Towards the On-line Recognition of Arabic Characters

Tim J. Klassen, and Malcolm I. Heywood, *Member, IEEE*

***Abstract*— A generic system is proposed for the recognition of on-line handwritten Arabic characters. Automatic extraction of features from on-line data using SOMs avoids heuristic extraction of features. Performance of a perceptron classifier is competitive with MLP and Genetic Programming based approaches and a better fit for handheld computing devices.**

***Index terms*—Arabic cursive handwriting, on-line character recognition, SOM, perceptron, MLP, Genetic Programming.**

I. Introduction

The keyboard has for a long period of time been regarded as the principle medium of data entry in human–computing environments. Such a status quo however is gradually changing as support for forms of data entry that do not require specialist skills are increasingly being sought. Examples include the use of voice recognition and hand-written data entry. The latter is particularly useful for mobile applications such as hand-held computing (e.g. mobile phones and Palm top computers). This is the particular interest in the present work.

Within such an environment the following assumptions are made,

- Characters are entered individually, thus avoiding the difficult task of segmenting individual letters from words;
- The characters are written in the natural script of the user. Hence, the user does not have to *a priori* learn to write using a script defined by the computing device;
- The on-line nature of the data implies that temporal as well as spatial characteristics are captured during the writing process. This makes the nature of the problem rather different from that of 'off-line' recognition.

Within this context a lot of effort has been given towards the recognition of Asian characters − Chinese, Korean, Japanese − where the 'word' based format of the characters makes the definition of equivalent keyboards difficult. Work in this area is therefore influenced by the line-based nature of the characters. In the case of this work, however, the language of interest is Arabic. Given the explicitly script based nature of the alphabet, a rather different approach to character recognition is necessary. In particular, a system is reported in which simplicity of the underlying learning algorithms is emphasized. The objective being to provide a good starting point from which users may 'train' the default recognition accuracy to the specific characteristics of their own script. Needless to say, if this process is to be performed, in real time, within the limited computational resources of a hand-held computing device, the learning algorithm must be highly efficient.

In the next section, characteristics of the isolated Arabic script recognition problem are summarized. Section III summarizes the character recognition system as a whole, and section IV details the learning systems used. Section V describes the results collected to date, and section VI concludes the work and makes recommendations regarding future work.

II. Characteristics of Arabic Script

The Arabic alphabet consists of 28 letters, written right to left, based on 18 distinct shapes that vary according to their connection to preceding or following letters. Using a combination of dots (one, two or three) or short hand symbols (line, circumflex, hamza) above and below these shapes, the full complement of 28 consonants is constructed. In the case of this work, we are initially only interested in recognizing the 'primary stroke' of each character. That is to say, the dots or secondary symbols are ignored, leaving a total of 15 distinct 'primary' characters to recognize, figure 1.

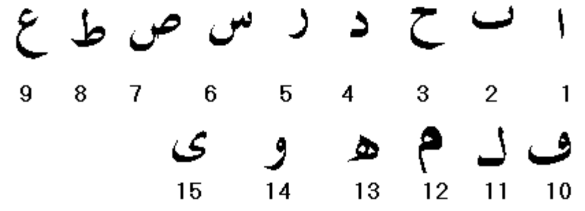

Fig. 1. Examples of the primary characters recognized.

Dalhousie University, Faculty of Computer Science, 6050 University Avenue, Halifax, Nova Scotia. B3H 1W5. Canada. http://www.cs.dal.ca

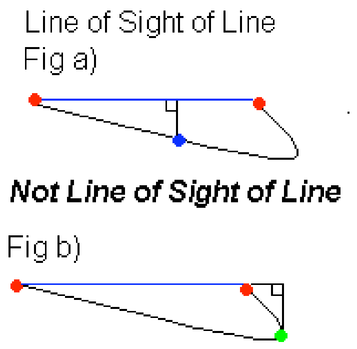

Fig. 2: Line of sight between two points.

In addition, Arabic has four forms for each letter depending on the position of the letter in each word – initial, medial, final and isolated. The complexity of the problem therefore increases significantly when recognition of entire words is necessary (60 separate classes rather than 15 classes). The target application area (independent characters) implies that it is sufficient to provide good recognition accuracy on characters from a single form.

III. Character Recognition System

Character recognition may be considered a specific instance of the general problem of pattern recognition. Thus there are several generic stages: Segmentation, Quantization, Normalization, Feature Extraction and Classification. In the following sections these steps are summarized.

A. Segmentation, Quantization and Normalization

In general, segmentation in character recognition systems is typically associated with extracting individual characters from cursive character sequences associated with a word. Naturally, in the case of this work, this particular aspect is not a problem. Moreover, the stylus naturally provides some hints as to where characters begin and end in the form of 'pen-up' and 'pen-down' information. As indicated later, this on its own is not sufficient to guarantee robust identification of character profiles, as the stylus will tend to bounce on impact with the tablet. However, for the purposes of this initial work, we will consider the availability of characters, which are not distorted in this way.

Given that data entry is an on-line activity in which users write on a touch sensitive 'pad' using a 'stylus', a stream of 'x-y' coordinate pairs corresponding to equally sampled positions of the stylus as it moves across the pad is produced. In practice, we are not actually interested in all such points, but only the points capable of providing a succinct summary of the character. One approach to this problem would be to leave an unsupervised learning algorithm to identify the 'important' points. To do so, however, would represent a computationally expensive exercise, even without limiting ourselves to a hand-held device. Instead, use is made of the observation that the most useful points lie in the areas of greatest variation in curvature [1]. In the past, however, such an observation has lead to the definition of many application / data specific methods for determining exactly which points are removed, and which remain [2-7]. In this work, use is made of the Critical Point definitions derived by Lee and Pan for on-line recognition of signatures [1]. Specifically, the Check Line of Sight or 'ChkLOS' algorithm is used as the sole criteria to extract critical points forwarded to the next stage. The algorithm is based on the premise that two points (on the boundary of the same object) are within 'line of sight' of each other if any point in between can be linked by following the curve and not describing a point outside of a line directly linking the two, figure 2.

Naturally, a threshold (δ) requires specification, which determines the sensitivity to curvature variation in between two end points, figure 2. In this case, the threshold is selected to provide an average of 17.36 points per character, as empirically determined over the training set ($\delta = 10$).

The next step recognizes that users do not produce uniformly sized handwriting. Normalization scales each set of critical point sequences (one set per character) to a uniform input space. This means that the following learning system 'sees' characters described over a constant input space. Again, with reference to the training data set, typical '*x*' and '*y*' pixel counts are taken to define the linear transformation applied to all character sequences. This works well for all characters other than character 1 in figure 1, Alif. In this case the '**l**' like nature of the character results in significant distortion in the '*x*' axis during the normalization process. In order to avoid this, a measure of the pre-normalized character *x* and *y* aspect ratio and a decision to normalize the character made on this basis, or

$$r = \frac{\max(x) - \min(x)}{\max(y) - \min(y)}$$

where $\begin{cases} not\ 'Alif' & if\ r > 0.25 \\ is\ 'Alif' & if\ r \le 0.25 \end{cases}$

In addition to normalizing the spatial information, it is necessary to normalize the temporal information. That is to say, depending on the user, or even mood of the same user, characters are likely to be written at different rates. Hence, a second linear transform is applied to provide characters taking the same time to write as far as the following stage is concerned.

B. *Feature Extraction and Classification*

The purpose of feature extraction is two-fold: to realize that not all data points are equally relevant or useful for pattern recognition and, in the case of neural networks, further reduction of the data input space to keep the network sizes computationally tractable. Most examples of on-line character recognition employs manually chosen features; examples include the number of strokes [2], trigonometric properties [3, 4], curvilinear velocity [5, 6], or position relative to secondary strokes [7]. The approach used here however, will be to use Kohonon's Self Organizing Map (SOM) approach to unsupervised learning [8]. A neural network approach to feature extraction allows automatic selection of relevant features. These features may represent obvious or subtle unseen relationships between the data points. Further study could extract which features the SOM found important to discriminate different Arabic letter classes e.g. dendrogram interpretations of features.

In addition to the selection of specific learning parameters and topology, another important design decision is the manner in which inputs are presented to the SOM network. Examples include providing separate coordinate and time i.e. (x, t) and (y, t) networks, or relying on a single network i.e. (x, y, t). Indeed it is likely that different characters may favor different topologies. The methodology followed in this work is to train an instance of each, review the ability of each SOM to extract useful features by way of a sensitivity analysis based on a plot of the confusion matrix, and then remove redundant SOM nodes. The resulting feature extraction stage is therefore the combination of best SOM nodes taken across all networks (five in total).

The final process is classification, a supervised learning process. In this case, we compare performance using three different classifiers: perceptron; multi-layer perceptron (MLP); and genetic programming (GP). The perceptron is a linear classifier, hence limited to distinguishing between linearly separable classes. The hypothesis here is that the wide input space provided by the SOM stage is sufficient to simplify the problem such that a linear classifier provides good classification accuracy. In addition, the intent is to adapt the classifier on-line from a pre-trained configuration; hence the low computational footprint of the perceptron may well prove a good application match.

Should non-linear performance be necessary, the MLP represents a classical alternative to the perceptron. However, the learning algorithm does not fit the application context, implying that should an MLP be necessary, then classification accuracy must be sufficient without additional online learning. Finally, given the tendency of BP to local minima effects (over learning), a GP classification model is also introduced for comparison. All three classifiers are based on the same SOM preprocessed data. The GP case will also be applied in an off-line training mode, however, the different nature of the learning process and decision functions on which GP is based may well provide a better classifier than provided by the MLP.

IV. Learning Algorithm Details

For completeness, the following four sub-sections summarize the learning algorithms employed. In the case of each neural network, Matlab 5.0 was used as the development environment [9], whereas GP was implemented using linearly structured individuals in Discipulus 2.0 [10].

A. *Self-Organizing Feature Map*

Kohonen's Self-Organizing Feature Map (SOM) algorithm is an unsupervised learning algorithm in which an initially 'soft' competition takes place between neurons to provide a topological arrangement between neurons at convergence [8]. The learning process is summarized as follows,

1. Assign random values to the network weights, w_{ij};
2. Present an input pattern, $\boldsymbol{x}$;
3. Calculate the distance between pattern, $\boldsymbol{x}$, and each neuron weight $\boldsymbol{w}_j$, and therefore identify the winning neuron, or
$$d = \min_j \left\{ \| x - w_j \| \right\}$$
where $\|\cdot\|$ is the Euclidean norm and $\boldsymbol{w}_j$ is the weight vector of neuron j;
4. Adjust all weights in the neighborhood of the winning neuron, or
$$w_{ij}(t+1) = w_{ij}(t) + \eta(t) K(j,t) \left\{ x_i(t) - w_{ij}(t) \right\}$$
where $\eta(t)$ is the learning rate at epoch t; and K(j, t) is a suitable neighborhood function, in this case of a Gaussian nature;
5. Repeat steps (2) – (4) until the convergence criteria is satisfied (in this case an iteration limit of 5,000 epochs).

Following convergence, presentation of an input vector, $\boldsymbol{x}$, results in a corresponding output vector, $\boldsymbol{d}$, the Euclidian distance between each neuron and input. Before forwarding to the classification stage, the mean and variance of each neuron, ($\underline{d}_i$, σ_i) over the training set, $\boldsymbol{P}$, is estimated and used to normalize the output of each neuron such that 68% of the time the neuron output lies over the interval [-1, 1]. That is,

$$d_j' = \frac{d_j - \underline{d}_j}{\sigma_j}$$

B. Perceptron Classifier

The perceptron is a widely studied supervised learning algorithm capable of forming linear discriminent functions [11]. In this case, a vanilla implementation based on Rosenblatt's original implementation is employed, as follows [11, 9],

1. Assign random values to the network weights, w_{ij}, where w_{0j} represent bias terms, or weights with a constant input of 1;
2. Present an input pattern, $\boldsymbol{x}(t)$, and desired output classification, $\boldsymbol{d}(t)$.
3. Calculate the network output(s), $y_j(t)$,

$$y_j(t) = \sum_{i=0}^{N} w_{ij}(t) x_i(t)$$

4. Update all the network weights on presentation t,

$$w_{ij}(t+1) = w_{ij}(t) + \eta \lfloor d(t) - y_j(t) \rfloor x_i(t)$$

5. Repeat steps (2) – (4) until the convergence criteria is satisfied (in this case a SSE over the training set of 0.02 or an epoch limit of 500).

Once trained, a policy is then necessary for distinguishing classification. In the case of the work reported here, a max(·) is taken over the network outputs and the argument of the maximum case taken as the classification. As indicated in section III, the SOM will tend to 'expand' the dimension of the original on-line data (x, y, t). The hypothesis being that such a feature space will more likely be linearly separable; verified in section V.

C. Multi-Layer Perceptron (MLP)

By way of comparison, the case of a non-linear neural architecture is also evaluated in the role of classification. Various learning algorithms where evaluated including the LMS algorithm with adaptive learning rates and momentum and second order derivative approaches [11]. In this case, best results where achieved using second order derivatives as estimated using BFGS quasi-Newton method [11] ('trainbfg' in [9]). The activation function employed was the 'tansig' function, hence desired in class and out of class labels were +0.75 and – 0.75 respectively. The training of a non-linear classifier on a problem with a wide input space is significantly more difficult than for a perceptron, both for computational reasons and due to poor scaling of error back-propagation with high levels of neuron fan in.

D. Genetic Programming

As indicated above, the MLP does not scale particularly well as the dimension of input spaces increase. An alternative non-linear classifier is therefore employed in the form of a Genetic Program [12, 13]. Genetic Programming is a generic beam search process [13] in which the contents of the beam (population) are maintained using the concept of survival of the fittest. The exploration-exploitation trade-off is addressed through genetically motivated operators – mutation and crossover respectively. Members of the population (individuals) take the form of programs composed from a terminal and functional set, where terminals are argument zero terms (e.g. inputs or constants) and functions usually denote the 'instruction set', where the instruction set as a whole is constrained such that syntactic closure is maintained [12, 13]. The resulting individuals represent symbolic expressions, evaluated across the dataset in order to assess their 'fitness' (count of the number of correctly classified patterns in this case). Needless to say, there are many variations on this overall process [13]. In the case of this work, linearly structured individuals are employed [13], with a steady-state tournament selection process [13, 10], and the concept of demes is used to divide the initial population into a series of subpopulations in order to minimize the chances of premature convergence or domination by an individual representing a local minima [13, 10] (all populations see all patterns).

GP typically only has a single output per individual, hence separate populations were evolved for each class. There are 15 classes and approximately 107 examplars per class (actually an unequal distribution of exemplars exists; section V) implying that if the classifier learned to 'do nothing' 93% of the classifications would be correct! This is a factor which any learning system needs to avoid, and is discussed further in the results section. Table I summarizes the learning parameters used to configure the linearly structured GP individuals.

Table I

Linear GP parameters

Function Set	+, -, *, %, exp, cosine, sine, sqrt
Terminal Set	SOM outputs
Crossover Rate	90 %
Mutation Rate	90 %
Max FPU Registers	4
Ratio Constants to Inputs	5 %
Instruction Limit	25(initial) 256 (max.)
Predefined Constants	0, 0.5, 1.0
Population Size	5,000
Max Tournaments	100,000
Demes	
Number of	10
Crossover Between	10 %
Migration Rate	5 %

V. Experimental Study

Given the lack of a public repository for on-line Arabic characters, a data capture application is written in C++ for the MS Windows™ PC platform. Examples of handwritten characters were captured using a digital tablet (Wacom Graphire Model ET-0405-U) with a resolution accuracy of 23 points/cm, an active surface of 9.2 x 12.8 cm and at a sampling frequency of 100 points/sec. Each candidate provided five copies of the entire Arabic alphabet in one sitting. This tended to have the effect that the first and the second passes through the alphabet provided the 'cleanest' examples and the latter two passes the most noisy, but possibly the most representative of handwriting as would be given in an application context. In all cases, the only constraint was that candidates write within the bounds of the data entry field and that characters be written one at a time. In total, 25 candidates contributed example handwriting, in all but two cases, the candidates wrote in Arabic as a first language.

As indicated in section II, we are only interested in recognizing the primary stroke; figure 1. Thus manual pre-processing is performed to separate primary and secondary strokes and check the integrity of the resulting primary strokes. That is to say, given the hardness of the writing surface, as provided by the digital tablet, the stylus would often 'bounce' producing multiple pen up and down signals in what should actually be considered a single sequence. Given the preliminary nature of this study, these artifacts were corrected (the out of sequence pen up/ down removed) and training conducted on the cleaned data. The hypothesis being that once trained to recognize valid examples, the system will be in a much better position to identify incomplete sequences automatically.

The data set is now separated into training, validation and test sequences. The training set consists of the 1st, 3rd and 5th example sequences collected from 20 of the 25 individuals ($\approx$1680 patterns). The validation set consists of the 2nd and 4th example sequences taken from the same 20 individuals ($\approx$1080 patterns). The test set is therefore all samples taken from the remaining 5 unseen individuals ($\approx$1000 patterns). This means that the validation set consists of stylistic properties already seen. This is therefore similar to the handheld computing context in which, given an initial configuration of the learning system, the user will tutor the system to recognize his/ her handwriting more explicitly. The test set is more synonymous with a system that needs to recognize the handwriting of multiple authors, where there is no constraint over the pool of participating authors.

As detailed in section III, Critical Points are then extracted and normalized for scale and translation invariance. Moreover, the temporal element is also normalized to ensure that the SOM network 'sees' the same degree of variation in each input [8]. It is emphasized that at this point no normalization for rotation or skew is provided. The implicit assumption is that the training data set is suitably representative of these properties and that the networks will account for these variances.

The following subsections detail the application of each learning system.

A. Extraction of features using SOM

As indicated in section III-B, we are interested in the automatic identification of features as opposed to accepting an *a priori* defined set of primitives. The on-line nature of the application implies that the initial input space is already small (three dimensions, x, y, t). Hence, the SOM will expand this space to provide a useful set of features on which classification will take place at the next stage in the spatial domain alone. To do, so the SOM is trained on (x, t), (y, t) and (x, y, t) pairs. In each case, the value of the corresponding winning neuron is retained. Thus, the classification stage sees as input the same number of 'winning points' as there are (x, y, t) tuples.

Initial experimentation with neuron counts and topologies leads to three sets of networks; table III. Performance was evaluated using a confusion matrix constructed across the training set examples. Naturally, from the classification stages' perspective, we desire clusters with high inter-class discrimination, but low intra-class variation. Factors that influence this are the network topology and initialization. The former is relatively easy to optimize, the latter however is usually addressed by repeatedly training the same network topology over different random number seeds, until acceptable performance is achieved.

This is a computationally expensive process. Instead, we take the set of topologies for a given initialization and use the confusion matrix of each to construct a new SOM from the best-case cluster performance on each class. In the case of a tie between class-wise classification errors, the SOM architecture that had the lowest overall error rate was chosen. Two cases were considered, partitioning on training data, table IV, or on validation, table V (raw error counts in each case). Second, third and forth characters are observed to have some regularity in the error (see [14] for confusion plots).

Table III

SOM Network Topologies

SOM identification	Input	SOM topology	Total Nodes
D-SOM 1	(x, t),	6 × 4, 7 × 5	59
D-SOM 2	(y, t)	7 × 5, 7 × 5	70
S-SOM	(x, y, t)	10 × 6	60

Table IV

SOM Classwise Error on Training Data

Character	D-SOM 1	D-SOM 2	S-SOM	partitioned
1	***3***	6	3	3
2	75	***17***	17	17
3	9	***9***	9	9
4	23	***11***	11	11
5	***19***	28	19	19
6	63	17	***15***	15
7	***3***	8	11	3
8	3	***2***	6	2
9	55	18	***9***	9
10	***11***	19	25	11
11	16	52	***8***	8
12	***1***	2	4	1
13	4	***4***	6	4
14	***0***	2	4	0
15	8	***7***	7	7
Total err.	293	202	154	119

Table V

SOM Classwise Partitioning on Validation Data

Character	D-SOM 1	D-SOM 2	S-SOM	partitioned
1	13	14	***6***	6
2	68	***16***	82	16
3	***15***	19	52	15
4	30	***15***	17	15
5	***18***	24	25	18
6	46	***15***	20	15
7	10	22	***10***	10
8	9	***5***	12	5
9	19	***16***	18	16
10	***15***	29	24	15
11	15	40	***12***	12
12	***6***	9	9	6
13	13	19	***6***	6
14	5	***3***	13	3
15	***12***	13	15	12
Total err.	294	259	321	170

B. *Classification*

The result from the SOM feature extraction stage is in the form of either, one of three networks (D-SOM 1 and 2 or S-SOM) or the combination of all three networks (hereafter referred to as P-SOM). A decision as to which configuration to use is now necessary. Although the P-SOM provides the best features, the input space to the classifier is now 189 nodes. To reduce this space, perceptrons are first trained on each of the three SOM topologies, but only on the classes for which the corresponding SOM provides the best features. A simple pruning procedure is then instigated in which SOM nodes are masked and perceptron classification performance assessed. If any decrease in classification accuracy occurs for that class, the node is retained else it is removed. Three variants are now possible: training the perceptrons without pruning – trial 1; training the perceptrons whilst pruning on training data – trial 2; training the perceptrons whilst pruning on validation data – trial 3. Table VI summarizes these results for in-class classification accuracy and the number of nodes from the SOM on which the classification is made.

Table VI

Percent In-class Perceptron Classification Performance

Configuration	Training	Validation	Test
SOM alone	83	75	63
Trial 1 (178 nodes)	88	77	64
Trial 2 (169 nodes)	93	84	77
Trial 3 (159 nodes)	90	89	79

Needless to say, when pruning is guided by the validation data set, the number of nodes removed increases and overall classification performance increases. However, from an application context, access to a separate validation data set might not exist, hence the remaining comparisons are conducted for the case of pruning on training data (trial 2). Table VII lists classification performance for perceptron, MLP and linear-GP for combined, positive and negative classification performance.

Table VII

Percent Classifier Performance under trial 2 conditions

Data Set		Perceptron	MLP	L-GP
Training	Overall	96.8	***99.7***	85.6
	Positive	93	***94***	91.2
	Negative	97	***100***	85.3
Validation	Overall	95.3	***96.65***	84.8
	Positive	84	73	***87.2***
	Negative	96	***98***	84.7
Test	Overall	***95***	***95***	83.3
	Positive	77	60	***82.1***
	Negative	96	***97***	83.4

From table VII it is apparent that there is little benefit in using an MLP as a non-linear classifier with respect to the performance achieved using a perceptron. Indeed, perceptron performance on in class exemplars for validation and test is always better than that of the MLP and negative class exemplars are within 1% of the MLP results. The major drawback using the MLP approach appears to be an over emphasis on classifying the negative (out of class examples) as opposed to the in class examples (the learning to do nothing effect), though reducing the architecture from 5 hidden layer neurons may account for this.

L-GP on the other hand does not achieve the same level of accuracy as either neural approach on training data. However, L-GP does provide good positive exemplar accuracy throughout and deviates least from training set performance on validation and test sets, a property that indicates good 'generalization'. In order to aid the further learning of negative as well as positive examples investigation of weighted cost functions is foreseen.

VI. Conclusion

A system for the on-line recognition of primary strokes in Arabic cursive characters has been proposed. Emphasis is given to computational efficiency as well as recognition accuracy. To do so significant effort is made to provide a good feature set before classification is attempted. Comparison against various non-linear classifiers indicates that good recognition of the primary strokes is possible with a relatively simple system. Given the success of this approach, a next natural step would be to consider performance using an LVQ type classifier [8] (not used in this initial work given our desire to separate the feature extraction and classification stages for experimental purposes).

Comparison to previous works is hampered by the lack of publicly available datasets. However, as far as the authors are aware, the proposed system was evaluated over a wider range of writers than is the case for all other reported systems. Moreover, this is the first system to explicitly target the handheld computing environment, and recognize the need to provide on-line adaptation of the classifier on such a platform.

References

[1] Lee S., Pan J.C., "Offline Tracing and Representation of Signatures," IEEE Transactions on Systems, Man and Cybernetics. 22(4), pp 755-771, 1992.

[2] El-Sheik, T.S. and El-Taweel, S.G., "Real-Time Arabic Handwritten Character Recognition" ,Pattern Recognition, volume 23 (1990), number 12 , pp. 1323-1332.

[3] Alimi, A. and Ghorbel, O. "The Analysis of Error in an On-Line Recognition System of Arabic Handwritten Characters", Proceedings of ICDAR 1995, 14-16 August 1995, Montreal, Canada. pp. 890-893.

[4] Al-Emami, S. and Usher, M. "On-Line Recognition of Handwritten Arabic Characters", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 7, July 1990. pp. 704-710.

[5] Bouslama, F. and Amin, A., "Pen-based Recognition System of Arabic Character Utilizing Structural and Fuzzy Techniques", 1998 Second International Conference on Knowledge-Based Intelligent Electronic Systems, 21-23 April 1998, Adelaide, Australia. Editors, L.C. Jain and R.K. Jain, pp 76-85.

[6] Alimi, A., "An Evolutionary Neuro-Fuzzy Approach to Recognize On-Line Arabic Handwriting", Proceedings of the 4th International Conference Document Analysis and Recognition (ICDAR '97), pp. 382-386, 1997.

[7] El-Wakil,M. and Shoukry, A., "On-Line Recognition of Handwritten Isolated Arabic Characters", Pattern Recognition, vol. 22, no.2, pp. 97-105, February 1989.

[8] Kangas J.A., Kohonen T., Laaksonen J.T., "Variants of Self-Organizing Maps," IEEE Transactions on Neural Networks, 1(1), pp 93-99, March 1990.

[9] Demuth H., Beale M., Matlab – Neural Network Toolbox, Users Guide 4.0; http://www.mathworks.com

[10] Discipulus 2.0; http://www.aimlearning.com

[11] Widrow B., Lehr M.A., "Adaptive Neural Networks and Their Applications," International Journal of Intelligent Systems," 8, pp 453-507. 1993.

[12] Koza J., Genetic Programming – I, MIT Press, 1992.

[13] Banzhaf W., Nordin P., Keller R.E., Francone F.D., Genetic Programming – An Introduction. Morgan Kaufmann, 1998.

[14] Klassen T.J., Towards NN Recognition of Handwritten Arabic Letters. Project Report, www.cs.dal.ca/~mheywood/Reports/, 2001.