been an Associate Professor with the Department of Electrical Engineering, Yunlin University of Science and Technology, Taiwan, R.O.C. His current research interests are human visual property, color science, wavelet transform, vector quantization and their applications to image/video processing and communication.

Kuo-Shu Chang received the M.Sc. degree in electrical engineering from Yunlin University of Science and Technology, Taiwan, R.O.C., in 1997.

He is currently an Engineer with the Electronic Department, Chung Shan Institute of Science and Technology, Yunlin, Taiwan. His primary interests are in image processing and its applications to national defense technology.