Engineering Applications of Artificial Intelligence 68 (2018) 53–62

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Accelerating nearest neighbor partitioning neural network classifier based on CUDA

Lin Wang [a,*], Xuehui Zhu [a], Bo Yang [a], Jifeng Guo [a], Shuangrong Liu [a], Meihui Li [a], Jian Zhu [c], Ajith Abraham [b]

[a] Shandong Provincial Key Laboratory of Network based Intelligent Computing, University of Jinan, Jinan, 250022, China
[b] Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, Auburn, WA, 98071, USA
[c] Department of Radiation Oncology, Shandong Cancer Hospital and Institute, Jinan 250117, China

ARTICLE INFO

Keywords:
Parallel nearest neighbor partitioning
Neural network classifier
Compute unified device architecture
Graphics processing units

ABSTRACT

The nearest neighbor partitioning (NNP) method is a high performance approach which is used for improving traditional neural network classifiers. However, the construction process of NNP model is very time-consuming, particularly for large data sets, thus limiting its range of application. In this study, a parallel NNP method is proposed to accelerate NNP based on Compute Unified Device Architecture(CUDA). In this method, blocks and threads are used to evaluate potential neural networks and to perform parallel subtasks, respectively. Experimental results manifest that the proposed parallel method improves performance of NNP neural network classifier. Furthermore, the application of parallel NNP in performance evaluation of cement microstructure indicates that the proposed approach has favorable performance.

1. Introduction

Classification is an useful tool in the field of data mining and machine learning. It aims to learn a classification function based on existing data or to construct a classification model, i.e., the "Classifier". After learning, the function or model not only maps the data records in the training data to a specific class but also is able to predict the unknown samples. Specifically, the classifier is a general term for classifying the samples in data mining, including neural networks (Lu et al., 1996; Misra et al., 2008; Gao and Ji, 2005; Hassan, 2011), support vector machines (López and Suykens, 2011; Vapnik, 1997; Chua, 2003), rule-based system, decision tree (Quinlan, 1986; Freund, 1995), data gravitation (Peng et al., 2014, 2017) etc. Artificial neural network is a general model of approximating nonlinear function, which directly maps samples to centroid which belongs a class. It has been successfully used in many classification tasks (Casta et al., 2011; An et al., 2011; Avci, 2012; Yaakob and Jain, 2012; Kang and Park, 2009).

In the traditional neural network, the position, number, and labels of centroids are permanent in training neural networks. However, in the optimization process, mapping the samples to fixed centroids reduces the possibility of finding optimal neural network. Thus the floating centroid method (FCM) (Wang et al., 2012) was proposed to solve the above problems. Its centroids are obtained through clustering method (Hartigan and Wong, 1979; Zhou et al., 2016) and distributed automatically in the entire partition space. However, the FCM also has its limitation, e.g., it cannot yield flexible decision boundaries. The nearest neighbor partitioning (NNP) (Wang et al., 2017) method overcomes that limitation by generating flexible decision boundary in a sphere-like zone. The goodness of each possible neural network in NNP is evaluated using a nearest-neighbor criterion. NNP is able to easily yield flexible decision boundaries and partitions, thus increases the probability of discovering optimal solution.

Despite the NNP's significant improvement of accuracy, its efficiency, particularly in solving large size data, is still problematic. The collection of scientific data is getting easier due to the progress of experimental devices and technologies. However, in recent years, the efficiency problem of approaches for big data has drawn a widely attention. In terms of NNP, the similarity computation in its training stage is an important procedure of the nearest-neighbor criterion (Baraldi et al., 2016; Yu et al., 2015) and its use in the target function results in the distribution of evaluated points are essential. However, the dramatic increase in data size increases the complexity of similarity computation and thus affects the efficiency of NNP. Furthermore, the efficiency of other stages in NNP, including sample mapping, normalization, and target function calculation, are also strongly influenced by the size of

* Corresponding author.
E-mail address: ise_wanglin@ujn.edu.cn (L. Wang).

https://doi.org/10.1016/j.engappai.2017.10.023
Received 26 May 2017; Received in revised form 1 October 2017; Accepted 30 October 2017
Available online 21 November 2017
0952-1976/

data. Although the flexible partition and boundary of the NNP method enlarge the possibility of finding an optimal neural network, learning large-scale data set is still difficult and time-consuming for NNP and restricts its scope of application.

The compute unified device architecture(CUDA), which runs on the Graphics Processing Unit (GPU) (Gong et al., 2016), brings the dawn to overcome the efficiency disadvantage of nearest neighbor partitioning on low-cost platform. CUDA regards GPU as a high-performance device that implements the management and allocation of tasks. With the use of CUDA, it is easier for scientists and engineers to develop general purpose computing intensive applications. Due to the high computational complexity that NNP faces, a CUDA parallel computing based strategy is proposed to accelerate NNP's training process. In this method, blocks and threads work together to perform the parallelization of nearest neighbor partitioning where blocks are responsible for the evaluation of neural networks and threads are responsible for the computation of parallel subtasks, respectively.

This paper is organized as follows. Section 2 presents the background, related works of the NNP and CUDA programming. Section 3 describes the motivation of this study. Section 4 outlines the message passing mechanism and thread allocation in the CUDA-based parallel computing. Section 5 describes the experimental results and shows its application to performance evaluation of cement microstructure, followed by the conclusions in Section 6.

2. Background and related works

2.1. Neural network classifiers

Neural network classifier has been used for solving different types of problems successfully. Oujaoura (2012) proposed a comparative evaluation of the image content annotation system through the nearest neighbor classifier and the multilayer neural networks. Kaur (2013) proposed a neural network classifier to check the state of a patient in its early stage and predicted the survival rate of a patient by extracted features. Kora and Kalva (2015) presented a method for extracting optimal characteristics from each cardiac beat with improved bat algorithm, which further improve the performance of the neural network. Attia et al. (2015) presented a computer-aided system for automatic classification of Ultrasound Kidney diseases.

For binary neural network classifier, its output is either one or zero. One for each class(one-per-class) method is applied to the multi-class classification problem, which has several output values and each output value is the estimated probability that a sample belongs to its corresponding class. But in that case, the probabilistic summary of outputs is unlikely to be equal to one. Therefore, Bridle (1990) used the softmax activation function (SoftMax) to adjust the total of probabilities so that it is always one. SoftMax method solved the aforementioned problem and the classes are mutually exclusive, i.e., an input vector can only be classified as a class. Dietterich and Bakiri (1995) presented an error-correcting output code (ECOC) method to further improve the robustness properties, which uses the redundant dimension to correct the error. Wang et al. (2012) proposed a floating centroid method to remove the limitation of fixed-centroids and further improved the performance of neural network classifier. However, it cannot generate flexible decision boundaries and its target function only applies to spherical centroid-based methods. Therefore, NNP (Wang et al., 2017) was proposed to deal with the aforementioned problems. The main advantage of the NNP is that it yields flexible decision boundaries between different clusters. Therefore, compared with the other methods, NNP generates flexible boundaries and removes the limitation of spherical shape boundary from centroid-based methods. This strategy increases the chance of finding the best neural network.

NNP adopts the particle swarm optimization (PSO) (Li et al., 2016a; Wang et al., 2014b, 2016b; Haddar et al., 2016) to optimize the weights and bias of neural networks (Wang and Orchard, in press). Each particle is encoded as a vector of weights. In each generation of optimization, when evaluating particles, samples are mapped to the partition space by decoded particle, i.e. neural networks, and is normalized using the Z-score (O'Neil, 2010). The normalized mapped samples (points) are further constrained into a hypersphere. A nearest neighbor criterion target function is used to evaluate the quality of the distribution of points in partition space. After optimization, a model of optimal neural network is found. In order to predict the unknown sample, the sample is mapped by optimal neural network at first and is further categorized by k-nearest neighbor. The general procedure of NNP is shown in Fig. 1.

2.2. Compute unified device architecture

The CUDA parallel computing architecture, which was launched by NVIDIA, is a revolutionary general purpose computing architecture in recent years. It contains instruction set architecture(ISA) and parallel computation engine within the GPU so that it enables GPU to solve complex computational problems. As a technology that supports both hardware and software, CUDA simultaneously uses multiple GPU cores to perform general computation processing. With the use of this architecture, the computing performance can be significantly improved. The program can be run in a high performance on CUDA processors. The program code based on CUDA is segmented into two categories in practice, one is the host code running on the CPU, the other one is the device code running on the GPU. The parallel program runs on the GPU is called the kernel. When CUDA deals with parallel computation, the threads constitutes block and blocks constitutes grid. Then the same kernel code performs on a grid in parallel. The framework of CUDA is shown in Fig. 2. It can be seen from the figure that there are several streaming multiprocessors(SM) in each GPU. Each multiprocessor includes several streaming processors(SP) and has a set of local 32-bit registers.

In recent years, CUDA has been used for accelerating intelligent computing methods. Anderson and Coupland (2008) explored the use of GPU and NVIDIA's CUDA for fast inference speeds in a flexible Mamdani type fuzzy inference system (FIS). Based on multi-core CPU and GPU, Jang et al. (2008) proposed a quick and effective method of implementation of neural networks. Li et al. (2011) presented GPUSVM, a core package for an efficient cross validation tool, a fast training tool and a predicting tool. Wu and Hong (2011) proposed an effective k-means algorithm based on CUDA, which reaches better efficiency using the triangle inequality. Juang et al. (2011) proposed the implementation of a zero-order Takagi–Sugeno–Kang (TSK)-type fuzzy neural network (FNN) on GPU. Sun et al. (2013) proposed a fast face detection algorithm by the Viola–Jones cascade classifier based on CUDA, which improves novel parallel methodologies of image integral calculation, scan window processing and the amplification and correction of classifiers. Wang et al. (2014c) proposed a parallel floating centroids method(PFCM) to accelerate the FCM in CUDA platform where blocks and threads are used for evaluating classifiers and performing subtasks separately.

3. Motivation

With the rapid development of technologies, humans are entering the age of big data. The total amount of data in the world is growing at a rate of double every two years. How to efficiently manage and take advantage of big data has been a hot issue today. Big data brings many issues for traditional computing technologies. Many traditional approaches, which are valid on processing small size data, is very time-consuming when the size of data increases sharply. Therefore, methods on high performance platform, are required for large size data to reduce its computational time.

As a promising neural network classification method, NNP exhibits its advantage of accuracy in many fields. However, despite its accuracy, the time complexity of adopted strategies is very high, especially for the computation of similarity matrix. With the increase of data size, its time complexity increases dramatically and thus slows down its

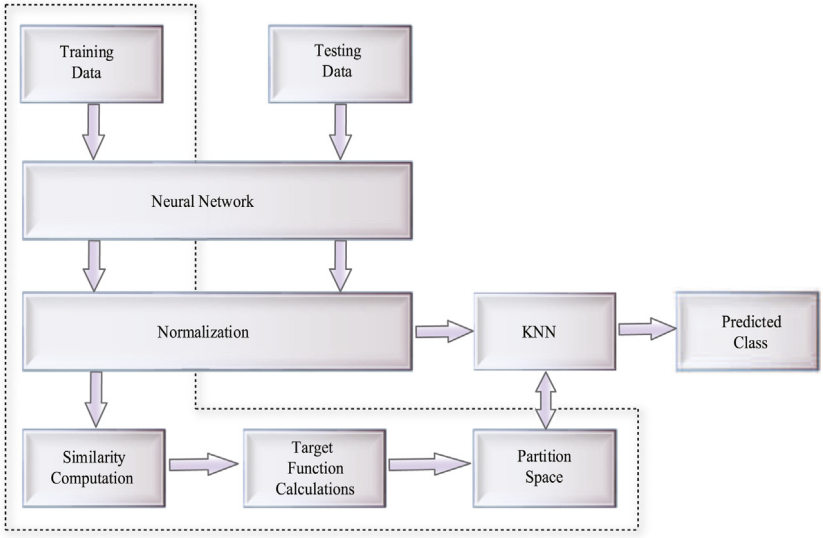

Fig. 1. The framework of nearest neighbor partitioning.

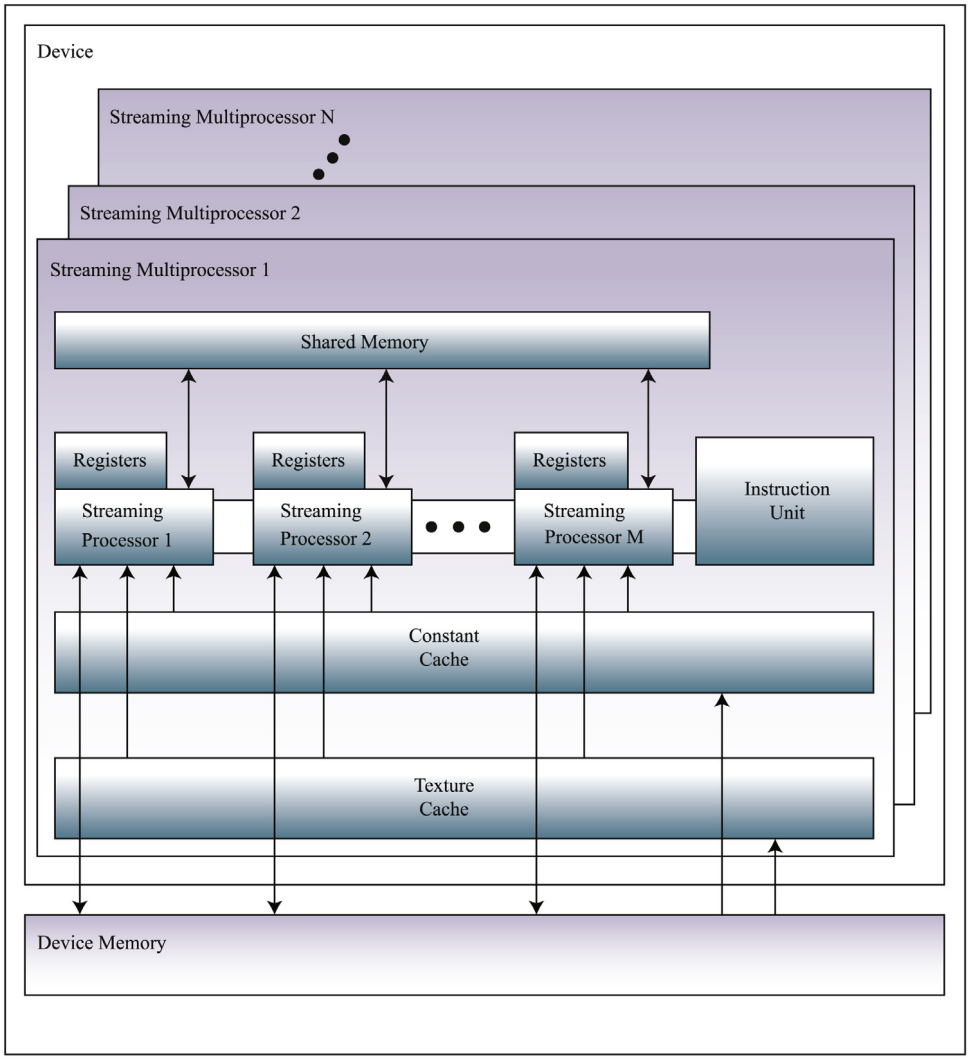

Fig. 2. The framework of CUDA.

execution speed. The computation of sample mapping, normalization and target function also have the same problem. All those issues together influenced its effect coping with large size data. Therefore, a question that arises here is how to accelerate nearest neighbor partitioning for large size data.

The CUDA opens a door to solve computationally expensive problems on low-cost platform. At the level of micro-architectures, GPU uses more resources for computation. It takes advantage of the parallel operation of non-logical relational data. In addition, CUDA regards GPU as a parallel machine of the single instruction multiple data stream (SIMD) which executes the same instruction simultaneously across multiple processors. In terms of nearest neighbor partitioning, CUDA provides a low-cost solution for time-consuming problem of NNP algorithm. In NNP, each particle corresponds to a neural network which is independent to each other during evaluation. Furthermore, the subtasks of evaluation also fit data parallelism. Therefore, the development of CUDA and the computationally expensive problem faced by NNP promote us to study a CUDA based parallelization approach across multiple GPU processors to accelerate the training process of NNP.

4. Parallel nearest neighbor partitioning in CUDA

For the learning of the NNP, an optional neural network is evaluated and is updated based on fitness evaluation. Because evaluation tasks are time consuming, thus parallelization is required. In each iteration, there are four stages for evaluating each neural network: sample mapping, normalization, similarity computation and target function calculation. The time complexities of all stages are high and all of these phases need to be parallelized, especially for large-scale data set. Therefore, a parallel nearest neighbor partitioning (PNNP) method based on CUDA is proposed in this paper.

Based on CUDA's parallel architecture, the particle swarm optimization is performed at the CPU side while the classifier is evaluated at the GPU side. That is, when the weight and bias of the neural network are provided on the CPU side, sample mapping, normalization, similarity computation and target function calculation are performed on the GPU side. On the one hand, updating weights on the CPU requires the GPU to calculate each step in parallel. Thus, CPU has been waiting until the fitness values of the model are returned by the GPU. On the other hand, each block is responsible for evaluating several potential models at the GPU side. Threads in each block control the computation of four sub-stages. The parallel relationship of the PNNP is illustrated in Fig. 3.

In application, an optimized neural network and its corresponding partition space can be established after training. Therefore, when an unknown sample arrives, it is mapped to the partition space at first by optimized neural network. Then, the sample $\mathbf{x}$ is normalized as follows

$$\mathbf{x}' = (\mathbf{x} - \mu)/\sigma \tag{1}$$

where μ and σ are obtained from training set. Then, the normalized sample is bounded to a hypersphere. Finally, the class of the unknown sample is labeled as a class by the weighted K-nearest neighbors.

PNNP focuses on improving efficiency of NNP, particularly for large data sets. In terms of granularity, a small granularity will result in the decrease of overall speed on account of the extra cost of transmission, but a very large granularity will also lower the utilization rate of resources. Thus, the sample mapping, normalization, similarity computation and target function calculation are processed per block to increase the computational efficiency and to eliminate unnecessary transmission cost. The parallel design of the sub-stages is described as follows.

4.1. Samples mapping in parallel

In the CUDA parallel computing, the information of neural network and sample data transmit between the GPU and CPU, i.e., the transmission is from the host to the device. The original samples are stored in the global memory and the information of neural networks is stored in the shared memory. Several neural networks are processed in a block at the GPU side, and each neural network corresponds to a particle. When a block has $threadnum$ threads, it will allocate $threadcount$ threads to each neural network. Then the entire data is divided into $threadcount$ groups of subdata $\{S^1, S^2, \ldots, S^{threadcount}\}$ for each neural network, and S^i $(i = 1, 2, \ldots, threadcount)$ contains $|S^i|$ sample where the number of sample is defined as $|S|$. The identity of thread in a block is defined as tx, then the value of the identity of particle is calculated by Eq. (3) and the value of the identity of thread in each neural network is calculated by Eq. (4).

$$|S^i| = \begin{cases} \lceil \dfrac{|S|}{threadcount} \rceil, & i < threadcount \\ |S| mod \lceil \dfrac{|S|}{threadcount} \rceil, & i = threadcount \end{cases} \tag{2}$$

The evaluation of the sample is executed by parallel tasks $\{T^1, T^2, \ldots, T^{threadnum}\}$. They are allocated to the $threadnum$ parallel threads in one block. Each task maps S^i to partition space using neural network. Then, the mapped samples are stored in the global memory.

$$particleid = tx/threadcount \tag{3}$$

$$threadid = tx \bmod threadcount \tag{4}$$

4.2. Normalization in parallel

After the samples are mapped, the new samples are normalized by the Z-score algorithm. The data are stored in array $Dataout$ and in the global memory. There are several steps involved in the parallel process. First, each thread calculates each dimension of sample and the average in each dimension of all the samples are obtained. The standard deviation of the dimension of each sample is calculated using the average value of data in parallel. Then, each thread is responsible for calculating each sample. The normalized value is obtained using Eq. (1). The parallel normalization is shown in Algorithm 1.

Algorithm 1: Parallel Normalization in CUDA

Input: Sample points $Dataout$, the dimension of partition space m, and thread identification $threadid$
Output: Standard value.

```
Set the value of mean and standard deviation of subdata threadid to 0;
for i = 0 to |S^threadid| do
    for j = 0 to |S| do
        Calculate the average value and standard deviation of each sample in subdata threadid;
    end
end
_syncthreads();
for i = 0 to |S^threadid| do
    for k = 0 to m do
        Calculate the standard value of the sample points in subdata threadid;
    end
    _syncthreads();
end
Return standard value.
```

4.3. Similarity computation in parallel

After the samples are normalized, the new data is saved in array $Dataout$. Each sample is evenly assigned to each thread. The d is resolved using

$$d = |\mathbf{x}| \frac{1 - e^{-|\mathbf{x}|/2}}{|\mathbf{x}| + |\mathbf{x}| \cdot e^{-|\mathbf{x}|/2}} \tag{5}$$

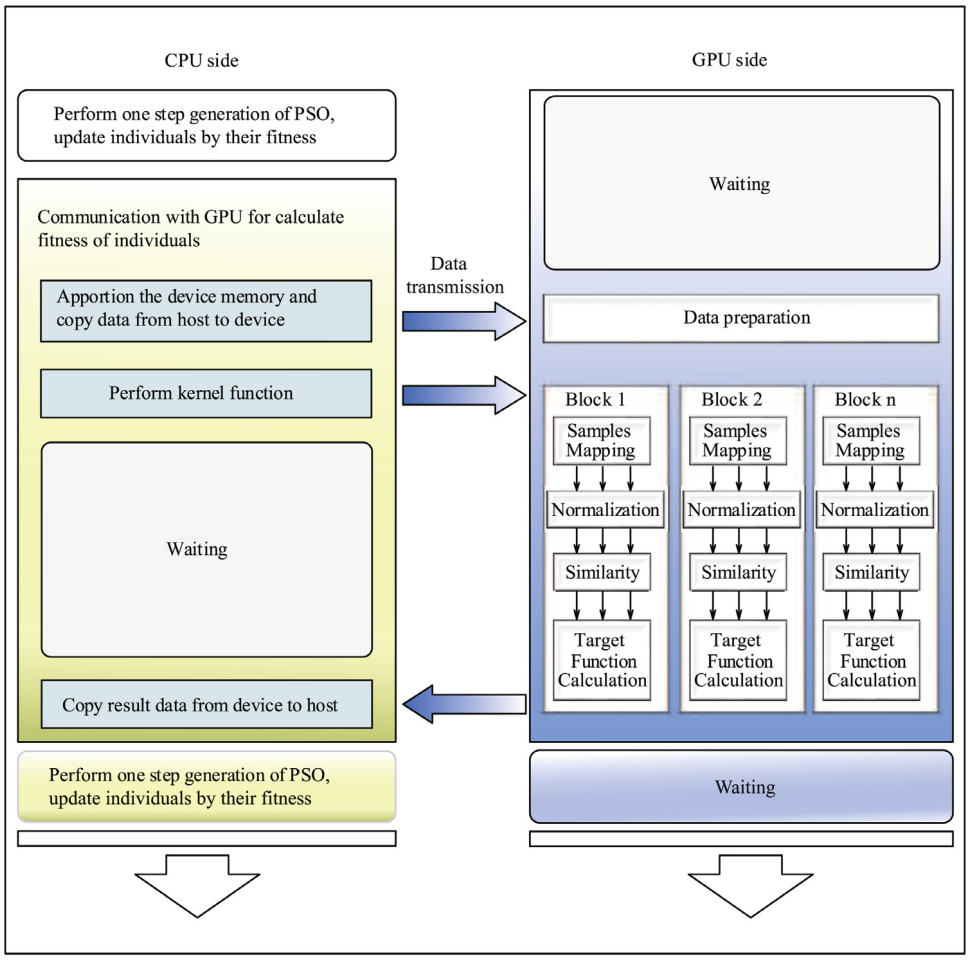

Fig. 3. The parallel relationship of PNNP.

where **x** represents a normalized sample, and then each subdata is normalized to a hypersphere in parallel. After calculating all the samples, the distance between the two samples is obtained by calculating the new sample point in array *Dataout*, and a fixed value minus the distance to obtain the similarity matrix of samples. The similarity matrix are stored in array *SimilarityArray* and in global memory. The parallel similarity computation is shown in Algorithm 2.

4.4. Target function calculation in parallel

After calculating the similarity of all sample pairs, the weight of each sample is calculated at the host side, and the data is transmitted to the device side and stored in array *weight*. There are several steps involved in the parallel process. First of all, each thread calculates each subdata in array *SimilarityArray*. Then the samples have to be listed in descending order according to the value of similarity and the corresponding index numbers of samples are stored in the array *index*. Secondly, each sample is divided into self-class or nonself-class according the similarity of two samples belonging to a definite class. Then each thread calculates the value of $S_{nonself}$ and S_{self} for each sample in parallel. The value of F is calculated on the basis of $S_{nonself}$, S_{self} and the weight of the current sample $\omega(\mathbf{x}_i)$

$$F = \omega(\mathbf{x}_i)(S_{nonself}(\mathbf{x}_i) - \alpha S_{self}(\mathbf{x}_i)) \tag{6}$$

where α is a adjustment coefficient. $S_{nonself}(\mathbf{x}_i)$ represents the sum of similarities between $\mathbf{x}_i$ and samples in other classes. $S_{self}(\mathbf{x}_i)$ is the sum of similarities between $\mathbf{x}_i$ and samples in same class. Finally, the fitness

Algorithm 2: Parallel Similarity Computation in CUDA

Input: Sample points *Dataout*, the dimension of partition space m, and thread identification *threadid*
Output: Similar matrix.

```
for i = 0 to |S^threadid| do
    for j = 0 to m do
        Calculate d use formula (5) in subdata threadid;
        Obtain the values by the normalized points into a hypersphere;
    end
end
_syncthreads();
for i = 0 to |S^threadid| do
    for j = 0 to |S| do
        Calculate the distance D between two points with Euclidean Distance in subdata threadid;
        if threadid: = j then
        The value of SimilarityArray_{threadid,j} is 2;
        else
        The value of SimilarityArray_{threadid,j} is 2-D;
    end
    _syncthreads();
end
Return the Similar matrix.
```

value of the current particle is obtained by adding up all the subdata values. The parallel target function calculation is shown in Algorithm 3.

Algorithm 3: Parallel Target Function Calculation in CUDA

Input: Similar matrix $SimilarityArray$, array $weight$ of each sample, space neighbor number L and thread identification $threadid$

Output: Fitness of each particle.

```
Initialize array index for subdata threadid;
for i = 0 to |S^threadid| do
    for j = 0 to L do
        if SimilarityArray_j < SimilarityArray_{j-1} do
        save the index numbers in array index according to the
        value of similarity
    end
end
_syncthreads();
for i = 0 to |S^threadid| do
    Initialize the sum of the weight of each sample point Wsum;
    for j = 0 to Wsum < L do
        Calculate S_self for point j in subdata threadid;
        Calculate S_nonself for point j in subdata threadid;
        Calculate fitness count for point j using S_self and S_nonself;
    end
    for k = 0 to |S| do
        Add the fitness of the current particle in subdata threadid;
    end
end
Returns the fitness of each particle.
```

5. Experiments

The computing platform used in experiments is a GPU-based desktop supercomputer with C programming environment and Linux operation system. The platform includes a high-performance Tesla K10 GPU produced by NVIDIA. High-speed shared memory and synchronization mechanism are adopted in the SM. There are eight SMs in the GPU, each of which has 192 SPs. The SP's clock frequency is 0.75 GHz. In each block, there are 65,536 available registers and the total amount of shared memory is 48 KB. Each multiprocessor has thousands of CUDA cores. On this platform, the proposed parallel nearest neighbor partitioning method is verified on commonly used data sets and is also applied to performance evaluation of cement microstructure.

5.1. Experiments on data sets

In this subsection, some famous classification data sets are applied to the experiments. All data sets are from the UCI machine learning repository and the web site is http://archive.ics.uci.edu/ml/. Table 1 displays the information of adopted data sets, containing the number of samples, attributes and classes.

In order to normalize data, the min–max normalization method is adopted. This is a linear transformation of the original data, so as to reduce the influence of different scale of attributes. Transformation function is as follows

$$x' = \frac{x - Min}{Max - Min} \tag{7}$$

where Max is maximum value of a attribution in a data set, Min is the minimum value of a attribution in a data set, x is the original value of this attribution.

We take a common test method 10-fold cross-validation. The data set is divided into ten subsets, where one subset is used as testing and the remaining subsets are used as training in turn. Then, the average performance is reported.

Table 1
Characteristics of data sets.

Data sets	No. of samples	No. of attributes	No. of classes
IONOSPHERE	351	34	2
UKM	403	5	4
VEHICLE	846	18	4
CMC	1 473	9	3
SEGMENT	2 310	19	7
PARKINSONS	195	22	2
VC	310	6	3
SEEDS	210	7	3
WINE	178	13	3
HABERMAN	306	3	2
SATELLITE	6 435	36	6
MAGIC	19 020	10	2

Table 2
Speedup ratio and time consumption in the process of training.

	NNP (s)	PNNP (s)	Speedup ratio
IONOSPHERE	1 283	23	55.78
UKM	1 158	15	77.2
VEHICLE	6 230	66	94.39
CMC	19 583	219	89.42
SEGMENT	71 198	783	90.93
SATELLITE	578 905	5 178	111.8
MAGIC	8 177 290	49 773	164.29

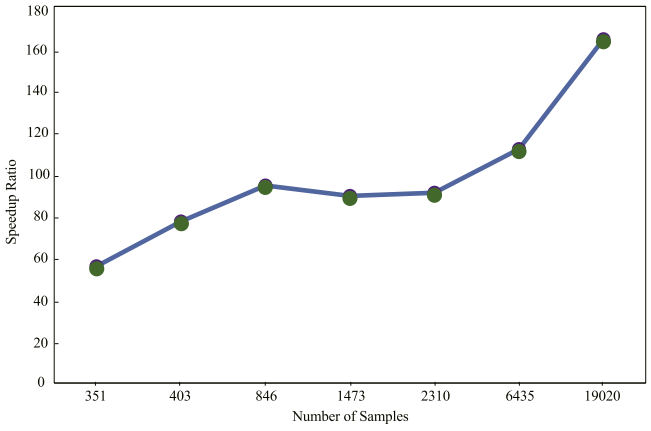

Fig. 4. Number of samples versus PNNP speedup ratio.

A three layers feedforward neural network is chosen for the comparison of all methods, the parameters of the hidden layer and the dimension of the partition space are adjusted according to the requirements of the different data sets. According to Ref. Wang et al. (2017), the number of nearest neighbors L is chosen from 1 to 50, the discrimination weight α is chosen from 10^{-2} to 10^{1} and the dimension of partition space m is chosen from 1 to 30. In the test, the number of nearest neighbors k is chosen from $\{1, 3, \ldots, 25\}$. In PSO, the population size is set to 40, Max Generation is set to 1000, $\varphi 1$ and $\varphi 2$ are set to 1.8, and $Vmax$ is set to 0.4.

Compared with the NNP, the time consumption of the PNNP is reduced by parallel computation. By setting the same parameters for same data set, the results in Table 2 indicate that the proposed method is able to accelerate NNP's speed, and the speedup ratio is more remarkable for the large data sets. Fig. 4 reveals the relationship between speedup ratio and number of samples. Although the speedup ratio is slightly lower on CMC and SEGMENT than on VEHICLE, speedup ratio is on the rise with the increase of sample size. Therefore, this phenomenon reflects that the proposed PNNP is more favorable to large data sets because, with the use of adopted strategy, the utilization rate of resources increases with the increase of samples.

Table 3 shows the comparison of accuracies between the PNNP and other methods. It can be observed from the table that PNNP has

Table 3
Accuracy on all data sets.

	Traditional (Wang et al., 2017)	SoftMax (Wang et al., 2017)	ECOC (Wang et al., 2017)	FCM (Wang et al., 2017)	PNNP
IONOSPHERE	88.06(±5.41)	N/A	N/A	**94.17**(±4.80)	**94.17**(±1.58)
PARKINSONS	87.00(±12.74)	N/A	N/A	87.00(±7.53)	**89.50**(±5.99)
HABERMAN	72.81(±4.43)	N/A	N/A	**73.75**(±3.67)	71.56(±5.4)
UKM	92.38(±5.12)	91.67(±6.66)	93.33(±2.46)	95.95(±1.96)	**97.62**(±1.94)
VEHICLE	81.74(±1.98)	83.26(±2.14)	79.88(±3.05)	79.19(±3.78)	**83.49**(±1.88)
CMC	53.56(±3.69)	52.55(±2.45)	51.21(±2.69)	**55.23**(±2.90)	53.76(±1.91)
SEGMENT	96.67(±1.78)	96.54(±1.43)	96.17(±1.38)	95.70(±1.71)	**96.92**(±0.91)
VC	77.74(±3.86)	84.19(±5.15)	79.35(±7.78)	85.16(±6.31)	**85.81**(±6.12)
SEEDS	93.33(±6.02)	92.86(±7.19)	94.29(±7.38)	95.24(±5.02)	**96.19**(±3.01)
WINE	97.78(±3.88)	98.33(±3.75)	97.78(±2.87)	98.89(±2.34)	**99.44**(±1.76)

Table 4
F-measure on all data sets.

	Traditional (Wang et al., 2017)	SoftMax (Wang et al., 2017)	ECOC (Wang et al., 2017)	FCM (Wang et al., 2017)	PNNP
IONOSPHERE	86.18(±6.42)	N/A	N/A	93.44(±5.43)	**93.59**(±1.63)
PARKINSONS	83.12(±16.57)	N/A	N/A	76.75(±17.39)	**85.52**(±9.07)
HABERMAN	59.42(±8.02)	N/A	N/A	55.93(±9.61)	**66.79**(±5.73)
UKM	92.73(±5.03)	85.22(±14.79)	93.26(±2.98)	96.11(±2.03)	**97.64**(±2.01)
VEHICLE	81.88(±2.14)	**83.28**(±2.19)	80.10(±3.12)	78.54(±4.17)	83.19(±2.24)
CMC	50.84(±4.88)	50.49(±2.89)	49.28(±3.16)	52.83(±2.55)	**53.08**(±2.5)
SEGMENT	96.66(±1.79)	96.55(±1.40)	96.16(±1.37)	95.72(±1.69)	**96.92**(±0.90)
VC	72.50(±5.72)	79.98(±6.31)	73.59(±9.61)	81.21(±7.77)	**83.02**(±6.68)
SEEDS	93.32(±6.01)	92.85(±7.15)	94.26(±7.43)	95.18(±5.08)	**96.16**(±3.06)
WINE	97.85(±3.69)	98.33(±3.77)	97.78(±2.88)	98.88(±2.36)	**99.49**(±1.62)

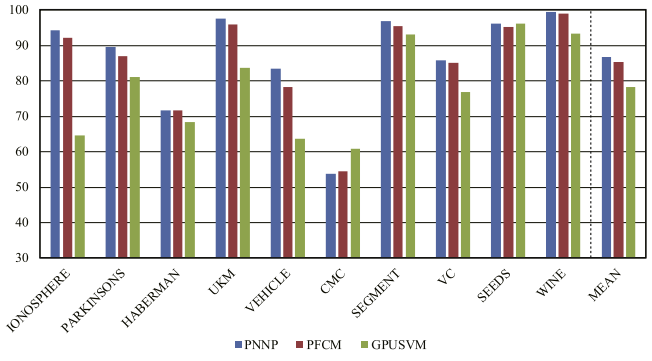

Fig. 5. Comparison of accuracy results for parallel classifiers.

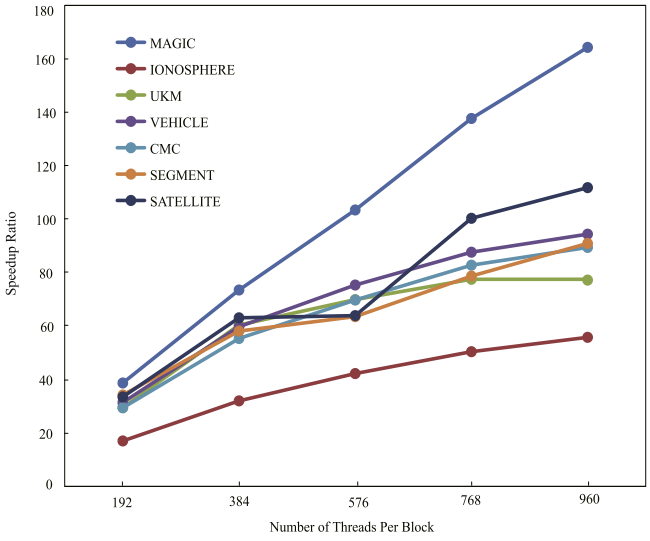

Fig. 6. Number of threads versus speedup ratio.

higher accuracy than other methods in the experiment. Although the PNNP's accuracy is lower than accuracy of the FCM on HABERMAN and CMC, it is the best one on most of the data sets, such as PARKINSONS, UKM, VEHICLE, SEGMENT, VC, SEEDS and WINE. The accuracy of PNNP and FCM are tied for first on IONOSPHERE. In addition, the smallest standard deviation is generated by the PNNP method except HABERMAN and VC, which means the PNNP is more stable. Therefore, PNNP has higher performance than Traditional, SoftMax and ECOC in classification. This means that the introduction of PNNP not only speeds up the learning process for the training set, but also improves the average accuracy. Table 4 shows F-measure of the PNNP, and further confirms its effectiveness, which lays the foundation for us to find the optimal neural network classifier.

Fig. 5 describes the accuracy results of the ten data sets for the three CUDA-based classification methods. It can be seen that PNNP has higher accuracy than other methods except HABERMAN, CMC and SEEDS. In addition, the accuracy of PNNP and GPUSVM are tied for first on SEEDS and the accuracy of PNNP and PFCM are tied for first on HABERMAN. Furthermore, the average accuracy of each method indicates that the performance of PNNP is superior to other methods.

Fig. 6 shows the relationship between the speedup ratio and the number of threads in each block. From this figure we can see clearly that speedup ratio gradually increases with the increase of the number of threads, especially on MAGIC. Due to GPU takes advantage of the processor's free time when thread's number increases, thus for large data sets, more threads are used to participate in the training process. Each SM has 192 CUDA cores, the number of threads in each block should be in multiples of 192 if the GPU performs its maximum efficiency. The samples are stored in the global memory because of conflict between the large data. CUDA uses thread switching to make up for disadvantage of increasing time consumption due to accessing data from global memory, thereby making GPU reach maximum efficiency.

5.2. Application in performance evaluation of cement microstructure

The performance, particularly strength, of hardened cement is largely influenced by the evolution of microstructure during cement hydration. The commonly used destructive methods, e.g. strength test, makes it impossible to evaluate the performance of cement paste *in situ*. Recently, the nondestructive approaches have been used for measuring cement paste, such as the surface wave method (Sun and Wang, 2005),

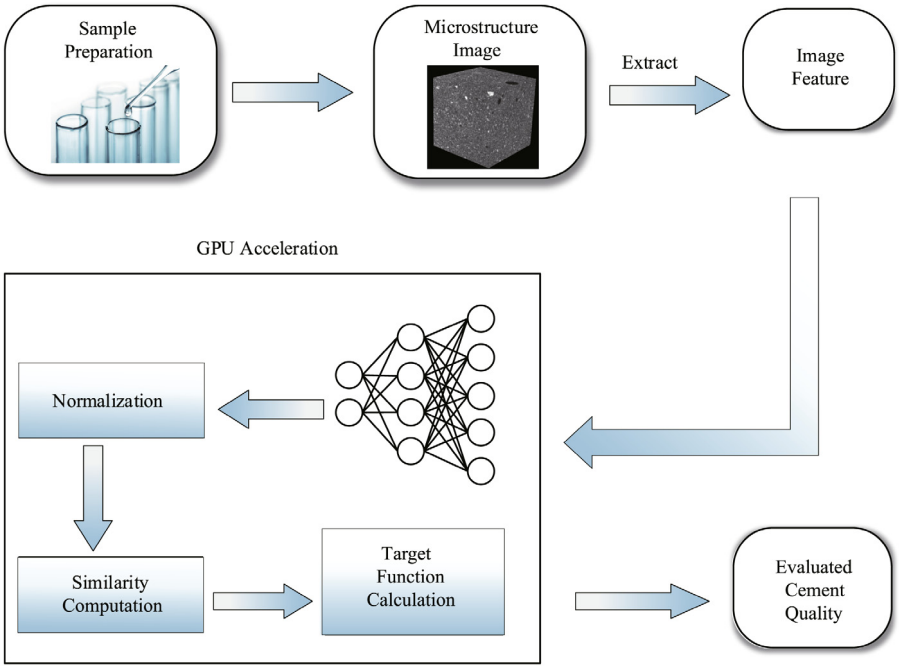

Fig. 7. Environment architecture of application.

Table 5
Results of speedup ratio and time consumption.

Validation	NNP (s)	PNNP (s)	Speedup ratio
1	113 178	1638	69.10
2	126 972	1827	69.50
3	126 080	1827	69.01
4	125 143	1826	68.53
5	127 465	1827	69.77
6	128 067	1826	70.01
7	127 046	1827	69.54
8	127 529	1826	69.84
9	126 729	1827	69.36
10	126 504	1826	69.28
Mean	125 471	1808	69.40

the ultrasonic method (Pham, 2014), and microwave method (Hasar, 2009) etc. Among those approaches, on account of the advantages of nondestructive and visualizability, the microtomography is widely used to observe cement paste (Bentz et al., 2002; Wang et al., 2014a).

The data based approaches have been introduced to model the cement hydration, including estimating strength using image analysis (Li et al., 2016b; Wang et al., 2015) and modeling development of degree of hydration (Wang et al., 2016a, 2010) using differential equations (Li and Rogovchenko, 2015, 2016; Han et al., 2017). In general, considering the great complexity of cement hydration, a large amount of data are required in training to increase the generalizability and robustness of established evaluation model. In this subsection, the nearest neighbor partitioning neural network classifier is applied to evaluate the performance of cement microstructure and PNNP is adopted to accelerate the training process. Nearest neighbor partitioning is used to classify the quality of cement based on images, i.e., whether cement paste meets the quality standard based on microstructural image features (Li et al., 2016b).

We collected 3000 microstructural images of cement paste and extracted 46 image features from each image, such as energy, entropy, correlation, inverse different moment etc. The corresponding strength of cement pastes are also measured physically, in which the quality standard is set to 15 MPa. The adopted approach is shown in Fig. 7. A three-layers feedforward neural network which have 24 neurons in its hidden layer is applied to this experiment. The number of nearest neighbors L is set to 40, the discrimination weight α is set to 1, and the dimension of partition space m is set to 8. In the test, the number of nearest neighbors k is set to 15. In PSO, the population size is set to 40, Max Generation is set to 1000, $\varphi 1$ and $\varphi 2$ are set to 1.8, and $V max$ is set to 0.4.

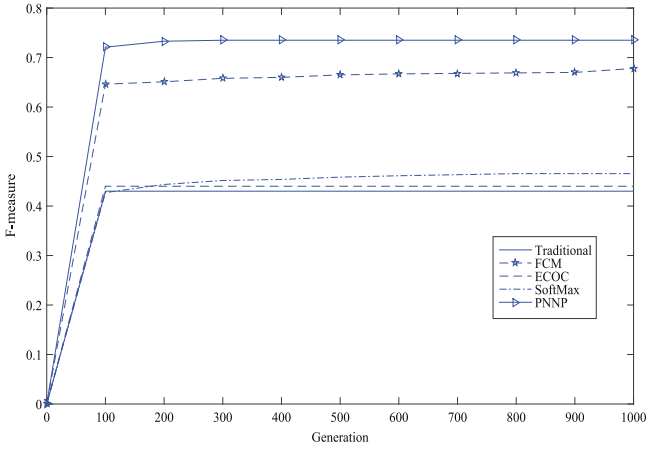

Fig. 8. The training process of the cement quality evaluation.

Table 5 reports the time consumption and speedup ratio for each subset in the 10-fold cross-validation. It can be observed from the table that PNNP is able to accelerate NNP in this application and the variance of speedup ratio is very stable. The adoption of CUDA platform remarkably reduces the time consumption of nearest neighbor partitioning in this application. Fig. 8 illustrates the development of F-Measure during training process of traditional method, SoftMax, ECOC, FCM and PNNP. For all of compared methods, the evolution process converges after 100 generations but PNNP is the fastest method among them.

Fig. 9 shows the comparison of accuracy of compared methods. The average training accuracy of PNNP is significantly higher than

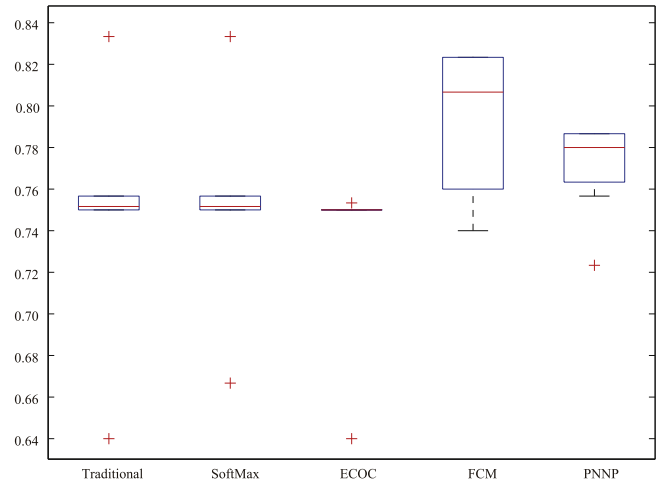

Fig. 9. Comparison of distribution of accuracy.

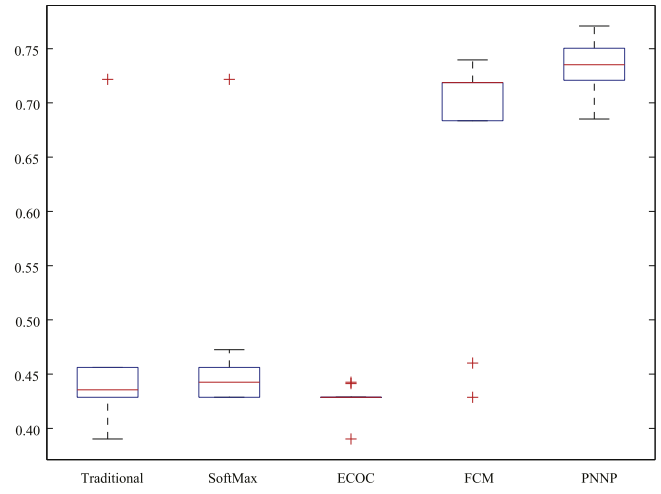

Fig. 10. Comparison of distribution of F-measure.

the other methods except FCM. Furthermore, compared with FCM, the small variance indicates that PNNP is stabler. Fig. 10 further verifies the effectiveness of PNNP from the perspective of F-measure. The performance of PNNP is the best one among all compared method. Therefore, the adoption of PNNP improves the performance of quality evaluation for cement pastes and also speeds up the training process of NNP.

6. Conclusion

In this study, in order to speed up the training process of NNP, particularly for large data sets, we proposed a parallel NNP method based on CUDA architecture. At the CPU side, the main optimization algorithm performs in serial. At the GPU side, each potential NNP neural network classifier is evaluated on blocks in parallel and its subtasks are also performed in parallel on threads.

When evaluating PNNP's performance, some famous data sets are chosen to compare the speedup ratio, accuracy and F-measure. The experimental results indicate that the parallel method of the NNP not only increases the speed but also improves those measurements.

Furthermore, the proposed PNNP is applied to the evaluation of cement quality based on microstructural image features. PNNP yields promising performance on this task and shows that it is able to solve real world problems.

In the future, PNNP can be used in the field of medical image analysis and bioinformatics on account of the big data size. Its applications on these fields will be studied to further evaluate PNNP's performance. Furthermore, in terms of its application in evaluation of cement quality, apart from microtomography images, data collected from other types of measurement approaches for cement paste will be further studied.

Acknowledgments

This work was supported by National Natural Science Foundation of China under Grant No. 61573166, No. 61572230, No. 81671785, No. 61373054, No. 61472164, No. 61472163, No. 61672262, No. 61640218. Shandong Provincial Natural Science Foundation, China, under Grant ZR2015JL025, ZR2014JL042. Science and technology project of Shandong Province under Grant No. 2015GGX101025. Project of Shandong Province Higher Educational Science and Technology Program under Grant no. J16LN07. Shandong Provincial Key R&D Program under Grant No. 2016ZDJS01A12, No. 2016GGX101001.

References

An, S.Y., Kang, J.G., Choi, W.S., Oh, S.Y., 2011. A neural network based retrainable framework for robust object recognition with application to mobile robotics. Appl. Intell. 35 (2), 190–210.

Anderson, D., Coupland, S., 2008. Parallelisation of fuzzy inference on a graphics processor unit using the compute unified device architecture. Recent. Progr. Med. 85 (3), 160–165.

Attia, M.W., Abou-Chadi, F.E.Z., Moustafa, E.D., Mekky, N., Classification of ultrasound kidney images using PCA and neural networks, 6 (2015) 53–57.

Avci, E., 2012. An expert target recognition system using a genetic wavelet neural network. Appl. Intell. 37 (4), 475–487.

Baraldi, P., Cannarile, F., Maio, F.D., Zio, E., 2016. Hierarchical k-nearest neighbours classification and binary differential evolution for fault diagnostics of automotive bearings operating under variable conditions. Eng. Appl. Artif. Intell. 56, 1–13.

Bentz, D.P., Mizell, S., Satterfield, S., Devaney, J., George, W., Ketcham, P., Graham, J., Porterfield, J., Quenard, D., Vallee, F., et al., 2002. The visible cement data set. J. Res. Natl. Inst. of Stand. Technol. 107 (2), 137.

Bridle, J.S., 1990. Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. Springer Berlin Heidelberg, pp. 227–236.

Casta, A.O., Ndez-Navarro, F., Mart, S., Nez, C., Gutierrez, P.A., 2011. Neuro-logistic models based on evolutionary generalized radial basis function for the microarray gene expression classification problem. Neural Process. Lett. 34 (2), 117–131.

Chua, K.S., 2003. Efficient computations for large least square support vector machine classifiers. Pattern Recognit. Lett. 24 (1–3), 75–80.

Dietterich, T.G., Bakiri, G., 1995. Solving multiclass learning problems via error-correcting output codes. J. Artificial Intelligence Res. 2 (1), 263–286.

Freund, B.Y., 1995. Boosting a weak learning algorithm by majority, information and computation. Inf. Comput. 121 (2), 256–285.

Gao, D., Ji, Y., 2005. Classification methodologies of multilayer perceptrons with sigmoid activation functions. Pattern Recognit. 38 (10), 1469–1482.

Gong, T., Fan, T., Guo, J., Cai, Z., 2016. GPU-based parallel optimization of immune convolutional neural network and embedded system. Eng. Appl. Artif. Intell..

Haddar, B., Khemakhem, M., Hanafi, S., Wilbaut, C., 2016. A hybrid quantum particle swarm optimization for the multidimensional knapsack problem. Eng. Appl. Artif. Intell. 55 (C), 1–13.

Han, S.Y., Chen, Y.H., Tang, G.Y., 2017. Fault diagnosis and fault-tolerant tracking control for discrete-time systems with faults and delays in actuator and measurement. J. Franklin Inst. B 354 (12), 4719–4738.

Hartigan, J., Wong, M., 1979. A k-means clustering algorithm. Appl. Stat. 28 (1), 100–108.

Hasar, U.C., 2009. Non-destructive testing of hardened cement specimens at microwave frequencies using a simple free-space method. NDT & E Int. 42 (6), 550–557.

Hassan, Y.F., 2011. Rough sets for adapting wavelet neural networks as a new classifier system. Appl. Intell. 35 (2), 260–268.

Jang, H., Park, A., Jung, K., 2008. Neural network implementation using cuda and openmp. In: Digital Image Computing: Techniques and Applications. pp. 155–161.

Juang, C.F., Chen, T.C., Cheng, W.Y., 2011. Speedup of implementing fuzzy neural networks with high-dimensional inputs through parallel processing on graphic processing units. IEEE Trans. Fuzzy Syst. 19 (4), 717–728.

Kang, S., Park, S., 2009. A fusion neural network classifier for image classification. Pattern Recognit. Lett. 30 (9), 789–793.

Kaur, R., 2013. Using some data mining techniques to predict the survival year of lung cancer patient. Int. J. Comput. Sci. Mob. Comput. 2 (4).

Kora, P., Kalva, S.R., 2015. Improved bat algorithm for the detection of myocardial infarction. SpringerPlus 4 (1), 1–18.

Li, K., Liu, L., Zhai, J., Khoshgoftaar, T.M., Li, T., 2016a. The improved grey model based on particle swarm optimization algorithm for time series prediction. Eng. Appl. Artif. Intell. 55, 285–291.

Li, M., Yang, B., Wang, L., Liu, Y., Zhao, X., Zhou, J., Zhang, L., 2016b. The prediction of cement compressive strength based on gray level images and neural network. In: International Conference on Informative and Cybernetics for Computational Social Systems, pp. 103–108.

Li, Q., Salman, R., Test, E., Strack, R., Kecman, V., 2011. GPUSVM: a comprehensive cuda based support vector machine package. Open Comput. Sci. 1 (4), 387–405.

Li, T., Rogovchenko, Y.V., 2015. Oscillation of second-order neutral differential equations. Math. Nachr. 288, 1150–1162.

Li, T., Rogovchenko, Y.V., 2016. Oscillation criteria for even-order neutral differential equations. Appl. Math. Lett. 61, 35–41.

López, J., Suykens, J.A.K., 2011. First and second order SMO algorithms for LS-SVM classifiers. Neural Process. Lett. 33 (1), 31–44.

Lu, H., Setiono, R., Liu, H., 1996. Effective data mining using neural networks. IEEE Trans. Knowl. Data Eng. 8 (6), 957–961.

Misra, B.B., Dehuri, S., Dash, P.K., Panda, G., 2008. A reduced and comprehensible polynomial neural network for classification. Pattern Recognit. Lett. 29 (12), 1705–1712.

O'Neil, B.P.V., 2010. Advanced Engineering Mathematics, fourth ed.

Oujaoura, M., 2012. Multilayer neural networks and nearest neighbor classifier performances for image annotation. Int. J. Adv. Comput. Sci. Appl. 3 (11), 165171.

Peng, L., Zhang, H., Yang, B., Chen, Y., 2014. A new approach for imbalanced data classification based on data gravitation. Inf. Sci. Int. J. 288 (20), 347–373.

Peng, L., Zhang, H., Zhang, H., Yang, B., 2017. A fast feature weighting algorithm of data gravitation classification. Inf. Sci. Int. J. 375 (C), 54–78.

Pham, S.T., 2014. Non-destructive characterization of cement materials using ultrasonic method - application to the study of carbonation. Adv. Mater. Res. 1065–1069, 1791–1794.

Quinlan, J.R., 1986. Introduction of decision trees, (1) 81–106.

Sun, J.Y., Wang, J.H., 2005. Research on surface-wave non-destructive detecting method of soil-cement wall. Underground Space.

Sun, L.C., Zhang, S.B., Cheng, X.T., Zhang, M., 2013. Fast face detection algorithm based on cuda. Comput. Modern. 1 (8), 11–14.

Vapnik, V.N., 1997. The nature of statistical learning theory. IEEE Trans. Neural Netw. 8 (6), 1564.

Wang, L., Orchard, J., 2017. Investigating the evolution of a neuroplasticity network for learning. IEEE Trans. Syst. Man Cybern.: Syst. (in press). http://dx.doi.org/10.1109/TSMC.2017.2755066.

Wang, L., Yang, B., Abraham, A., 2016a. Distilling middle-age cement hydration kinetics from observed data using phased hybrid evolution. Soft Comput. 20 (9), 3637–3656.

Wang, L., Yang, B., Abraham, A., Qi, L., Zhao, X., Chen, Z., 2014a. Construction of dynamic three-dimensional microstructure for the hydration of cement using 3d image registration. Pattern Anal. Appl. 17 (3), 655–665.

Wang, L., Yang, B., Chen, Y., 2014b. Improving particle swarm optimization using multi-layer searching strategy. Inform. Sci. 274 (8), 70–94.

Wang, L., Yang, B., Chen, Y., Abraham, A., Sun, H., Chen, Z., Wang, H., 2012. Improvement of neural network classifier using floating centroids. Knowl. Inf. Syst. 31 (3), 433–454.

Wang, L., Yang, B., Chen, Y., Chen, Z., Sun, H., 2014c. Accelerating FCM neural network classifier using graphics processing units with CUDA. Appl. Intell. 40 (1), 143–153.

Wang, L., Yang, B., Chen, Y., Zhang, X., Orchard, J., 2017. Improving neural-network classifiers using nearest neighbor partitioning. IEEE Trans. Neural Netw. Learn. Syst. 28 (10), 2255–2267.

Wang, L., Yang, B., Orchard, J., 2016b. Particle swarm optimization using dynamic tournament topology. Appl. Soft Comput. 48, 584–596.

Wang, L., Yang, B., Wang, S., Liang, Z., 2015. Building image feature kinetics for cement hydration using gene expression programming with similarity weight tournament selection. IEEE Trans. Evol. Comput. 19 (5), 679–693.

Wang, L., Yang, B., Zhao, X., Chen, Y., Chang, J., 2010. Reverse extraction of early-age hydration kinetic equation from observed data of portland cement. Sci. China Technol. Sci. 40 (5), 582–595.

Wu, J., Hong, B., 2011. An efficient k-means algorithm on CUDA. In: IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, pp. 1740–1749.

Yaakob, S.N., Jain, L., 2012. An insect classification analysis based on shape features using quality threshold ARTMAP and moment invariant. Appl. Intell. 37 (1), 12–30.

Yu, Z., Liu, Y., Yu, X., Pu, K.Q., 2015. Scalable distributed processing of k nearest neighbor queries over moving objects. IEEE Trans. Knowl. Data Eng. 27 (5), 1383–1396.

Zhou, J., Chen, L., Chen, C.L.P., Zhang, Y., Li, H.X., 2016. Fuzzy clustering with the entropy of attribute weights. Neurocomputing 198 (C), 125–134.