

Java Characters Recognition using Evolutionary Neural Network and Combination of Chi2 and Backpropagation Neural Network

GregoriusSatiaBudhi^{1*} and Rudy Adipranata¹

¹ Informatics Department, Industrial Technology Faculty, Petra Christian University, Surabaya, Indonesia.

*E-mail: gregorius@petra.ac.id

Abstract

Javanese language is the language used by the people on the island of Java and it has its own form of letters called Java characters. Recognition of Java characters is quite difficult because it consist of basic characters, numbers, complementary characters, and so on. In this research we developed a system to recognize Java characters and compared two methods of neural network namely evolutionary neural network and combination of Chi2 and backpropagation neural network. Input for the system is a digital image of Java characters. Before entering into the neural network, the digital image is processed by reducing noise, segmentation and thinning and feature extraction. From experimental result, evolutionary neural network has 60% average recognition accuracy, while combination of Chi2 and backpropagation network has 70% average recognition accuracy.

Keywords: Java characters recognition, Evolutionary neural network, Back-propagation neural network, Chi2.

Introduction

Javanese language is the language used by the people on the island of Java and it has its own form of letters called Java characters. Recognition of Java characters is quite difficult because it consist of basic characters, numbers, complementary characters, and so on. Several researches for recognition of Java characters have been made. Recognition using fuzzy logic with input of a 15x15 pixel of Java character image which will be recognized as one of the 20 basic Java characters (*carakan* characters) is done in [1]. The research was conducted by using MATLAB Toolbox.

The researcher noted that the results are satisfactory. Another research conducted by [2], also perform basic Java character recognition using backpropagation neural network. Implementation is also done using MATLAB toolbox. Accuracy of the recognition results is approximately 61%. Budhi and Rudy [3] also have done research comparing bidirectional associative memory, counterpropagation and evolutionary neural network for Java character recognition. Their results showed that the evolutionary neural network has a higher accuracy than the other two methods.

In this research we developed a system to recognize Java characters not only basic Java characters, but also numbers and *sandhangan* characters. We used evolutionary neural network for recognition and also we wanted to improve the accuracy of backpropagation neural network results by combine it with Chi2. We compared both results to determine which one better for Java character recognition. After being recognized, the Java characters will be converted to *Hanacaraka* font.

Input for system is a digital image of Java characters. The digital image will be processed to reduce its noise. Later, to get the character, segmentation and thinning process is done on digital image. After that, we do feature extraction to get the feature of each Java characters. The feature will become input for the neural network.

Java Characters

Java characters are differs from the commonly used Latin characters. Java characters have different shape and structure with the Latin characters. *Carakan* characters is the core of Java characters consisting of 20 syllables called Dentawyanjana, can be seen in Figure 1 [4].



Figure 1. Basic characters (*Carakan*)

For numbers in Java characters can be seen in Figure 2 [4].

1	2	3	4	5	6	7	8	9	0
---	---	---	---	---	---	---	---	---	---

Figure 2. Numbers

Sandhangan character is commonly used as complementary character, vowel or consonant that are commonly used in everyday language. *Sandhangan* can be seen in Figure 3 [4].






Sandhangan name	Java character	Description
<i>Wulu</i>		Vowel i
<i>Suku</i>		Vowel u
<i>Taling</i>		Vowel é
<i>Pepet</i>		Vowel ê
<i>Talingtarung</i>		Vowel o

Figure 3. *Sandhangan* characters

Skeletonizing

Skeletonizing or thinning is the process to get rid of the extra pixels and produces images that are more modest. The purpose of skeletonizing is made simpler image so that the image can be analyzed further in the way of its shape and suitability. Problem encountered in conducting thinning is how to determine the pixels are redundant. If we cannot determine it, the thinning process is more likely to an erosion process where erosion can cause a region is deleted. Skeleton should remain intact and have some basic properties such as [5]:

- Must consist of several thin regions, with a width of 1 pixel.
- Pixels that form the skeleton should be near the middle are of the cross section of the region.
- Skeletal pixel must be connected to each other to form several regions that are equal to the number of region in original image.

The basic idea is to determine whether a pixel can be removed just by looking at the 8 neighbors of the pixel. There are two terms that are used to determine whether a pixel can be removed or not. The first requirement is as follows:

- A pixel can be removed only if it has more than one and less than 7 neighbors.
- A pixel can be removed only if it has a counting index equal to one.
- A pixel can be removed only if at least one of its neighbors that are in the direction of 1, 3, or 5 is the pixel that indicates the background.
- A pixel can be removed only if one of its neighbors that are in the direction of 3, 5, or 7 is the pixel that indicates the background.

The second requirement is approximately the same as the first but different requirements on the last two steps [5]:

- A pixel can be removed only if at least one of its neighbors that are in the direction of 7, 1, or 3 is the pixel that indicates the background.
- A pixel can be removed only if one of its neighbors that are in the direction of 1, 5, or 7 is the pixel that indicates the background.

Evolutionary Neural Network

Evolutionary neural network (ENN) is a combination of a neural network with evolutionary algorithm. Although the neural network can be used to solve various kinds of problems, it still has some limitations. A common limitation is usually associated with network training. Backpropagation learning algorithms are often used as flexible and easy to implement had serious drawbacks, which cannot guarantee that the optimal solution is given. Another difficulty is related to selecting the optimal network topology for the neural network. Network architecture that is appropriate for certain cases more often chosen from heuristic methods, and neural network topology design is still an art than a technique. This shortcoming can be addressed using evolutionary algorithm.

Evolutionary algorithm refers to a probabilistic adaptation algorithm inspired from natural evolution. This method follows the statistical search strategies in a population of individuals, each representing a possible solution to the problem. Evolutionary algorithm divides into three main forms, namely: evolution strategies, genetic algorithms, and evolutionary programming [6].

In this research, the evolutionary algorithm used is the genetic algorithm. Genetic algorithm is an effective optimization technique that could help both the optimization of weight and selecting the network topology. In order to use genetic algorithm, first a problem must be represented as a chromosome. For example, when we want to look for a set of optimal weight of a multilayer feed forward neural network, the first step in solving this problem is the system should make the process of encoding of the network into a chromosome, can be seen in Figure 4.

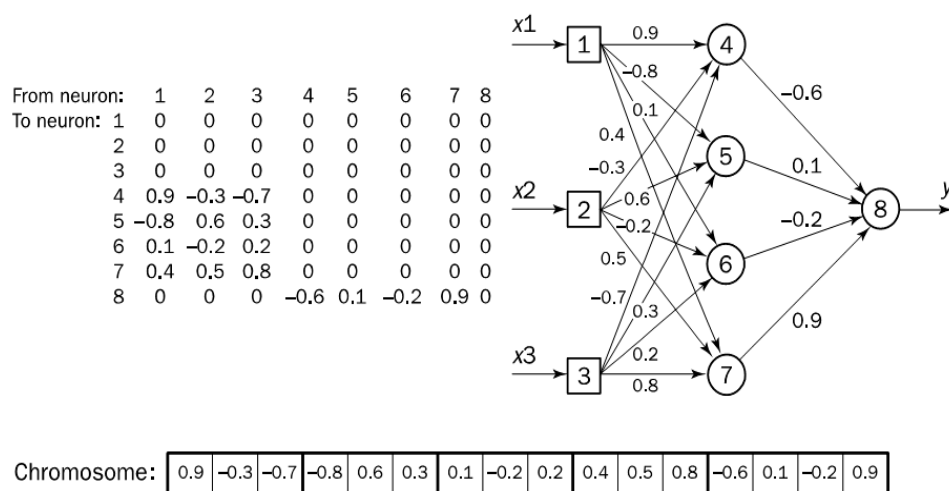


Figure 4. Encoding a network into a chromosome

The second step is to define the fitness function to evaluate the performance of the chromosome. This function must be calculated given the performance of the neural network. We can implement a simple function from squared errors. To evaluate

the fitness of the chromosomes, each chromosome weight is given to each link in the network. Training of examples collections are then presented to the network, and the number of squared errors is calculated. Small squared errors indicate that the chromosome is more fit than the other. In other words, genetic algorithm seeks to find a set amount of weight that has the smallest squared errors.

The third step is to choose the genetic operators, namely crossover and mutation. Crossover operator requires two parent chromosomes and creates a child with genetic material from both of its parent. Each gene of the child chromosome is represented by the corresponding genes of randomly selected parent. Mutation operator randomly selects a gene and replaces it with a random result between -1 to 1. By doing so, the system is ready to apply genetic algorithms. However, users still need to define the number of population, the number of networks with different weights, the probability of crossover and mutation as well as the number of generation [7]. Crossover and mutation process can be seen in Figure 5 and Figure 6.

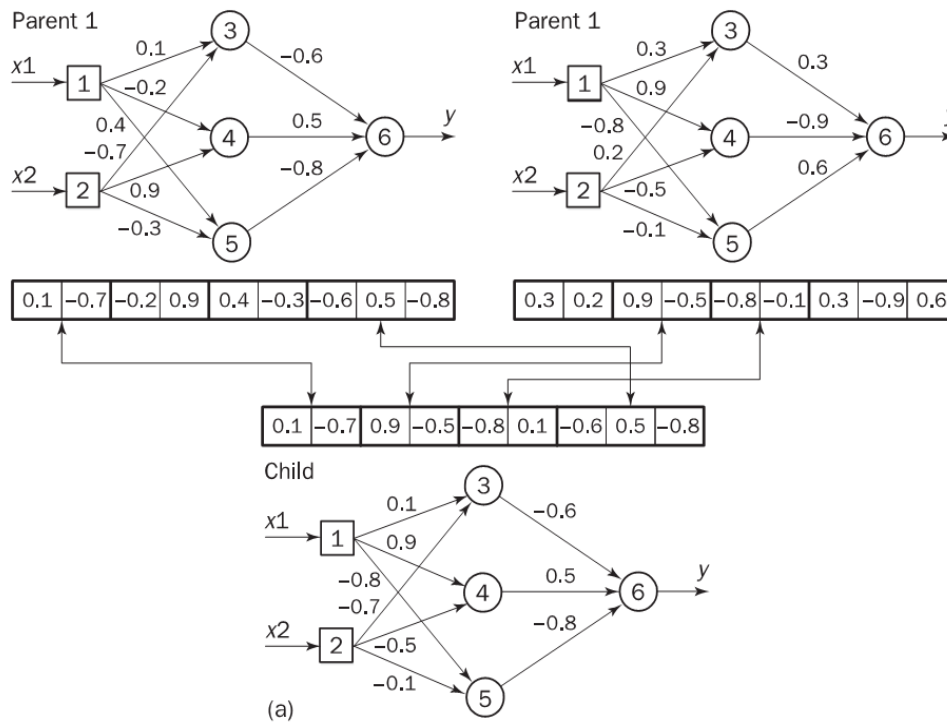


Figure 5. Crossover operation

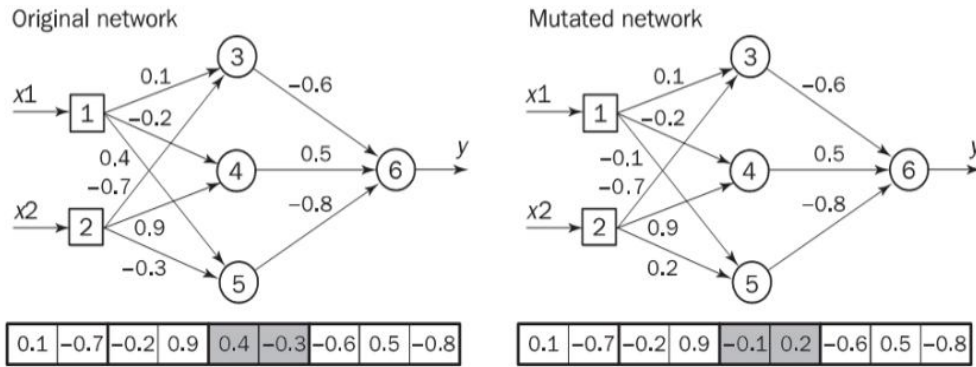


Figure 6. Mutation operation

Backpropagation

Backpropagation neural network is a neural network that uses a multilayer feed-forward architecture and trained using backpropagation algorithm. This method is widely used to solve many problems such as pattern recognition, classification and generalization [8].

Stages of backpropagation training algorithm are as follows [9]:

Feed-forward phase (7 steps):

- 0: Initialize weight (random value between 0-1) and the learning rate α
- 1: While the stop condition is not met, do step 2-13
- 2: Perform steps 3-13 as the desired number of training
- 3: For each hidden layer and output layer do steps 4-13
- 4: Calculate the input of each node in the hidden layer using equation 1

$$z_in_j = \sum_{i=1}^n x_i * w_{ij} \quad (1)$$

- 5: Calculate the output of each node in the hidden layer activation function using equations 2 and 3

$$z_j = f(z_in_j) \quad (2)$$

$$f_1(x) = \frac{1}{1 + \exp(-x)} \quad (3)$$

- 6: Calculate the input of each node in the output layer using equation 4

$$y_in_k = \sum_{j=1}^n z_j * w_{jk} \quad (4)$$

- 7: Calculate the output at each node in the output layer using equation 5

$$y_k = f(y_in_k) \quad (5)$$

Error-backpropagation phase (6 steps):

- 8: Calculate the error of each node in the output layer with the deactivation function using equations 6 and 7

$$\delta_k = (t_k - y_k) * f'(y_in_k) \quad (6)$$

$$f'_1(x) = f_1(x)[1 - f_1(x)] \quad (7)$$

- 9: Calculate the change in weights in each output node in each layer using equation 8

$$\Delta w_{jk} = \alpha * \delta_k \quad (8)$$

10: Calculate the error of each node in the hidden layer to deactivate the function using the equation 9

$$\delta_j = \left(\sum_{k=1}^n \delta_k * w_{jk} \right) * f'(z_{in_k}) \quad (9)$$

11: Calculate the change in weight on each node in each hidden layer using equation 10

$$\Delta w_{ij} = \alpha * \delta_j \quad (10)$$

12: Update the weights at each node in the output layer using equation 11

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk} \quad (11)$$

13: Update the weights on each node in each hidden layer using equation 12

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij} \quad (12)$$

Chi2

Chi2 algorithm [10] is an algorithm that uses the χ^2 statistic to discretize numeric valued attributes. Therefore, this algorithm is quite effective if used in the selection of the important features of a group of numerical attributes. By using the features that are relevant, then this algorithm can speed up the training process and improve the prediction accuracy of classification algorithms in general. And as addition, there are many classification algorithms that require and work better on discrete training data.

In use, the Chi2 algorithm is divided into two phases. The first phase begins with the high enough significance value, e.g 0.5, on all attributes for discretization. The process of merging the data will continue for χ^2 value does not exceed the specified value of significance (0.5, yielding a value 0.455 with degree of freedom equal to 1). This phase will be repeated by reducing the value of significance, until the number of inconsistent data in the discretization exceeds the specified limits. The equation to calculate the value of χ^2 can be seen at equation 13.

$$\chi^2 = \sum_{i=1}^2 \sum_{j=1}^k \frac{(A_{ij} - E_{ij})^2}{E_{ij}} \quad (13)$$

k = number of classification,

A_{ij} = number of pattern at interval - i, classification - j,

R_i = number of pattern at interval - i = $\sum_{j=1}^k A_{ij}$,

C_j = number of pattern at interval - j = $\sum_{i=1}^2 A_{ij}$,

N = total number of pattern = $\sum_{i=1}^2 R_i$,

E_{ij} = the pattern expected from $A_{ij} = R_i * C_j / N$, if R_i or C_j equal to 0, E_{ij} should change to 0.1.

The second phase is the optimization stage of the first phase. The most visible difference is the calculation of inconsistency. In second phase, calculation is done after all the attributes through the merger process. While in the first phase, inconsistent value is calculated at the end of each attribute discretization process. The second phase will be repeated until there are no values of attributes can be discretized or combined. Inconsistency occurs when there are several samples that all of its

attributes have the same value, but, they belong to different groups.

System Design and Implementation

Input for the system is a Java characters digital image in JPEG, PNG or bitmap format. We do several digital image processing, namely grayscaling, highboost filtering, low-pass filtering, and thresholding. Highboost filtering and low-pass filtering is used to reduce digital image noise. Thresholding is used to convert grayscale image to binary image.

Then the digital image is segmented and skeletonized to get each Java character. After that, feature extraction is done using ICZ-ZCZ method [11]. The all steps for those processes can be seen in Figure 5.

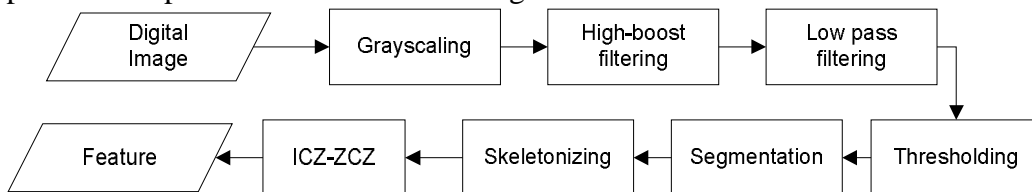


Figure 5. Digital image processing and feature extraction

For each input image, segmentation is done to get all the Java characters that are in the picture. Then we do resizing into 40x40 pixels for each Java characters. Before doing feature extraction, we skeletonized to eliminate the line thickness of character. For feature extraction, we used ICZ-ZCZ method. ICZ (Image Centroid and Zone) – ZCZ (Zone Centroid and Zone) is zoning type feature extraction that utilizing centroid of the image or centroid of the zone. In this research, each image is divided into 20 zones (4 * 5) so that we got 20 output values for ICZ and 20 output values for ZCZ value. This 40 output values from ICZ-ZCZ will become input neurons for neural network.

For evolutionary neural network (ENN), the input layer consists of 40 neurons corresponding to the number of outputs from ICZ-ZCZ, but for backpropagation neural network (BPNN), before entering into the input layer, the data from ICZ-ZCZ will be processed using the Chi2. The results of Chi2 are 60 values, combined with the 40 initial values, will be the input neurons for BPNN.

Output layer for both of ENN and BPNN consist of 31 neurons. Those 31 neurons are in accordance with the number of Java character used in this research. Each neuron has a value of 0 or 1. For the first character, then the first neuron is 1, while other neurons are 0. For the second character, then the second neuron is 1, whereas other neurons are 1, etc. The number of neuron in other layer is 60 neurons for both of ENN and BPNN. Interface for our system can be seen in Figure 6 and Figure 7.

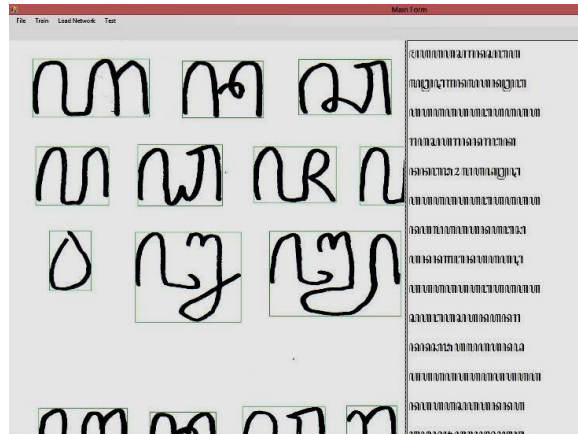


Figure 6. System interface for recognition

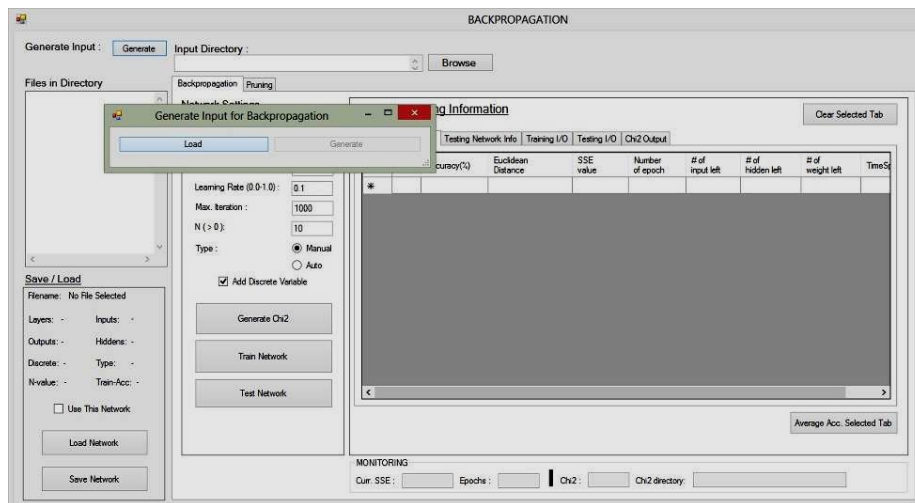


Figure 7. System interface for neural network training

Result and Discussion

To get the results of the comparison, we perform an experiment using the sampled data that has been trained before and the sample data that hasn't been trained before. The number of sample data for training is 15 samples for each character (total 31 Java characters). And for testing we also use 15 sample data for each Java character. We perform two kinds of experiments in which the first experiment conducted training and testing for all types of Java character, while in the second experiment conducted training and testing for each type of Java character (basic, number and *sandhangan*). In Figure 8 can be seen one example of our sample.

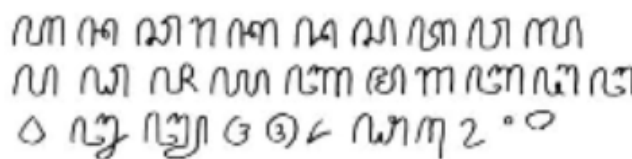


Figure 8. Example of sample data

For evolutionary neuralnetwork method, we use the followingparameters: number of neuron for each layer: 60, crossover probability: 100%, mutation probability: 50%, maximum population: 50, maximum epoch: 10 million and error limit: 0.1. The experimental result can be seen in Table 1.

Table 1. The recognition accuracy for all type of Java characters using ENN

Number of layer	Data type	Recognition accuracy (%)
1 layer	Training data	93.33
	Testing data	48.83
2 layer	Training data	92.21
	Testing data	60.23

This experimental result in Table 1 is done using all type of Java characters, while the experimental result for each type of Java characters (basic / *carakan*, numbers and *sandhangan*) can be seen in Table 2.

Table 2. The recognition accuracy for each type of Java characters using ENN

Java characters type	Number of layer	Data type	Recognition accuracy (%)
Basic character/ <i>carakan</i>	1 layer	Training data	96.73
		Testing data	45.38
	2 layer	Training data	95.21
		Testing data	49.72
Numbers	1 layer	Training data	98.54
		Testing data	54.25
	2 layer	Training data	97.62
		Testing data	60.10
<i>Sandhangan</i>	1 layer	Training data	92.30
		Testing data	60.52
	2 layer	Training data	88.46
		Testing data	53.87

From experimental results, it can be seen that average recognition accuracy of evolutionary neural network is about 93% for data that has been trained before and 60% for testing data.

Experiment using combination Chi2 and BPNN is done using parameters: one hidden layer, number of neuron in hidden layer: 60, maximum epoch: 1000, learning rate: 0.1, initial input neuron: 40, combined with Chi2 algorithm to become 100 input neurons, output neuron: 31. Experimental result for all Java characters can be seen in Table 3.

Table 3. Recognition accuracy for all Java characters type using Chi2 and BPNN

Data type	Recognition accuracy (%)
Training data	95.25
Testing data	70.50

For experiment using each of Java character type can be seen in Table 4.

Table 4. Recognition accuracy for each Java characters type using Chi2 and BPNN

Java characters type	Data type	Recognition accuracy (%)
Basic character / <i>carakan</i>	Training data	97.84
	Testing data	62.68
Numbers	Training data	98.26
	Testing data	76.69
<i>Sandhangan</i>	Training data	99.37
	Testing data	81.42

The overall comparison between ENN and combination of Chi2 and BPNN can be seen in Table 5.

Table 5. Overall comparison

Java characters type	Data type	Recognition accuracy (%)		
		ENN 1 layer	ENN 2 layers	Chi2 and BPNN
All characters type	Training data	93.33	92.21	95.25
	Testing data	48.83	60.23	70.50
Basic character/ <i>carakan</i>	Training data	96.73	95.21	97.84
	Testing data	45.38	49.72	62.68
Numbers	Training data	98.54	97.62	98.26
	Testing data	54.25	60.10	76.69
<i>Sandhangan</i>	Training data	92.30	88.46	99.37
	Testing data	60.52	63.87	81.42

From experimental result in Table 5, it can be seen that average recognition accuracy of Chi2 and BPNN is greater than ENN. The average recognition accuracy of Chi2 and BPNN using data has been trained before is 95% and 70% for data hasn't been trained before. This accuracy is affected by some Java characters that similar one to each other. Their features extracted may be recognized as same character.

Conclusion

In this research, we have developed Java character recognition system and compare the use of evolutionary neural network and combination of Chi2 and backpropagation neural network. From experimental result, combination of Chi2 and backpropagation neural network could perform better accuracy than evolutionary neural network for Java characters recognition. Its recognition accuracy could reach 95% for data has been trained before and 70% for data hasn't been trained before. For future research could be focused using another method for segmentation and feature

extraction that can differentiate similar Java characters in order to increase recognition accuracy.

Acknowledgement

This research was funded by Research Competitive Grant DIPA-PT Coordination of Private Higher Education Region VII, East Java, Indonesia (20/SP2H/PDSTRL_PEN/LPPM-UKP/IV/2014), fiscal year 2014. We also thank Edwin Prasetyo Nandra, Danny Setiawan Putra, Eric Yogi Tjandra, Evan Sanjaya, Jeffry Hartanto, Ricky Fajar Adi Edna P., and Christopher H. Imantaka for their help in doing the system coding.

Reference

- [1] Priyatma, J. E. dan Wahyuningrum, S. E., 2005. Java Character Recognition Using Fuzzy Logic. SIGMA vol 8, No 1, pp 75-84.
- [2] Nurmila, N., Sugiharto, A., dan Sarwoko, E. A., 2010. Back Propagation Neural Network Algorithm For Java Character Pattern Recognition, Jurnal Masyarakat Informatika vol 1, no 1, pp 1-10. (<http://ejournal.undip.ac.id/index.php/jmasif/issue/view/38>). Last access: 1 June 2013
- [3] Budhi, Gregorius Satia and Rudy Adipranata, 2014. Comparison of Bidirectional Associative Memory, Counterpropagation and Evolutionary Neural Network for Java Character Recognition. In Proc. of The 2014 International Conference on Advanced Informatics: Concepts, Theory and Applications, August 2014.
- [4] Daryanto, 1999. *Kawruh Basa Jawa Pepak*. Apollo, Surabaya.
- [5] Parker, J.R., 2010. Algorithm for Image Processing and Computer Vision. New York: John Wiley and Sons, Inc.
- [6] Dewri, R., 2003. Evolutionary Neural Networks: Design Methodologies (<http://ai-depot.com/articles/evolutionary-neural-networks-design-methodologies/1/>). Last access: 30 January 2013.
- [7] Negnevitsky, M., 2005. Artificial Intelligence: A Guide to Intelligence Systems (2nd ed.). New York: Addison Wesley.
- [8] Rao, Hayagriva V. and Valluru B. Rao., 1993. C++ Neural Networks And Fuzzy Logic. New York: Henry Holt and Company.
- [9] Fausett, Laurene, 1994. Fundamentals of Neural Networks. New Jersey: Prentice Hall.
- [10] Liu, H. and Setiono, R., 1995. Chi2: Feature Selection and Discretization of Numeric Attributes. In Proc. of the 7th International Conference on Tools with Artificial Intelligence, Washington D.C., November, 1995. pp. 388-391
- [11] Rajashekararadhya, S.V., Ranjan, Vanaja, 2005. Efficient zone based feature extraction algorithm for handwritten numeral recognition of four popular South Indian scripts. Journal of Theoretical and Applied Information Technology 4(12). pp. 1171-1181.