

Logo Recognition by Recursive Neural Networks

E. Francesconi \diamond P. Frasconi \diamond M. Gori \ast
S. Marinai \diamond J. Q. Sheng \diamond G. Soda \diamond A. Sperduti \bullet

\diamond Dipartimento di Sistemi e Informatica - Università di Firenze - Italy

\ast Dipartimento di Ingegneria dell'Informazione - Università di Siena - Italy

\bullet Dipartimento di Informatica - Università di Pisa - Italy

<http://mcculloch.ing.unifi.it/~docproc>

Abstract. In this paper we propose recognizing logo images by using an adaptive model referred to as recursive artificial neural network. At first, logo images are converted into a structured representation based on contour trees. Recursive neural networks are then learnt using the contour-trees as inputs to the neural nets. On the other hand, the contour-tree is constructed by associating a node with each exterior or interior contour extracted from the logo instance. Nodes in the tree are labeled by a feature vector, which describes the contour by means of its perimeter, surrounded area, and a synthetic representation of its curvature plot. The contour-tree representation contains the topological structured information of logo and continuous values pertaining to each contour node. Hence symbolic and sub-symbolic information coexist in the contour-tree representation of logo image. Experimental results are reported on 40 real logos distorted with artificial noise and performance of recursive neural network is compared with another two types of neural approaches.

1 Introduction

The adaptive computational scheme of statistical models and artificial neural networks is very well-suited for pattern recognition. It is commonly recognized, however, that these adaptive models, especially feedforward neural networks, can only handle graphs by encoding them into a fixed-size vectorial representations. This is motivated by the fact that feedforward neural networks only have a fixed number of input units. On the other hand, most of the models proposed in the field of syntactic pattern recognition are very well-suited for incorporating pattern grammatical regularities. Unfortunately, symbolic approaches are not strongly oriented to face the presence of noise. This scenario seems to be mainly due to the development of computational approaches that are basically oriented to either dealing with symbols or with unstructured information (structured information is usually represented with trees or graphs) while in the field of pattern recognition one is often looking for a balanced combination of the symbolic and sub-symbolic processing levels. This paper is intended to bridge these two distinct paradigms by applying recursive neural networks for solving logo recognition problems, a typical task in which the significant presence of

noise and the structured information associated with the patterns might require both paradigms be taken into account.

Logo recognition has been a research area of great interest in the last few years, particularly in applications in the field of document image analysis. In our work, it is required for identifying commercial invoice class in order to facilitate the modeling and recognition of documents [1]. Many proposals for logo recognition are based on syntactic approaches (see e.g. [2], [3], [4]) or neural network approaches (see e.g. [5], [6]). Particularly, Cesarini et al. [5] propose to estimate logo membership by multi-layer perceptrons (MLPs) acting as auto-associators that are fed by a vector obtained by fitting the logo image into a fixed size grid. Cesarini et al. [6] also describe an extension of the approach proposed in [5] in order to deal with spot noise where strips and blobs produce a partial obstruction of logo. Such a neural network approach completely relies on the assumption that auto-associators can learn the logo regularities and face the presence of noise. Although very satisfactory results have been achieved, in particular in the presence of spot noise, a structural approach is in demand to be investigated in order to deal with very similar logos, such as `logo13` and `logo14` or `logo29` and `logo30` in Figure 6, where the Euclidean distance between the corresponding vectorial representations is small, but the logo structures are quite different. In this paper, we propose recognizing logos described by a structured representation by using recursive neural networks.

Recursive neural networks have been recently proposed as a model for learning data structures such as labeled trees or graphs [7]. The recognition strategy proposed here is composed of two steps. First, logo images are converted into a structured representation based on contour trees [3]. Then recursive neural networks are employed to classify these structured representations. One of the problems with the first step is that noisy images can produce contour trees having many children corresponding to small contours. We propose filtering out these irregularities by means of a contour tree pruning procedure.

The paper is organized as follows: at first in Section 2 we describe the recursive neural network and its relevance for logo recognition. In Section 3, we describe the contour tree construction algorithm of logos and the tree transformation. In Section 4 we report the experimental results on 40 logos and the performance comparison between recursive neural networks and some other neural network approaches. Finally some conclusions are drawn in Section 5.

2 Recursive Neural Networks

We describe here briefly how the recursive model works in the graphical structured domain. Firstly, an extended dynamical system operating on a tree is described. Secondly, the use of neural networks for implementing such a dynamical system is introduced.

2.1 Dynamical System for Tree Structured Domains

Dynamical systems are usually defined for modeling data generated in temporal domains. A discrete time dynamical system operates on serially ordered entities and can be used to map a sequence into a class label, according to the following recursive state space representation:

$$\begin{aligned}x_t &= f(x_{t-1}, u_t) \\ y &= g(x_T)\end{aligned}\tag{1}$$

where f is the state transition function and g maps the final state x_T into the category y . For example, if inputs (u_t) and states (x_t) are both symbolic, the above equation defines a finite state automaton and can be regarded as a function from labeled lists to a scalar quantity representing the category of the input list. Dynamical systems for sequences can be generalized to dynamical systems for labeled graphs, by properly redefining the state transition function. In the symbolic domain, such extensions have been thoroughly studied during Seventies, in research areas such as computational language theory and in syntactic pattern recognition, leading to the definition of symbolic recognizers known as tree or graph automata [8, 9]. In the present context, the purpose is to define a scalar function whose input is a labeled tree, but vertex labels are not necessarily symbolic. Let v denote a generic vertex in an m -ary tree and let u_v denote the label attached to v . Then, the generalized recursive state space representation is based on the following equations:

$$\begin{aligned}x_v &= f(x_{\text{ch}[v]}, u_v) \\ y &= g(x_r)\end{aligned}\tag{2}$$

where r corresponds to the root of the tree and $x_{\text{ch}[v]}$ denotes the set of state labels attached to the children of v . Whereas in eq. 1 the input is scanned sequentially from left to right, computation in eq. 2 must begin simultaneously at each vertex on the frontier of the input tree (the leaves of the tree) and proceeds along parallel paths towards the root. Figure 1 shows an example of such computation. Whenever a child is missing for vertex v , the corresponding entry in $x_{\text{ch}[v]}$ is replaced by a special *frontier state*, that generalizes the concept of initial state in sequential systems. Note that the “final” state x_r in eq. 2 is a summary of the whole input tree, in the same way as the final state x_T in eq. 1 is a summary of the whole input sequence. Hence, the categorization of the tree only depends on x_r . In a tree automaton, u_v and x_v are symbolic and g is a binary valued function that takes on the value 1 iff the argument is an accepting state.

In this paper we use a generalized view of tree automata in which u_v and x_v are continuous valued vectors. In the present approach the functions f and g are realized by means of feedforward neural networks and, hence, can be adapted to fit a given set of training examples.

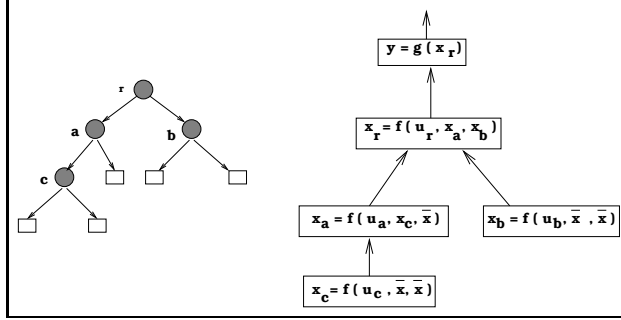


Fig. 1. An example of simple tree and its internal state transition. Mainly there are two functions f, g are involved in the developing of dynamical system. Function f is internal state transition function and g is the prediction function.

2.2 Recursive Neural Network for Tree Classification

Very recently, *recursive* neural networks have been introduced as an extension of recurrent neural network to accept labeled directed ordered acyclic graphs (i.e., graphs in which a total order is defined on the m children of each node) as input [7]. In that case, functions f, g , which are used for the evolution of dynamical system, are realized by using a multilayer feedforward neural network. In such case, the function variables are the connection weights of the neural network. In this paper, we limit our discussion to recursive nets operating on ordered m -ary trees, in which each node is presumed to have equally m children.

A recursive neural net is an 8-tuple $\{\mathcal{U}, \mathcal{Y}, \mathcal{X}, f, g, \bar{x}, \theta, \vartheta\}$ where \mathcal{U} , \mathcal{Y} , and \mathcal{X} are the input label space, the output label space, and the state space, respectively. The input label space \mathcal{U} corresponds to the external input at each vertex, the state space \mathcal{X} contains the state elements for the corresponding node, the output label space \mathcal{Y} is used for the prediction of the associated structure. Given an input m -ary tree \mathcal{U} , internal states in the network are recursively updated according to the equation

$$\mathbf{x}_v = f(\mathbf{x}_{\text{ch}[v]}, \mathbf{u}_v; \theta) \quad (3)$$

where v is a generic node in the input tree, \mathbf{u}_v is the label attached to node v , $\mathbf{x}_v \in \mathcal{X}$ is the state variable at node v and $\mathbf{x}_{\text{ch}[v]}$ is the ordered *set* of states at the children of node v . $f: \mathcal{X}^m \times \mathcal{U} \rightarrow \mathcal{X}$ is the *state transition function* and is realized by a feed-forward neural network with sigmoidal units (with activation function $f(x) = \frac{1}{1+e^{-x}}$) and connection weights θ between the interlayer nodes.

In the special case $m = 1$ (i.e. the tree is a sequence), eq. (3) reduces to the usual state updating in sequential dynamical systems and the node position v corresponds to a discrete time point ($\mathbf{x}_{\text{ch}[v]}$ would be replaced by the *previous state* \mathbf{x}_{v-1}). Whereas a recurrent net scans an input sequence element by element from past to future, a recursive net must begin simultaneously at each node on the frontier of the input tree and proceeds along parallel paths towards the root.

Whenever a child of the generic node v is missing, the corresponding entry in $\mathbf{x}_{\text{ch}[v]}$ is replaced by a node with the *frontier state* \mathbf{x}_0 . Frontier-to-root automata (see e.g. [8], page 158) for the recognition of syntactic structure process trees in the same fashion. The class prediction $\mathbf{y} \in \mathcal{Y}$ is obtained as a function of the state at the root node r

$$\mathbf{y} = g(\mathbf{x}_r; \boldsymbol{\vartheta}). \quad (4)$$

Again, we remind that g is also realized by a feed-forward neural network with weights $\boldsymbol{\vartheta}$.

The supervised learning problem in recursive neural nets can be solved in the usual framework of error minimization (or maximum likelihood from the statistical viewpoint). Searching the optimal parameters $\boldsymbol{\theta}$ and $\boldsymbol{\vartheta}$ can be accomplished by gradient descent techniques [10]. In this case, gradient can be computed using the back-propagation through structure algorithm, an extension of back-propagation through time [11] that unrolls the recursive network in a larger feed-forward network, following the topology of the input graph [7]. An example of unrolled recursive network obtained from the contour tree reported in Figure 3 is shown in Figure 2.

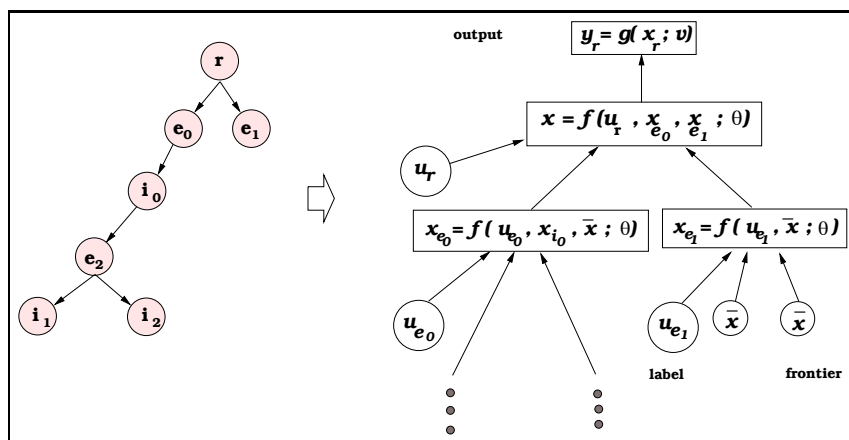


Fig. 2. Left: A logo contour tree (see the details in Section 3). Right: Fragment of the recursive network unrolled according to the contour tree. The fragment corresponds to nodes r , e_0 and e_1 . Circles represent the labels attached to nodes in the contour tree. Rectangles represent hidden states, updated by means of the transition function f . The output \mathbf{y}_r is a function g of the state at the root of the tree. Both f and g are realized by feed forward nets.

When realizing the recursive neural network by feedforward neural network, the number of input neurons N_0 for the feedforward neural network is fixed as follows

$$N_0 = N_l + mN_s \quad (5)$$

N_s is the dimension of the state space, N_l is the dimension of the label space, and m is the maximum order of the tree as described above.

3 Structured Logo Representation

An appropriate description of logo structures is based on a tree representation referred to as contour tree which is created by recursively detecting contours in logo instance (see an example in Figure 3). The nodes of the contour tree correspond to the contours of logo and the edges of the tree define the topological inclusion relationships among the contours. The root of the tree represents an upright rectangle surrounding the logo item. The nodes in odd levels of the tree are associated to exterior contours and nodes in even levels are associated to interior contours. In this paper, the tree obtained by means of the contour extraction algorithm is called *primary* contour tree. A *secondary* contour tree is generated by deleting some nodes and links from the primary contour tree and adding new nodes to the tree. This transformation is intended for limiting the maximum out-degree of the tree (the maximum number of children for each node of the tree), and consequently reducing the complexity of the recursive network.

Each node v in both primary and secondary contour tree is labeled by a set of numerical features measured on the object associated to the node and collected in a multivariate variable \mathbf{u}_v . The features are the perimeter of the contour, the area surrounded by the contour and a sub-vector containing a synthetic representation of its curvature plot. Appropriate normalization of continuous labels ensures scale invariance. Rotation invariance is gained through an ordering operation of the children for a given node according to the contour length.

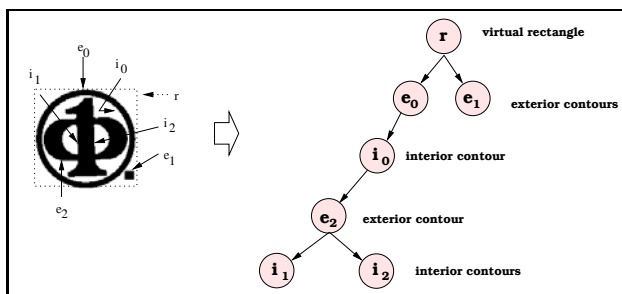


Fig. 3. Left: A logo image. Right: The contour-tree representation of the logo.

3.1 Contour Tracing and Feature Extraction

Contour tracing [12] is a local operation on a binarized image that traverses adjacent contour pixels to derive a chain coded representation. The exterior

contours are formed with clockwise ordered pixels and interior contours with counterclockwise pixels. Perimeter of contour is obtained directly in the phase of contour tracing. Area surrounded by an exterior contour is the total number of black pixels inside the contour, while area enclosed by an interior contour is the total number of white pixels inside the contour. As stated before, in order to maintain the consistency for the root node, r , of the tree, an upright rectangle surrounding the logo is virtually applied to the image and corresponding feature vector can still be obtained. By doing so, all nodes in the contour tree, including the root node, have a homogeneous representation with one feature vector associated. The perimeter for the root is $P_r = 2(w + h)$, where w and h are width and height of upright rectangle respectively, and $A_r = wh$.

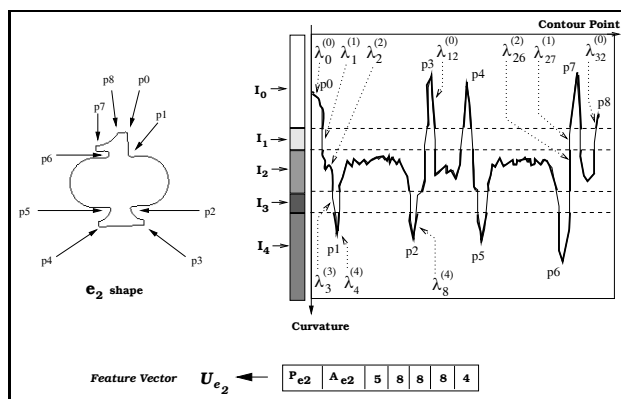


Fig. 4. A contour extracted from a logo instance, and its curvature plot. The curvature plot contains also the five intervals I_0, \dots, I_4 and some *curvature-runs* indicated by thin lines and thick lines denoted by $\lambda_0^{(0)}, \lambda_1^{(1)}, \dots, \lambda_{32}^{(0)}$. The elements of the feature vector, relative to curvature-runs, are: $D_{e_2}^{(0)} = 5$, $D_{e_2}^{(1)} = 8$, $D_{e_2}^{(2)} = 8$, $D_{e_2}^{(3)} = 8$, and $D_{e_2}^{(4)} = 4$.

The analysis of the curvature plot of a contour allows to describe the contour in a synthetic way, since corners in the contour correspond to peaks in the curvature plot (critical points). In order to properly extract the critical points, we have used the smoothed compensated extended Freeman coding of the contour [13], followed by a derivative smoothing filter for finding critical points. We have set up five intervals for derivative values corresponding to the dimension of the sub-vector describing the curvature plot. The number of groups of contiguous contour pixels whose curvature values fall in each of the intervals I_i ($i = 0, \dots, 4$) is chosen as value for the corresponding value of the sub-vector. This sub-vector $\mathcal{D} = (D^{(i)})$ ($i = 0 \dots 4$) together with the perimeter P and area A forms the feature vector pertaining to the node. The set of contiguous points whose curvature values fall in interval I_j is defined as a *curvature-run* $\lambda^{(j)}$, so a contour is composed of a series of *curvature-runs*, denoted as $\lambda_1^{(j_1)}, \lambda_2^{(j_2)}, \dots, \lambda_n^{(j_n)}$, where n is the total number of *curvature-runs*, and $j_k \in [0, \dots, 4]$ ($k = 1, \dots, n$) indicate

that the corresponding run is in interval I_j . For a node v in the contour tree, the feature vector is defined as $\mathbf{u}_v = (P_v, A_v, D_v^{(0)}, \dots, D_v^{(4)})$, where $D_v^{(i)}$ is equal to the total number of *curvature-runs* in interval I_i (see example in Figure 4). Note that the number of intervals I_i is chosen after some preliminary trials.

It is customary for neural network to accept normalized values as inputs in order to avoid neuron saturation. Therefore, all the seven elements in the feature vector are normalized in an appropriate way. The perimeter is normalized with the maximum contour length found in the logo. The area is normalized with the logo area of surrounding rectangle. The number of *curvature-runs* is normalized with the maximum number of *curvature-run* in the five intervals for all the contours extracted from the logo. In Figure 4, an example of logo contour and its derivative value at each point by means of the above-mentioned approach are depicted.

3.2 Tree Construction and Transformation

The contour tree is constructed by tracing contours and searching holes in the region enclosed by the contours. Once a contour is found, a node associated to the contour is created and added to the tree. The node associated to an interior (or exterior) contour is added as child of the node to which its nearest outside exterior (or interior) contour is associated. This procedure is performed recursively until all contours have been found and no holes exist [3]. The contour tree constructed in this way, called primary contour tree, is a general tree, whose nodes have an unspecified number of children. The recursive neural network, on the other hand, operates on m -ary trees. Transforming general trees into m -ary trees is straightforwardly obtained by choosing m as the maximum number of children in the contour tree, and by filling in with null pointers the missing children. Since the net complexity increases with m , and since the maximum number of children in the contour tree may become quite large (because of irrelevant or noisy details in the image), it makes sense to prune the contour tree by collapsing subtrees associated to contours which are judged not to be very significant.

The contours order as well as feature values can be altered due to the presence of noises. However, these different structures are learnt by the recursive nets, since noisy logo instances are generated and used for the net training.

As detailed in Algorithm 1, pruning is performed by fixing m and by depth-first traversing the primary contour tree for finding nodes having more than m children. When the number of children for a given node is greater than m the following operations are performed (see Figure 5). First, the children associated to small contours (with length less than a threshold τ) are concentrated into a representative virtual node, whose features are calculated by a simple averaging operator. Second, the children associated to big contours (including the above added virtual node) are iteratively split until v has less than m children. Usually, the tree transformation reduces the out-degree of some nodes and increase the tree depth.

```

Algorithm 1 TRANSFORM( $v, m, \tau$ )
1  Let  $\mathcal{C}$  be the list of children of  $v$ ;
2  if  $|\mathcal{C}| > m$  then
3    Let  $\mathcal{C}_S \leftarrow \{w \in \mathcal{C} : P_w < \tau\}$ ;
     $\triangleright$  List of nodes associated to small contours
4    Let  $\mathcal{C}_B \leftarrow \mathcal{C} \setminus \mathcal{C}_S$ ;
     $\triangleright$  List of nodes associated to big contours
5    if  $\mathcal{C}_S \neq \emptyset$  then
6      Create a node  $g_0$  with features obtained from the features of nodes in  $\mathcal{C}_S$ ;
7       $\mathcal{C} \leftarrow \mathcal{C}_B \cup \{g_0\}$ ;
8      Create a node  $g_1$  with features obtained from the features of descendants
      of nodes in  $\mathcal{C}_S$ ;
9      Let  $\{g_1\}$  be the list of  $g_0$ 's children;
10     while  $|\mathcal{C}| > m$  do
11       if  $|\mathcal{C}| \geq m^2$  then
12          $N \leftarrow \lceil \frac{|\mathcal{C}|}{m} \rceil$ ;  $\triangleright$  Number of new nodes to be created
13          $S_N \leftarrow 0$   $\triangleright$  Number of nodes in  $\mathcal{C}_B$  to be maintained
14         else
15            $N \leftarrow \lfloor \frac{|\mathcal{C}| - (m+1)}{m-1} \rfloor + 1$ ;
16            $S_N \leftarrow m - N - 1$ 
17           Create an ordered partition of  $\mathcal{C}$ :  $\mathcal{C}_0 \cup \mathcal{C}_1 \cup \dots \cup \mathcal{C}_N = \mathcal{C}$  such that  $|\mathcal{C}_0| = S_N$ 
           and  $|\mathcal{C}_1| = |\mathcal{C}_2| = \dots = |\mathcal{C}_{N-1}| = m$ ;
18           for  $i \leftarrow 0$  to  $N - 1$  do
19             Create a node  $h_i$  with features obtained from the features of nodes in
              $\mathcal{C}_{i+1}$ ;
20             Let  $\mathcal{C}_{i+1}$  be the list of  $h_i$ 's children;
21             Let  $\mathcal{C} \leftarrow \mathcal{C}_0 \cup \{h_0, \dots, h_{N-1}\}$ ;
22     foreach  $w \in \mathcal{C}$ 
23       TRANSFORM( $w, m, \tau$ )

```

4 Experiments of Logo Recognition

Some experiments were carried out for logo recognition using the proposed recursive neural networks. The comparison of its performance to MLP (Multilayer Perceptron) approach and autoassociator-based neural networks [5] is made.

The data used in our experiments, 40 distinct company logos (from **logo2** to **logo41**) from logo database created by the *Document Processing Group, Center for Automation Research, University of Maryland*. In Figure 6 it is shown some examples which is an subset of 40 logos we used in our experiments. It can be noted that some logos (such as **logo12**, **logo14** and **logo37**) have a very shallow tree structures. Some logos such as **logo2**, **logo4** and **logo5** have a relatively higher outdegree for some nodes and some logos such as **logo11** and **logo5** have a nested contour-tree structural representations. Some logos, for example **logo29**, **logo30** and **logo34**, are composed mainly by some associated text while **logo5** is composed of iconic region surrounding a text region. Furthermore, although **logo29** and **logo30** or **logo13** and **logo14** have similar appearances, the contour-tree representations for them are quite different.

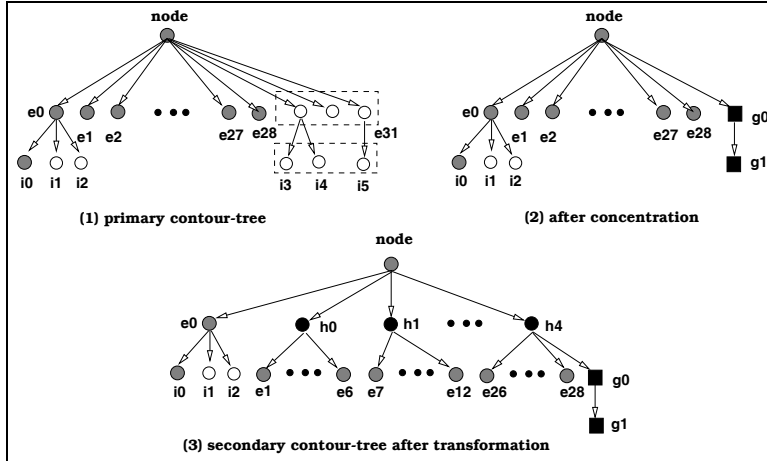


Fig. 5. An example of tree transformation. Shaded circles indicate big nodes, white nodes indicate small nodes, black squares indicate new big nodes created by concentration process, black circles indicate new nodes. Note the pre-defined valence for secondary contour tree is 6.

Source images were randomly distorted by using a program inspired by Baird’s image defect model [14]. The noise simulation includes rotation within a range of $[-40^\circ, 40^\circ]$, kerning in the axis X and Y with values in $(0, 1]$, optical blurring due to scanning process which is modeled as a circularly symmetric Gaussian filter with mean value 0.25 and variance 0.25. We generated 200 instances for each logo class, half of them was used for net learning and the other half for accuracy estimation. In total we generated in this way a dataset \mathcal{D} of 8000 instances (200 distorted images for each logo) that was split into a learning set \mathcal{D}_l and a test set \mathcal{D}_t , both containing 4000 instances. These data were processed by the contour extraction algorithm to obtain primary contour trees. Two types of secondary contour trees were obtained by running Algorithm 1 with $m = 6$ and $m = 8$, resulting in four datasets $\mathcal{D}_l^{(6)}, \mathcal{D}_t^{(6)}, \mathcal{D}_l^{(8)}, \mathcal{D}_t^{(8)}$. In order to compare recursive neural networks to MLPs and auto-associators, we also generated unstructured data by sub-sampling images in \mathcal{D} to fit a 16×16 grid, as in [5]. The resulting sets \mathcal{D}_l^u and \mathcal{D}_t^u were used to train and test both MLP and autoassociator models.

Table 1 summarizes the results obtained using MLP, auto-associators, and recursive neural nets on 6-ary and 8-ary secondary tree representations. The selection of the maximum permitted outdegree m depends on real applications through a few experiments. The reasons of doing so is that high number of outdegree can result very complex net structure and low outdegree can decrease the generalization capability of network by deforming the representation of logo structures. For these reasons, the value of permitted outdegree, depending on the population of logo database under study, should be properly selected through experimentations. Recursive nets had 7 state units. The state transition function

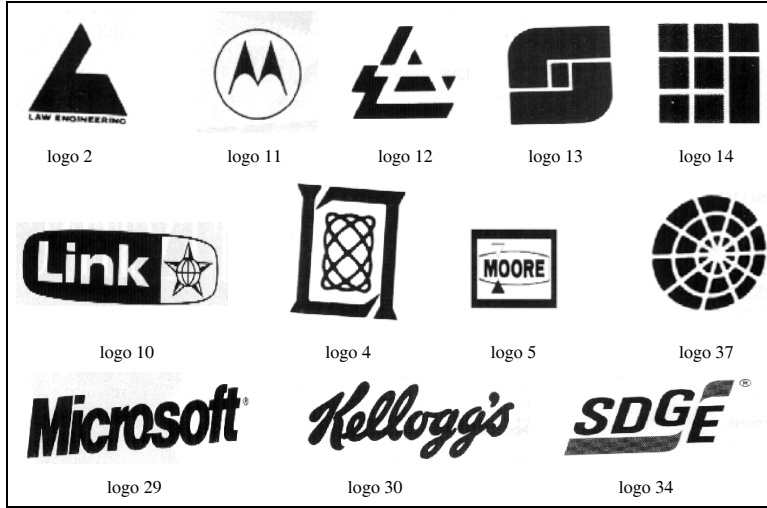


Fig. 6. Some logo examples used in our experiments.

f was realized by a feed-forward net having $7m + 7$ inputs, 25 hidden units and 7 outputs. The output function g was realized by a feed-forward network having 7 inputs, 25 hidden units and 6 output units for encoding the class label (binary coding was used). The total number of free parameters was 1406 for $m = 6$ and 1756 for $m = 8$. The MLP had 256 inputs, 25 hidden units, and 6 output units, resulting in 6581 free parameters. Each of the 40 auto-associators had 256 inputs (and outputs) and 25 hidden units, resulting in 13081 free parameters for each net (about $0.5 \cdot 10^6$ parameters for the whole recognition system). The number of hidden units is selected through experiments in order that the learning examples can be learnt properly. We used batch mode and random initialization of weights for the learning of all the networks.

In the case of recursive neural networks and MLP, classification is based on the distance $D^{(\kappa)} = \|\mathbf{y} - \mathbf{t}^{(\kappa)}\|^2$ between the output of the network, \mathbf{y} , and the coding of each class, $\mathbf{t}^{(\kappa)}$. The class is the κ for which $D^{(\kappa)}$ has the minimum value. If $D_{min}^{(\kappa)}$ is greater than a threshold, then the pattern is rejected. For the auto-associator, the classification is based on the input-output error for each network. The minimum error corresponds to the class most similar to the pattern, that is rejected when the minimum error is greater than a predefined threshold [5].

The experimental results are reported with different neural network paradigms in terms of CPU time for training and recognition, estimated accuracy and rejection rate (see detailed caption of Table 1). By comparing the experimental results obtained using recursive neural networks and MLP, it can be observed that recursive networks achieve a better generalization capability to the test set than MLP.

Network	CPU time		Number of epochs	Estimated accuracy %		Rejection threshold	Rejection rate(%)
	Training (hours)	Recognition (ms)		Learn set	Test set		
MLP	47.00	6	10000	99.90	96.45	0.5	2.00
Auto-associators	71.67	300	200	99.48	99.20	2.0	0.00
Recursive (6)	130.60	4	20000	99.75	99.15	0.5	0.25
Recursive (8)	137.56	4	20000	99.75	99.32	0.5	0.15

Table 1. Experimental results. Recursive (6) and Recursive (8) indicates recursive neural networks with maximum out-degree 6 and (8) respectively. The learning time for auto-associators is the sum of training time for all the auto-associators. The accuracies % for Learn and Test are the percentages of the correct classified patterns, and Rejection rate % is the percentage of rejected patterns over the total number of patterns for learning or test set. The CPU time of recognition is the time for recognizing one logo pattern.

Since learning recursive neural networks by structured representations is made recursively according to the topology of contour tree, the learning time is much higher than that of MLP and auto-associators. However, recursive neural networks and MLP are extremely advantageous over auto-associators from the viewpoint of recognition time, which is a crucial aspect in designing effective logo recognizer. On the other hand, the recursive neural networks with different valences appear to have very similar performances. The higher valence network shows a slightly better recognition performance than the other one. One straightforward explanation is that the higher valence network has higher dimension of architecture as a side-effect of maintaining more structured information. Consequently, learning of lower valence network is faster and higher valence network is more accurate.

The autoassociator-based classifier seems to perform very well with very lower uncertainty. However one drawback is that much more time is needed in the recognition phase, but the advantage is the modularity of recognizer [5].

The rejection rate of recursive network is much lower than that of MLP, hence giving rise to a recognition system with high confidence.

5 Discussion and Conclusion

We have presented a recursive neural networks based technique for logo image classification. The experimental results show that recursive nets are able to learn from structured representation (contour-tree) of logo images. Such logo representation has never been used before in conjunction with sub-symbolic learning models.

It can be also noted that the recursive neural networks can learn contour-trees with either shallow structures or deep structures. In the case of shallow structure, the representation of logo might be only one node or with depth of 1. If there is only one node, the recursive neural network becomes a feedforward neural network with the associated feature vector as input. Usually, logo images

are represented with nested contour trees, this representation is suitable to be processed by recursive neural networks. Although too deep structure can come unstuck, logo images usually do not generate too deep contour-trees.

Naturally, there are also other kind of representations of logo, such as Line Adjacency Graph (LAG) [12] or quad-tree. However, preliminary experimental results show us that the representation of logo components by using LAG has a larger number of nodes than contour-tree description. Therefore, learning of recursive net using LAG can be harder than that using contour-tree. Additionally, LAG can be sensitive to object rotation. Quad-tree can be used to represent logo by fitting logo with 2×2 frame recursively to a certain depth. But it seems that local features of logo are missed and it is likely to be too sensitive to logo rotation.

The strength of the recursive neural approach lies in the integration of structured representation and recursive net learning without explicitly transforming logo structures into fixed-size vectorial representations. Another strength of the approach is the robustness of the recognition process obtained by using local features of logo contours as inputs of neural model. This is analogous to the method proposed in [2], which used local invariants obtained through contour signature extraction.

On the contrary, the weakness of the recursive neural network arises from the learning of *similar* logos, e.g. logos with few difference in their text contents. Our dataset does not include such kind of similar logos, however, it can be foreseen that distinguishing those logos may need apply character recognition techniques, such as proposed in [2]. Another concern using the approach proposed in this paper approach comes from the fact that logo representations might have same structures and close feature vectors, such as some examples used in [3]. In this case, other structured representations and features are desired. Furthermore, when dealing with veryhigh number of classes, the proposed approach can be applied on subsets of whole set of classes. Each subset of classes can be derived from a clustering procedure.

One further developments also include the introduction of a non-stationary recursive network models, in which the state transition function may vary from node to node, possibly also taking a variable number of arguments. Such an extension might render superfluous the pruning algorithm we have described in Section 3.2.

We are currently investigating the possibility of applying the proposed technique to handwritten character recognition, using the line adjacency graph representations of characters.

Acknowledgment: We thank Marco Maggini (DII, Università di Siena) for useful suggestions and his software assistance in the development of BPTS. We thank the research groups at University of Maryland for making their logo databases available on Internet. We also appreciate the reviewers for their useful comments on the paper.

References

1. F. Cesarini, E. Francesconi, M. Gori, S. Marinai, J. Q. Sheng, and G. Soda, "Rectangle labelling for an invoice understanding system," in *Proceedings of the International Conference on Document Analysis and Recognition*, pp. 324–330, Ulm, Germany, August 18–20 1997.
2. D. Doermann, E. Rivlin, and I. Weiss, "Applying algebraic and differential invariants for logo recognition," *Machine Vision and Applications*, vol. 9, no. 2, pp. 73–86, 1996.
3. G. Cortelazzo, G. A. Mian, G. Vezzi, and P. Zamperoni, "Trademark shapes description by string-matching techniques," *Pattern Recognition*, vol. 27, no. 8, pp. 1005–1018, 1994.
4. K. Hachimura, "Decomposition of hand-printed patterns," *Proceedings of the International Conference on Pattern Recognition*, pp. 417–420, 1992.
5. F. Cesarini, M. Gori, S. Marinai, and G. Soda, "A hybrid system for locating and recognizing low level graphic items," in *Graphics Recognition* (R. Kasturi and K. Tombre, eds.), pp. 135–147, LNCS, Vol. 1072, Springer Verlag, March 1996.
6. F. Cesarini, E. Francesconi, M. Gori, S. Marinai, J. Sheng, and G. Soda, "A neural based architecture for spot-noisy logo recognition," in *Proceedings of the International Conference on Document Analysis and Recognition*, pp. 175–179, Ulm, Germany, August 18–20 1997.
7. A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Transaction on Neural Networks*, vol. 8, no. 3, pp. 714–735, 1997.
8. J. Thacher, "Tree automata: An informal survey," in *Currents in the Theory of Computing* (A. Aho, ed.), pp. 143–172, Englewood Cliffs: Prentice-Hall Inc., 1973.
9. R. C. Gonzalez and M. G. Thomason, *Syntactic Pattern Recognition, An Introduction*. Reading, Massachusettes: Addison Wesley, 1978.
10. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing*, vol. 1, ch. 8, pp. 318–362, Cambridge: MIT Press, 1986.
11. R. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," *Backpropagation: Theory, Architectures, and Applications*, In Y. Chauvin and D.E. Rumelhart (Eds.), Lawrence Erlbaum Associates, 1995.
12. T. Pavlidis, *Algorithms for Graphics and Image Processing*. Computer Science Press, 1982.
13. W. McKee and J. Aggarwal, "Computer recognition of partial views of curved objects," *IEEE Trans. Computers*, vol. 26, no. 8, pp. 790 – 800, 1977.
14. H. S. Baird, "Document image defect models," in *Structured Document Image Analysis* (H.S. Baird, H. Bunke and K. Yamamoto eds.), pp. 546–556, Springer Verlag, 1992.