

Input-Dependent Neural Network trained by Real-Coded Genetic Algorithm and its Industrial Applications

¹*S.H. Ling, ²F.H.F. Leung, and ³H.K. Lam*

¹*School of Electrical, Electronic and Computer Engineering, The University of Western Australia, WA, Australia*

²*Centre for Signal Processing, Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*

³*Division of Engineering, King's College London, Strand, London, United Kingdom*

Abstract: This paper presents an input-dependent neural network (IDNN) with variable parameters. The parameters of the neurons in the hidden nodes adapt to changes of the input environment, so that different test input sets separately distributed in a large domain can be tackled after training. Effectively, there are different individual neural networks for different sets of inputs. The proposed network exhibits a better learning and generalization ability than the traditional one. An improved real-coded genetic algorithm (RCGA) [1] is proposed to train the network parameters. Industrial applications on short-term load forecasting and hand-written graffiti recognition will be presented to verify and illustrate the improvement.

Keywords: *Neural network, real-coded genetic algorithm, short-term load forecasting, hand-written recognition.*

I INTRODUCTION

Inspired by the human nervous system, neural networks began to develop in 1940's. The McCulloch-Pitts neuron, which was the foundation of the neural network development, was presented in around 1943 [2]. No training algorithm was developed for this neuron. After that, a trainable adaptive linear (Adaline) element [2-3], which is the basic building block used in many neural networks, was proposed. Adalines are able to model some linearly separable logic functions. In order to model a nonlinear separation boundary, layered networks such as the multiple Adaline (Madaline) and multi-layer feed-forward networks were proposed. The architecture of the Madaline network consists of a linear combination of some Adaline elements, which form a single feed-forward network structure. Based on these ideas and further exploration, different kinds of neural network were then developed,

including mapping networks, self-organizing networks, recurrent networks, radial basis function networks, etc. to meet the need of different applications.

One of the important issues for neural networks is the learning or training process. The learning process aids to find a set of optimal network parameters. At the early stage, two major classes of learning rules, i.e. the error correction and gradient rules, were proposed. The error correction rules [2-3], such as the α -LMS algorithm, perception learning rules and May's rule, adjust the network parameters to correct the network output error corresponding to some input patterns. Some of the error correction rules are only applicable to linear separable problems. The gradient rules [2-4], such as MRI, MRII, MRIII rules and the backpropagation techniques, adjust the network parameters based on the gradient information of the error function to reduce the mean square errors over all input patterns. Different variations of backpropagation algorithms, such as the backpropagation algorithms with momentum [5], backpropagation algorithms with variable learning rate [5] and conjugate gradient algorithm [6], were proposed to reduce the learning time. However, the derivative information of the optimized function is needed (meaning that the error function has to be continuous and differentiable), and the learning process may be trapped in a local optima, especially when the problems are multimodal. The learning rules are also network-structure dependent. Some global search algorithms such as Tabu search [7], simulated annealing [7] and genetic algorithm [7-9, 14-15] were proposed. Unlike the gradient rules, these search algorithms are less likely to be trapped in local optima, and do not require a differentiable or even continuous error function. Thus, these search algorithms are more suitable for searching in a large, complex, non-differentiable and multimodal domain [10].

Real-coded genetic algorithm (RCGA) is a good training algorithm for neural or neural-fuzzy networks [11-13, 44-48]. The same RCGA can be used to train many different network regardless of whether they are feed-forward, recurrent, or of other structure types. This generally saves a lot of human efforts in developing training algorithms for different types of networks.

It is well known that a neural network can approximate any smooth and continuous nonlinear functions in a compact domain to any arbitrary accuracy [2-3]. Three-layer feed-forward neural networks have been successfully applied in wide range of applications such as system modelling and control [19], function estimation [16], forecasting [11, 17], recognition [12], [18] etc. Thanks to its specific structure, a neural network can be used to realize a learning process [2-5] that consists of two steps: designing a network structure and choosing

an algorithm for the learning process. The structure of the neural network governs the non-linearity of the modelled function. The learning algorithm is used to provide a rule to optimize the weights' values within the training period. A typical neural network structure offers a fixed set of weights after the learning process. This single set of weights is used to model all input data sets. However, a fixed set of weights may not be enough to learn the features of the whole data sets if the data sets are distributed in a vast domain separately and/or the number of network parameters is too small.

If the network parameters' values vary with respect to the input data, we might separate the information contained in a vast domain into that in a number of sub-domains. The network parameters' values change accordingly to cope with the information in different sub-domains. In this paper, an input-dependent neural network (IDNN) tuned by an improved RCGA [1] is proposed. In this IDNN, the parameters of the activation functions vary according to the input signals of the proposed network. In other words, the parameters are variables depending on the input data after the training process. For applications involving a large domain of input-output mappings, an IDNN should be able to perform the mapping task more efficiently in terms of accuracy and network complexity (number of parameters). The enhancement is a result of adaptive property the proposed network with respect to the contingent changes of the environment. Effectively, the IDNN operates as if different individual neural networks are handling different sets of input data. An RCGA [1] will be employed to train the parameters of the proposed network. Two industrial applications of short-term daily load forecasting and hand-written graffiti recognition will be presented. By applying the proposed IDNN, it is found that better results with increased learning ability than conventional feed-forward neural networks are given. In this paper, a 3 layer feed-forward fully-connected neural network (FFCNN) [3] and a wavelet neural network (WNN) [41-43] will be used as references for comparison.

This paper is organized as follows. The proposed IDNN will be presented in Section II. Training of the proposed network using the RCGA will be presented in Section III. Application examples will be presented in Section IV to illustrate the applicability of the proposed approach. A conclusion will be drawn in Section V.

II INPUT-DEPENDENT NEURAL NETWORK MODEL

The proposed input-dependent neural network is shown in Fig. 3. In this network, the parameters of the activation functions vary according to some intermediate signals of the

proposed network. Consequently, adaptation ability is endowed to the proposed network to cope with the contingent changes of the environment. The learning and generalization abilities of the network are thus enhanced.

The proposed neural network can be regarded as consisting of two units, namely the network memory (NM) and the data-processing (DP) neural network as shown in Fig. 1. The NM stores some parameters (rules) governing how the DP neural network handles the input data. By using this network architecture, some problems that cannot be handled by the traditional feed-forward neural networks with a limited number of parameters can now be tackled. To illustrate this point, Fig. 2 shows two sets of data S1 and S2 separated in a far distance. In general, there can be even more data sets separated in far distances within the large domain. If we model these data sets using a traditional feed-forward neural network, the weights of the neural network will be trained in a mapping problem to minimise the error between the network output and the desired value. However, with a limited number of parameters, the network may only model the data set S instead as shown in Fig. 1. When the network architecture as shown in Fig. 1 is used instead, we can see that if the input data belongs to S1, the NM will adopt parameter set 1 to drive the DP neural network. Similarly, if the input data belongs to S2, the parameters corresponding to S2 will be employed by the NM to drive the DP neural network. In other words, it operates like two individual neural networks handling the corresponding input data. An improved GA [1], which exhibits good performance for multimodal problems, can be employed to train the parameters of the proposed neural network.

Referring to Fig. 3, $\mathbf{z}(t) = [z_1(t) \ z_2(t) \ \cdots \ z_{n_{in}}(t)]$ denotes the input vector, n_{in} denotes the number of input nodes; t denotes the current of input vector number, which is a non-zero integer; $w_{ji}^{(1)}, j = 1, 2, \dots, n_h; i = 1, 2, \dots, n_{in}$, denote the connection weights between the input layer and the hidden layer; n_h denotes the number of hidden nodes; $w_{kj}^{(2)}, k = 1, 2, \dots, n_{out}; j = 1, 2, \dots, n_h$, denote the connection weights between the hidden layer and the output layer; n_{out} denotes the number of output nodes. m_j, r_j are parameters related to a proposed activation function of the hidden nodes; $tf^1(\cdot)$ denotes the activation function in the hidden node and $tf^2(\cdot)$ denotes the activation function in the output nodes. The details of the proposed network will be presented as follows.

A. Proposed Neuron

Fig. 4 shows the details of the proposed neuron in the hidden layer. In this figure, $x_1(t)$ to $x_n(t)$ denote the input of the node; w_1 to w_n denote the connection weights; m and r denote the intermediate connection weights. The output of the summation block, $f_s(t)$, is given by,

$$f_s(t) = \sum_{i=1}^n w_i x_i(t) \quad (1)$$

The output of the node is defined as,

$$f(t) = tf^1(f_s(t), m, r) \quad (2)$$

where the activation function is to evaluate the fitness of the input and is defined as,

$$tf^1(f_s(t), m, r) = \frac{2}{1 + e^{-\frac{(f_s(t) - (m + \Delta m))}{2((r + \Delta r))^2}}} - 1 \in [-1 \ 1] \quad (3)$$

$$\Delta m = m \left(\frac{2}{1 + e^{-2f_s(t)}} - 1 \right) \quad (4)$$

$$\Delta r = r \left(\frac{2}{1 + e^{-2f_s(t)}} - 1 \right) \quad (5)$$

It can be seen from (3) that the proposed activation function is characterized by the mean (m) and standard deviation (r) respectively. When Δm and Δr are both zero, the values of m (which is functionally equivalent to the static bias of the traditional neural network) and r govern the zero-crossing point and the steepness of the activation function respectively. Δm and Δr are to adjust the zero-crossing point and the steepness of the activation function according to the value of $f_s(t)$. Referring to Fig. 1, the network memory stores a set of parameters m and r , and these parameters are used to manipulate the zero-crossing point and the steepness of the activation function of the proposed neuron. From (4) and (5), we can see that the values of the Δm and Δr depend on the network inputs (Δm and Δr are functions of $f_s(t)$) and the parameters m and r . In other words, it operates as if the neural network handles different input data with different network parameters Δm and Δr . Notice that the value of the parameters m and r are determined during the training process. After training, these parameters are fixed.

B. Input-Output Relationship of the Proposed Neural Network

Referring to Fig. 3, the network output is defined as,

$$y_k(t) = tf^2\left(\sum_{j=1}^{n_h} w_{kj}^{(2)} f_{s_j}(t)\right), j = 1, 2, \dots, n_{out} \quad (6)$$

where

$$f_{s_j}(t) = tf^1\left(\sum_{i=1}^{n_{in}} w_{ji}^{(1)} z_i(t), m_j, r_j\right), j = 1, 2, \dots, n_h \quad (7)$$

denotes the output of the j -th hidden node. In the proposed neural network, the values of the parameters $w_{ji}^{(1)}$, $w_{kj}^{(2)}$, m_j , r_j will be trained by the RCGA in [1]. These parameters are fixed after the training process. The network will use these trained parameters and the input testing data to adjust the parameters of each hidden-node activation function; hence, the form of the activation function is input dependent. The total number of tunable parameters of the proposed neural networks is $(n_{in} + n_{out} + 2)n_h$.

C. Parameters design for IDNN

C.1. Number of hidden node (n_h)

The size of the hidden layer is a question often raised on designing multilayer feed-forward neural networks for real-life applications. An analytical method to determine the number of hidden node is difficult to obtain owing to the complexity of the network structure and the limited knowledge of the application. Hence, the number of hidden node is usually determined experimentally. In practice, the number of the hidden nodes depends on the application and the dimension of the input space.

C.2. Parameters in network memory (m and r)

There are two parameters in network memory: m and r . The value of m governs the zero-crossing point of the activation function and the value of r governs the steepness of the activation function. In (3), the output value of the proposed activation function is ranged from -1 to 1 . The shape of the activation function is shown in Fig. 5. In Fig. 5(a), the effect of the value of m to the shape of the activation function $tf(\cdot)$ is shown. The value of r is fixed at 1 , and the value of m is chosen from -4 to 4 . In Fig. 5(b), the effect of the value of r is shown. The value of m is fixed at 0 , and the value of r is chosen from 0.6 to 1.4 . It can be observed that the zero-crossing point and the steepness of the activation function are governed by the values of m and r respectively.

C.3. Network parameters (weights)

RCGA is the tool to search the optimal values of m , r and the network parameters (weights), $w_{ji}^{(1)}$, $w_{kj}^{(2)}$ of the IDNN. The training process for the IDNN is to minimize the errors between the desired outputs and the actual ones. More details about the tuning of network parameters using RCGA will be presented in the next section.

III TRAINING OF NETWORK PARAMETERS USING REAL-CODED GENETIC ALGORITHM

In this section, the improved RCGA [1] will be employed to obtain the optimal parameters of the proposed neural network. The details of the RCGA [1] are shown in the Appendix. RCGA is a powerful global search algorithm that has been widely applied in various optimization problems. By using RCGA, the detailed information of the nonlinear system, in particular the derivative information of the fitness function, is not necessarily known. Hence, RCGA is suitable to handle some complex optimization problems. As the RCGA process is basically using a random search technique, the convergence to a global optimum may be in doubt within a finite search time. In this paper, the improved RCGA in [1] is employed to optimize the fitness function, which is characterized by the parameters of the IDNN. In general, the fitness function is a mathematical expression quantitatively measures the performance of the RCGA tuning process. A larger fitness value indicates better tuning performance. By adjusting the values of the network parameters, the fitness function is maximized by using the RCGA. During the tuning process, offspring with better fitness values is ensured. Also, the mutation operation contracts gradually in the search domain with respect to the current iteration number. This helps the convergence of the searching process. After the tuning process, the obtained network parameter values will be used by the proposed neural network. As the proposed neural network is a feed-forward one, the outputs are bounded if its inputs are bounded, which happens in most of the real-life applications. Consequently, no convergence problem is present in the neural network itself.

To learn the input-output relationship of a neural network by using the improved RCGA, let the relationship be described by,

$$\mathbf{y}^d(t) = \mathbf{g}(\mathbf{z}^d(t)), t = 1, 2, \dots, n_d \quad (8)$$

where $\mathbf{z}^d(t) = [z_1^d(t) \ z_2^d(t) \ \dots \ z_{n_{in}}^d(t)]$ and $\mathbf{y}^d(t) = [y_1^d(t) \ y_2^d(t) \ \dots \ y_{n_{out}}^d(t)]$ are the given inputs and the desired outputs of an unknown nonlinear function $\mathbf{g}(\cdot)$ respectively, n_d denotes the number of input-output data pairs. The fitness function is defined as,

$$fitness = \frac{1}{1 + err} \quad (9)$$

In (9), err can be the mean square error (MSE), mean absolute error (MAE), mean absolute percentage error (MAPE), etc. The MSE is defined as:

$$err = \frac{\sum_{t=1}^{n_d} \sum_{k=1}^{n_{out}} (y_k^d(t) - y_k(t))^2}{n_d n_{out}} \quad (10)$$

The MAE is defined as:

$$err = \frac{\sum_{t=1}^{n_d} \sum_{k=1}^{n_{out}} |y_k^d(t) - y_k(t)|}{n_d n_{out}} \quad (11)$$

The MAPE is defined as:

$$err = \frac{\sum_{t=1}^{n_d} \sum_{k=1}^{n_{out}} \frac{|y_k^d(t) - y_k(t)|}{y_k^d(t)}}{n_d n_{out}} \quad (12)$$

The objective is to maximize the fitness value of (9) using the improved RCGA by setting the chromosome to be $[w_{ji}^1 \ m_j \ r_j \ w_{kj}^2]$ for all i, j and k . The range of the fitness value of (9) is from 0 to 1. It can be seen from (9) and (12) that a larger $fitness$ implies a smaller err . The actual fitness function depends on the type of application. The one presented in this section is only for illustrating some general mapping problems using supervised learning. Different fitness functions should be employed for different types of applications.

IV APPLICATION EXAMPLES

In this section, two examples will be given to illustrate the merits of the proposed IDNN.

A. Short-term load forecasting

We consider the short-term load forecasting (STLF) for the power supply system in Hong Kong. STLF is important because it plays a role in the formulation of economic, reliable, and secure operating strategies for the power system [23]. The objectives of STLF is i) to derive the scheduling functions that determine the most economic load dispatch with operational constraints and policies, environmental and equipment limitations; ii) to assess the security of the power system at any time point; iii) to provide system dispatchers with timely information. The non-linear and large-scale characteristics generally make the STLF problem difficult to solve [24]. Neural/neural-fuzzy networks [24-34] offer a promising and effective platform for tackling this problem.

The proposed IDNN is applied to do STLF. The idea is to construct seven multi-input multi-output neural networks, one for each day of a week. Each neural network has 24 outputs representing the expected hourly load for a day. A diagram of one of the seven NNs for the load forecasting is shown in Fig. 6. The network has 28 inputs and 24 outputs. Among the 28 inputs nodes, the first 24 nodes (z_1, \dots, z_{24}) represent the previous 24 hourly loads [34] and are denoted by $z_i = L_i^d(t-1)$, $i = 1, 2, \dots, 24$. Node 25 (z_{25}) and node 26 (z_{26}) represent the average temperatures of the previous day ($T(t-1)$) and the forecasted averaged temperatures of the present day ($T(t)$) respectively provided by the local observatory. Node 27 (z_{27}) and node 28 (z_{28}) represent the average relative humidity at the previous day ($RH(t-1)$) and the forecasted average relative humidity at the present day ($RH(t)$) respectively provided by the local observatory. The output layer consists of 24 output nodes that represent the forecasted 24 hourly loads of a day, and are denoted by $y_k(t) = L_i(t)$, $i = 1, 2, \dots, 24$. Such a network structure is chosen based on the assumption that the consumption patterns of the seven days within a week would differ significantly among each other, while the patterns among the same day of weeks are similar. By using the past 24 hourly loads as the inputs, the relationship between a given hour's load and the 24 hourly loads of the pervious day can be considered. Temperature information (Node 25 and Node 26) is important inputs to the STLF. For any given day, the deviation of the temperature values may cause such significant load changes as to require major modifications in the unit commitment pattern. Humidity is similar to temperature that affects the system load particularly in hot and humid areas.

In this paper, we use a data set in year 2000 provided by CLP Power Hong Kong Ltd to illustrate the performance of the proposed IDNN on doing STLF. The networks were trained using 20 weeks of load data in residential area from Mar 20 to Aug 6, 2000. The load pattern for every Thursday and Sunday from Mar 20 to Aug 6, 2000 are shown in Fig. 7(a)

and Fig. 7(b) respectively. In these two figures, we can see that the shape of every load pattern is similar but the power consumption is much different. Conventional feed-forward type neural networks, such as 3-layer feed-forward fully-connected neural networks (FFCNN) [3], wavelet neural networks (WNN) [41-43] (which combine feed-forward neural networks with the wavelet theory that provides a multi-resolution approximation for discriminate functions to enhance function learning) are only good at minimizing the average error of the system. It is difficult to model all the load patterns accurately. IDNN has the ability to divide the input-output domain into several sub-domains, and the parameters in the activation functions are allowed to change to cope with the changes of the network inputs in different sub-domains. As a result, the IDNN operates as different individual neural networks to handle the inputs of different operating sub-domains. The load forecasting problem should then be better handled. Referring to (6), the proposed IDNN used for doing STLF can be described by,

$$y_k(t) = tf^2 \left(\sum_{j=1}^{n_h} w_{kj}^{(2)} tf^1 \left(\sum_{i=1}^{n_m} w_{ji}^{(1)} z_i(t), m_j, r_j \right) \right), k = 1, 2, \dots, 23, 24. \quad (13)$$

where $tf^2(\alpha) = \alpha \in [-\infty \ \infty]$, $\alpha \in \mathfrak{R}$ is a pure linear activation function.

The RCGA in [1] is used to tune the parameters of the proposed IDNN of (13). The fitness function is defined as follows,

$$fitness = \frac{1}{1 + err} \quad (14)$$

$$err = \frac{\sum_{t=1}^{20} \sum_{k=1}^{24} \frac{|y_k^d(t) - y_k(t)|}{y_k^d(t)}}{20 \times 24} \quad (15)$$

The value of *err* indicates the mean absolute percentage error (MAPE) of the load forecasting system. The objective is to maximize *fitness* (minimize *err*). For comparison, a wavelet neural network (WNN) and a 3-layer fully connected neural network (FFCNN) are also applied to do the same job. All parameters of the various networks are trained by the RCGA. For all cases, the initial values of the parameters of the neural networks are randomly generated. The lower and upper bounds of the IDNN parameters are defined as $w_{ij}^1, w_{jk}^2, m_j^1 \in [-3 \ 3]$ and $r_j^1 \in [0.8 \ 1.2]$. The lower and upper bounds of the WNN and FFCNN parameters are -3 and 3 respectively. The number of iteration to train the neural networks is 10000. For the RCGA, the probability of crossover (p_c) and the probability of mutation (p_m) are 0.8 and 0.01 respectively; the weights of crossover w_a and w_b are set at

0.5 and 1 respectively; the WM shape parameter ζ is 2. The population size is 50. All the results are averaged ones out of 20 runs. Table I shows the average, best training and forecasting results offered by the proposed IDNN, WNN [41] and FFCNN [3] for Thursday respectively. Also, the number of parameters of the network (n_{para}) is given. From Table I, it can be shown that the proposed IDNN performs better than the WNN [41] and FFCNN [3]. With the proposed IDNN, the best average training errors are 2.8304% at $n_h = 26$. This implies 11.1% and 21.6% improvement over the best performed WNN ($n_h = 26$) and FFCNN ($n_h = 26$). To test the forecasting ability, the data of 4 weeks from 7 Aug to 03 Sep 2000 are used. The best average forecasting error for the IDNN is 2.2345%. It implies 20.4% improvement for the WNN and 36.8% improvement for the FFCNN.

The results of the daily load forecasting with different numbers of hidden nodes (n_h) for Sunday are tabulated in Table II. In this table, it can be shown that the proposed IDNN performs better. Also, the best results are achieved when the number of hidden node (n_h) is set as 24. Comparing with the WNN ($n_h = 24$) and the FFCNN ($n_h = 24$), the improvements brought by the IDNN are 14.7% and 19.7% respectively. The best average forecasting error for the IDNN is 2.2609% for Sunday. It implies 13.7% improvement for the WNN and 17.8% improvement for the FFCNN.

The best average training and forecasting errors offered by the proposed IDNN, WNN and FFCNN for Monday to Sunday are tabulated in Table III. The average training error and the forecasting error offered by the IDNN are 2.7919% and 2.1971%, which imply 12.1% and 13.0% improvement respectively over the WNN, and 19.4% and 19.8% improvement respectively over the FFCNN. Fig. 8 and Fig. 9 show the comparison of the average forecasting errors (MAPE) for Thursday and Sunday respectively. In these figures, the solid line is the result of the IDNN, the dotted line is the result of the WNN, and the dash-dot line is the result of the FFCNN. In these two figures, we can see that the average forecasting errors are smaller for the IDNN. In conclusion, the IDNN gives a better result on solving the STLF problem.

B. Hand-written graffiti recognition

A hand-written graffiti pattern recognition problem is given to illustrate the superior learning and generalization abilities of the proposed IDNN to classification problems with a large number of input sets. The recognition is realized practically in a prototyped electronic book. In general, the neural/neural-fuzzy network approaches are model-free and are good for graffiti recognition [35-40]. In this example, the digits 0 to 9 and three (control) characters (*backspace*, *carriage return* and *space*) are recognized by the IDNN. These graffiti patterns are shown in Fig. 10. A point in each graffiti pattern is characterized by a number based on the x - y coordinates on a writing area. The size of the writing area is x_{\max} by y_{\max} . The bottom left corner is set as $(0, 0)$. Ten uniformly sampled points of the graffiti pattern will be taken as the inputs of the interpreter. The points are taken in the following way. First, the input graffiti is divided into 9 uniformly distanced segments characterized by 10 points, including the start and the end points. Each point is labelled as (x_i, y_i) , $i = 1, 2, \dots, 10$. The first 5 points, (x_i, y_i) , $i = 1, 3, 5, 7$ and 9 taken alternatively are converted to 5 numbers ρ_i respectively by using the formula $\rho_i = x_i x_{\max} + y_i$. The other 5 points, (x_i, y_i) , $i = 2, 4, 6, 8$ and 10 are converted to 5 numbers respectively by using the formula $\rho_i = y_i y_{\max} + x_i$. These ten numbers, z_i , $i = 1, 2, \dots, 10$, will be used as the inputs of the proposed graffiti recognizer. The hand-written graffiti recognizer as shown in Fig. 11 is proposed to perform the graffiti recognition. In this figure, the inputs are defined as follows,

$$\bar{\mathbf{z}}(t) = \frac{\mathbf{z}(t)}{\|\mathbf{z}(t)\|} \quad (16)$$

where $\bar{\mathbf{z}}(t) = [\bar{z}_1(t) \ \bar{z}_2(t) \ \dots \ \bar{z}_{10}(t)]$ denotes the normalized input vectors of the proposed graffiti recognizer; $\mathbf{z}(t) = [z_1(t) \ z_2(t) \ \dots \ z_{10}(t)]$ denotes the ten points in the writing area; $\|\cdot\|$ denotes the l_2 vector norm. The 16 outputs, $y_j(t)$, $j = 1, 2, \dots, 16$, indicate the similarity between the input pattern and the 16 standard patterns respectively. The input-output relationship of the neural network is defined as that the output $y_i(t) = 1$ and others are zero when the input vector belongs to pattern i , $i = 1, 2, \dots, 16$. For examples, the desired outputs of the pattern recognition system are $[1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$ for the digit “0(a)”, $[0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$ for the digit “0(b)”, and $[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1]$ for the character “space” respectively. After training, the graffiti determiner is used to determine the output of the recognizer. A larger value of $y_j(t)$ implies that the input pattern j matches more closely to the input graffiti pattern. For instance, a large value of $y_0(t)$ implies that the input pattern is recognized as “0”.

To train the neural network of the hand-written graffiti recognition system, a set of training patterns governing the input-output relationship will be employed. 1600 training patterns (100 for each graffiti pattern) will be used in this example. The training patterns consist of the input vectors and its corresponding expected output. The fitness function is given by (9), with

$$err = \sum_{k=1}^{16} \frac{\sum_{t=1}^{100} \left(\frac{y_k(t)}{\|\mathbf{y}(t)\|} - \frac{y_k^d(t)}{\|\mathbf{y}^d(t)\|} \right)^2}{16 \times 100} \quad (17)$$

where $\mathbf{y}^d(t) = [y_1^d(t) \ y_2^d(t) \ \dots \ y_{16}^d(t)]$ denotes the expected output vector and $\mathbf{y}(t) = [y_1(t) \ y_2(t) \ \dots \ y_{16}(t)]$ is the actual network output defined as,

$$y_j(t) = tf^2 \left(\sum_{k=1}^{n_h} w_{jk}^2 tf^1 \left(\sum_{l=1}^{10} w_{kl}^1 x_l(t), m_k^1, r_k^1 \right) \right), j = 1, 2, \dots, 16. \quad (18)$$

where $tf^2(\alpha) = \alpha \in [-\infty \ \infty]$, $\alpha \in \mathfrak{R}$ is a pure linear activation function.

For comparison purpose, a WNN [41] and a FFCNN [3] are also used in this example. For all cases, the initial values of the parameters of the neural network are randomly generated. The lower and upper bounds of the network parameters for the IDNN are defined as $w_{ij}^1, w_{jk}^2, m_j \in [-4 \ 4]$ and $r_j \in [0.8 \ 1.2]$. The lower and upper bounds of the WNN and FFCNN parameters are -4 and 4 respectively. The number of iteration to train the neural networks is 15000. For the RCGA [1], the probability of crossover (p_c) and the probability of mutation (p_m) are 0.8 and 0.05 respectively; the weights of crossover w_a and w_b are set at 0.5 and 1 respectively; the WM shape parameter ζ is 2. The population size is 50. All the results are averaged ones out of 20 runs.

The training results and recognition accuracy rate of all approaches with different numbers of hidden nodes (n_h) are tabulated in Table IV. Also, the numbers of parameters (n_{para}) for different approaches with different n_h are given. It can be seen from Table IV that the recognition system implemented by the proposed IDNN outperforms that of the other approaches in terms of the average training mean square error and the recognition accuracy rate. The best results are achieved when the number of hidden nodes (n_h) is set at 22 for the IDNN and 24 for the WNN and FFCNN. The average training errors are 0.0223 for the IDNN, 0.0309 for the WNN and 0.0360 for the FFCNN. This implies 38.56% and 61.43%

improvements respectively. The average recognition accuracy rate of the IDNN is 95.83% , as compared with 93.23% of the WNN and 91.50% of the FFCNN.

In order to test the generalization ability of the proposed neural networks, a set of testing patterns consisting of 480 input patterns (30 for each graffiti pattern) is used. The results in terms of the average testing error (mean square error), best testing error and recognition accuracy rate are tabulated in Table V. In this Table, we can see that the IDNN gives better performance. The average testing errors are 0.0247 for the IDNN at $n_h = 22$, 0.0329 for the WNN at $n_h = 24$, and 0.0374 for the FFCNN at $n_h = 24$. This implies 33.20% and 51.42% improvements respectively. The average recognition rate of the IDNN is 95.92% as compared with 93.92% of the WNN and 91.25% of the FFCNN.

Table VI and VII shows the overall training and testing recognition rate for the digits (0-9) and commands achieved by the IDNN, WNN, and FFCNN. Referring to Table VI and VII, it can be seen the proposed network performs better. The underlined results show more than 4% recognition accuracy rate difference between the results of the IDNN and the WNN or the FFCNN. The digits '1', '6', '9', and the command 'space' are done particularly well by the IDNN. Fig. 12 shows the output values of the IDNN, WNN and FFCNN for the 480 (30 for each digit/command) testing graffiti patterns. In this figure, the x -axis represents the pattern number for corresponding digit/command. Pattern numbers 1 to 30 are for the digit '0'(a), numbers 31-60 are for the digit '0' (b), and so on. The y -axis represents the output y . As mentioned previously, the input-output relationship of the network is defined as that the output $y_i(t) = 1$ and others are zero when the input vector belongs to pattern i , $i = 1, 2, \dots, 16$. For instance, referring to Fig.12(b), we can see that the output y of pattern numbers within 31-60 for the IDNN are nearest to 1 and others are nearest to zero when comparing with other approaches. This figure shows that the recognition ability of the IDNN is good.

V. CONCLUSION

An input-dependent neural network has been proposed in this paper. The parameters of the proposed neural network are trained by the RCGA. Thanks to the property of input independence in the activation functions of the network, the learning and generalization abilities have been increased. Two application examples of short-term load forecasting and hand-written graffiti recognition have been given to show the merits of the proposed network.

REFERENCES

- [1] S.H. Ling and F.H.F. Leung, "An improved genetic algorithm with average-bound crossover and wavelet mutation operations," *Soft Computing*, vol. 11, no. 1, pp. 7-31, Jan. 2007.
- [2] F.M. Ham and I. Kostanic, *Principles of Neurocomputing for Science & Engineering*. McGraw Hill, 2001.
- [3] B. Widrow and M.A. Lehr, "30 years of adaptive neural networks: Perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415-1442, Sept. 1990.
- [4] J.M. Zurada, *Introduction to Artificial Neural Systems*. West Info Access, 1992.
- [5] S. Haykin, *Neural Network: A Comprehensive Foundation*. Prentice Hall, 1999.
- [6] M.F. Moller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6. no. 4, pp. 525-533, 1993.
- [7] D.T. Pham and D. Karaboga, *Intelligent Optimization Techniques, Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. Springer, 2000.
- [8] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.
- [9] Z. Michalewicz, *Genetic Algorithm + Data Structures = Evolution Programs*, 2nd extended ed. Springer-Verlag, 1994.
- [10] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423-1447, Sept. 1999.
- [11] S.H. Ling, F.H.F. Leung, H.K. Lam, Y.S. Lee, and P.K.S. Tam, "A novel GA-based neural network for short-term load forecasting," *IEEE Trans. Industrial Electronics*, vol. 50, no. 4, pp. 793-799, Aug. 2003.
- [12] K.F. Leung, F.H.F. Leung, H.K. Lam, and S.H. Ling "On interpretation of graffiti digits and characters for eBooks: Neural-fuzzy network and genetic algorithm approach," *IEEE Trans. Industrial Electronics*, vol. 51, no. 2, pp. 464-471, Aug. 2004.
- [13] F.H.F. Leung, H.K. Lam, S.H. Ling, and P.K.S. Tam, "Tuning of the structure and parameters of neural network using an improved genetic algorithm," *IEEE Trans. Neural Networks*, vol. 14, no. 1, pp. 79-88, Jan. 2003.
- [14] L. Davis, *Handbook of Genetic Algorithms*. NY: Van Nostrand Reinhold, 1991.
- [15] M. Srinivas and L.M. Patnaik, "Genetic algorithms: a survey," *IEEE Computer*, vol. 27, issue 6, pp. 17-26, June 1994.
- [16] H.K. Lam, S.H. Ling, F.H.F. Leung, and P.K.S. Tam, "Function estimation using a

- neural-fuzzy network and an improved genetic algorithm," *International Journal of Approximate Reasoning*, vol. 30, no. 3, pp. 243-260, Jul. 2004.
- [17] S.H. Ling, F.H.F. Leung, L.K. Wong, and H.K. Lam, "Computational intelligence techniques for home electric load forecasting and balancing," *International Journal of Computational Intelligence and Applications*, vol. 5, no. 3, pp. 371-391, Sep. 2005.
- [18] H.K. Lam and F.H.F. Leung, "Design and stabilization of sampled-data neural-network-based control systems," *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 36, no. 5, pp. 995-1005, Oct. 2006.
- [19] H.K. Lam and F.H.F. Leung, "Stability analysis, synthesis and optimization of radial-basis-function neural-network based controller for nonlinear systems," in *Proc. 30th Annual Conference of the IEEE Industrial Electronics Society, IECON, Busan, Korea*, Nov 2004, pp. 2813-2818.
- [20] I. Daubechies, "The wavelet transform, time-frequency localization and signal analysis," *IEEE Trans. Information Theory*, vol. 36, no. 5, pp. 961-1005, Sep. 1990.
- [21] S.G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674-693, Jul. 1989.
- [22] J. Joines and C. Houck, "On the use of non-stationary penalty functions to solve constrained optimization problems with genetic algorithm," in *Proc. 1994 Int. Symp. Evolutionary Computation*, Orlando, 1994, pp. 579-584.
- [23] G. Gross and F.D. Galiana, "Short-term load forecasting," *IEEE Proceedings*, vol. 75, pp. 1558-1573, 1987.
- [24] R. Aggarwal and Y.H. Song, "Artificial neural networks in power systems," *Power Engineering Journal*, pp. 279-287, Dec. 1998.
- [25] C.C. Hsu and C.Y. Chen, "Regional load forecasting in Taiwan—applications of artificial neural networks," *Energy Conversion and Management*, vol. 44, pp. 1941-1949, Jul. 2003.
- [26] K.H. Yang, J.J. Zhu, B.S. Wang, and L.L. Zhao, "Design of short-term load forecasting model based on fuzzy neural networks," in *Proc. 5th World Congress on Intelligent Control and Automation 2004*, vol. 3, pp. 15-19, June 2004.
- [27] R.H. Liang and C.C. Cheng, "Short-term load forecasting by a neuro-fuzzy based approach," *Energy Power and Energy Systems*, vol. 24, pp. 103-111, 2002.

- [28] J.W. Taylor and R. Buizza, "Neural network load forecasting with weather ensemble predictions," *IEEE Trans. Power Syst.*, vol. 17, no. 4, pp. 626-632, Aug. 2002.
- [29] T. Senjyu, H. Takara, and T. Funabashi, "One-hour-ahead load forecasting using neural network," *IEEE Trans. Power Syst.*, vol. 17, no.1, pp. 113-118, Feb. 2002.
- [30] W. Charytoniuk and M.S. Chen, "Very short-term load forecasting using artificial neural networks," *IEEE Trans. Power Syst.*, vol. 15, no.1, pp. 263-268, Feb. 2000.
- [31] I. Drezga and S. Rahman, "Short-term load forecasting with local ANN predictors," *IEEE Trans. Power Syst.*, vol. 14, no.3, pp. 844-850, Aug. 1999.
- [32] T. Saksornchai, W.J. Lee, K. Methaprayoon, J.R. Liao, and R.J. Ross, "Improve the unit commitment scheduling by using the neural-network-based short-term load forecasting," *IEEE Trans. Ind. App.*, vol. 41, no. 1, pp. 169-179, Jan. 2005.
- [33] E.T.H. Heng, D. Srinivasan, and A.C. Liew, "Short term load forecasting using genetic algorithm and neural networks," in *Proc. 1998 Int. Conf. Energy Management and Power Delivery*, vol. 2, 1998, pp. 576-581.
- [34] J.A. Momoh, Y. Wang, and M. Elfayoumy, "Artificial neural network based load forecasting," in *Proc. IEEE Int. Conf. System, Man, and Cybernetics: Computational Cybernetics and Simulation*, vol. 4, 1997, pp. 3443-3451.
- [35] R. Buse, Z.Q. Liu, and J. Bezdek, "Word recognition using fuzzy logic," *IEEE Trans. Fuzzy Systems*, vol. 10, no. 1, Feb. 2002.
- [36] P.D. Gader and M.A. Khabou, "Automatic feature generation for handwritten digit recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 18, no. 12, pp. 1256-1261, Dec. 1996.
- [37] B. Zhang, M. Fu, H. Yan, and M.A. Jabri, "Handwritten digit recognition by adaptive-subspace self-organizing map (ASSOM)," *IEEE Trans. on Neural Networks*, vol. 10, no. 4, pp. 939-945, Jul. 1997.
- [38] D.R. Lovell, T. Downs, and A.C. Tsoi, "An evaluation of the neocognitron," *IEEE Trans. on Neural Networks*, vol. 8, no. 5, pp. 1090-1105, Sep. 1997.
- [39] C.A. Perez, C.A. Salinas, P.A. Estévez, and P.M. Valenzuela, "Genetic design of biologically inspired receptive fields for neural pattern recognition," *IEEE Trans. on Systems, Man, and Cybernetics – Part B: Cybernetics*, vol. 33, no. 2, pp. 258-270, Apr. 2003.
- [40] L. Holmström, P. Koistinen, J. Laaksonen, and E. Oja, "Neural and statistical classifiers – Taxonomy and two case studies," *IEEE Trans. on Neural Networks*, vol. 8, no. 1, pp. 5-17, Jan. 1997.

- [41] S. Yao, C.J. Wei, and Z.Y. He, "Evolving wavelet neural networks for function approximation," *Electron. Lett.*, vol.32, no.4, pp. 360-361, Feb. 1996.
- [42] Q. Zhang and A. Benveniste, "Wavelet networks," *IEEE Trans. on Neural Networks*, vol. 3, no. 6, pp. 889-898, Nov. 1992.
- [43] J. Zhang, G.G. Walter, Y. Miao, and W.W.N. Lee, "Wavelet neural networks for function learning," *IEEE Trans. on Signal Processing*, vol. 43, no. 6, pp. 1485-1497, Jun. 1995.
- [44] V. Kecman, *Learning and Soft Computing, Support Vector machines, Neural Networks and Fuzzy Logic Models*. Cambridge, MA: The MIT Press, 2001.
- [45] K.C. Tan, M.H. Lim, X. Yao, and L.P. Wang (Eds.), *Recent Advances in Simulated Evolution and Learning*. Singapore: World Scientific, 2004.
- [46] J.E. Fieldsend and S. Singh, "Pareto evolutionary neural networks," *IEEE Trans. Neural Networks*, vol. 16, no. 2, pp. 338 - 354, March 2005.
- [47] L.P.Wang and X.J. Fu, *Data Mining with Computational Intelligence*. Berlin: Springer, 2005.
- [48] N. Garcia-Pedrajas, C. Hervas-Martinez, and J. Munoz-Perez, "COVNET: a cooperative coevolutionary model for evolving artificial neural networks," *IEEE Trans. Neural Networks*, vol. 14, no. 3, pp. 575 - 596, May 2003.

ACKNOWLEDGEMENT

The work described in this paper was substantially supported by a grant from the Hong Kong Polytechnic University (PhD Student Account Code: RG9T). The data in the application example are offered by CLP Power Hong Long Limited for illustration purpose only.

Appendix

In this paper, all parameters of the neural networks are trained by the RCGA with new genetic operations (crossover and mutation) [1]. The RCGA process is shown in Fig. 13. First, a set of population of chromosomes $P = [P_1, P_2, \dots, P_{pop_size}]$ is created (where pop_size is the number of chromosomes in the population). Each chromosome \mathbf{p} contains some genes (variables). Second, the chromosomes are evaluated by a defined fitness function.

The better chromosomes will return higher values in this process. The form of the fitness function depends on the application. Third, some of the chromosomes are selected to undergo genetic operations for reproduction by the method of normalized geometric ranking [22]. Forth, genetic operations of crossover are performed. The crossover operation is mainly for exchanging information between the two parents that are obtained by the selection operation. In the crossover operation, one of the parameters is the probability of crossover p_c which gives the expected number of chromosomes that undergo the crossover operation. We propose a new crossover operation described as follows: 1) four chromosomes (instead of two in the conventional RCGA) will be generated, 2) the best two offspring in terms of the fitness value are selected to replace their parents. After the crossover operation, the mutation operation follows. It operates with the parameter of the probability of mutation (p_m), which gives an expected number ($p_m \times pop_size \times no_vars$) of genes that undergo the mutation. The mutation operation is to change the genes of the chromosomes in the population such that the features inherited from their parents can be changed. After going through the mutation operation, the new offspring will be evaluated using the fitness function. The new population will be reproduced when the new offspring replaces the chromosome with the smallest fitness value. After the operations of selection, crossover and mutation, a new population is generated. This new population will repeat the same process iteratively until a defined condition is met. The details about the crossover and mutation operations are given as follows.

A. Average-Bound Crossover

The crossover operation is called the average-bound crossover (ABX), which is mainly for exchanging information from the two parents, chromosomes \mathbf{p}_1 and \mathbf{p}_2 , obtained in the selection process. The two parents will produce four offspring. The ABX comprises two operations: average crossover and bound crossover. The details of the crossover operation are as follows,

Average crossover

$$\mathbf{o}_{s_c^1} = \frac{\mathbf{p}_1 + \mathbf{p}_2}{2} \quad (A1)$$

$$\mathbf{o}_{s_c^2} = \frac{(\mathbf{p}_{\max} + \mathbf{p}_{\min})(1 - w_a) + (\mathbf{p}_1 + \mathbf{p}_2)w_a}{2} \quad (A2)$$

Bound crossover

$$\mathbf{o}_{s_c^3} = \mathbf{p}_{\max}(1 - w_b) + \max(\mathbf{p}_1, \mathbf{p}_2)w_b \quad (A3)$$

$$\mathbf{o}_{s_c^4} = \mathbf{p}_{\min} (1 - w_b) + \min(\mathbf{p}_1, \mathbf{p}_2) w_b \quad (\text{A4})$$

where

$$\mathbf{o}_{s_c^k} = \left[o_{s_1^k} \quad o_{s_2^k} \quad \cdots \quad o_{s_{no_vars}^k} \right], \quad k = 1, 2, 3, 4.$$

$$\mathbf{p}_i = \left[p_{i_1} \quad p_{i_2} \quad \cdots \quad p_{i_j} \quad \cdots \quad p_{i_{no_vars}} \right], \quad i = 1, 2; j = 1, 2, \dots, no_vars, \quad (\text{A5})$$

$$para_{\min}^j \leq p_{i_j} \leq para_{\max}^j, \quad (\text{A6})$$

$$\mathbf{p}_{\max} = \left[para_{\max}^1 \quad para_{\max}^2 \quad \cdots \quad para_{\max}^{no_vars} \right], \quad (\text{A7})$$

$$\mathbf{p}_{\min} = \left[para_{\min}^1 \quad para_{\min}^2 \quad \cdots \quad para_{\min}^{no_vars} \right], \quad (\text{A8})$$

no_vars denotes the number of variables to be tuned; $para_{\min}^j$ and $para_{\max}^j$ are the minimum and maximum values of p_{i_j} respectively for all i ; $w_a, w_b \in [0 \ 1]$ denotes the weights of average crossover and bound crossover to be determined by users respectively, $\max(\mathbf{p}_1, \mathbf{p}_2)$ denotes the vector with each element obtained by taking the maximum between the corresponding element of \mathbf{p}_1 and \mathbf{p}_2 . For instance, $\max([1 \ -2 \ 3], [2 \ 3 \ 1]) = [2 \ 3 \ 3]$. Similarly, $\min(\mathbf{p}_1, \mathbf{p}_2)$ gives a vector by taking the minimum value. Among $\mathbf{o}_{s_c^1}$ to $\mathbf{o}_{s_c^4}$, the two with the largest fitness value are used as the offspring of the crossover operation. These two offspring are put back into the population to replace their parents.

The rationale behind the ABX is that if the offspring spreads over the domain, a higher chance of reaching the global optimum can be obtained. As seen from (A1) to (A4): (A1) and (A2) will move the offspring near the centre region of the concerned domain (as w_a in (A2) approaches 1, $\mathbf{o}_{s_c^2}$ approaches $\frac{\mathbf{p}_1 + \mathbf{p}_2}{2}$, which is the average of the selected parents; and as w_a approaches 0, $\mathbf{o}_{s_c^2}$ approaches $\frac{\mathbf{p}_{\max} + \mathbf{p}_{\min}}{2}$, which is the average of the domain boundary), while (A3) and (A4) will move the offspring near the domain boundary (as w_b in (A3) and (A4) approaches 0, $\mathbf{o}_{s_c^3}$ and $\mathbf{o}_{s_c^4}$ approaches \mathbf{p}_{\max} and \mathbf{p}_{\min} respectively). The result of the crossover depends on the values of the weights w_a and w_b . They vary with the optimisation problem and are chosen by trial and error. For many problems, the value of w_a can be set as 0.5. Changing the value of the weight w_b will change the characteristics of the bound crossover operation.

B. Wavelet Mutation

The wavelet theory [20-21] is applied to do the mutation operation.

Wavelet theory

Certain seismic signals can be modelled by combining translations and dilations of an oscillatory function with finite duration called a “wavelet”. A continuous-time function $\psi(x)$ is called a “mother wavelet” or “wavelet” if it satisfies the following properties:

Property 1:

$$\int_{-\infty}^{+\infty} \psi(x) dx = 0 \quad (\text{A9})$$

In other words, the total positive energy of $\psi(x)$ is equal to the total negative energy of $\psi(x)$.

Property 2:

$$\int_{-\infty}^{+\infty} |\psi(x)|^2 dx < \infty \quad (\text{A10})$$

where most of the energy in $\psi(x)$ is confined to a finite duration and bounded. The Morlet wavelet is an example mother wavelet:

$$\psi(x) = e^{-x^2/2} \cos(5x) \quad (\text{A11})$$

The Morlet wavelet integrates to zero (*Property 1*). Over 99% of the total energy of the function is contained in the interval of $-2.5 \leq x \leq 2.5$ (*Property 2*). In order to control the magnitude and the position of $\psi(x)$, $\psi_{a,b}(x)$ is defined as:

$$\psi_{a,b}(x) = \frac{1}{\sqrt{a}} \psi\left(\frac{x-b}{a}\right) \quad (\text{A12})$$

where a is the dilation parameter and b is the translation parameter. Notice that

$$\psi_{1,0}(x) = \psi(x) \quad (\text{A13})$$

As

$$\psi_{a,0}(x) = \frac{1}{\sqrt{a}} \psi\left(\frac{x}{a}\right), \quad (\text{A14})$$

it follows that $\psi_{a,0}(x)$ is an amplitude-scaled version of $\psi(x)$. Fig. 14 shows different dilations of the Morlet wavelet. The amplitude of $\psi_{a,0}(x)$ will be scaled down as the dilation parameter a increases. This property is used to do the mutation operation to enhance the performance.

Operation of Wavelet Mutation

The mutation operation is to change the genes of the chromosomes inherited from their parents. In general, various methods like uniform mutation or non-uniform mutation

[14-15] can be employed to realize the mutation operation. We propose a Wavelet Mutation (WM) operation based on the wavelet theory, which has a fine-tuning ability. The details of the operation are as follows. Every gene of the chromosomes will have a chance to mutate governed by a probability of mutation, $p_m \in [0, 1]$, which is defined by the user. For each gene, a random number between 0 and 1 will be generated such that if it is less than or equal to p_m , the mutation will take place on that gene. For instance, if $\mathbf{o}_s = [o_{s_1}, o_{s_2}, \dots, o_{s_{no_vars}}]$ is the selected chromosome and the element o_{s_j} is randomly selected for mutation (the value of o_{s_j} is inside $[para_{\min}^j, para_{\max}^j]$), the resulting chromosome is given by $\hat{\mathbf{o}}_s = [o_{s_1}, \dots, \hat{o}_{s_j}, \dots, o_{s_{no_vars}}]$, where $j \in 1, 2, \dots, no_vars$, and

$$\hat{o}_{s_j} = \begin{cases} o_{s_j} + \delta \times (para_{\max}^j - o_{s_j}) & \text{if } \delta > 0 \\ o_{s_j} + \delta \times (o_{s_j} - para_{\min}^j) & \text{if } \delta \leq 0 \end{cases}, \quad (\text{A15})$$

By using the Morlet wavelet in (A11) as the mother wavelet,

$$\delta = \frac{1}{\sqrt{a}} e^{-\left(\frac{\varphi}{a}\right)^2 / 2} \cos\left(5\left(\frac{\varphi}{a}\right)\right) \quad (\text{A16})$$

If δ is positive ($\delta > 0$) approaching 1, the mutated gene will tend to the maximum value of o_{s_j} . Conversely, when δ is negative ($\delta \leq 0$) approaching -1 , the mutated gene will tend to the minimum value of o_{s_j} . A larger value of $|\delta|$ gives a larger searching space for o_{s_j} . When $|\delta|$ is small, it gives a smaller searching space for fine-tuning the gene. Referring to *Property 1* of the wavelet, the total positive energy of the mother wavelet is equal to the total negative energy of the mother wavelet. Then, the sum of the positive δ is equal to the sum of the negative δ when the number of samples is large. That is,

$$\frac{1}{N} \sum_N \delta = 0 \text{ for } N \rightarrow \infty, \quad (\text{A17})$$

where N is the number of samples.

Hence, the overall positive mutation and the overall negative mutation throughout the evolution are nearly the same. This property gives better solution stability (smaller standard deviation). As over 99% of the total energy of the mother wavelet function is contained in the interval $[-2.5, 2.5]$, φ can be generated from $[-2.5, 2.5]$ randomly. The value of the dilation parameter a is set to vary with the value of τ/T in order to meet the fine-tuning purpose, where T is the total number of iteration and τ is the current number of iteration. In

order to perform a local search when τ is large, the value of a should increase as τ/T increases so as to reduce the significance of the mutation. Hence, a monotonic increasing function governing a and τ/T is proposed as follows.

$$a = e^{-\ln(g) \times \left(1 - \frac{\tau}{T}\right)^\zeta + \ln(g)} \quad (\text{A18})$$

where ζ is the shape parameter of the monotonic increasing function, g is the upper limit of the parameter a . In this paper, g is set as 10000. The value of a is between 1 and 10000. Referring to (A16), the maximum value of δ is 1 when the random number of $\varphi=0$. Then referring to (A15), the offspring gene $\hat{o}_{s_j} = o_{s_j} + 1 \times (\text{para}_{\max}^j - o_{s_j})$. It ensures that a large search space for the mutated gene is given. When the value τ/T is near to 1, the value of a is so large that the maximum value of δ will become very small. For example, at $\tau/T = 0.9$ and $\zeta = 1$, the dilation parameter $a = 4000$. If the random value of φ is zero, the value of δ will be equal to 0.0158. With $\hat{o}_{s_j} = o_{s_j} + 0.0158 \times (\text{para}_{\max}^j - o_{s_j})$, a smaller searching space for the mutated gene is given for fine-tuning.

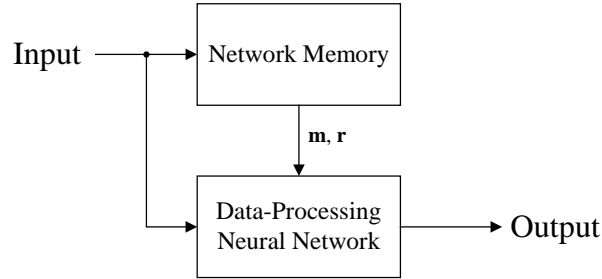


Fig. 1. Proposed architecture of the neural network.

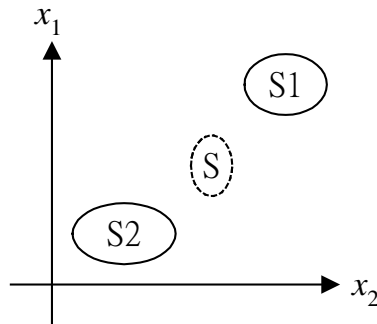


Fig. 2. Diagram showing two sets in the spatial domain.

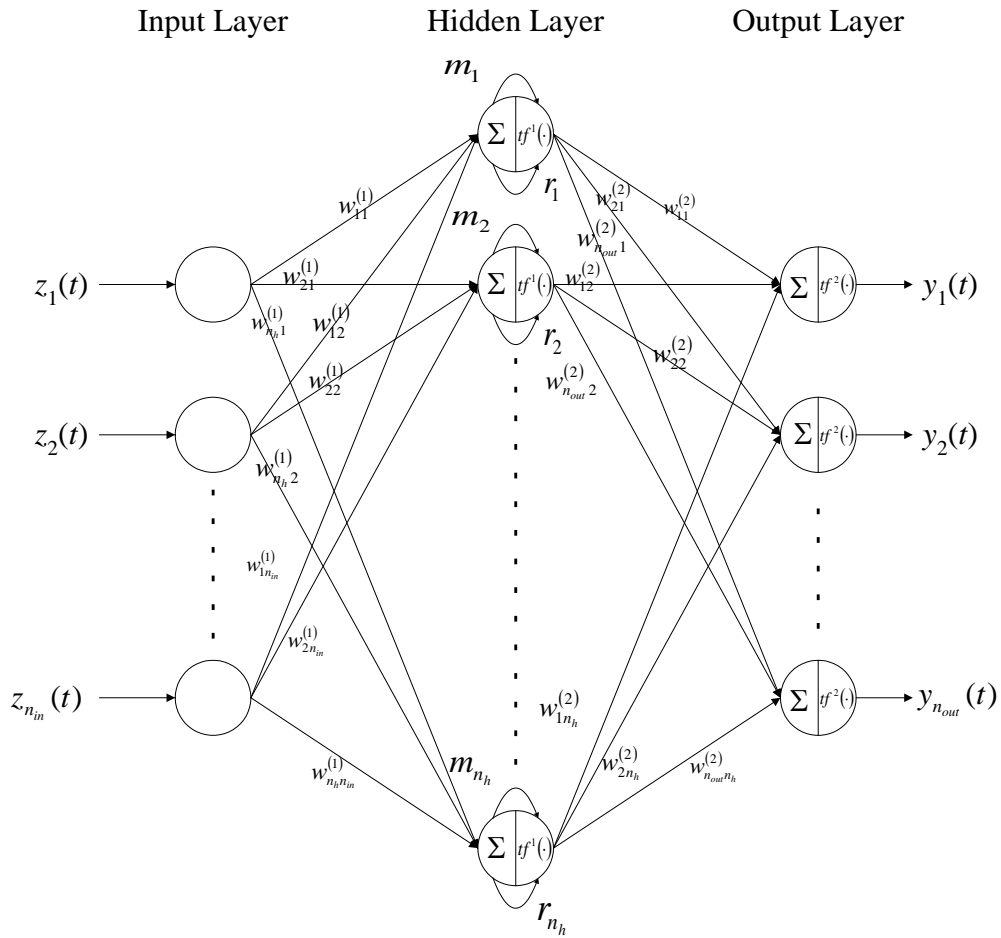


Fig. 3. Proposed input-dependent neural network.

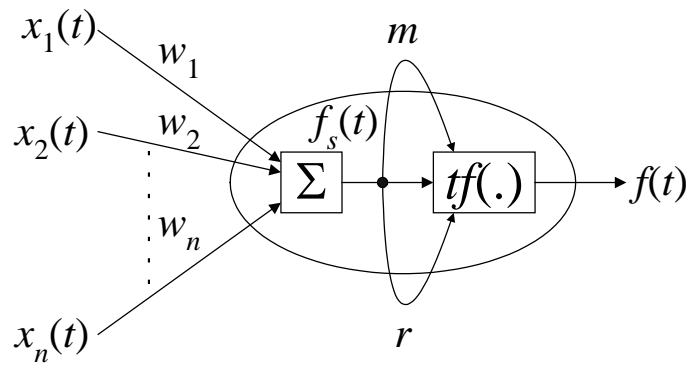


Fig. 4. Proposed neuron.

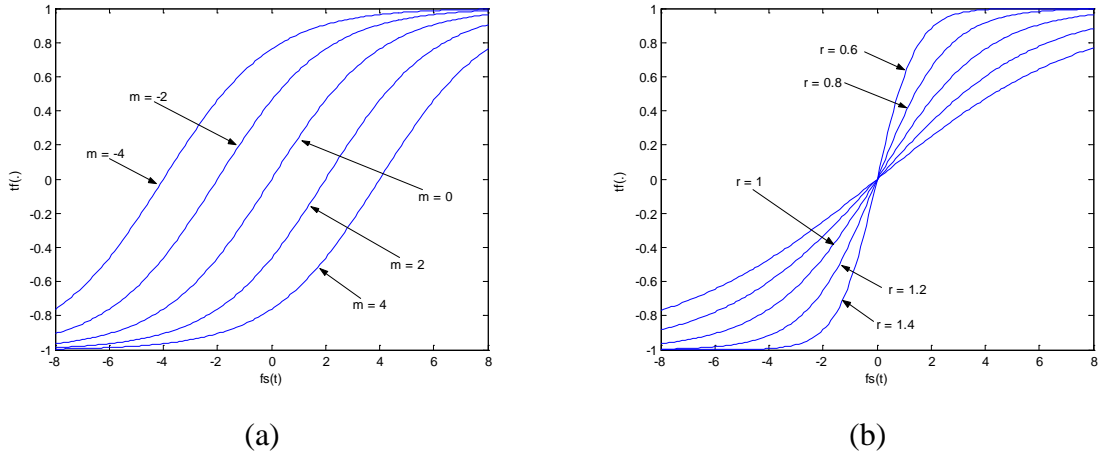


Fig. 5 The shape of the activation function: (a) under different values of m , (b) under different values of r .

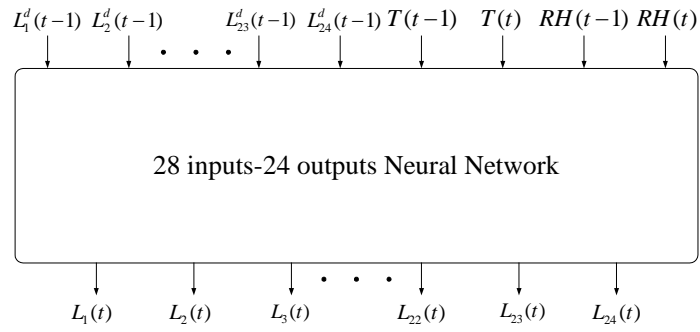


Fig. 6 Proposed neural network based short-term load forecaster.

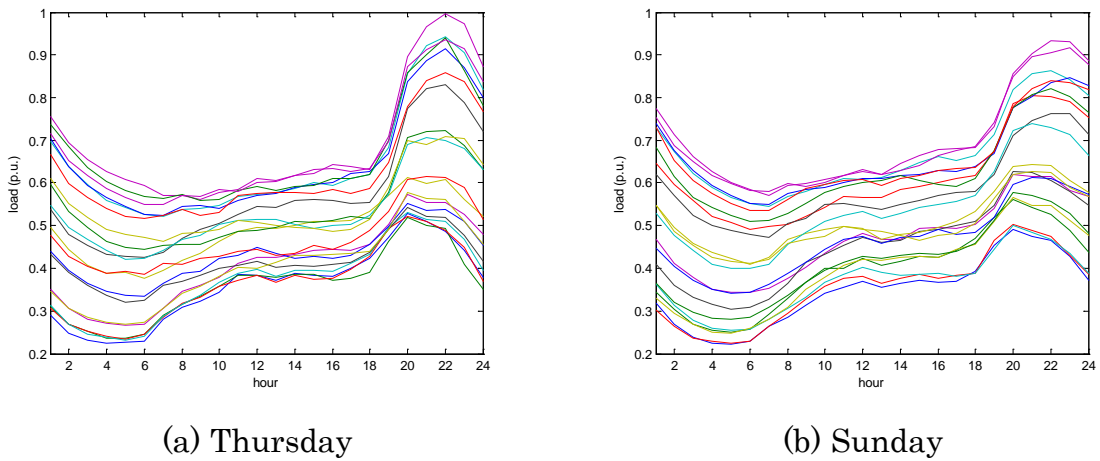


Fig. 7 The load patterns for every Thursday and Sunday from March 20 to August 6.

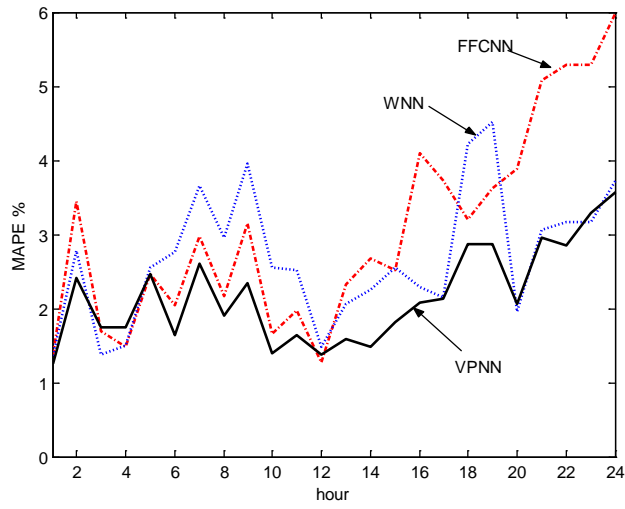


Fig. 8 Comparison of average forecasted load errors (MAPE) for Thursday.

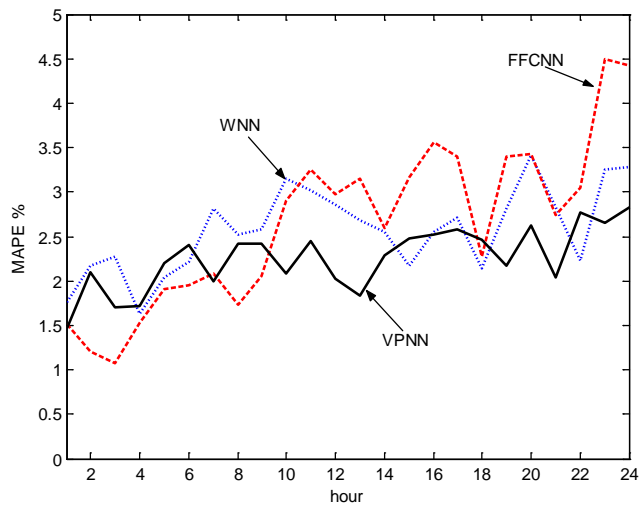


Fig. 9 Comparison of average forecasted load errors (MAPE) for Sunday.

















| Digits or Characters | Strokes | Digits or Characters | Strokes |
|----------------------|---|----------------------|---|
| 0(a) |  | 6 |  |
| 0(b) |  | 7 |  |
| 1 |  | 8(a) |  |
| 2 |  | 8(b) |  |
| 3 |  | 9 |  |
| 4 |  | Backspace |  |
| 5(a) |  | Carriage Return |  |
| 5(b) |  | Space |  |

Fig. 10. Graffiti digits and characters (with the dot indicating the starting point of the graffiti).

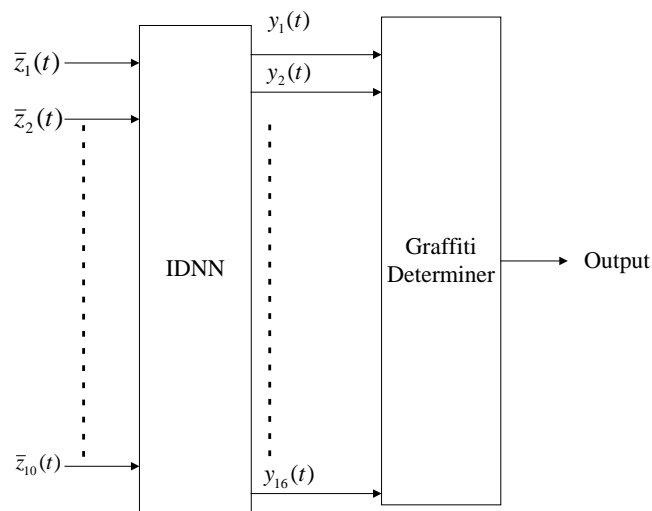


Fig. 11. Architecture of the hand-written graffiti recognizer.

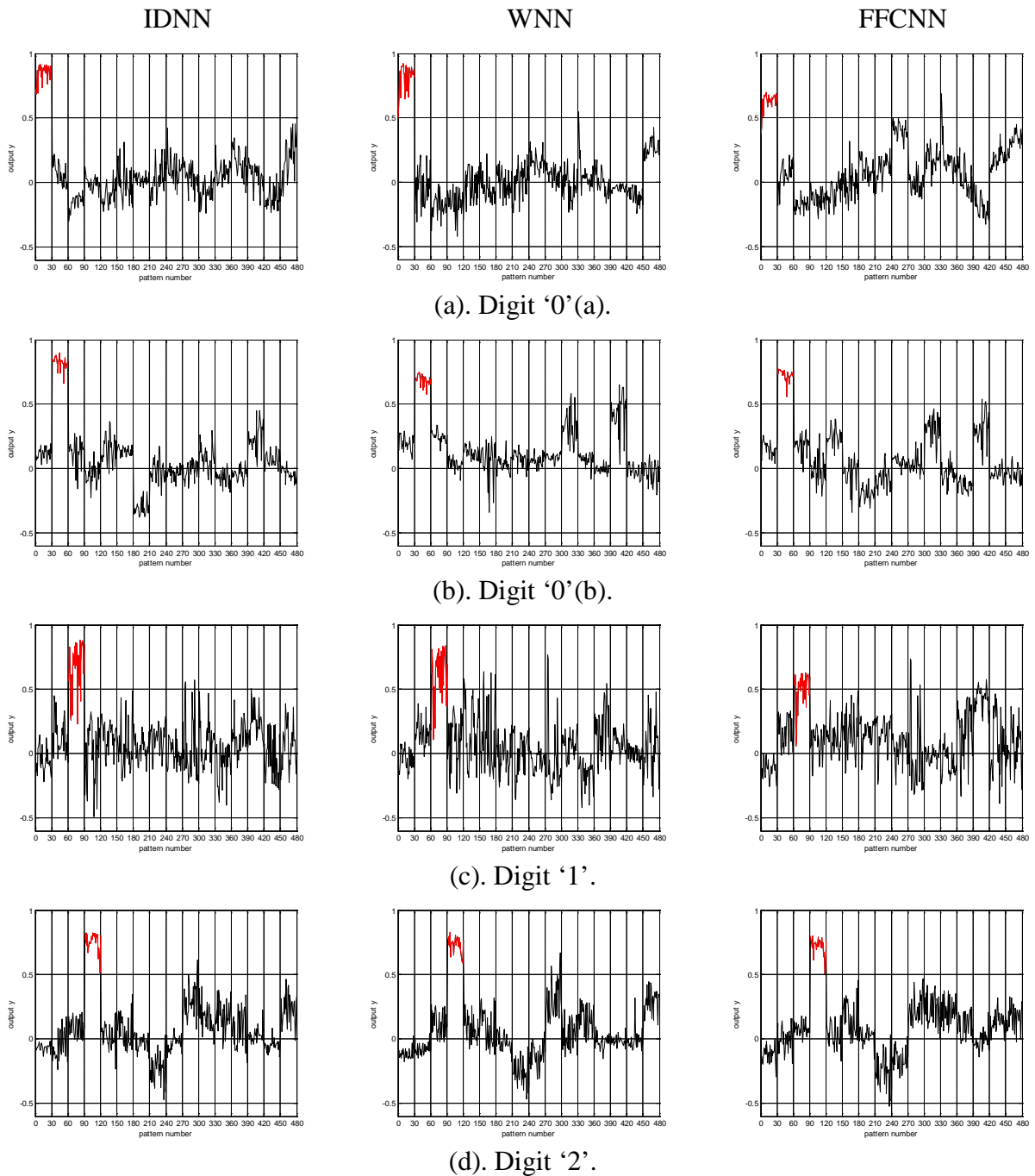
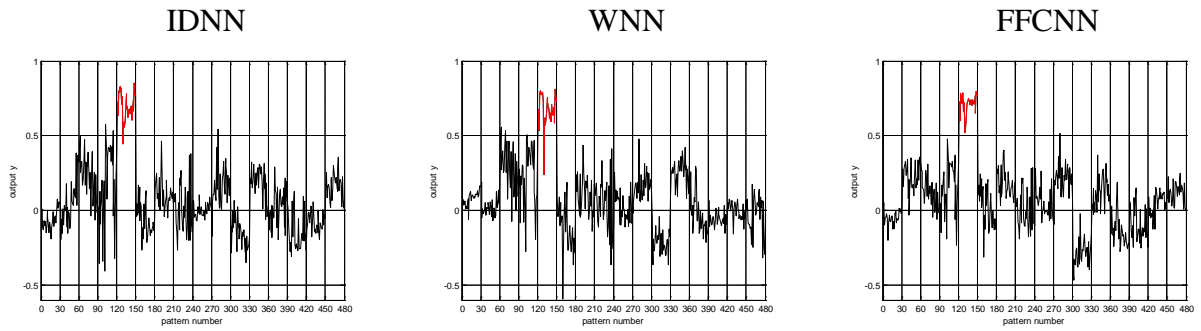
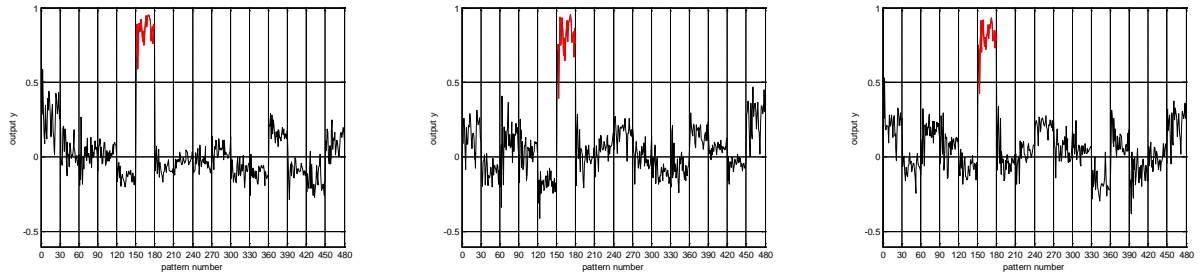


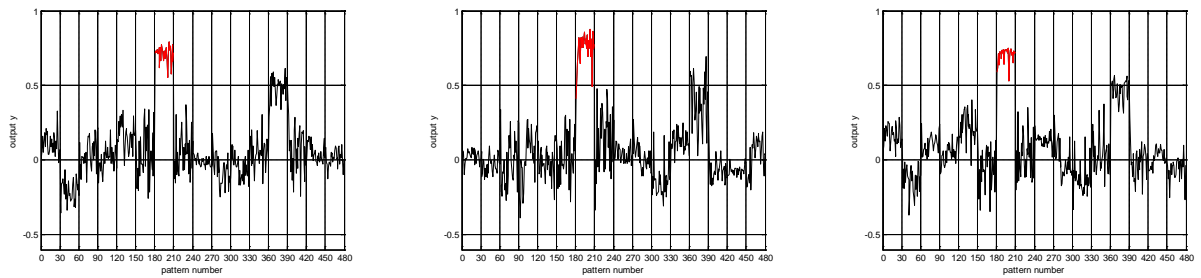
Fig. 12 The output values of the IDNN, WNN, and FFCNN for the 480 (30 for each type) testing graffiti patterns.



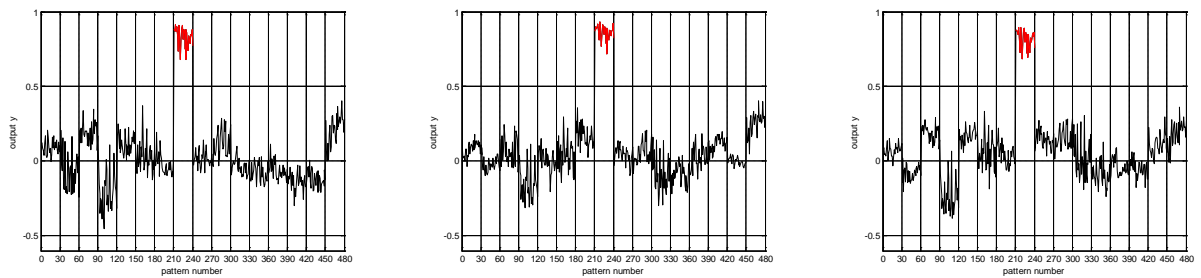
(e). Digit '3'.



(f). Digit '4'.



(g). Digit '5'(a).



(h). Digit '5'(b).

Fig. 12 (continued.) The output values of the IDNN, WNN, and FFCNN for the 480 (30 for each type) testing graffiti patterns.

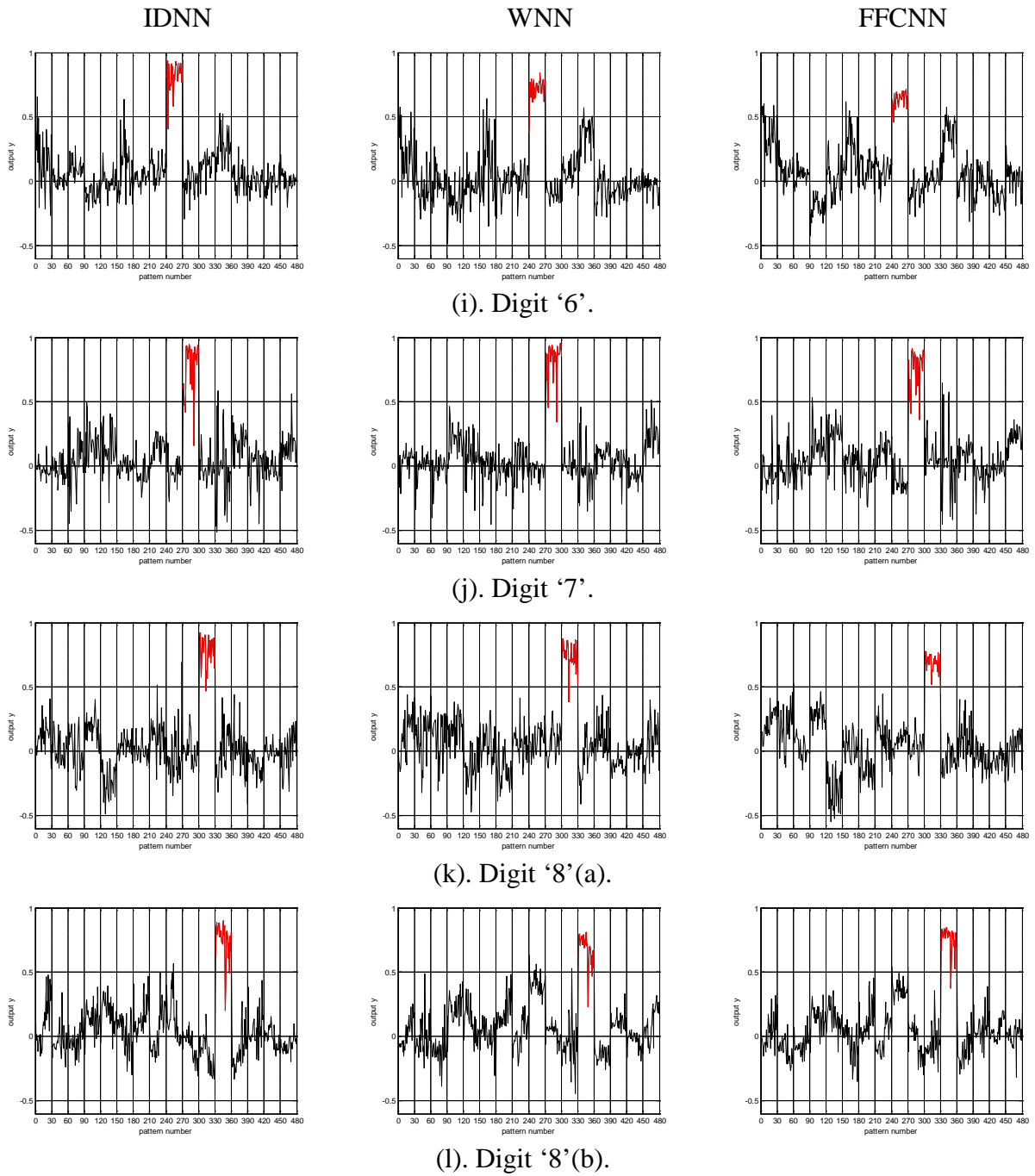


Fig. 12 (*continued.*) The output values of the IDNN, WNN, and FFCNN for the 480 (30 for each type) testing graffiti patterns.

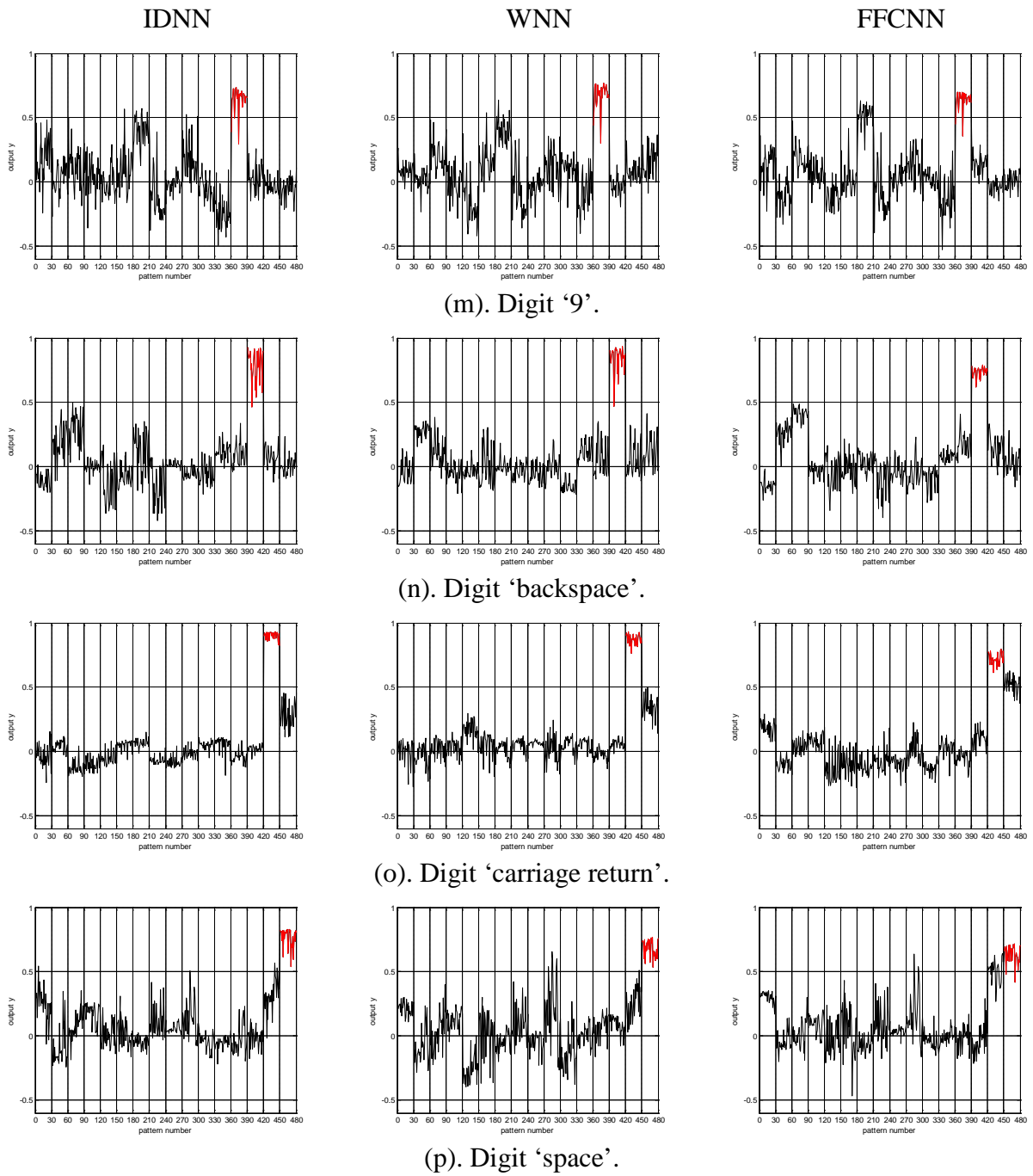


Fig. 12 (*continued.*) The output values of the IDNN, WNN, and FFCNN for the 480 (30 for each type) testing graffiti patterns.

```

Procedure of the RCGA
begin
     $\tau \rightarrow 0$  //  $\tau$ : iteration number
    Initialize  $P(\tau)$  //  $P(\tau)$ : population for iteration  $\tau$ 
    Evaluate  $f(P(\tau))$  //  $f(P(\tau))$ : fitness function
    while (not termination condition) do
        begin
             $\tau \rightarrow \tau + 1$ 
            Perform selection operation
            Determine the number of crossover based on  $p_c$ 
            Select 2 parents  $\mathbf{p}_1$  and  $\mathbf{p}_2$  from  $P(\tau-1)$ 
            Perform crossover operation
            Four chromosomes will be generated
            Select the best 2 offspring in terms of the fitness value
            Perform mutation operation for the whole population based on  $p_m$ 
            Reproduce a new  $P(\tau)$ 
            Evaluate  $f(P(\tau))$ 
        end
    end

```

Fig.13. The procedure of the RCGA

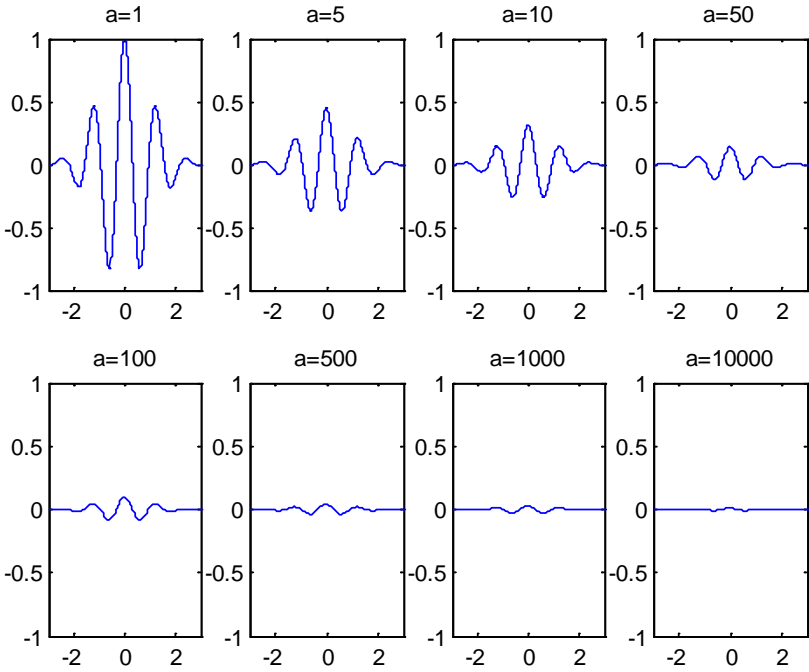


Fig. 14. A Morlet wavelet dilated by various values of the parameter a (x-axis: x , y-axis: $\psi_{a,0}(x)$.)

| | | | $n_h = 20$ | $n_h = 22$ | $n_h = 24$ | $n_h = 26$ | $n_h = 28$ | $n_h = 30$ |
|-----------|-------------|---------|------------|------------|----------------|----------------|------------|------------|
| IDNN | n_{para} | | 1106 | 1214 | 1322 | 1430 | 1530 | 1646 |
| | Training | Ave. | 3.0085% | 2.9412% | 2.9088% | 2.8304% | 2.8681% | 2.9321% |
| | | Best | 2.8021% | 2.7601% | 2.7561% | 2.7153% | 2.7190% | 2.7186% |
| | Forecasting | Ave. | 2.5011% | 2.4572% | 2.4102% | 2.2345% | 2.3861% | 2.4594% |
| Best | | 2.3830% | 2.3136% | 2.3036% | 2.1151% | 2.1200% | 2.2993% | |
| WNN [41] | n_{para} | | 1060 | 1166 | 1272 | 1378 | 1484 | 1590 |
| | Training | Ave. | 3.3026% | 3.2684% | 3.2653% | 3.1449% | 3.2989% | 3.3710% |
| | | Best | 2.8715% | 2.7417% | 2.7474% | 2.6143% | 2.7795% | 3.0313% |
| | Forecasting | Mean | 2.8803% | 2.8724% | 2.8734% | 2.6902% | 2.8785% | 2.9711% |
| Best | | 2.6711% | 2.6462% | 2.5376% | 2.5571% | 2.6360% | 2.7148% | |
| FFCNN [3] | n_{para} | | 1084 | 1190 | 1296 | 1402 | 1508 | 1614 |
| | Training | Ave. | 3.5511% | 3.4862% | 3.4431% | 3.4417% | 3.5006% | 3.4715% |
| | | Best | 3.4810% | 3.4394% | 3.3978% | 3.3674% | 3.4615% | 3.4033% |
| | Forecasting | Ave. | 3.1982% | 3.1448% | 3.0782% | 3.0565% | 3.1805% | 3.0124% |
| Best | | 3.1211% | 3.0440% | 2.9429% | 2.9386% | 3.1109% | 2.9565% | |

Table I. Results of the proposed IDNN, WNN and FFCNN for Thursday (Ave.: Average).

| | | | $n_h = 20$ | $n_h = 22$ | $n_h = 24$ | $n_h = 26$ | $n_h = 28$ | $n_h = 30$ |
|-----------|-------------|---------|------------|----------------|----------------|------------|------------|------------|
| IDNN | n_{para} | | 1106 | 1214 | 1322 | 1430 | 1530 | 1646 |
| | Training | Ave. | 2.8008% | 2.7632% | 2.7381% | 2.7820% | 2.8291% | 2.8309% |
| | | Best | 2.5691% | 2.5455% | 2.5326% | 2.6262% | 2.6681% | 2.6512% |
| | Forecasting | Ave. | 2.3121% | 2.2923% | 2.2609% | 2.3081% | 2.3465% | 2.3503% |
| Best | | 2.2654% | 2.2200% | 2.2191% | 2.2716% | 2.3002% | 2.2964% | |
| WNN [41] | n_{para} | | 1060 | 1166 | 1272 | 1378 | 1484 | 1590 |
| | Training | Ave. | 3.2867% | 3.1709% | 3.1419% | 3.1688% | 3.1831% | 3.2606% |
| | | Best | 3.1981% | 3.0430% | 2.9886% | 3.0240% | 3.0187% | 3.1423% |
| | Forecasting | Mean | 2.8633% | 2.6197% | 2.5706% | 2.6421% | 2.7287% | 2.8502% |
| Best | | 2.8411% | 2.6552% | 2.5533% | 2.6033% | 2.6751% | 2.8364% | |
| FFCNN [3] | n_{para} | | 1084 | 1190 | 1296 | 1402 | 1508 | 1614 |
| | Training | Ave. | 3.3677% | 3.3106% | 3.2765% | 3.2912% | 3.3211% | 3.3286% |
| | | Best | 3.2081% | 3.1483% | 3.1101% | 3.1366% | 3.1251% | 3.1680% |
| | Forecasting | Ave. | 2.9066% | 2.7329% | 2.6620% | 2.7199% | 2.7406% | 2.8734% |
| Best | | 2.8608% | 2.6884% | 2.6288% | 2.6756% | 2.7002% | 2.8106% | |

Table II. Results of the proposed IDNN, WNN and FFCNN for Sunday (Ave.: Average).

| | IDNN | | WNN [41] | | FFCNN [3] | |
|-------------|---------------------|------------------------|---------------------|------------------------|---------------------|------------------------|
| | Ave. training error | Ave. forecasting error | Ave. training error | Ave. forecasting error | Ave. training error | Ave. forecasting error |
| Mon | 2.9096% | 2.5012% | 3.2022% | 2.7095% | 3.2970% | 2.7881% |
| Tue | 2.8008% | 2.0231% | 3.0686% | 2.2908% | 3.3042% | 2.4530% |
| Wed | 2.7801% | 2.1076% | 3.1012% | 2.4121% | 3.4188% | 2.5804% |
| Thu | 2.8304% | 2.2345% | 3.1449% | 2.6902% | 3.4417% | 3.0565% |
| Fri | 2.6033% | 1.9911% | 3.0581% | 2.1026% | 3.2941% | 2.2300% |
| Sat | 2.8812% | 2.2610% | 3.1817% | 2.6010% | 3.3054% | 2.6612% |
| Sun | 2.7381% | 2.2609% | 3.1419% | 2.5706% | 3.2765% | 2.6620% |
| Ave: | 2.7919% | 2.1971% | 3.1284% | 2.4824% | 3.3340% | 2.6330% |

Table III. Average training and forecasting errors of the proposed IDNN, WNN, and FFCNN for Monday to Sunday.

| | | | $n_h = 16$ | $n_h = 18$ | $n_h = 20$ | $n_h = 22$ | $n_h = 24$ | $n_h = 26$ |
|---------------|---------------|---------------|------------|------------|------------|---------------|---------------|------------|
| IDNN | n_{para} | | 464 | 520 | 576 | 632 | 688 | 744 |
| | Ave. training | MSE | 0.0271 | 0.0264 | 0.0251 | 0.0223 | 0.0243 | 0.0256 |
| | | Accuracy rate | 94.10% | 94.88% | 95.10% | 95.83% | 95.50% | 95.00% |
| | Best training | MSE | 0.0241 | 0.0238 | 0.0225 | 0.0173 | 0.0208 | 0.0233 |
| | | Accuracy rate | 95.17% | 95.75% | 96.00% | 96.75% | 96.62% | 95.81% |
| | WNN [41] | n_{para} | | 448 | 504 | 560 | 616 | 672 |
| Ave. training | | MSE | 0.0355 | 0.0349 | 0.0321 | 0.0316 | 0.0309 | 0.0314 |
| | | Accuracy rate | 92.34% | 92.35% | 92.42% | 93.10% | 93.23% | 93.20% |
| Best training | | MSE | 0.0340 | 0.0315 | 0.0292 | 0.0280 | 0.0278 | 0.0279 |
| | | Accuracy rate | 93.19% | 93.31% | 93.81% | 94.00% | 94.13% | 94.00% |
| FFCNN [3] | | n_{para} | | 448 | 502 | 556 | 610 | 664 |
| | Ave. training | MSE | 0.0409 | 0.0393 | 0.0385 | 0.0380 | 0.0360 | 0.0368 |
| | | Accuracy rate | 89.94% | 90.17% | 90.46% | 90.73% | 91.50% | 91.42% |
| | Best training | MSE | 0.0400 | 0.0370 | 0.0388 | 0.0361 | 0.0326 | 0.0338 |
| | | Accuracy rate | 90.25% | 90.50% | 91.69% | 92.56% | 93.06% | 92.50% |

Table IV. Training results of the proposed IDNN, WNN, and FFCNN for hand-written graffiti recognition (Ave: average).

| | | | $n_h = 16$ | $n_h = 18$ | $n_h = 20$ | $n_h = 22$ | $n_h = 24$ | $n_h = 26$ |
|--------------|--------------|---------------|------------|------------|------------|---------------|---------------|------------|
| IDNN | n_{para} | | 464 | 520 | 576 | 632 | 688 | 744 |
| | Ave. testing | MSE | 0.0298 | 0.0288 | 0.0270 | 0.0247 | 0.0271 | 0.0279 |
| | | Accuracy rate | 94.00% | 94.58% | 95.00% | 95.92% | 95.12% | 94.88% |
| | Best testing | MSE | 0.0277 | 0.0265 | 0.0243 | 0.0194 | 0.0235 | 0.0252 |
| | | Accuracy rate | 95.40% | 95.58% | 96.25% | 96.88% | 96.50% | 96.19% |
| | WNN [41] | n_{para} | | 448 | 504 | 560 | 616 | 672 |
| Ave. testing | | MSE | 0.0380 | 0.0365 | 0.0346 | 0.0344 | 0.0329 | 0.0332 |
| | | Accuracy rate | 91.91% | 92.08% | 92.22% | 92.71% | 93.92% | 93.60% |
| Best testing | | MSE | 0.0359 | 0.0329 | 0.0320 | 0.0322 | 0.0308 | 0.0315 |
| | | Accuracy rate | 92.40% | 92.59% | 93.54% | 93.75% | 94.38% | 94.12% |
| FFCNN [3] | | n_{para} | | 448 | 502 | 556 | 610 | 664 |
| | Ave. testing | MSE | 0.0428 | 0.0410 | 0.0404 | 0.0393 | 0.0374 | 0.0386 |
| | | Accuracy rate | 89.93% | 90.07% | 90.58% | 90.68% | 91.25% | 90.69% |
| | Best testing | MSE | 0.0420 | 0.0404 | 0.0393 | 0.0388 | 0.0361 | 0.0374 |
| | | Accuracy rate | 90.42% | 90.42% | 91.67% | 92.06% | 92.92% | 92.50% |

Table V. Testing results of the proposed IDNN, WNN, and FFCNN for hand-written graffiti recognition (Ave: average).

| | IDNN | | WNN [41] | | FFCNN [3] | |
|----------------|--------------|--------------|--------------|--------------|--------------|--------------|
| | Ave. | Best | Ave. | Best | Ave. | Best |
| 0(a) | 96.40 | 95.00 | 97.00 | 94.00 | 96.05 | 97.00 |
| 0(b) | 99.40 | 99.00 | 97.80 | 100.0 | 99.40 | 100.0 |
| 1 | <u>89.10</u> | <u>98.00</u> | 73.60 | 79.00 | 61.70 | 72.00 |
| 2 | 96.40 | 98.00 | 98.15 | 99.00 | 97.65 | 99.00 |
| 3 | 94.70 | <u>97.00</u> | 93.20 | 92.00 | 87.60 | 89.00 |
| 4 | 97.05 | 97.00 | 98.25 | 99.00 | 98.50 | 99.00 |
| 5(a) | 96.95 | 93.00 | 95.60 | 97.00 | 90.90 | 92.00 |
| 5(b) | 97.20 | 97.00 | 95.50 | 94.00 | 97.40 | 98.00 |
| 6 | <u>94.15</u> | <u>94.00</u> | 85.60 | 90.00 | 85.20 | 86.00 |
| 7 | 95.20 | <u>100.0</u> | 94.90 | 94.00 | 95.40 | 95.00 |
| 8(a) | 99.40 | 99.00 | 98.40 | 99.00 | 97.45 | 99.00 |
| 8(b) | 99.40 | 100.0 | 99.70 | 100.0 | 100.0 | 100.0 |
| 9 | <u>81.20</u> | <u>86.00</u> | 76.10 | 79.00 | 78.40 | 80.00 |
| Back Space | 99.25 | 99.00 | 97.60 | 100.0 | 98.05 | 100.0 |
| Return | 99.80 | 99.00 | 99.45 | 99.00 | 98.20 | 99.00 |
| Space | <u>97.60</u> | <u>97.00</u> | 91.15 | 91.00 | 82.15 | 84.00 |
| Overall | 95.83 | 96.75 | 93.23 | 94.13 | 91.50 | 93.06 |

Table VI. Average and best recognition accuracy rate (%) for the training patterns for IDNN at $n_h = 22$, WNN at $n_h = 24$, and FFCNN at $n_h = 24$. (The underlined results show bigger than 4% difference.)

| | IDNN | | WNN [41] | | FFCNN [3] | |
|----------------|--------------|--------------|--------------|--------------|--------------|--------------|
| | Ave. | Best | Ave. | Best | Ave. | Best |
| 0(a) | 100.0 | 100.0 | 98.67 | 96.67 | 94.00 | 93.33 |
| 0(b) | 100.0 | <u>100.0</u> | 98.67 | 93.33 | 99.33 | 100.0 |
| 1 | <u>87.33</u> | <u>90.00</u> | 77.33 | 76.67 | 57.33 | 66.67 |
| 2 | <u>98.67</u> | 96.67 | 77.33 | 96.67 | 100.0 | 100.0 |
| 3 | 93.33 | <u>100.0</u> | 94.67 | 93.33 | 97.33 | 100.0 |
| 4 | 100.0 | 100.0 | 98.00 | 100.0 | 96.66 | 96.67 |
| 5(a) | 96.67 | <u>100.0</u> | 95.33 | 90.00 | 90.00 | 100.0 |
| 5(b) | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| 6 | <u>92.00</u> | 93.33 | 85.33 | 96.67 | 84.00 | 86.67 |
| 7 | 91.33 | 90.00 | 93.33 | 93.33 | 93.33 | 86.67 |
| 8(a) | 95.33 | 96.67 | 92.67 | 93.33 | 94.67 | 96.67 |
| 8(b) | 90.67 | 93.33 | 91.33 | 90.00 | 95.33 | 96.67 |
| 9 | <u>90.67</u> | 90.00 | 86.00 | 90.00 | 83.33 | 90.00 |
| Back Space | 99.33 | 100.0 | 100.0 | 100.0 | 98.67 | 100.0 |
| Return | 100 | 100.0 | 98.67 | 100.0 | 94.67 | 96.67 |
| Space | <u>99.33</u> | 100.0 | 95.33 | 96.67 | 80.67 | 76.67 |
| Overall | 95.92 | 96.87 | 93.92 | 94.38 | 91.25 | 92.92 |

Table VII. Average and best recognition accuracy rate (%) for the testing patterns for IDNN at $n_h = 22$, WNN at $n_h = 24$, and FFCNN at $n_h = 24$. (The underlined results show bigger than 4% difference.)