A Pattern Recognition and Classification Network for Digitized-Freehand Characters

Moses E. Ekpenyong
Department of Mathematics, Statistics and Computer Science
University of Uyo, P.M.B. 1017, 520001, Uyo, 520001, Uyo, Akwa Ibom State, Nigeria.
ekpenyong_moses@yahoo.com, (+234)8037933961

Abstract

*Conventional recognition approaches do not permit noisy patterns and significant information cannot be detected from incomplete data. This paper designs an Artificial Neural Network (ANN) that can recognize digitized-freehand characters. The characters considered include (i) letters A-Z (ii) digits 0-9 and (iii) symbols or special characters (+, -, *, /, =, (,), ^ and %). We use the competitive-learning approach to learn and recognize the digital writings. The designed network has a Graphic User Interface (GUI) where the user draws the desired input pattern in a drawing area with the help of a mouse. The input pattern is then digitized by fitting the resulting character into a 6×8 pixel grid. The character is finally trained before recognition. Implementation shows that our system can recognize noisy patterns and incomplete inputs. While correctly written inputs could learn faster, incomplete inputs took sometime to learn. The efficiency of this design is directly proportional to the number of training sets for each pattern. The design is also adaptable to other pattern classifiers.*

Keywords: *ANN, Competitive Learning, Kohonen Network*

1. Introduction

The goal of pattern recognition is to apply a set of example solutions to certain problems to infer an underlying regularity, which subsequently can be used to solve new problem instances, [1]. Examples are hand-written digit recognition, medical image screening and finger print identification. One central issue in any pattern recognition application is that of generalization, i.e. the performance of the trained model when applied to previously unseen data.

[2] uses a superficial model to train a perceptron neural network to recognize digits 0-9 by pattern classification. They implement a single-layer perceptron by applying the "Perceptron learning rule" using bipolar inputs with a typewritten modality. Input to their design is a text file from which the user builds a matrix of desired digits. The output of the design is the classification results. Their design is restrictive and cannot classify noisy patterns and incomplete inputs.

[3] reports on a classifier design project for recognizing typing digits. He follows a conventional classifier design process discussed in [4] to derive two classifiers using decision tree and nearest neighbour methods.

The construction of locally linear generative models with a collection of pixel-based digital images used in capturing different writing styles is seen in [5]. Their research classifies new images by evaluating their log-likelihood under each model.

Feature extraction is an essential step towards a good classifier. An automated reading system like a human reader should meet two requirements. It should have omni-writing capabilities to recognize different handwritings and a mono-writing capability that takes into account the potential fantasy of each writer. Hence, learning machines requires sophisticated and highly adaptable pattern recognition algorithms to enable them read hand-written texts. The machine also needs to manage in general, the various levels of interpretations (i.e. from the graphical level up to the lexical and syntactical levels). The human expertise in managing these interpretation levels depends on some abilities of learning the current handwriting. Current recognition systems do not possess these learning abilities. The recognition is considered to be a pure omni-writer problem, [6]. Recognition systems try to recognize handwritten words or letters, one independently from the other in a sequential manner, [7], [8].

Two approaches are used to perform handwritten cursive words recognition. The first is the analytical, a data-driven bottom-up approach where letters are recognized before a lexical analysis is performed, [9], [10]. Here to offset the letter recognition problem before recognition, several segmentation hypothesis must be managed making the letter recognition modules more complex since it must be able to reject the bad segmentation hypothesis. However, the final decision can only be taken by the lexical verification module. The second approach, is the holistic, a top-down approach with verification. In this approach, the letters segmentation is offset by recognizing a whole word and selecting word candidates from a lexicon. This approach leans either on the detection of holistic features in the word, [11], [12] or on the verification that some letters or parts of the letters are present at some word positions [13]. The first approach is well adapted to word recognition belonging to a large lexicon or even without a lexicon. The second approach is rather well adapted to limited lexicon applications. These two approaches could be combined to improve recognition, [14]. Some recent studies try to accommodate the problem of handwriting variability by clustering handwritings into families of handwriting styles, [15]. The recognizers are then trained for each specific family, but an intermediary-style choice is required to select the fitted recognizer, before the recognition phase, leaning itself on problem-specific recognition schemes.

The writer's invariants concept, which defines a set of similar patterns automatically extracted from handwriting segmentations is presented in [16]. They illustrate how this concept allows for the derivation of new contextual graphical knowledge that can be used to adapt the recognition task to a particular handwriting and allow for robust decisions making when neither simple lexical nor syntactical rules can be used. [6] explains how the recognition system can adapt itself to the current handwriting for recognition by exploiting the graphical context defined by the writer's invariants. They justify the need for an open multi-agent architecture to support the implementation of such an adaptation principle. The proposed platform allows for the plugging of expert treatments dedicated to handwriting analysis.

This paper implements an ANN that can recognize and classify freehand characters (letters, digits and symbols) through digitization. It applies the competitive learning approach and allow training cycles, where the input pattern is learned by the recognition system. To fully train the network, the system should undergo at least three cycles. The learned data can be stored in file(s) and used for the recognition of subsequent inputs. The recognized pattern is then classified and the pattern matching probabilities for each character displayed.

The justification of this research lies in the ability of our network to recognize and classify noisy and incomplete inputs/patterns, which conventional recognition approaches do not permit. The approach adopted by our network could match noisy and incomplete inputs, outputting the degree (in percentage) of how close or far that pattern is to learned characters. The character with the highest percentage is selected as the best matched pattern. The design could also be transformed into a more sophisticated network for noisy/incomplete detection.

2. Design components and methods

Various learning rules that can cause a network to learn have been applied in the field of neural networks. Some of these rules include:

(i) Perceptron Rule

The perceptron is a network in which the neuron unit calculates the linear combination of its real-valued or Boolean inputs and passes it through a Threshold activation function:

$$\Theta = Threshold(\sum_{i=0} d_{w_i x_i}) \quad (1)$$

where x_i are the components of the input $x_e = (x_{e1}, x_{e2}, ..., x_{el})$ from the set $\{(x_e, y_e)\}_{e=1} N$.

Threshold is the activation function defined as Threshold(s) = 1 if s>0 and -1 otherwise. The perceptron rule is a sequential learning procedure for updating the weights and is given by:

$$w_i = w_i + \eta(y_e - \Theta_e)x_{ie} \quad (2)$$

where

y_e is the desired output,

Θ_e is the output generated by the perceptron,

w_i is the weight associated with the ith connection,

η is the learning rate parameter.

(ii) Hebb's Rule

The Hebb's rule states that if a neuron receives an input from another neuron, and both are highly active, i.e. mathematically have the same sign, the weights between the neurons should be strengthened.

Hebbian rule, which are used mainly to train auto-associative networks, can also be used to train pattern associating networks. Hebbian rule receive strong support from neurophysiological studies that demonstrate synaptic weight changes associated with pre-synaptic and post-synaptic activity, [17].

(iii) Hopfield Network

The Hopfield network is a recurrent neural network in which all connections are symmetric. This network guarantees that its dynamics will converge. If the connections are trained using Hebbian learning, then the Hopfield network can perform as robust content-addressable memory, resistant to connection alteration. The Hopfield network is particularly useful in image detection, [18].

(iv) Delta Rule

The delta rule is the gradient descent learning rule for updating the weights of the artificial neurons in a single-layer perceptron. For a neuron j with activation function g(x), the delta rule for j's ith weight w_{ji} is given by:

$$\Delta w_{ji} = \alpha(t_j - y_i)g'(h_j)x_i \qquad (3)$$

where

α is a constant called learning rate

g(x) is the neuron's activation function

t_j is the target output

y_i is the actual output

x_i is the ith input

It holds that $h_j = \sum x_i w_{ji}$ and $y_i = g(h_j)$.

The delta rule is commonly stated in a simplified form for a perceptron with a linear activation function thus:

$$\Delta w_{ji} = \alpha(t_j - y_i)x_i \qquad (4)$$

(iv) Competitive Learning

The competitive rule makes use of a two-layered network; i.e. the input layer and the output layer. Each neuron in the input layer is connected to all the neurons in the output layer. The network adjusts weights linking an input neuron to an output neuron according to the algorithm:

$$\begin{cases} \Delta w_{ij} = \{ n (w^*_{ij} - w_{ij}) \mu \} \\ \\ x_{ij} = x^*_{ij} \; otherwise \end{cases} \qquad (5)$$

The above algorithm means that: if a particular input neuron is active (represented by the black dots in Fig. 5.), the weights connecting the input to the desired output should be increased by an amount equal to:

$$\Delta w_{ij} = \{ n (w^{*}_{ij} - w_{ij}) \mu \} \quad (6)$$

where

w_{ij} = weight connecting the neuron to the desired output.

w^{*}_{ij} = desired weight that should connect the input neuron to the desired output neuron.

n = learning rate, usually $0 \le n \le 1$

μ = state of input (1 = on, 0 = off)

else,

the weight should not be increased $(\Delta w_{ij} = 0)$

x_{ij} = any of the links

x^{*}_{ij} = link connecting the input neuron to the desired output neuron.

This implies that if an input neuron is activated, then the weight connecting it to the desired output neuron should be adjusted, and for all other links, no adjustments should be made. The weights connecting the outputs to each other are also adjusted such that only the loop is increased while the other weights are decreased.

In this paper we implement the competitive learning rule. Our network permits three training cycles (i.e. the maximum allowable weights defined by the user). Thus for the first training cycle an incremental amount of 1 is produced as computed below using equation (5):

$$\frac{1}{3-0}(3-0).1 = \frac{1}{3}.3 = 1$$

For an inactive neuron, $\mu = 0$. Hence no increment takes place.

During use, in order to recognize an input character, the weights connecting each output neuron to all the input neurons are summed for all output neurons and the output with the highest weight-sum wins and is activated. The network derives its name here since all the output neurons compete among themselves for activation. As a result, only one output is selected at a time. This phenomenon is also known as a winner takes all network. Our choice for this type of network stems from the fact that we can make use of any number of input and output neurons and no back-propagation is involved.

3. System Model

Our design as earlier mentioned has a GUI with a drawing area of 6×8 pixel grid box. The system has an underlying neural network engine, which processes the input from the interface window and produces as output, the recognized character. During learning, the neural network engine will adjust weights linking the input neurons to the output neurons. Thus a simple model to illustrate the system is shown in Fig. 3.

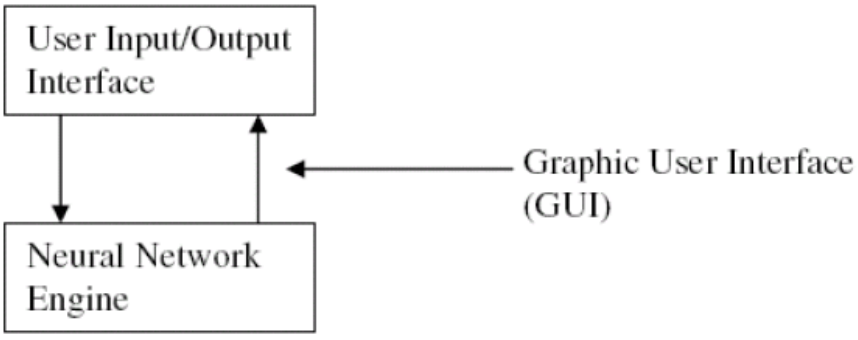

Fig. 3. Our ANN system model

The system operates in two modes: the learn mode and the use mode. During the learn mode, the system trains the underlying neural network with data supplied by the user. The trained data is saved to a file. Several training files are allowed. During the use mode, the system identifies the inputs, classifying them either as an alphabet A-Z, digit 0-9 or a symbol: +, -, *, /, =, (,), ^, and %. The character inventory module in the program could be expanded to accommodate other input symbols and characters.

Neural Network (NN) Engine Architecture

The NN engine consists of a NN based on a competitive learning architecture. It is two-layered. The input layer contains 48 neurons and the output layer contains 26 neurons. The architecture has no hidden layer. This engine processes the input data and produces the output. The engine is also responsible for weights adjustment linking the input neurons to the output neurons. The NN architecture is a simple two-layered Kohonen network and is illustrated in Fig. 4.

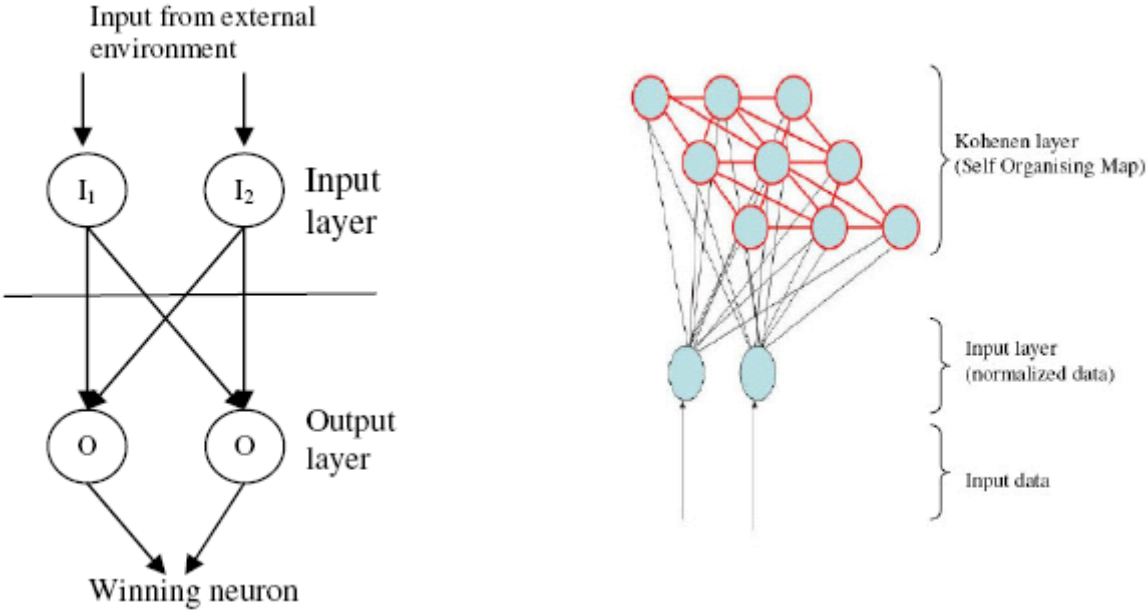

Fig. 4. (a). A simple Kohonen Network (b) Kohonen Neural Network with SOM

The Kohonen network (SOM) is able to organize the neuron values into separate groups, keeping similar values together in order to form "clusters'. The NN is unsupervised; i.e. it is able to group the input data into clusters without any pre-held information concerning how these clusters will eventually look like. Each output node contains a vector of randomly set numbers which has the same length as the input vector. During training, every input vector is connected with every node on the output layer (see Fig. 4(b).)

Applications

Kohonen's Self organizing feature maps algorithm have been applied to cortical maps to detect orientation and direction preference in cats, ferrets and monkeys, [20].

Detection of certain behaviours and inferring meaning from them has already been attempted by the London Underground train stations that tested CCTV "smart software" designed to spot any abnormal behaviour, [19]. The strategy employed by this software, though does not attempt any complex modelling of behaviour, simply compares the CCTV real-time footage with an image of the empty station and alerts staff members of any suspicious object "loitering" in the same position for an abnormal amount of time.

Monitoring behaviour via computers has been attempted, as many airports have increased security and need for accurate measures to observe people. Systems are put in place to track airport users via the use of their air ticket. Obviously not only for check in. Some airports have to swipe

the ticket if users buy items in a duty free shop or enter the toilets. All information are recorded about the time of certain events as a way of keeping track of individuals.

Learning/Training the Network

Many training vectors are presented to the network and a winner output node is chosen based on the "Best Matching Unit" (BMU) – the node whose vector is the closest to the input data. Once a BMU is chosen, its values are adjusted to better match the input vector which effectively strengthens the ability of that node to recognize similar patterns. The surrounding units are moved closer to each BMU eventually forming a cluster of similar data. The training process is repeated until the NN has effectively categorized all input patterns.

Once the entire training set has been presented to the network, an epoch is reached. Usually the NN undergoes many epochs during its training phase and for each row of data a winning node is selected and strengthened to better represent the input set. After each epoch, some data may not be "well represented"; in this case our network selects the least likely node as the winning node. This node is trained to represent the underrepresented row by altering its weight accordingly. This way, the network avoids only representing occurring data inputs.

In this design, each input neuron will have 26 outward-weighted links to the 26 output neurons in the output layer. The output layer will consist of 26 output neurons each connected to all others by weights and one loop back to itself. Thus each output neuron will have 26 outward weighted links in addition to 48 inward weighted links from all the input neurons. A partial view of connections identifying a sample alphabet H as a class of the input neurons (black dots) is shown in Fig. 5.

The learning rate of our network is computed by making n - the learning rate, the subject in equation (6). During training, the function that adjusts the learning rate to ensure that the number of learning steps is maintained is shown below:

```
void CompeteNet::adjustLearningRate(double _wt_change){
if(round(_wt_change) == 0.0)
learning_rate = 1;
else
learning_rate = no_of_steps / _wt_change;
}
```

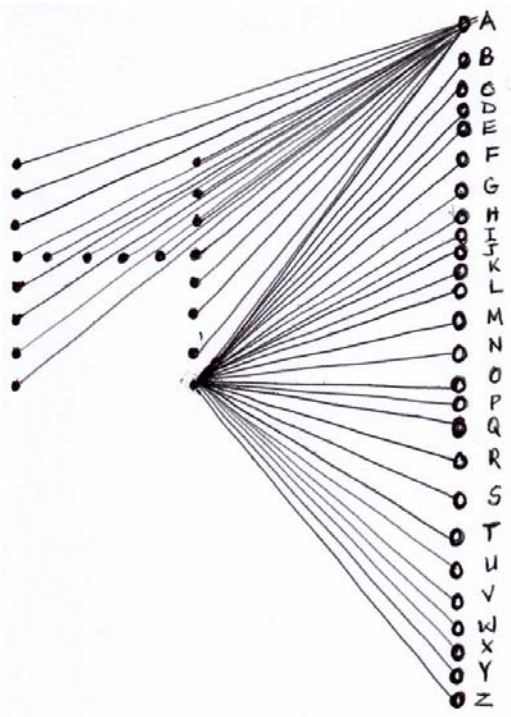

Fig. 5. The Neural Network Architecture (e.g. identification of the letter H).

The percentage learning rates for the characters are stored in a matrix and computed thus:

```
learn_rates[desired_output] = (1 – ((expected – diff) / expected)) * 100.0;
```

where

$expected = (no\_of\_active\_cells * desired\_weight);$

$diff = output[desired\_output] \rightarrow weighted\_score;$

A fully trained network must produce 100%.

The function that returns the winning neuron is implemented with the following code:

```
int CompeteNet::getWinner(){
    int win  = 0;
            int maxS = score_percent[0];// set maximum score to that of the first Oneuron
            for(int j = 0; j < oneurons; j++){// get highest score
            getProbability(j);
            if (maxS < score_percent[j]){
                    maxS = score_percent[j];
                    win  = j;
        }
    }
    return win; // return highest
     }
```

Each character has a matching score (probability), which represent how close or far the recognized character is to the input. The percentage pattern matching probability of the characters is implemented with the if statement below:

```
if(neuron_score == 0)// if total score of Oneuron is 0
    score_percent[desired_output] = 0; // its probability should be set to 0.
else{// its not zero so calculate probability in percentage
    score_percent[desired_output] = (( neuron_score * 100.0) / expected_score);
}}
```

Algorithm

The algorithm implementing our NN is given below:

(1) Start
(2) Select mode (1-Learn mode, 2-Use mode)
(3) If mode=2? then Goto (15)
(4) Call draw_image(); enable user to draw image or activate desired pixel in a grid box.
(5) Call InterfaceStatus_transfer_to_NN(); a function that causes the state of the cells to be transferred to the NN engine.
(6) Call TrainNetwork(); function that trains the network
(7) Call adjust_weight(); weight adjustment function
(8) Call SetLearning_steps(); function that sets the learning cycles
(9) Call ComputeWeightedScore(); Compute weights function
(10) GetProbability();
(11) GetWinner();
(12) Call Learn_rate_Interface(); return status report
(13) Call Output_display(); display learning rate
(14) Goto (21)
(15) Call draw_image(); enable user to draw image or activate desired pixel in a grid box.
(16) Call InterfaceStatus_transfer_to_NN(); a function that causes the state of the cells to be transferred to the NN engine.
(17) Call TrainNetwork(); function that trains the network
(18) Call Process_input(); a function to recognize the character
(19) Return WinnerWeighted_sum and MatchProbability
(20) Using WinnerWeighted_sum and MatchProbability, Call Output_display(); display resembling character and probability values.
(21) Exit? (Yes/No)
(22) If yes, then End, else, Goto (2)
(23) Stop

4. Implementation

The design of a system is incomplete without testing, implementation and documentation. In this section we test the recognition system with sample input data. The system was trained to recognize freehand written digital characters. Both complete and incomplete (inc) data were input. We observed that it took some time to learn/train incomplete and wrong inputs. Necessary corrections were made to the program and the program subjected to several trials.
Sample input/output of our recognition system are shown below:

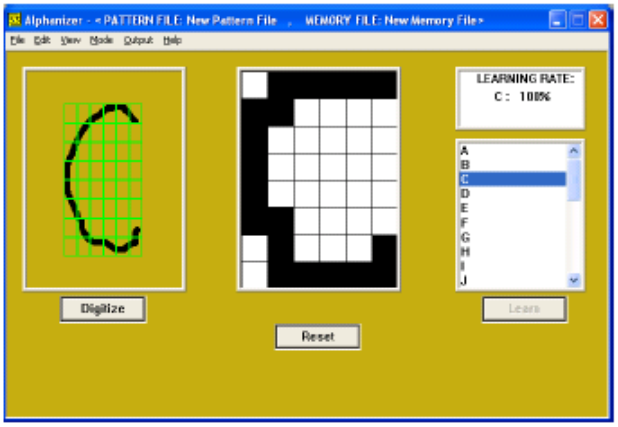

Fig. 6(a). Output-letter C after training

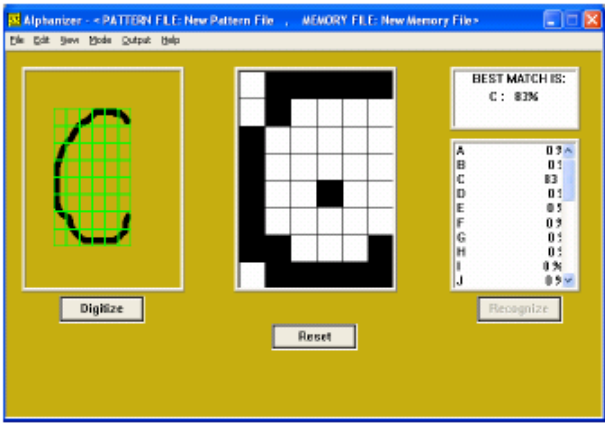

Fig.6(b). Output-letter C after recognition

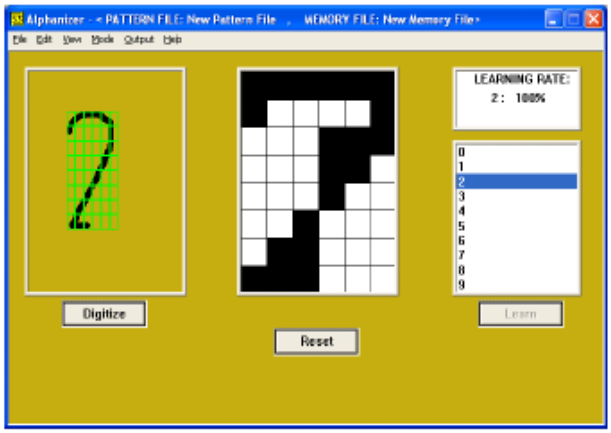

Fig.7(a).Output-Digit 2 (inc) after training

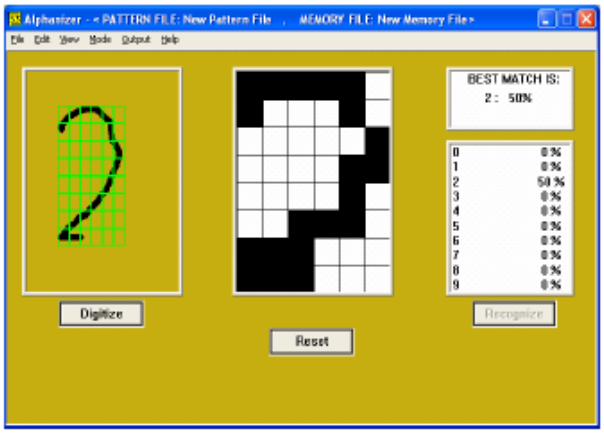

Fig.7(b).Output-Digit 2 (inc) after recognition

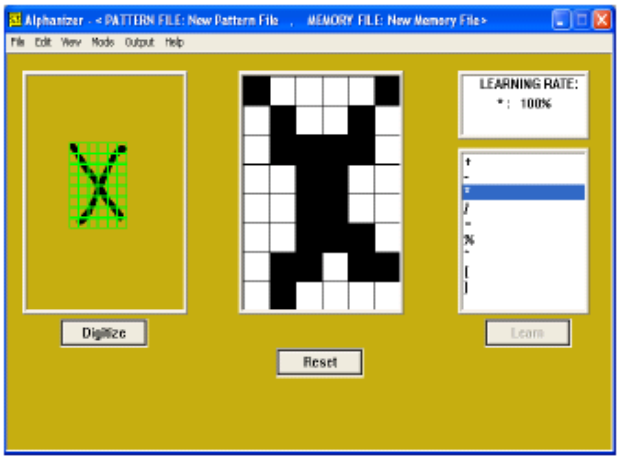

Fig.8(a). Output-Symbol * (inc) after training

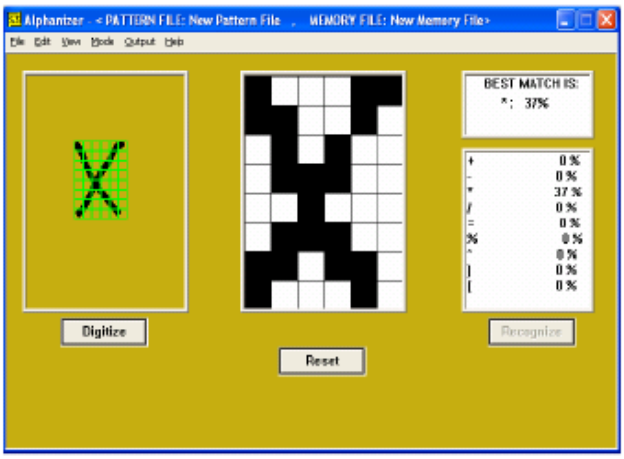

Fig.8(b). Output-Symbol * (Inc) after recognition

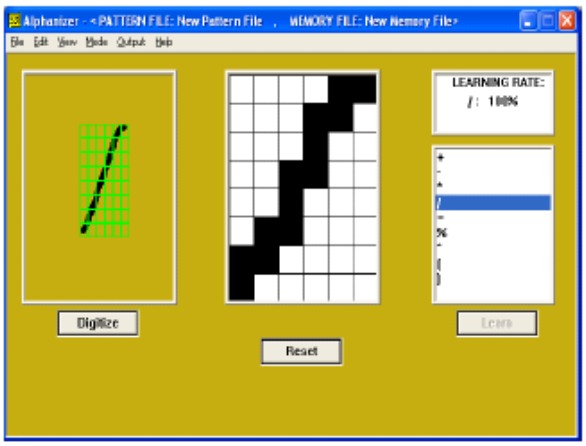

Fig. 9(a). Output- Symbol / after training

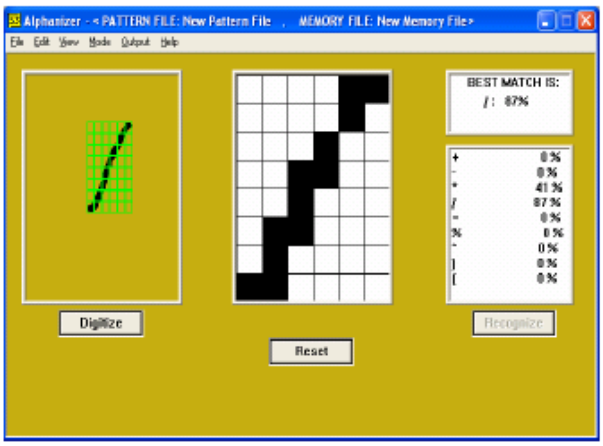

Fig. 9(b). Output-Symbol / after recognition

From the above results, we observe that the system could perfectly recognize characters C and / with pattern matching probabilities of 0.83 and 0.87 respectively (see figs. 6(b) and 9.(b)). For incomplete inputs (digit 2 and symbol *), the system fairly recognized the characters with pattern matching probabilities of 0.50 and 0.37 respectively (see figs. 7(b) and 8(b)).

5. Conclusion

Pattern recognition systems built with the NN technology are being utilized in many aspects of life to enhance efficiency and also provide better services to customers. This paper has shown how freehand characters are recognized using NNs. When compared with the ordinary system of pattern recognition process, it has the ability to tolerate noisy patterns and also recognize incomplete data. Pattern recognition systems have been found to be very useful in online applications such as banking (e.g. for signature verification), and at security checkpoints (e.g. thumb print verification). This new area of technology will be the next direction of our research. We aim at developing further sophisticated models by building on the current paper.

Although the efficiency of NN approach depends on the number of training sets, it has been discovered that it still remains a better approach to pattern recognition. Also, since the automation of pattern recognition processes are becoming increasingly important in data processing, the speed of the neural network approach hence makes it more suitable for real time applications.

6. References

[1] Bishop, C. (1999). Pattern Recognition and Feed-forward Networks. In MIT Encyclopedia of the Cognitive Sciences. R. A. Wilson and F. C. Keil (eds), MIT Press, 629-632.

[2] Ekpenyong, M. and Bello, M. (2002). A Neural Computing Approach to Pattern Recognition of Digits using Bipolar Inputs, Journal Of Natural and Applied Sciences: 79-88.

[3] Xiao, H. (2004). *Typing digit classifier*. *http://www.cs.queensu.ca/home/xiao/pr.html*.

[4] Hart, P., Duda, R., and Stork (2001). *Pattern classification*, second ed., John Wiley & Sons, Inc.

[5] Hinton, G., Revow, M. and Dayan, P. (1995). Recognizing Handwritten Digits using Mixtures of Linear Models. http://cs.utoronto.ca/~hinton/pancake.ps.gz

[6] Heutte, L., Paquet, T., Nosary, A. and Hernoux, C. (2000). Handwritten Text Recognition Using a Multiple-Agent Architecture to Adapt the Recognition Task. In: L. R. B. Schomaker and L. G. Vuurpijl (Eds.), Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition, September 11-13 2000, Amsterdam, Nijmegen: International Unipen Foundation: 413-422.

[7] Srihari, S. (1993). Recognition of handwritten and machine-printed text for postal address interpretation. Pattern Recognition Letters 14, no. 4: 291-302.

[8] Tang, Y., Lee, S. and Suen, C. (1996). Automatic document processing: a survey. Pattern Recognition 29, 12: 1931-1952.

[9] Kim, C., Govindaraju, V. (1997). A lexicon driven approach to handwriting word recognition for real-time application. IEEE PAMI 18, no. 4: 366-379.

[10] Shridhar, M., Houle, G. and Kimura, F. (1997). Handwritten word recognition using lexicon free and lexicon directed word recognition algorithms. Proc. ICDAR'97, Germany: 861-865.

[11] Bramal, P. and Higgins, C. (1995). A cursive recognition system based on human reading models. Machine Vision Application, Vol. 8: 224-231.

[12] Leroy, A. (1996). Correlation between handwriting characteristics. In Handwriting and Drawing Research: Basic Applied Issues, M. L. Simner and C. G. Leedham: 403-417.

[13] Farouz, C., Gilloux, M. and Bertille, J. (1998). Handwritten word recognition with contextual hidden markov models. Proc. 6th IWFHR, Korea: 133-142.

[14] Plessis, B., Sicsu, A., Heutte, L., Menu, E., Lecolinet, E., Debon, O. and Moreau, J. (1993). A multi-classifier combination strategy for the recognition of handwritten cursive words. Proc. ICDAR'93, Japan: 642-645.

[15] Crettez JP (1995). A set of handwriting families: style recognition. Proc. ICDAR'95, Montreal, Canada: 489-494.

[16] Nosary, A., Heutte, L, Paquet, T and Lecourtier, Y. (1990). Defining writer's invariants to adapt the recognition task. Proc. ICDAR'99, India: 765-768.

[17] Anastasio, T. (2002). Neural Network Learning. http://www.cs.rtu.lv/dssg/download/ publications/2002/Pchelkin-EROAT-2002.pdf

[18] Gurney K. (1997) An Introduction to Neural Networks. CRC Press, London.

[19] Hogan J. (2003) Smart Software linked to CCTV That can Spot Dubious Behaviour. New Scientist. July 11th . http://www.newscientist.com/

[20] Swindale, N. and Bauer, H-U. (1998). Application of Kohonen's Self-Organizing Feature Map Algorithm to Cortical Maps of Orientation and Direction Preference. In *Proceedings: Biological Sciences*, Vol. 265, No. 1398 (May 7, 1998): 827-838.

Article received: 2009-01-26