ECAI 2006
G. Brewka et al. (Eds.)
IOS Press, 2006
© 2006 The authors. All rights reserved.

Graph Neural Networks for Object Localization

Gabriele Monfardini and **Vincenzo Di Massa** and **Franco Scarselli** and **Marco Gori**[1]

Abstract. Graph Neural Networks (GNNs) are a recently proposed connectionist model that extends previous neural methods to structured domains. GNNs can be applied on datasets that contain very general types of graphs and, under mild hypotheses, they have been proven to be universal approximators on graphical domains. Whereas most of the common approaches to graphs processing are based on a preliminary phase that maps each graph onto a simpler data type, like a vector or a sequence of reals, GNNs have the ability to directly process input graphs, thus embedding their connectivity into the processing scheme. In this paper, the main theoretical properties of GNNs are briefly reviewed and they are proposed as a tool for object localization. An experimentation has been carried out on the task of locating the face of a popular Walt Disney character in comic covers. In the dataset the character is shown in a number of different poses, often in cluttered backgrounds, and in high variety of colors. The proposed learning framework provides a way to deal with complex data arising from image segmentation process, without exploiting any prior knowledge on the dataset. The results are very encouraging, prove the viability of the method and the effectiveness of the structural representation of images.

1 Introduction

Object localization [4, 3], image classification [7], natural language processing [14, 6], bioinformatic [2], Web page scoring, social networks analisys [16] and relational learning are examples of machine learning applications where the information of interest are encoded into the relationships between a set of basic entities. In those domains, data is suitably represented as sequences, trees, and, more generally, directed or undirected graphs. In fact, nodes can be naturally used to denote concepts with edges specifying their relationships. For instance, an image can be represented by a Region Adjacency Graph (RAG), where the nodes denote the homogeneous regions of the image and the edges represent their adjacency relationship (Fig. 1).

In those applications, the goal consists of designing a function τ that maps a graph $\boldsymbol{G}$ and one of its nodes n to a vector of reals $\tau(\boldsymbol{G}, n)$. For example, the localization of an object in a image can be implemented by a function τ that classifies the nodes of the RAG according to whether the corresponding region belongs to the object or not. In the case of Fig. 1, where the object to be localized is the plane, $\tau(\boldsymbol{G}, n)$ might be 1 for the black nodes, which correspond to the parts of the plane, and -1 otherwise.

Traditional methods usually deal with graphs using a preprocessing procedure that transforms the graphs into simpler representations, as vectors or sequences of reals, which are then elaborated with common machine learning techniques. However, valuable information may be lost during the preprocessing and, as a consequence, the application may suffer from a poor performance and generalization.

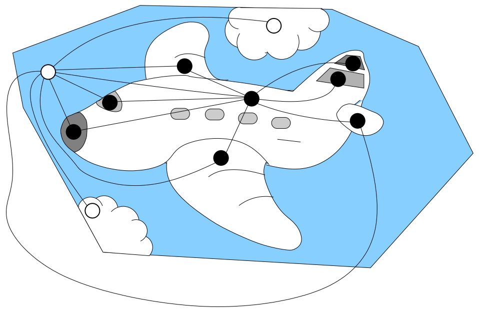


Figure 1. An image and its graphical representation by a RAG. Black nodes are associated to the parts of the plane, white nodes to the background.

Recently, a new connectionist model, called Graph Neural Network (GNN) model, has been proposed to better exploit the graphical representation of data [9, 20]. In GNNs, the preprocessing procedure is not required since the relationships between the basic entities of the input domain are directly embedded into the processing scheme of the model. It has been proved that, under mild hypotheses, GNNs are universal approximators on graphical domains, i.e. the can approximate any function $\tau(\boldsymbol{G}, n)$.

GNNs extend the previous connectionist methods for structured domains [11] like Recursive Neural Networks (RNNs) [21, 8], and SOM for structured domains (SOM–SD) [10]. Actually, whereas RNNs and SOM-SD were limited by the fact that they can be applied only on a restricted class of structures, i.e. the directed acyclic graphs, GNNs can virtually elaborate any type of graphs, including cyclic and acyclic, directed and undirected ones. Moreover, the original RNN model produces one output for each graph and can be applied only on those problems where $\tau(\boldsymbol{G}, n)$ does not depend on the node n. Thus, for example, RNNs are suitable for image classification, but not for object localization[2].

However GNNs had not been widely validated experimentally since now and it was unknown whether they could be applied with success on practical problems. In this paper, the results on an object localization problem are presented. The task consists of locating the face of a popular Walt Disney character in comic covers. Each cover was segmented and represented by a RAG, with labeled nodes and edges.

[1] Università degli Studi di Siena, Dip. Ingegneria dell'Informazione, Italy, email: {monfardini,dimassa,franco,marco}@dii.unisi.it

[2] More precisely, object localization in images can be carried out even by RNNs, but, in this case, the input RAGs must undergo a preprocessing step that transforms RAGs to direct acyclic graphs [4] (see Sect. 3).

In comic covers the character is shown in a number of different poses, often in cluttered backgrounds, and in high variety of colors. The proposed learning framework provides a way to deal with complex data arising from image segmentation process, without exploiting any prior knowledge on the dataset.

Moreover, common approaches to object localization are usually based on a window which is moved over the image [23, 22]. At each step of such a procedure, a classification technique is used to verify whether the window currently contains the target object. On the other hand, GNNs process the whole image representation once and for all, exploiting the graphical structure encoded naturally in the RAG.

The paper is organized as follows. In the next section, the GNN model and its main properties are reviewed. Section 3 illustrates the object localization problem and shows the results obtained by GNNs. Finally, the conclusions are drawn in Sect. 4.

2 Graph Neural Networks

In the following, a *graph* $\boldsymbol{G}$ is a pair ($\boldsymbol{N}$,$\boldsymbol{E}$), where $\boldsymbol{N}$ is a set of *nodes* (or vertices), and $\boldsymbol{E}$ is a set of *edges* (or arcs) between nodes: $\boldsymbol{E} \subseteq \{(u,v)|u,v \in \boldsymbol{N}\}$. If the graph is *directed*, each edge (u,v) has a direction, i.e. a head and a tail, whereas, in *undirected* graphs, the ordering between u and v is not defined, i.e. $(u,v) = (v,u)$. Nodes and edges may be labeled and labels are represented by $\boldsymbol{l}_n$ and $\boldsymbol{l}_{(u,v)}$, respectively. By convention, the symbol $\boldsymbol{l}$ with no further specifications defines all the labels of the graph, while operator $|\cdot|$ denotes the cardinality or the absolute value of its argument, according to whether it is applied to a set or to a number. A graph is called *acyclic* if there is no path, i.e. a sequence of connected edges, that starts and ends in the same node, otherwise it is *cyclic*. Finally, the nodes adjacent to some node n (or neighbors of n) are those connected to it by an edge and are represented by ne$[n]$.

2.1 The GNN model

The GNN model has been conceived to approximate a target function τ that maps a graph $\boldsymbol{G}$ and one of its nodes n into an output vector $\boldsymbol{o}_n = \tau(\boldsymbol{G}, n)$. In categorization problems $\boldsymbol{o}_n \in \mathbb{N}^m$, while in regression problems the outputs belong to $\mathbb{R}^m$.

In GNN framework, a node represents an object or a concept of the domain of interest. Each node can be described using a fixed-length vector of real numbers $\boldsymbol{x}_n \in \mathbb{R}^s$ called *state*, where the dimension s is a predefined parameter of the model. In order to derive a flat but adaptive description of each node n in a distributed processing scheme, the state must be evaluated locally at each node. The goal here is to combine label symbolic information with subsymbolic contributions embedded in graph topology. Such a task is accomplished modeling $\boldsymbol{x}_n$ as the output of a parametric function $f_{\boldsymbol{w}}$ (called *state transition function*), that depends on the node label $\boldsymbol{l}_n$ and on the relationships between n and its neighbors

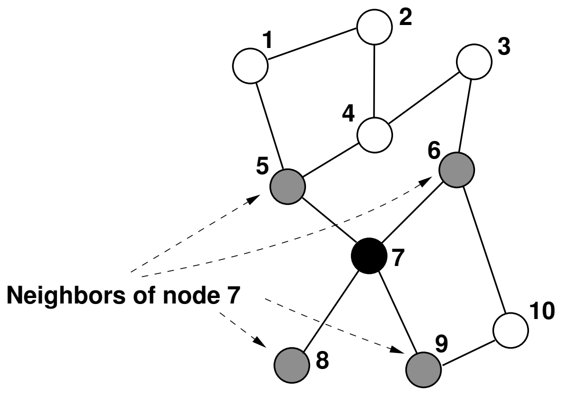
$$\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}, \boldsymbol{l}_{(n,\mathrm{ne}[n])}), \quad n \in \boldsymbol{N}, \tag{1}$$

where ne$[n]$ is the set of neighbors of node n. Here, $\boldsymbol{x}_{\mathrm{ne}[n]}$ and $\boldsymbol{l}_{\mathrm{ne}[n]}$ are the states and the labels of the nodes in ne$[n]$, respectively, and $\boldsymbol{l}_{(n,\mathrm{ne}[n])}$ collects the labels of the edges connected to n (Fig. 2).

For each node n, the state $\boldsymbol{x}_n$ is also combined with node label $\boldsymbol{l}_n$ to produce an output vector $\boldsymbol{o}_n$

$$\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n), \quad n \in \boldsymbol{N}, \tag{2}$$

where $g_{\boldsymbol{w}}$ is another parametric function, called *output function*.



$$\boldsymbol{x}_7 = f_{\boldsymbol{w}}(\boldsymbol{l}_7, \boldsymbol{x}_5, \boldsymbol{x}_6, \boldsymbol{x}_8, \boldsymbol{x}_9, \boldsymbol{l}_5, \boldsymbol{l}_6, \boldsymbol{l}_8, \boldsymbol{l}_9, \boldsymbol{l}_{(7,5)}, \boldsymbol{l}_{(7,6)}, \boldsymbol{l}_{(7,8)}, \boldsymbol{l}_{(7,9)})$$

Figure 2. The state $\boldsymbol{x}_n$ depends on the information in its neighborhood.

In conclusion, Eqs. (1) and (2) define a method to compute an output $\boldsymbol{o}_n = \varphi_{\boldsymbol{w}}(\boldsymbol{G}, n)$ for each node n of the graph $\boldsymbol{G}$, taking into account the local descriptions and the relationships of all the nodes in $\boldsymbol{G}$. The goal of the learning procedure is to adapt the parameters $\boldsymbol{w}$ so that $\varphi_{\boldsymbol{w}}$ approximates the data in the learning set $\mathcal{L} = \{(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{n_{i,j}})| 1 \le i \le p, 1 \le j \le q_i\}$, where each triple $(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{n_{i,j}})$ denotes a graph $\boldsymbol{G}_i$, one of its nodes $n_{i,j}$ and the desired output $\boldsymbol{t}_{n_{i,j}}$. Moreover, p is the number of graphs in $\mathcal{L}$ and q_i is the cardinality of the set of supervised nodes in graph $\boldsymbol{G}_i$ In practice, the learning problem can be implemented as the minimization of the quadratic error function

$$e_{\boldsymbol{w}} = \sum_{i=1}^{p} \sum_{j=1}^{q_i} (\boldsymbol{t}_{n_{i,j}} - \varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}))^2. \tag{3}$$

Notice, however, that the following three issues must be solved in order to implement the proposed model.

1. How can the transition function $f_{\boldsymbol{w}}$ and the output function $g_{\boldsymbol{w}}$ be implemented?
2. The state $\boldsymbol{x}_n$ should be uniquely defined, but Eq. (1) provides a recursive specification of $\boldsymbol{x}_n$. Thus, how can be ensured that Eq. (1) has a unique solution?
3. How can the parameters be adapted to minimize the error function?

In GNNs, both $f_{\boldsymbol{w}}$ and $g_{\boldsymbol{w}}$ can be implemented by two feedforward neural networks. However, the fact that $f_{\boldsymbol{w}}$ doesn't have a fixed number of input parameters, as different nodes have different numbers of neighbors (see Eq (1)), raises some practical problems. In fact, the network implementing $f_{\boldsymbol{w}}$ should be designed to accommodate the maximal number of input parameters, even if several of them may remain unused in most of the cases. A better solution consists of specializing Eq. (1)

$$\boldsymbol{x}_n = \sum_{i=1}^{|\mathrm{ne}[n]|} h_{\boldsymbol{w}}\left(\boldsymbol{l}_n, \boldsymbol{x}_{\mathrm{ne}_i[n]}, \boldsymbol{l}_{\mathrm{ne}_i[n]}, \boldsymbol{l}_{(n,\mathrm{ne}_i[n])}\right), \quad n \in \boldsymbol{N} \tag{4}$$

where ne$_i[n]$ is the i-th neighbor of n, $|\mathrm{ne}[n]|$ is the number of neighbors of n, and $h_{\boldsymbol{w}}$ is parametric function implemented by a feedforward neural network. Here, the idea is that $\boldsymbol{x}_n$ can be evaluated as a sum of "contributions", one for each of the neighbors of node n.

In order to solve the second issue, notice the set of Eqs. (1) can be also represented as

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix} = \begin{bmatrix} f_{\boldsymbol{w}}(\boldsymbol{l}_1, \boldsymbol{x}_{\mathrm{ne}[1]}, \boldsymbol{l}_{\mathrm{ne}[1]}, \boldsymbol{l}_{(1,\mathrm{ne}[1])}) \\ f_{\boldsymbol{w}}(\boldsymbol{l}_2, \boldsymbol{x}_{\mathrm{ne}[2]}, \boldsymbol{l}_{\mathrm{ne}[2]}, \boldsymbol{l}_{(2,\mathrm{ne}[2])}) \\ \vdots \\ f_{\boldsymbol{w}}(\boldsymbol{l}_N, \boldsymbol{x}_{\mathrm{ne}[N]}, \boldsymbol{l}_{\mathrm{ne}[N]}, \boldsymbol{l}_{(N,\mathrm{ne}[N])}) \end{bmatrix}$$

where N is the cardinality $|\boldsymbol{N}|$ of node set $\boldsymbol{N}$. Introducing a vectorial function $\boldsymbol{F}_{\boldsymbol{w}}$ obtained stacking together the instances of the transition function $f_{\boldsymbol{w}}$, collecting all the states in the symbol $\boldsymbol{x}$ and all the labels in the symbol $\boldsymbol{l}$, the previous equation can be rewritten as

$$\boldsymbol{x} = \boldsymbol{F}_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})\,. \tag{5}$$

By Banach Theorem, if $\boldsymbol{F}_{\boldsymbol{w}}$ is a contraction mapping[3], then Eq. (5) has a unique solution [13]. Thus, the second issue can be solved by designing $f_{\boldsymbol{w}}$ such that the global function $\boldsymbol{F}_{\boldsymbol{w}}$ results to be a *contraction mapping* w.r.t. the state $\boldsymbol{x}$.

In practice, this goal can be achieved by adding a penalty term to the error function

$$e_{\boldsymbol{w}} = \sum_{i=1}^{p} \sum_{j=1}^{q_i} (\boldsymbol{t}_{n_{i,j}} - \varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}))^2 + \beta L\left(\left\| \frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{x}} \right\|\right),$$

where $L(y) = (y - \mu)^2$, if $y > \mu$, and $L(y) = 0$ otherwise [20]. Moreover, the real number β balances the importance of the penalty term and the error on patterns, the parameter $\mu \in (0, 1)$ defines a desired upper bound on the contraction constant of $F_{\boldsymbol{w}}$, and $\|\frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{x}}\|$ denotes the norm of the Jacobian of $F_{\boldsymbol{w}}$.

Next section answers briefly to the last issue, discussing the learning algorithm used to adapt model parameters. More details can be found in [20].

2.2 The learning algorithm

The functions $h_{\boldsymbol{w}}$ and $g_{\boldsymbol{w}}$ may be implemented by a variety of feedforward neural network architectures. In our experiments, they are designed as three layered (i.e. with one hidden layer) feedforward neural networks. According to Banach fixed point theorem, if ρ is a generic contraction mapping, then the dynamical system $\boldsymbol{x}(t+1) = \rho(\boldsymbol{x}(t))$, where $\boldsymbol{x}(t)$ denotes the t-th iterate of $\boldsymbol{x}$, converges exponentially fast to the solution of the equation $\boldsymbol{x} = \rho(\boldsymbol{x})$ for any initial state $\boldsymbol{x}(0)$. Thus Eq. (5) can be solved simply by iteration. Then, fixed point states are used by function $g_{\boldsymbol{w}}$ to evaluate the outputs $\boldsymbol{o}_n$ (Eq. (2)), while the weight updating is accomplished deriving the error according to Eq. (3) and employing a gradient descent strategy as in traditional backpropagation algorithm.

The main difference with respect to the traditional backpropagation is due to the fact that fixed point states depend on the parameters of $f_{\boldsymbol{w}}$ through the iteration procedure. For this reason the error should be backpropagated from one iteration to the previous, accumulating the gradient until convergence. The adopted gradient computation is not completely new: it combines the backpropagation through structure algorithm, designed to train Recursive Neural Networks [21, 8], and the recurrent backpropagation algorithm [1, 18].

[3] A generic function $\rho : \mathbb{R}^n \to \mathbb{R}^n$ is a contraction mapping w.r.t. a vector norm $\|\cdot\|$, if it exists a real number μ, $0 \le \mu < 1$, such that for any $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n$, $\|\rho(\boldsymbol{x}_1) - \rho(\boldsymbol{x}_2)\| \le \mu\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$.

When the gradient is available, the weights can be updated using resilient backpropagation [19], a well-known variant of standard backpropagation that speeds up the convergence.

3 Application to object localization

In order to test the performance of the Graph Neural Network model, an object localization problem was designed. It consists of finding the face and the hat of a famous Walt Disney character, Uncle Scrooge, in a set of cartoon covers taken from two different Italian publications. The full dataset comprises 319 images, all of which show the character in various sizes and poses. Some pictures have multiple copies of the face, with or without the hat, whereas other images also show the faces of other Walt Disney characters, that are quite difficult to distinguish from Uncle Scrooge, f.i. Donald Duck. or other ducks. A training set, a validation set and a test set have been built randomly using, respectively, 127, 64 and 125 images from the dataset.

Images were encoded into Region Adjacency Graphs (RAGs). RAGs [17] are a classical image representation, where nodes stand for homogeneous regions and edges encode the relationship "is adjacent to" (Fig. 1). Since adjacency is a symmetric relationship, the obtained graphs are undirected. Previous attempts to elaborate RAGs with Recursive Neural Networks [4, 3], showed encouraging results, even if RAGs had to be transformed into acyclic graphs, because of the limitations of Recursive Neural Networks. On the contrary, GNNs are able to directly process undirected graphs, so no preprocessing was needed.

The RAGs have labels on both nodes and edges: node labels containe features of the corresponding image regions, while edge labels store information on adjacent regions pairs[4] (Fig. 3). Tab. 1 lists all the features in detail. Each node n also has a manually assigned tar-

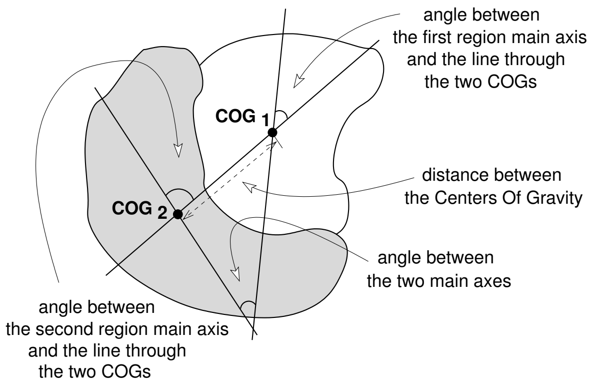


Figure 3. The geometric features stored in each edge label. COG is an abbreviation for "center of gravity".

get $\boldsymbol{t}_n$ which is positive for regions belonging to Uncle Scrooge's face or hat and negative elsewhere. The targets were assigned using a graphical interface that shows the segmented image and allows to select the regions that composes each object.

In order to obtain the homogeneous regions a segmentation procedure composed by three steps was adopted. In the first, each image was filtered using the Mean Shift algorithm [5], that acts like a low–pass filter in a mixed 5D feature space composed by the tristimulus

[4] The used features have been computed using the LTI–Lib library [15], developed by the Chair of Technical Computer Science, RWTH-AachenUniversity, and released under the GNU Lesser General Public License.

Table 1. Features contained in node and edge labels

Label type	Feat. type	Description
Node label	color	average color
	geometric	area
		perimeter
		coordinates of the center of gravity (COG)
		coordinates of the bounding box
		compactness: $4 * \pi * area/(perim.)^2$
		eccentricity
		main axis orientation
		inertia parall. and orthog. to main axis
	moments	Several central moments m_{xy} of various orders w.r.t. x and y: $m_{02}, m_{03}, m_{11}, m_{12}, m_{20}, m_{21}, m_{30}$
Edge label	color	RGB–color distance between the average colors of the two regions
	geometric	distance between the two COGs
		angle between the main axes
		angle between the main axis of the first region and the line through the COGs
		angle between the main axis of the second region and the line through the two COGs

values in CIE–Luv color space and the bidimensional image coordinates (x,y). The main advantage of this technique is that it is especially designed to preserve gradient discontinuities. Thus smallest textures, often oversegmented by techniques employing only color information, are heavily smoothed, while significant edges remain almost crisp, easing region boundaries detection. This filtering step plays a key role in removing the noise introduced by image digitalization and JPEG compression[5].

Then, a k–means [12] color quantization procedure was adopted to locate an initial large number of homogeneously colored regions. Finally, pairs of adjacent regions were recursively merged, using a heuristic function that evaluated similarities in order to choose the best pair. Such a procedure was repeated until the desired final number of regions was achieved. The optimal number of regions was automatically defined using a function that extimates the image complexity through the average value of the luminance gradient.

Several tests were performed using GNNs with three different architectures. More precisely, the state dimensions s and the number of hidden units r in the neural networks that implement the functions f_w and g_w were varied[6]. In the first GNN architecture, $s = 3$ and $r = 3$ hold, in the second architecture $s = 5$ and $r = 5$, and in third architecture $s = 10$ and $r = 10$.

To evaluate the importance of the subsymbolic information and the effectiveness of the structural representation of objects, the GNN model was compared with a three layered feedforward neural networks (MLP), that solved the problem using only the node labels as input patterns. Since the MLPs cannot exploit the graph connectivity, while GNNs can, the performance difference is mostly due to the subsymbolic information embedded in RAG topology.

Table 2 shows the accuracies obtained by the various architectures measured by the percentage of the correctly classified regions.

[5] Mean Shift algorithm was implemented using the LTI–Lib library [15].

[6] For sake of simplicity, in all the experiments the two neural network were three layered feedforward networks with the same number r of hidden neurons.

Table 2. Percent of correctly predicted regions in the positive examples (Object), in the negative (Non-obj.), and on average (Total)

Model	Configuration	Object	Non-obj.	Total
MLP	10 hidden u.	84.39%	68.91%	70.44%
	20 hidden u.	**85.15%**	69.45%	70.90%
	30 hidden u.	84.96%	69.27%	70.73%
GNN	state dimension = 3 3 hidden u. in funct. f 3 hidden u. in funct. g	78.90%	75.97%	74.72%
	state dimension = 5 5 hidden u. in funct. f 5 hidden u. in funct. g	77.29%	78.00%	77.93%
	state dimension = 10 10 hidden u. in funct. f 10 hidden u. in funct. g	78.81%	**78.54%**	**78.56%**

MLPs reached very high accuracies on the class of positive examples and quite low accuracies on the other nodes, while GNNs behaved similarly on the two classes. The reason is that the positive regions have features that belong to a restricted set, while the features of the negative regions have a wider range. Thus, false positives are more frequent than false negatives in MLPs. On the other hands, GNN are able to exploit also the information given the regions adjacent to the possible face, that usually represents the clothes or the hat. Such an information allows to largely decrease the number of false positives. Notice that the unbalanced results obtained by MLPs has a negative impact on the total accuracy of the MLPs w.r.t. the results achieved by GNNs, as clearly shown in Tab. 2. The error function considers both the positive and the negative examples, thus the training tried to force the MLPs to increase the performance on the negative examples, eventually decreasing the accuracy on the positive examples, but such a goal cannot be reached because of the task difficulty.

Since the dimension of the regions greatly varies in the dataset and larger regions are more important than smaller regions in object localization applications, the obtained networks were also evaluated on the base of the number of pixels that have been correctly classified. Table 3 shows the achieved results with accuracy defined as the ratio between the correctly predicted pixels and the total number of pixels of the image.

Table 3. Percent of correctly predicted pixels in the positive examples (Object), in the negative (Non-obj.), and on average (Total)

Model	Configuration	Object	Non-obj.	Total
MLP	10 hidden u.	83.72%	82.68%	82.73%
	20 hidden u.	**86.76%**	84.07%	84.20%
	30 hidden u.	83.92%	84.55%	84.52%
GNN	state dimension = 3 3 hidden u. in funct. f 3 hidden u. in funct. g	76.49%	87.91%	87.38%
	state dimension = 5 5 hidden u. in funct. f 5 hidden u. in funct. g	75.70%	**88.92%**	**88.30%**
	state dimension = 10 10 hidden u. in funct. f 10 hidden u. in funct. g	73.91%	88.56%	87.88%

Comparing the results of Tab. 2 and Tab. 3, we notice that all neural models better classify the largest regions w.r.t. the smallest ones. In fact, it can be shown that the mean size of the mistaken regions is about a half of the mean size of the correctly predicted ones, in all tests and with all architectures. This behavior is probably due to the fact that geometric features are likely to be progressively less significant when the regions become too small.

Finally, the correctly processed images were counted. In this case, an image was defined as correctly processed if the number of correctly classified pixels exceeded a given threshold. Table 4 shows the achieved accuracies using a 80% threshold and a 90% threshold. The

Table 4. Percent of images where the total pixel–based accuracy exceeded 80% and 90%

Model	Configuration	Acc. $\geq$ 80%	Acc. $\geq$ 90%
MLP	10 hidden u.	75.59%	17.32%
	20 hidden u.	76.38%	19.69%
	30 hidden u.	78.74%	20.47%
GNN	state dimension = 3 3 hidden u. in funct. f 3 hidden u. in funct. g	83.46%	41.73%
	state dimension = 5 5 hidden u. in funct. f 5 hidden u. in funct. g	**90.55%**	51.18%
	state dimension = 10 10 hidden u. in funct. f 10 hidden u. in funct. g	82.68%	**52.76%**

results show that GNNs outperform MLPs beacuse the structural information make GNNs less prone to spurious features. In fact, sets of adjacent positive regions reinforce themselves mutually, and the same happens for each cluster of negative regions.

Finally it is worthy to remark that GNNs succeeded in correctly classifying over the ninety percent of the pixels for more than a half of the images. Since the faced problem was particularly difficult, those results are encouraging and prove that the application of the GNN model on structural domains is viable.

4 Conclusions

This paper has presented the preliminary results obtained by the application of the Graph Neural Network model to an object localization task. The results were encouraging and have shown that the model can really combine the symbolic and the subsymbolic information stored in structured data. Future matter of research includes a more systematic experimentation of the GNN model on object localization problems and the application of the model to other tasks where the information is naturally represented by graphs as relational learning problems and social network applications.

REFERENCES

[1] L.B. Almeida, 'A learning rule for asynchronous perceptrons with feedback in a combinatorial environment', in *Proceedings of the IEEE International Conference on Neural Networks*, eds., M. Caudill and C. Butler, volume 2, pp. 609–618, San Diego, 1987, (1987). IEEE, New York.

[2] P. Baldi and G. Pollastri, 'The principled design of large-scale recursive neural network architectures-dag-rnns and the protein structure prediction problem', *Journal of Machine Learning Research*, **4**, 575–602, (2003).

[3] M. Bianchini, M. Maggini, L. Sarti, and F. Scarselli, 'Recursive neural networks for processing graphs with labelled edges: Theory and applications', *Neural Networks - Special Issue on Neural Networks and Kernel Methods for Structured Domains*, **18**, 1040–1050, (October 2005).

[4] Monica Bianchini, Paolo Mazzoni, Lorenzo Sarti, and Franco Scarselli, 'Face spotting in color images using recursive neural networks', in *Proceedings of the 1st ANNPR Workshop*, Florence (Italy), (2003).

[5] Dorin Comaniciu and Peter Meer, 'Mean shift: A robust approach toward feature space analysis', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**(5), 603–619, (2002).

[6] F. Costa, P. Frasconi, V. Lombardo, and G. Soda, 'Towards incremental parsing of natural language using recursive neural networks', *Applied Intelligence*, **19**(1–2), 9, (July 2003).

[7] E. Francesconi, P. Frasconi, M. Gori, S. Marinai, J.Q. Sheng, G. Soda, and A. Sperduti, 'Logo recognition by recursive neural networks', in *GREC '97: Selected Papers from the Second International Workshop on Graphics Recognition, Algorithms and Systems*, eds., Karl Tombre and Atul K. Chhabra, 104–117, Springer-Verlag, (1998).

[8] P. Frasconi, M. Gori, and A. Sperduti, 'A general framework for adaptive processing of data structures', *IEEE Transactions on Neural Networks*, **9**(5), 768–786, (1998).

[9] M. Gori, G. Monfardini, and F. Scarselli, 'A new model for learning in graph domains', in *Proc. International Joint Conference on Neural Networks (IJCNN2005)*, pp. 729–734, (2005).

[10] M. Hagenbuchner, A. Sperduti, and Ah Chung Tsoi, 'A self-organizing map for adaptive processing of structured data', *IEEE Transactions on Neural Networks*, **14**(3), 491–505, (May 2003).

[11] B. Hammer and J. Jain, 'Neural methods for non-standard data', in *Proceedings of the 12th European Symposium on Artificial Neural Networks*, ed., M.Verleysen, pp. 281–292, (2004).

[12] J. A. Hartigan and M. A. Wong, 'A k–means clustering algorithm', *Applied Statistics*, **28**, 100–108, (1979).

[13] Mohamed A. Khamsi, *An Introduction to Metric Spaces and Fixed Point Theory*, John Wiley & Sons Inc, 2001.

[14] E. Krahmer, S. Erk, and A. Verleg, 'Graph-based generation of referring expressions', *Computational Linguistics*, **29**(1), 53–72, (2003).

[15] LTI-Lib. An object oriented library with algorithms and data structures for image processing and computer vision. Developed at the Chair of Technical Computer Science (Lehrstuhl fuer Technische Informatik) LTI at the Aachen University of Technology.
Available at *http://ltilib.sourceforge.net/*.

[16] M. E. J. Newman, 'From the cover: The structure of scientific collaboration networks', *Proc. National Academy of Sciences*, 404–409, (2001).

[17] T. Pavlidis, *Structural pattern recognition*, Springer, Series in Electrophysics, 1977.

[18] F.J. Pineda, 'Generalization of back–propagation to recurrent neural networks', *Physical Review Letters*, **59**, 2229–2232, (1987).

[19] M. Riedmiller and H. Braun, 'A direct adaptive method for faster backpropagation learning: the rprop algorithm', in *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pp. 586–591, San Francisco, CA, USA, (1993).

[20] F. Scarselli, M. Gori, G. Monfardini, Ah Chung Tsoi, and M. Hagenbuchner, 'A new neural network model for graph processing', Technical Report DII 01/05, Department of Information Engineering, University of Siena, (2005).

[21] A. Sperduti and A. Starita, 'Supervised neural networks for the classification of structures', *IEEE Transactions on Neural Networks*, **8**, 429–459, (1997).

[22] H.C. van Assen, M. Egmont-Petersen, and J.H.C. Reiber, 'Accurate object localization in gray level images using the center of gravity measure: accuracy versus precision', *IEEE Transactions on Image Processing*, **11**(12), 1379–1384, (December 2002).

[23] M.-H. Yang, J. Kriegman, and N. Ahuja, 'Detecting faces in images: A survey', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**(1), 34–58, (January 2002).