A K-Means-Based Multi-Prototype High-Speed Learning System with FPGA-Implemented Coprocessor for 1-NN Searching

Fengwei AN†a), Nonmember, Tetsushi KOIDE†, and Hans Jürgen MATTAUSCH†, Members

SUMMARY   In this paper, we propose a hardware solution for overcoming the problem of high computational demands in a nearest neighbor (NN) based multi-prototype learning system. The multiple prototypes are obtained by a high-speed K-means clustering algorithm utilizing a concept of software-hardware cooperation that takes advantage of the flexibility of the software and the efficiency of the hardware. The one nearest neighbor (1-NN) classifier is used to recognize an object by searching for the nearest Euclidean distance among the prototypes. The major deficiency in conventional implementations for both K-means and 1-NN is the high computational demand of the nearest neighbor searching. This deficiency is resolved by an FPGA-implemented coprocessor that is a VLSI circuit for searching the nearest Euclidean distance. The coprocessor requires 12.9% logic elements and 58% block memory bits of an Altera Stratix III E110 FPGA device. The hardware communicates with the software by a PCI Express (×4) local-bus-compatible interface. We benchmark our learning system against the popular case of handwritten digit recognition in which abundant previous works for comparison are available. In the case of the MNIST database, we could attain the most efficient accuracy rate of 97.91% with 930 prototypes, the learning speed of $1.3 \times 10^{-4}$ s/sample and the classification speed of $3.94 \times 10^{-8}$ s/character.

key words: multi-prototype learning system, K-means clustering, software-hardware cooperation, one nearest neighbor (1-NN), FPGA-implemented coprocessor, nearest euclidean distance searching

1. Introduction

The main limitation of the usage of the nearest neighbor (NN) rule based learning systems [1] in practice even today is the high computational demands of the minimum distance searching, although such systems have been studied since 1954 [2].

The distances between a test sample and references are often defined as the Euclidean distance. In order to minimize the computational efforts in the classification stage, the cluster-based prototype learning algorithms such as K-means clustering [3]–[5], the fuzzy c-means [6], [7], and the learning vector quantization algorithm (LVQ) [8], have been proposed to reduce the number of references. Instead of recruiting all training samples, the prototypes (references) are the centroids of clusters derived from the average of training samples. The prototypes are vectors that reside in the same vector space as the feature vectors of the training samples. The number of prototypes, produced by the K-means clustering algorithm, is sufficiently compressed to much lower order than that of the training samples. However, the high computational demands of the nearest Euclidean distance searching in cluster-based prototype learning and NN-based classification cannot be fundamentally resolved by conventional implementations.

Unlike the sequential processing of conventional implementations, recently, the graphics processing units (GPU) provide a powerful processing platform which is specialized for parallel processing. Indeed, most of the parallelizable problems can be significantly accelerated by a GPU which has a fixed computational and programming model. By taking advantage of the GPU’s capability for parallel processing, many methods were proposed to accelerate the NN-based learning algorithms [9]–[12]. Because of the progress in semiconductor technology, the possibility of massively parallel processing is enabled by hardware implementations. Such hardware implementations of the K-means clustering algorithm have attracted large attention in recent years [13]–[18]. The Field Programmable Gate Array (FPGA) which has comparable computational speed and lower cost in comparison to an Application Specific Integrated Circuit (ASIC) also attracts many researchers [19]–[21]. However, K-means requires large adder to accumulate the values of the training samples which have been assigned to the same class categories and needs large divider for the centroids updating. Owing to the related enormous processing efforts, the dimension of the vector of each training sample in these hardware implementations had to be low enough so that these previous proposals are not suitable for a machine learning algorithm with high dimensional requirements. K nearest neighbor (KNN) classifier is one of the simplest machine learning algorithms for classifying samples based on the NN rule. Hardware implementations of KNN that exploit parallel processing and pipelining have been reported in [22]–[24]. However, KNN needs to find the $k$ smallest distances to the test sample among all references which are the whole training data. This leads to a lot of numerical efforts for finding an optimal number of $k$.

Several space-partitioning methods (e.g. the KD-tree [25]) have been developed for reducing the search time of the nearest neighbor searching. In order to further reduce the bottleneck of high computational time, the approximate nearest neighbor (ANN) searching retrieves an approximation as the nearest neighbor. ANN algorithms, such as Euclidean Locality-Sensitive Hashing (E2LSH) [26], fast approximate nearest neighbors (FLANN) [27], spectral hashing (SH) [28], and inverted file system with asymmetric dis-
occurrence of 1-NN operation. The learning system is tested by significantly improved by the coprocessor for the frequently occurring 1-NN operation. Consequently, the dedicated hardware can significantly improve the speed of basic nearest-neighbor searching with low hardware-resource cost in most cases.

Beside the hardware implementation of K-means and KNN, a significant amount of work has been done in developing hardware for neural networks. A few surveys of neural network hardware have been published in [30]–[34] where several hundred works were listed. Support vector machine (SVM) hardware also seems attractive [35]–[41]. As Heemskerk [30] indicates, the hardware implementation of neural networks is expensive in terms of development time and hardware resources. This is because the neural networks are so complicated that the dedicated circuits lead to large silicon area, high fabrication cost, and bad cost-to-performance ratio. In addition, little is known about the real commercial prospects for real-world applications. On the other hand, the NN-based learning algorithms such as KNN, and cluster-based prototype learning methods are simpler than neural networks and SVMs. This does not always mean the NN-based learning algorithms are worse than the neural networks or SVMs. Which is better depends on what the application is and what kind of feature extraction technique is used [42]. The NN-based learning algorithms have been widely used in pattern recognition [43]–[45], text categorization [46], [47], ranking models [48], event tracking [49], and object recognition [50].

The problems with above works can be attributed to four causes. Firstly, the hardware K-means requires large adders and dividers for updating the centroids. This restricts the application in machine learning algorithms which have a high-dimensional feature vector. Then, finding an optimal k and recruiting all training samples lead to a high computational cost for KNN. Furthermore, GPU implementation of learning algorithms is less efficient than hardware in most cases [24]. The fourth aspect is that the neural networks and SVMs are so complicated that they are not suitable for practical hardware implementation [30], although a lot of hardware related works have been done. Here, in order to cope with the computational barrier in clustering-based prototype learning algorithms, we develop an FPGA-implemented coprocessor for the nearest Euclidean distance searching which is the critical path both in K-means-based learning and 1-NN-based classification. For the complete system, we exploit the flexibility of the software in programming for updating the centroids of clusters which would require large adders and dividers in the hardware implementation. The overall speed of the learning and classification is significantly improved by the coprocessor for the frequently occurring 1-NN operation. The learning system is tested by the problem of handwritten-digit recognition in this work. Due to the optimized combination of software and hardware architecture, high-speed learning and classification are attained.

The rest of this paper is organized as follows: Sect. 2 illustrates the algorithm of the developed K-means-based multi-prototype learning system. The hardware architecture of the FPGA-implemented coprocessor for 1-NN searching is described in Sect. 3. Section 4 explains how the software-hardware cooperation is working. In Sect. 5, the results of resource usage, the speed of learning and classification, and accuracy rate are shown. This verifies that our learning system is efficient due to not only the high speed, which is accelerated by the hardware, but also the reasonable accuracy rate which is better than KNN (K = 3) in the handwritten-digit recognition although the bit precision is only 16-bit. At last we conclude in Sect. 6.

2. K-Means Based Prototype Learning

The K-means algorithm is a widely used in machine learning and data compression. The goal of this algorithm is to partition a set of data samples into k clusters by considering the boundary information of confusing classes. Suppose that a set of n data samples ($\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$) with a sample $\mathbf{x}_j \in \mathbb{R}^m$ and the number of clusters $k (0 < k < n)$ are given. The k optimal cluster centroids ($\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_k$) with a centroid $\mathbf{c}_k$ as a special case of KNN, the 1-NN is simpler than the KNN ($K > 1$) and exhibits satisfactory performance [1]. It was proven that the asymptotic classification error of 1-NN is bounded by twice the Bayesian error rate [51]. However, the NN classifier (either 1-NN or KNN) requires a large number of training samples and therefore suffers from a heavy burden of storage and computation requirements.

The clustering algorithm is applied to generate new prototypes to replace the original training samples, which significantly reduces the number of references in 1-NN classification [53]. Each prototype is a representative of a group of training samples. The goal of the K-means algorithm [52] is to adjust the prototypes of the 1-NN classifier by considering the boundary information of confusing classes so that the training samples are well classified by a small number of prototypes. Suppose that the training samples are described with n feature vectors ($\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$), where each sample $\mathbf{x}_i = (l^{K\text{-means}}_i, x_{i1}, x_{i2}, \cdots, x_{im})^T$ is an m-dimensional feature vector and has an unknown label ($l^{K\text{-means}}_i$). The Euclidean distance is used as the distance metric to measure the similarity. K-means is to find k optimal cluster centroids ($\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_k$). Then the objective function (1) is locally minimized.

$$J_{K\text{-means}} = \min \sum_{i=1}^{k} \sum_{j=1}^{m} ||\mathbf{x}_i - \mathbf{c}_j||$$

$$||\mathbf{x}_i - \mathbf{c}_j|| = \sqrt{(x_{i1} - c_{j1})^2 + \cdots + (x_{im} - c_{jm})^2}$$

Randomly given k cluster centroids of m-dimensional vec-
tors \( \overrightarrow{c_1}, \overrightarrow{c_2}, \ldots, \overrightarrow{c_k} \) are used as the initial prototypes and the distances among the \( n \) training samples and the \( k \) cluster centroids are calculated. Then, every individual \( \overrightarrow{x_i} \) of the \( n \) training samples is assigned to its nearest cluster centroid. The cluster centroid \( \overrightarrow{c_i} \) is updated according to (2) where \( |\overrightarrow{c_i}| \) is the total number of training samples assigned to this centroid. Updating the centroids in (2) is the main limitation of hardware K-means because it requires large adders and dividers when the dimension of the feature is high. The updating process is reiterated until the cluster centroids stabilize.

\[
\overrightarrow{c_i} = \frac{1}{|\overrightarrow{c_i}|} \sum_{\overrightarrow{x_j} \in \overrightarrow{c_i}} \overrightarrow{x_j} \quad (2)
\]

In K-means-based prototype learning, each training sample is described by a feature vector with two labels \( \overrightarrow{x_i} = (l_{i1}^{(1-NN)}, l_{i2}^{(K-means)}, x_{i1}, x_{i2}, \ldots, x_{im})^T \). The K-means algorithm clusters the training samples which have the same class label \( (l_{i1}^{(1-NN)}) \) into multiple groups, while the unknown label for K-means \( (l_{i2}^{(K-means)}) \) is assigned according to the specified number of groups with the same class label \( (l_{i1}^{(1-NN)}) \). The cluster centroids of the obtained groups are used as prototypes and are the representatives of the training samples which are in the same class category.

The configuration of prototypes for each class, \( k \) of K-means, was pre-fixed in previous works [44]. In the learning algorithm reported here, we define a desired accuracy rate criterion to dynamically obtain the optimal \( k \) of the cluster centers. The desired accuracy rate is a parameter that the real accuracy rate is expected to attain. The real accuracy rate is determined by using the prototypes to recognize a set of test samples that are different from the training samples. The result of comparing the real accuracy rate to the desired accuracy rate is used as a feedback to confirm whether the current \( k \) is already appropriate or not. The value of \( k \) for the next iteration using K-means to produce prototypes will be increased when the real accuracy rate is less than the desired accuracy rate. The procedure for producing prototypes is terminated when the real accuracy rate reaches the predefined accuracy or \( k \) is equal to a pre-defined \( k_{\text{max}} \) which is derived from the upper limitation of the storage capacity for the prototypes in our system. In the case of \( k = k_{\text{max}} \), we select the value of \( k (k \leq k_{\text{max}}) \) which has yielded the best accuracy rate. The K-means-based prototype learning algorithm is performed according to the following steps.

**Step 1** Initialize \( k \) cluster centroids for one class according to the label \( (l_{i1}^{(1-NN)}) \).

**Step 2** For each training sample \( \overrightarrow{x_i} \), find the cluster center \( \overrightarrow{c_j} \) which is closest to \( \overrightarrow{x_i} \). Assign the class category of this cluster center \( \overrightarrow{c_j} \) to the label for K-means \( (l_{i2}^{(K-means)}) \) of \( \overrightarrow{x_i} \). Then, \( \overrightarrow{x_i} \) is added to the cluster accumulator \( c_j \).

**Step 3** Repeat step 2 until all training samples with class label \( (l_{i1}^{(1-NN)}) \) are processed.

**Step 4** The mean value of the each cluster accumulator \( c_j \) becomes the new centroid \( \overrightarrow{c_j} \).

**Step 5** Repeat step 1, step 2, step 3, and step 4 until the algorithm has converged towards stable centroids \( \overrightarrow{c_j} \).

**Step 6** Repeat step 1 to step 5 until all class categories of training samples have been clustered.

**Step 7** Use the prototypes to recognize the test samples by 1-NN rule. Calculate the real accuracy rate.

**Step 8** Increase the value of \( k \) by a pre-defined amount and repeat the above procedure of step 1 to step 7 until the real accuracy rate reaches the desired accuracy rate or until \( k = k_{\text{max}} \).

**Step 9** Choose the \( k \) prototypes with the best training result for the final recognition.

Because of the high computational costs in the cluster-based prototype learning stage, a software implementation only can produce less than 10 prototypes for each class [44]. Moreover, finding the optimal number of prototypes requires much more time. Fortunately, since the hardware coprocessor in our system implementation accelerates the processing speed for the nearest Euclidean distance searching, the most efficient case of 1024 prototypes for all classes can be attained. The architecture of the developed K-means-based learning system with software-hardware cooperation is illustrated in Fig. 1, where the hardware coprocessor communicates with the CPU by a PCI Express (x4) interface. Before the learning, the training samples and test samples are stored in the external memory. At the beginning, the initialized \( k \) prototypes are transferred via PCI-e bus to the on-chip memory of the FPGA device. Then, the nearest distance searching for all training samples is performed by hardware. The results of the nearest distance searching are addresses in each of which the prototype has the minimum distance to the current training sample. After the searching, the addresses, which are stored in a memory, are transferred back to the host PC to perform the centroids updating by software. Subsequently, the updated centroids overwrite the
prototypes in on-chip memory via PCI-e bus. The learning procedure will not be terminated until the prototypes become stable. Furthermore, the test samples are classified by the prototypes. The result of the accuracy rate comparing against the desired accuracy rate determines whether the learning procedure will be terminated or repeated. It is worth noting that the configuration of prototypes for each class has the same value. The value of \( k \) starts from 1 and ends at \( k_{\text{max}} \) or a smaller value \( (k < k_{\text{max}}) \) when the real accuracy rate reaches the desired accuracy rate. The \( k \) is increased one by one or by a jump value which is pre-fixed by user’s experience. A large jump value can reduce the cycle number of repeated K-means application, even though the largest amount of computational complexity for the nearest neighbor searching has been accelerated by the coprocessor.

3. FPGA-Implemented Coprocessor

The conventional methods for nearest distance searching are often based on a general purpose computer with a sequential processing. Recently, mixed digital/analog solutions with a fully-parallel VLSI implementation have been presented for finding the nearest Manhattan and Hamming distance in previous works. The reported circuits, also called associative memory, have small silicon area (few sq mm), low power dissipation (few mW) and high matching speed (few deca ns) [54]–[56]. However, the Hamming and Manhattan distance are too simple for most machine learning algorithms. The VLSI circuit of an associative memory for searching the nearest Euclidean distance by a mixed digital-analog solution was developed in [57]. However, because the Euclidean distance is expressed by a current-mode representation, which is affected by random fabrication process variations, the reliability of the mixed digital/analog VLSI circuit may be insufficient. A fully-digital VLSI circuit for the nearest Euclidean distance searching is furthermore expensive in developing time and fabrication cost. Fortunately, the Field Programmable Gate Array (FPGA), which is a reconfigurable VLSI circuit, provides us sufficient resources to develop a digital hardware circuit for nearest Euclidean distance searching.

The Euclidean distance is defined in (3) where \( \overrightarrow{IN} = (IN_1, IN_2, \cdots, IN_m) \) is the \( m \)-dimensional test feature vector and \( \overrightarrow{REF}_i = (REF_{i1}, REF_{i2}, \cdots, REF_{im}) \) is the reference vector. Because the square root is difficult to be implemented and not necessary for nearest distance searching, (3) is simplified to (4). This simplification has no influences on the results of searching for the nearest distance.

\[
D_1 = \sqrt{\sum_{0 \leq i \leq m} (\overrightarrow{IN}_i - \overrightarrow{REF}_i)^2}
\]  

(3)

\[
D_1 \approx \sum_{0 \leq i \leq m} (\overrightarrow{IN}_i - \overrightarrow{REF}_i)^2
\]  

(4)

Both \( IN_i \) and \( REF_{ij} \) are real numbers which are represented by a fixed point number in our hardware design.

The bit precision, the length of one word, is important for the output resolution. A statistical study for the precision requirements in the neural networks hardware showed that a fixed point encoding of 16 bit is sufficient [61]. Therefore, 16-bit precision is adopted so that the storage of one feature vector requires \( 16 \times m \) bits.

The coprocessor for the nearest Euclidean distance searching is implemented on an Altera Stratix III E110 FPGA device by parallel processing and pipelining. The parallelism (\( \rho \)) in horizontal direction of Fig. 3 refers to how many dimensions of the feature vector can be processed in each clock cycle. It determines the throughput (\( \theta \)) in (5) when the bit precision is 16-bit.

\[
\theta = \frac{\rho}{16}
\]  

(5)

In the design of the memory for storing the prototypes (Fig. 2), each feature vector, which has \( 16 \times m \) bits, is represented by multiple words. Each word, which has \( \rho - \text{bit} \) width, is composed of \( \beta \) blocks as shown in (6). The word length of one block (\( \nu \)-bit) is determined by the bit width of the on-board local bus and the external memory which are explained in Sect. 4.

\[
\beta = \frac{\rho}{\nu}
\]  

(6)

The number of words (\( \eta \)) per prototype can be obtained from (7). In other words, to calculate the Euclidean distance between one test and one reference feature vector, \( \eta \) clock cycles are required. Because of the storage limitations of the on-chip memory, we only allocate \( 16 \times m \) bits for one test sample. The rest of the memory blocks is used for storing the prototypes which are the centroids of the clusters when K-means is performing.

\[
\eta = \frac{16 \times m}{\rho} = \frac{16 \times m}{\beta \times \nu}
\]  

(7)

The pipelining concept is used to split the calculation of the Euclidean distance into a series of independent stages (\( \lambda \)) in terms of arithmetic operations as shown in vertical
direction of Fig. 3. The results of each arithmetic operation, which are stored in the registers at the end of each stage, become the inputs of the next stage. The pipelining makes sure that $\theta$ dimensions are processed in each clock cycle. Otherwise, the next $\theta$ dimensions should wait $\eta$ clock cycles to be processed.

The more stages ($\lambda$) the pipeline has, the fewer logic gates are required in each stage. This means the propagation delay is decreased for all stages so as to increase the coprocessor’s operating frequency. However, a larger $\lambda$ requires also more processing units for arithmetic operations, because the processing units cannot be reused. The final result after passing the arithmetic units (AUs) of the distance calculation hardware with parallel and pipelining architecture in Fig. 3 is the squares of the Euclidean distance for $\theta$ dimensions. In this work, the pipeline depth ($\lambda$) is defined by the parallelism and throughput according to (5). A larger $\lambda$ can be obtained by inserting sub-stages in the critical AUs. For example, when the multiplier stage in Fig. 3 becomes the critical path, inserting additional sub-stages in each multiplier (i.e. using a pipelined multiplier) can improve the frequency.

The maximum clock frequency $f_{\text{max}}$ of the coprocessor is determined by the clock period. The clock period of the critical AU, which contains the delay of the AU ($t_C$), the clock skew between registers ($t_2-t_1$), the clock-to-output delay of the register A ($t_{CO}$), and the setup time of the register B ($t_{SU}$), is illustrated in Fig. 4. The critical AU may be the multiplier or the larger adder (40-bit) in this case. Usually, the delay of the multiplier rather than an adder determines the critical path. Table 1 shows the improved maximum clock frequency attained by increasing the stage of multiplier. As a consequence, a larger $\lambda$, which is obtained by inserting sub-stages into the time-critical AUs, can improve the frequency according to the example for pipelined multiplier in Table 1.

The parallelism and throughput also determine the number of adders. Suppose that $\theta = 2^N$. In order to calculate the Euclidean-distance square for these $\theta$ dimensions, $N$ stages for add operations are required. The adder in Fig. 5 is the add operations after the $N^{th}$ stage of Fig. 3. The top register of Fig. 5 stores the intermediate Euclidean distances during the accumulation of the partial distances for $\eta$ words of one prototype. The result of the Euclidean distance between one prototype and one training or test sample is loaded to the middle register when $\eta$ words have been processed. In other words, the “One sample” signal is asserted when the address for the memory, which is used to store one training or test sample, is pointing to the last location that is the $\eta^{th}$ word. At this moment, the output of the multiplexer is “0” for calculating the Euclidean distance of the next prototype. The local minimum Euclidean dis-
tance and its reference address are respectively updated in the bottom registers only when the “One sample” signal is asserted and the current Euclidean distance is less than the previously stored local minimum. The final output result is the address in which the prototype is most similar to the test sample according to the Euclidean distance. Hence, the number of adders \( S_{adder} \), which are used to calculate the Euclidean distance, can be obtained from (8) where the last adder is shown in Fig. 5.

\[
S_{adder} = 2^{N-1} + 2^{N-2} + \ldots + 2^1 + 2^0 + 1 = \theta 
\] (8)

As shown in Fig. 3 and (8), the depth of the pipeline can be calculated. Therefore, both the calculation of the Euclidean distance and the nearest distance searching can be easily controled by a “reset” signal and the addresses of the memory for the prototypes and the test sample rather than by a state machine.

The parallelism \( \rho \) plays an important role in the resource usages, since \( \rho \) determines the number of AUs and the depth of the pipeline. The higher the parallelism is, the more resources for AUs are required. In this work, the coprocessor for the nearest Euclidean distance searching is implemented on an FPGA device which has sufficient resources and low cost.

4. Software-Hardware Cooperation

The K-means algorithm produces multiple prototypes which are the cluster centroids. Although the FPGA device provides us a low cost hardware implementation platform with sufficient resources, updating the centroids which requires not only large adders but also large dividers is still expensive due to the cost-to-performance ratio in a fully-hardware implemented K-means. Therefore, the software implementation in a general-purpose PC for updating the centroids is better than a hardware implementation when K-means is used for machine learning algorithms with high dimensional feature vectors.

The learning system in Fig. 6 is implemented on a GiDEl PROCeIII FPGA board which has an Altera stratiX III E110 FPGA device and provides a PCI-e bus interface. The maximum bandwidth of the PCI-e \( \times 4 \) bus is 2 GB/s which provides us an efficient way for data exchange. The data transferring via the local bus and the PCI-e bus is performed by on-board Direct Memory Access (DMA) controllers to obtain a speed advantage. The software opens a DMA channel and transfers all of the test samples at once. This allows the hardware to search for the nearest distance while the software can perform other tasks. At last, the addresses, in each of which the feature vector is most similar to the corresponding test sample according to the Euclidean distance, are the output from the hardware which is transferred back to the host PC by DMA.

5. Experimental Results

5.1 Handwritten Digit Recognition

Handwritten digit recognition is a widely researched attractive task in the pattern recognition area. Many classification algorithms have been proposed to solve this challenging problem. Solutions based on machine learning such as neural networks, support vector machine and KNN have been employed to recognize the handwritten characters. Constructing a classification system that can recognize any character with high reliability to every application is not possible because handwriting depends on the writer and even the same writer does not always write the same character in an identical way. Handwritten digit recognition has already contributed to the evaluation of classification algorithms. Certainly, a number of related works such as pre-processing and feature extraction have also been accomplished for handwritten digit recognition.

In this paper, we used the well known MNIST database [62], which contains handwritten digit samples normalized and fixed to the same size, to validate the K-means-based prototype learning system. The gradient feature is extracted to represent the digits. For gradient feature extraction, the Sobel operator is used to compute the gradient in \( x \) and \( y \) direction respectively. A number of other feature extraction techniques have also been illustrated in [63]. We apply the gradient feature vector from gray-scale images which has 200-dimensions for 8-directions. In the hardware design, the feature vector is expressed by multiple words in the memory space. Therefore, we defined a 256-dimensional vector \( m = 256 \) where the remaining 56-dimension components are simply filled up by 0. When \( m = 200 \), the part of the reserved memory space on the hardware remains unused. This example also indicates that the developed learning system can be easily ported to solve any problem with a feature vector less or equal to 256 dimensions.

5.2 Accuracy Rate of the K-Means-Based Learning System with Software-Hardware Cooperation

The MNIST database of handwritten digits contains 60000 training samples and 10000 test samples. As described in Sect. 2, the training samples have two class labels \( (l_i^{NN}, l_i^{K\text{-means}}) \) where \( 0 \leq l_i^{NN}, l_i^{K\text{-means}} \leq 9 \) is the label provided by
the MNIST database. The digits are first grouped based on $l^1_{\text{NN}}$. The K-means algorithm is then used to assign the $l_K^{\text{K-means}}$ ($1 \leq l_K^{\text{K-means}} \leq k$) values in the clustering procedure. Consequently, the number of the prototypes is $10 \times k$. At last, only $l^1_{\text{NN}}$ is used in the classification stage by the coprocessor.

Three experiments were carried out to check the performance with different numbers of the training samples. The first experiment was performed with 10000 training samples to produce the prototypes. The accuracy rate was then obtained by classifying 10000 test samples. Subsequently, two groups of the prototypes, which were respectively generated from larger numbers of 30000 and 60000 training samples, were used to estimate the possible accuracy rate improvements. The accuracy rates, using the three different groups of the prototypes to recognize the test samples, are illustrated in Fig. 7. A larger number of the training samples yields as expected a more efficient accuracy rate according to the results in Fig. 7. It implies that our learning system can attain competitive results provided that sufficient training data are available. The number of the prototypes can significantly improve the accuracy rate until it reaches 400. After that, the accuracy rates using the prototypes produced by 60000 and 30000 training samples fluctuate around a certain value when the number of the prototypes is increasing. The most efficient accuracy rate of 97.91% occurs with 930 prototypes.

Many previous learning and recognition methods have been tested with the MNIST training samples and test samples. The accuracy rate of our learning system is compared against KNN with the same feature extraction technique. KNN requires not only high computational costs to choose an optimal $k$ but also a very large number of prototypes (references). The number of required prototypes to achieve the same real accuracy rate for recognizing the test samples in MNIST is reduced with our developed system by a factor of 20 or more as shown in Fig. 8. Although the precision of each dimension in the feature vector was decreased to 16-bit in the hardware implementation, the accuracy rate is reasonable and higher than 97.6% which is obtained by KNN when $k = 3$ in [64]. The classifier of boosted LeNet4 in [64] which combines three neural network classifiers and has therefore extremely high complexity attains the best accuracy rate of 99.3% for the handwritten digits recognition.

It may seem unfair that a learning system using 1-NN in the classification stage is compared against the FPGA implementation of KNN when $k = 20$ in the previous works [11], [24]. The large $k$ in KNN is expected to reduce the influences of noise and improve the accuracy rate. However, the experimental results in Fig. 9 prove that 1-NN is better than 3-NN or 5-NN and therefore the best choice in this work. The results in Fig. 9 are using the prototypes which were produced by clustering 60000 training samples. The accuracy rates using the prototypes generated by clustering 10000 and 30000 samples have the same tendency where $k = 1$ is the better than $k = 3$ or $k = 5$. For this reason, the coprocessor for 1-NN was used both in K-means-based learning and 1-NN-based classification.
5.3 Effects of the Bit Precision

The definition of the bit precision is a fixed point encoding of 16-bit in this work according to a statistical study for the precision requirements in the neural networks hardware [61]. After all, the neural networks hardware is different from the NN-based learning algorithm. The scaling factor is a power of 10 owing to the flexible software pre-processing. The 16-bit precision was obtained by the truncation technique which chops the lowest order bits after 16\textsuperscript{th} bit. The bit precision affects the resource usages of AUs for calculating the Euclidean distance. According to the experimental results in Fig. 10, the accuracy rate has no obvious difference between the 32-bit and 16-bit precision.

5.4 Realization of the K-Means-Based Multi-Prototype Learning System

The software-hardware cooperation was used to implement the K-means-based prototype learning system. The coprocessor was implemented on an Altera Stratix III E110 FPGA device which is a memory- and multiplier-rich device and has 107.5 K logic elements (LEs). The cost of data transferring depends on the transmission rate of the DMA controller and the bandwidth of both the PCI-e bus and the local bus. Meanwhile, the centroids updating is decided by the speed of the CPU in the host PC. The nearest distance searching, which is the critical path in a conventional software implementation, was accelerated by the hardware in this work. When the speed of the nearest distance searching is close to anyone of the other two tasks (data transferring and centroids updating), the implemented parallelism of the hardware processing can be considered as optimal.

According to the bit width of the local bus and the external memory on the PROCeIII board, the word length of each memory block (\(\nu\)-bit) was defined as 128-bit. The write operation to the memory for prototypes is simplest when \(\rho = 128\)-bit. The iteration to find the stabilized prototypes for all class categories by K-means constitutes one learning cycle. The K-means-based multi-prototype learning system was applied to recognize the handwritten digits. In Sect. 5.2, the accuracy rates have been estimated by three different group sizes of the training samples. The reserved FPGA on-chip memory for the prototypes, the cluster centroids in K-means, can store 1024 feature vectors. This means the maximum number of prototypes for each digit becomes 102 (\(k_{\max} = 102\)). In this work, the parallelism (\(\rho\)) plays an important role in improving the speed for 1-NN. A higher parallelism requires certainly more hardware resources and attains higher speed. The comparison of the resource usages between \(\rho = 128 - \text{bit}\) and \(\rho = 512 - \text{bit}\) is illustrated in Table 2 to provide the evidence for judging the optimal parallelism. The data exchange in the K-means based learning between the coprocessor and the host PC is achieved via the on-board local bus and the PCI-e bus. The interface unit for the local bus which is automatically generated by the GiDEL software was implemented on the same FPGA device in addition to the coprocessor. The resource usages of the interface unit is listed in Table 3. The coprocessor with a parallelism of \(\rho = 512 - \text{bit}\) requires almost the same number of resources as the interface unit for the communication with the host PC.

Then the speed comparison of the learning procedure using different numbers of the training samples and different numbers of prototypes for each digit is shown in Table 4. One learning cycle contains the processing of step 1 and step 2 in Sect. 2. The cost of each learning cycle has three components: (a) the data transferring via PCI-e bus and on-board local bus by DMA, (b) the nearest distance searching by the FPGA-implemented coprocessor for 1-NN searching, and (c) updating the cluster centroids by software. The host PC used to update the prototypes is a Phenom II \(\times 4\) 840 3.2 GHz

### Table 2: Comparison of the resource usages of the FPGA-implemented coprocessor for 1-NN.

| Resources | \(\rho = 128 - \text{bit}\) | \(\rho = 512 - \text{bit}\) |
|-----------|-----------------|-----------------|
| LEs       | 3350 (3.9%)     | 11028 (12.9%)  |
| Registers | 3564 (4.2%)     | 8279 (9.7%)    |
| Memory bits | 4,806,672 (58%) | 4,806,672 (58%) |
| AU        |                 |                 |
| Subtractor | 8               | 32              |
| Squerer   | 8               | 32              |
| Adder     | 8               | 32              |
| Storage capability for prototypes | 1024 |                 |
| Dimension of feature vector | 256 |                 |
| Working frequency | 250 MHz |                 |

### Table 3: Resource usages of the interface unit for the data exchange between the FPGA-implemented coprocessor for 1-NN and host PC via on-board local bus and PCI-e.

| Resources | \(\rho = 128 - \text{bit}\) | \(\rho = 512 - \text{bit}\) |
|-----------|-----------------|-----------------|
| LEs       | 10931 (12.8%)   | 10969 (12.9%)   |
| Registers | 13024 (15.3%)   | 13048 (15.3%)   |
| Memory ALUTs | 640 (2%) |                 |
with 4 GB of DDR memory. In order to obtain the converged prototypes, a certain number of the learning cycles is required. The speed of the learning procedure (s/sample) of both the easy case ($k = 10$) and the hard case ($k = 102$) is shown in Table 4. The higher parallelism attains the more efficient learning speed. In the analysis for the accuracy rate, the learning system with software-hardware cooperation for learning can attain competitive results provided that sufficient training data are available and that there is sufficient computing capability to process them.

### 5.5 Comparison of the Nearest Distance Searching Speed

A fair comparison of learning algorithms is very difficult because the used feature vector plays an important role in the classification as well. In order to avoid the effects of the size of the feature vector, the speed was normalized to how many bits can be proceed per second. Besides the speed of each calculation, the parallelism is also important because higher parallelism can attain a more efficient searching speed, which determines the usage of hardware resources and the throughput. The process technology for the gate length of each transistor affects the operating frequency as well. The hardware resource of FPGAs is usually expressed by the number of logic elements, which are the basic hardware units of an FPGA. The size and complexity of a logic element vary from manufacturer to manufacturer. Particularly, the gate count is an ASIC metric while logic cell count is an FPGA metric. Either metric for hardware-complexity comparison is meaningful only in its own world. Therefore, it is difficult to compare the complexity of those two classes of hardware implementations.

The differences among the previous works and our work are illustrated in Table 5. The process technology also plays a significant role for the speed because smaller transistors switch faster. The shrinking effect of transistors can be estimated by the ideas of scaling [58]–[60]. A larger transistor is scaled down by a factor $\alpha$ to attain the corresponding behavior of a smaller transistor. The scaling factor can be coarsely estimated from the ratio of the gate length.

For example the scaling factor $\alpha$ is 0.72 when the previous FPGA implementation with 90 nm technology in [24] is scaled down to produce similar hardware performance as the FPGA with 65 nm technology in this work. With the scaling factor $\alpha$ for the gate length, the scaling factor for the speed becomes $1/\alpha$. At last, the normalized recognition speed ($\text{bit}/s$) in Fig. 11 also considers this scaling factor. However, the ASIC implementation for the nearest Euclidean distance searching [57] is difficult to be scaled down since an analog circuit is used, which expresses the distance by a voltage difference. The paper [57] indicates that the voltage has to be large enough to guarantee the requested search reliability, so that scaling is not appropriate.

The recognition speed ($\text{bit}/s$) of this work outperforms the normalized speed of the software, GPU, previous FPGA, and even the ASIC implementations as illustrated in Fig. 11. Each dimension of the feature vector is considered as $32$-bit in the CPU and GPU implementation. To attain the best accuracy rate in [11], [24], $k$ is equal to 20, which requires high computational costs. Except for the circuit area and power consumptions, the FPGA-implemented coprocessor for 1-NN used in both the learning and recognition outperforms even the mixed digital-analog circuit [57] in the cost, applicability, speed, and reliability.

We also compare our FPGA-implemented coprocessor for 1-NN searching to advanced nearest neighbor search-

### Table 4 Execution time of one learning cycle for different levels of parallelism.

| $k$ | $\rho = 128$-bit | $\rho = 512$-bit |
|-----|------------------|------------------|
|     | $N = 10000$ | $N = 30000$ | $N = 60000$ | $N = 10000$ | $N = 30000$ | $N = 60000$ |
| 10  | 2.1  | 2.2  | 2.2  | 2.1  | 2.1  | 2.1  |
| 102 | 2.1  | 2.1  | 2.1  | 2.1  | 2.1  | 2.1  |

### Table 5 Differences in parallelism, process technology, and hardware-resource utilization among the previous works and this work.

|                     | GPU [11] | FPGA [24] | ASIC [57] | This work |
|---------------------|----------|-----------|-----------|-----------|
| Distance metric     |          |          |           |           |
| Throughput          | 96       | 128       | 1024      | 32        |
| $k$                 | 20       | 20        | 1         | 1         |
| Hardware platform   | GeForce 8800 GTX | Virtex-II Pro | n/a       | Stratix-III E110 |
| Process technology  | 90 nm    | 90 nm     | 350 nm    | 65 nm     |
| Hardware resource   | 175 M gates | 22 K logic cells | 46.5 K gates | 11 K logic cells |
ing techniques such as the KD-tree and the hash table approach. In the case of a hash table, one of the most popular approximate nearest neighbor algorithms is the Exact Euclidean Locality-Sensitive Hashing (E2LSH) which is an algorithm for solving both exact and approximate nearest neighbor searches in high dimensional spaces. As shown in Fig. 12, the hardware implementation outperforms even this sophisticated approach of software implementation on a PC with a 3.2 GHz CPU and 4 GB of DDR memory. The success probability of E2LSH for the approximate nearest neighbor searching was specified by 90%. We increased the number of references, each of which has 256 dimensions, to estimate the search time. The search time curve in Fig. 12 shows that our hardware implementation with low cost is about 100 times faster than the KD-tree, and still 10 times faster than as software approach with E2LSH for approximate nearest neighbor search.

6. Conclusion

As an alternative to the hardware implementations of neural networks, support vector machines, K-means, or KNN, the K-means-based multi-prototype learning system with software-hardware cooperation is proposed in this work. The FPGA-implemented coprocessor for 1-NN solves the computational burden in both K-means-based learning and 1-NN-based classification. The learning system is benchmarked against the previous works on handwritten digit recognition. The experimental results demonstrate that the recognition speed is remarkably faster than the software, GPU, previous FPGA, and even analog circuit implementations owing to the highly parallel and pipelined architecture. The accuracy rate for handwritten digit recognition of the MNIST database is better than KNN although the precision of feature-vector component representation was decreased to 16-bit in the hardware implementation. It is worthwhile to point out that the handwritten digit recognition is used to demonstrate the efficiency of our learning system, which has high flexibility to be adapted to other applications with feature vector less or equal to 256 dimensions.

References

[1] T.M. Cover and P.E. Hart, “Nearest neighbor pattern classification,” IEEE Trans. Inf. Theory, vol.13, no.1, pp.21–27, 1967.
[2] P.J. Clark and F.C. Evans, “Distance to nearest neighbor as a measure of spatial relationships in population,” Ecology, vol.35, no.4, pp.445–453, 1954.
[3] J. Max, “Quantizing for minimum distortion,” IEEE Trans. Inf. Theory, vol.6, no.1, pp.7–12, 1960.
[4] Y. Linde, A. Buzo, and R. Gray, “An algorithm for vector quantizer design,” IEEE Trans. Commun., vol.19, no.1, pp.84–95, 1980.
[5] S. Lloyd, “Least squares quantization in PCM,” IEEE Trans. Inf. Theory, vol.28, no.2, pp.129–137, 1982.
[6] J.C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum, New York, 1981.
[7] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler, Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition, John Wiley & Sons, New York, 1999.
[8] T. Kohonen, Self-Organization and Associated Memory, Springer-Verlag, Berlin, 1988.
[9] T. Hiroyuki and K. Hiroaki, “Hierarchical parallel processing of large scale data clustering on a PC cluster with GPU co-processing,” J. Supercomputing, vol.36-3, pp.219–234, 2006.
[10] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover, “GPU cluster for high performance computing,” ACM/IEEE Supercomputing
AN et al.: A K-MEANS-BASED MULTI-PROTOTYPE HIGH-SPEED LEARNING SYSTEM WITH FPGA-IMPLEMENTED COPROCESSOR FOR 1-NN SEARCHING

[11] V. Garcia, E. Debreuve, and M. Barland, “Fast k nearest neighbor search using GPU,” Proc. Computer Vision and Pattern Recognition Workshops, pp.1–6, 2008.

[12] B. Bustos, O. Deussen, S. Hiller, and D. Keim, “A graphics hardware accelerated algorithm for nearest neighbor search,” Proc. 6th International Conference on Computational Science, Vol.3994 of Lecture Notes in Computer Science, Springer, pp.196–199, 2006.

[13] B. Malisaitis and O. Yadid-Pecht, “Hardware-driven adaptive K-modes clustering for real-time video imaging,” IEEE Trans. Circuits Syst. Video Technol., vol.15, no.1, pp.164–166, 2005.

[14] Y. Ma and T. Shibata, “A real-time K-means learning processor architecture extendible to multiple-chip large-scale systems,” Proc. 34th Eur. Solid-State Circuits Conf., p.3, 2008.

[15] Y. Ma and T. Shibata, “A binary-tree hierarchical multiple-chip architecture for real-time large-scale learning processor systems,” Jpn. J. Appl. Phys., vol.49, no.4, 04DE08, 2010.

[16] T. Chen, C. Sun, J. Bai, H. Chen, and S. Chien, “Architectural analyses of K-modes silicon intellectual property for image segmentation,” Proc. IEEE Int. Symp. Circuits Syst., pp.2578–2581, 2008.

[17] T. Chen and S. Chien, “Bandwidth adaptive hardware architecture of K-modes clustering for video analysis,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.18, no.6, pp.957–966, 2010.

[18] T. Chen and S. Chien, “Flexible hardware architecture of hierarchical K-means clustering for large cluster number,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.19, no.8, pp.1336–1345, 2011.

[19] M. Esfandi, M. Leeser, J. Theiler, and J.J. Szymanski, “Algorithmic transformations in the implementation of K-means clustering on reconfigurable hardware,” Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays, pp.103–110, 2001.

[20] A.G. Da, S. Filho, A.C. Frery, C.C. De Araújo, H. Alice, J. Cerqueira, J.A. Loureiro, M.E. De Lima, M. Das, G.S. Oliveira, and M.M. Horta, “Hyper spectral images clustering on reconfigurable hardware using the K-means algorithm,” Proc. Symp. Integrated Circuits Syst. Design, pp.99–104, Sept. 2003.

[21] T. Saegusa and T. Maruyama, “An FPGA implementation of real-time K-means clustering for color images,” J. Real-Time Image Process., vol.2, no.4, pp.309–318, 2007.

[22] Y. Yeh, H. Li, W. Hwang, and C. Fang, “FPGA implementation of k-NN classifier based on wavelet transform and partial distance,” Scandinavian Conf. on Image Analysis, vol.4633, pp.512–521, 2007.

[23] J. Liu, B. Li, and D. Liang, “Design and implementation of FPGA-based modified BKNN classifier,” Int. J. Comput. Sci. Netw. Secur., vol.7, no.3, pp.67–71, 2007.

[24] E.S. Manolakos and I. Stamoulis, “IP-cores design for the KNN classifier,” Proc. 2010 ISCAS, pp.4133–4136, 2010.

[25] J. Friedman, J.L. Bentley, and R.A. Finkel, “An algorithm for finding best matches in logarithmic expected time,” ACM Trans. Mathematical Software, vol.3, no.3, pp.209–226, 1977.

[26] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, “Locality sensitive hashing scheme based on p-stable distributions,” Proc. Symposium on Computational Geometry, pp.253–262, 2004.

[27] M. Muja and D.G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” Proc. International Conference on Computer Vision Theory and Applications, pp.331–340, 2009.

[28] Y. Weiss, A. Torralba, and R. Fergus, Spectral hashing, Advances in Neural Information Processing Systems, 2009.

[29] H. Jegou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” IEEE Trans. Pattern Anal. Mach. Intell., vol.33, no.1, pp.117–128, 2010.

[30] J.N.H. Heemskerk, Overview of Neural Hardware. Neurocomputers for Brain-Style Processing. Design, Implementation and Application, PhD Thesis, Unit of Experimental and Theoretical Psychology Leiden University, Netherlands, 1995.

[31] M. Misra, “Parallel environments for implementing neural networks,” Neural Computing Survey, vol.1, pp.48–60, 1997.

[32] J. Zhu and P. Sutton, “FPGA implementation of neural networks—a survey of a decade of progress,” Proc. 13th International Conference on Field Programmable Logic and Applications (FPL 2003), pp.1062–1066, 2003.

[33] F.M. Dias, A. Antunes, and A.M. Mota, “Artificial neural networks: A review of comercial hardware,” Engineering Applications of Artificial Intelligence, vol.17, pp.945–952, 2004.

[34] J. Misra and I. Saha, “Artificial neural networks in hardware: A survey of two decades of progress,” Neurocomputing, vol.74, pp.239–255, 2010.

[35] R.A. Reyna, D. Esteve, D. Houzet, and M.-F. Albenge, “Implementation of the SVM neural network generalization function for image processing,” 5th IEEE International Workshop on Computer Architectures for Machine Perception, p.147, 2000.

[36] R. Genov and G. Cauwenberghs, “Kernelon: Support vector machine in silicon,” IEEE Trans. Neural Netw., vol.14, no.5, pp.1426–1434, 2003.

[37] D. Anguita, A. Boni, and S. Ridella, “A digital architecture for support vector machines: Theory, algorithm, and FPGA implementation,” IEEE Trans. Neural Netw., vol.14, no.5, pp.993–1009, 2003.

[38] B.A. Boni and A. Zorat, “A reconfigurable parallel architecture for SVM classification,” IEEE International Joint Conference on Neural Networks, vol.5, pp.2867–2872, 2005.

[39] F. Khan, M. Arnold, and W. Pottenger, “Hardware-based support vector machine classification in logarithmic number systems,” IEEE International Symposium on circuits and systems, p.5154, 2005.

[40] B.A. Boni and A. Zorat, “FPGA implementation of support vector machines with pseudo-logarithmic number representation,” Neural Networks, pp.618–624, 2006.

[41] O. Pina-Ramirez, R. Valdes-Cristerna, and O. Yanez-Suarez, An FPGA implementation of linear kernel support vector machines, IEEE International Conference on Reconfigurable Computing and FPGA’s, pp.1–6, 2006.

[42] M. Grca, B. Fortuna, D. Mladenic, and M. Grobelnik, “KNN versus SVM in the collaborative filtering framework,” ACM WebKDD 2005 Taming evolving, Expanding and Multi-faceted Web Click streams Workshop, pp.251–260, 2005.

[43] W. Duelemans, A. Bosch, Memory-based Language Processing, Cambridge University Press, 2005.

[44] C.L. Liu and M. Nakagawa, “Evaluation of prototype learning algorithms for nearest neighbor classifier in application to handwritten character recognition,” Pattern Recognit., vol.34, no.3, pp.601–615, 2001.

[45] V. Vaidhehi and S. Vasuhi, “Person authentication using face recognition,” Proc. World Congress on Engineering and Computer Science, 2008.

[46] Han E, Text categorization using weight adjusted k-nearest neighbor classification. PhD thesis, University of Minnesota, 1999.

[47] Y. Liao and V.R. Vemuri, “Using text categorization techniques for intrusion detection,” Proc. 11th USENIX Security Symposium, pp.51–59, 2002.

[48] X. Geng, T.Y. Liu, T. Qin, A. Arnold, H. Li, and H.-Y. Shum, “Query dependent ranking using k-nearest neighbor,” Proc. 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.115–122, 2008.

[49] Y. Yang, T. Ault, T. Pierce, and C.W. Lattimer, “Improving text categorization methods for event tracking,” Proc. 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.65–72, 2000.

[50] F. Bajramovic, F. Mattern, N. Butko, and J. Denzler, “A comparison of nearest neighbor search algorithms for generic object recognition,” Proc. Advanced Concepts for Intelligent Vision Systems (ACIVS), pp.1186–1197, 2006.

[51] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, Wiley Interscience, New York, 1973.

[52] J.B. MacQueen, “Some methods for classification and analysis of multivariate observations,” Proc. 5th Berkeley Symposium on Mathe-
Fengwei An received the B.E. of computer science from Qingdao University of Science and Technology, China, in July 2006. Then he worked in an IT Company from August 2006. He received his M.E. from Graduate School of Information Engineering of Hiroshima University, Japan, in March 2010. Now, he is studying his doctoral degree in Graduate School of Advanced Science of Matter of Hiroshima University where he has been engaged the hardware-implemented Machine learning for real-time applications.

Tetsushi Koide received the B.E. degree in Physical Electronics, the M.E. and the Ph.D. degrees in Systems Engineering from Hiroshima University in 1990, 1992, and 1998, respectively. Now, he is an Associate Professor in the Research Center for Nanodevices and Systems and Graduate School of Advanced Sciences of Matter, Hiroshima University respectively. His research interests include system design and architecture issues for functional-memories-based intelligent architecture and real-time image processing VLSI. Dr. Koide is a member of IEEE.

Hans Jürgen Mattausch received the doctor degree from the University of Stuttgart, Stuttgart, Germany, in 1981. He is a Professor at the Research Institute for Nanodevice and Bio Systems and the Graduate School for Advanced Sciences of Matter, Hiroshima University, Higashi-hiroshima, Japan. From 1982 to 1996 he was with Siemens AG in Munich, Germany, where he was involved in the development of CMOS technology, memory and telecommunication circuits, power semiconductor devices, chip-card ICs and compact models. Since 1996 he is a Professor with Hiroshima University researching in the fields of VLSI design, nano-electronics and compact modeling. Dr. Mattausch is a senior member of IEEE.

[53] J.C. Bezdek, T.R. Reichherzer, G.S. Lim, and Y.A. Attikiouzel, Multiple-prototype classifier design, IEEE Trans. Syst. Man Cybern. C, Appl. Rev., vol.28, no.1, pp.67–69, 1998.
[54] H.J. Mattausch, T. Gyohten, Y. Soda, and T. Koide, “Compact associative-memory architecture with fully-parallel search capability for the minimum hamming distance,” IEEE J. Solid-State Circuits, vol.37, no.2, pp.218–227, 2002.
[55] Y. Yano, T. Koide, and H.J. Mattausch, “Associative memory with fully parallel nearest-manhattan-distance search for low-power real-time single-chip applications,” Asia and South Pacific Design Automation Conference, pp.543–544, 2004.
[56] H.J. Mattausch, W. Imafuku, T. Ansari, A. Kawabata, and T. Koide, “Low-power word-parallel nearest-hamming-distance search circuit based on frequency mapping,” 36th European Solid-State Circuits Conference, pp.538–541, 2010.
[57] Md. A. Abedin, Y. Tanaka, A. Ahmadi, T. Koide, and H.J. Mattausch, “Mixed digital-analog associative memory enabling fully parallel nearest euclidean distance search,” Jpn. J. Applied Phys., vol.46, pp.2231–2237, 2007.
[58] R.H. Dennard, F.H. Gaensslen, H.N. Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc, “Design of ion-implanted MOSFETs with very small physical dimensions,” IEEE J. Solid-State Circuits, vol.SC-9, no.4, pp.256–268, 1974.
[59] D.L. Critchlow, “MOSFET scaling—The driver of VLSI technology,” Proc. IEEE, vol.87, pp.659–667, 1999.
[60] D.J. Frank, R.H. Dennard, E. Nowak, P.M. Solomon, Y. Taur, and H.P. Wong, “Device scaling limits of Si MOSFETs and their application dependencies,” Proc. IEEE 89, pp.259–288, 2001.
[61] J. Holt and J.N. Hwang, “Finite precision error analysis of neural network hardware implementations,” IEEE Trans. Comput., vol.42, no.3, pp.281–290, 1993.
[62] http://yann.lecun.com/exdb/mnist/, The MNIST database of handwritten digits.
[63] C. Liu, K. Nakashima, H. Sako, and H. Fujisawa, “Handwritten digit recognition: Benchmarking of state-of-the-art techniques,” Pattern Recognit., vol.36, pp.2271–2285, 2003.
[64] Y.A. LeCun, L.D. Jackel, L. Bottou, A. Brunot, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, U.A. Muller, E. Sackinger, P.Y. Simard, and Y.N. Vapnik, “Comparison of learning algorithms for handwritten digit recognition,” Proc. International Conference on Artificial Neural Networks, vol.2, pp.53–60, 1995.