2020 25th International Conference on Pattern Recognition (ICPR)
Milan, Italy, Jan 10-15, 2021

TreeRNN: Topology-Preserving Deep Graph Embedding and Learning

Yecheng Lyu, Ming Li, Xinming Huang, Ulkuhan Guler, Patrick Schaumont, and Ziming Zhang
Electrical and Computer Engineering
Worcester Polytechnic Institute
{ylyu, mli12, xhuang, uguler, pschaumont, zzhang15}@wpi.edu

***Abstract*—General graphs are difficult for learning due to their irregular structures. Existing works employ message passing along graph edges to extract local patterns using customized graph kernels, but few of them are effective for the integration of such local patterns into global features. In contrast, in this paper we study the methods to transfer the graphs into trees so that explicit orders are learned to direct the feature integration from local to global. To this end, we apply the breadth first search (BFS) to construct trees from the graphs, which adds direction to the graph edges from the center node to the peripheral nodes. In addition, we proposed a novel projection scheme that transfer the trees to image representations, which is suitable for conventional convolution neural networks (CNNs) and recurrent neural networks (RNNs). To best learn the patterns from the graph-tree-images, we propose TreeRNN, a 2D RNN architecture that recurrently integrates the image pixels by rows and columns to help classify the graph categories. We evaluate the proposed method on several graph classification datasets, and manage to demonstrate comparable accuracy with the state-of-the-art on MUTAG, PTC-MR and NCI1 datasets.**

I. Introduction

Deep graph learning has been attracting increasing research interests in recent years [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]. As a widely used data structure to store the topological features, a graph saves the point features in a node list and their affiliations as node edges. The nodes in a graph are orderless and the affiliations are sparse, which makes it difficult for deep graph learning. Trees are ordered graphs with a clear hierarchy, but they still fail to serve as tensors for network processing. In contrast, images have a tensor-like structure with densely ordered pixels in local regions. Such local regularity is beneficial for fast convolutions and recurrent processing that efficiently and effectively learn the local pattern from pixels within different applications.

Motivation. Existing graph neural networks (GNNs) try to collect the features from adjacent nodes through message passing [8] to perceive local pattern. In GNN kernels, the center node plays the same role as adjacent nodes or is just a bit higher weighted in convolution. Surprisingly, we find that there are few existing works contributing to the hierarchy that integrates the local features to global features. In addition, existing GNN works mainly focus on specialized graph convolution kernels, which cannot benefit from the conventional neural networks. These observations motivate us to address the following question:

How to effectively and efficiently project graphs into an ordered and regularized space so that we can take advantage of pattern extraction in conventional neural networks for graph learning?

Approach. The question above is critical. A bad projection function easily leads to the loss of topological information in a graph with, for instance, misplaced parents and children in a tree. Such topological loss is fatal as it may introduce so much noise that the pattern is completely changed. Therefore, a good graph projection function is the key to ordered and regularized representations of input graphs.

At the system level, the pipeline of our method can be summarized as follows: (1) construct trees from graphs, (2) project the trees into image space, and (3) classify graph-tree-images using TreeRNN, our proposed novel network architecture.

We are motivated by the DeepWalk [13] that generates a batch of ordered node list from a graph. In DeepWalk, a random walk scheme is applied to a graph, which starts from a selected root node and goes through the graph edges for several steps, resulting in a list of nodes it passed. By applying the random walker multiple times starting from the same node, a batch of node lists is constructed and reformed as an image for network processing. However, the random walk fails to guarantee the tree to coverage over all graph nodes, and it fails to stop the walker visiting a node more than once. Those failures in topological-preserving confuse the neural network to encode the graph topology. In contrast, we propose to construct trees, a kind of directed acyclic graph, which guarantee coverage to all graph nodes and no repeated nodes on the tree. Within this projection, nodes are clearly ordered in the tree space, which helps extract the local topological features. To further transfer the tree to a structure that is feasible to conventional CNN and RNN, motivated by the Ordered Neurons [14], we employ a block-view like projection from the tree to image space. The block view image explicitly presents the hierarchy of a tree in the pixel context, which contributes to better feature extraction and classification.

Now the graph-tree-images are ready for conventional CNN and RNN processing. To better take advantage of the order and regularity in graph-tree-images, we propose TreeRNN, a novel RNN architecture that integrates the pixels in the images following the tree structures. Specifically, we employ a vanilla

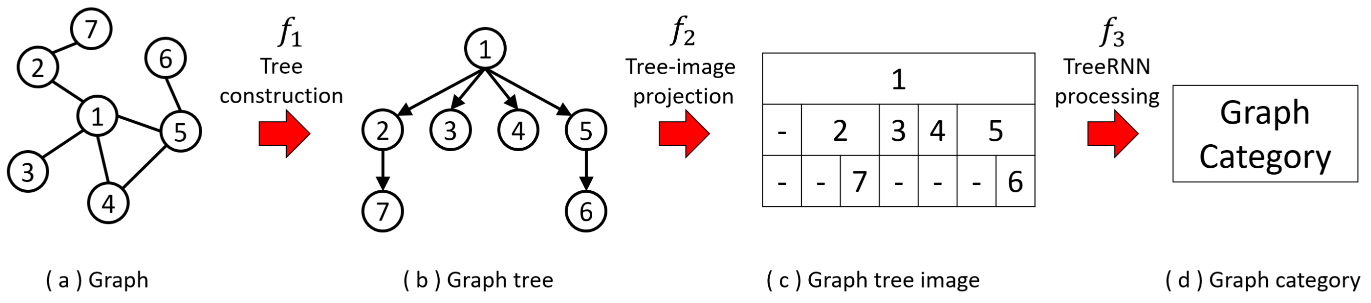


Figure 1. System overview of the proposed graph classification method. (a) a given graph; (b) a tree construct from the corresponding graph; (c) an image representation projected from the graph-tree; (d) graph category predicted by the TreeRNN from the graph-tree-image.

RNN unit and designed a novel network module to achieve 2D recurrent integration on image rows and columns by turns. The pixels in the same row represent the graph nodes on the same layer in the tree, while the pixels in the same column represent the graph nodes connected across the tree layers. By employing this novel network module, we succeed to extract features form graph-tree-images within few parameters, which makes the TreeRNN light-weighted.

Contribution. In summary, our key contributions in this paper are as follows:

- We are the *first*, to the best of our knowledge, to explore the graph-tree-image projection in the context of graph classification.
- We accordingly propose TreeRNN, a novel RNN architecture to process the graph-tree-images that recurrently process on image rows and columns in turns, which implicitly passes through the tree structure.
- We apply the integrated method to the graph classification application and experiment on three graph classification datasets named MUTAG, PTC and NCI1. Our work results in comparable performance with the state-of-the-art works on those benchmark datasets, which demonstrate the success of our graph-tree-image projection scheme and TreeRNN architecture.

II. Related Work

Graph Embedding. Graph Embedding has been studied for decades [15], whose goal is to find a low dimensional representation of the graph nodes in some metric space so that the given similarity (or distance) function is preserved as much as possible. Force-directed graph layout is a series of graph embedding algorithms that try to project a graph to a 2D plane while preserving the distance between graph nodes using a force-directed function. Widely used methods in that series are Kamada-Kawai [16], Fruchterman-Reingold [17] and FM^3 [18]. These algorithms achieved great success in graph visualization. For graph embedding aimed at deep learning, however, those methods result in a high topological disparity on complicated graph embedding. In a recent survey paper [19], a comprehensive understanding of graph embedding techniques is introduced including the problems, techniques, and applications.

In our paper, different from the graph embedding methods that try to represent the graphs in low-dimension space for visualization, we propose an ordered and regular representation of a graph so that conventional convolution kernels and recurrent operators can apply to it.

Tree Construction from Graphs. Tree construction from graphs tries to generate a connected and directed sub-graph with no cycles. Minimum spanning tree (MST) [20] is a kind of tree construction method that generates a tree with a minimum sum of edge weights. Graph tree search is another kind of method that traverses all graph nodes from a selected root node and generates a tree along the path. Depth-first search (DFS) [21] explores as far as possible along each branch before backtracking, while breadth-first search (BFS) [22] explores all of the neighbor nodes at the present depth before moving on to the nodes at the next depth level. Other tree construction methods includes K-MST [23], AVL tree [24] and B-tree [25].

In this paper, we employ the BFS to construct trees from graphs because it covers all the connected graph nodes within the fewest layers, which minimizes the memory allocation of the image representations described in Section IV-B.

Deep Graph Learning. Deep graph learning is an extension of conventional deep learning algorithms to the graph, an orderless and irregular data structure connecting the paired nodes with edges. Graph convolution [1] is a big family of deep graph learning that fuses the local graph subsets by collecting the adjacent nodes' features via message passing and applying a pooling operation (max, sum or average) on them. GCN [2], DGCNN [4], ECConv [3], GraphSAGE [5], GraphConv [7], and GINConv [6] are good examples in this family. Another family of deep graph learning methods embed graphs into other feature spaces followed conventional neural network processing. This family includes DeepWalk [13], DDGK [11], DNGR [10], PSCN [12], LINE [26], M-NMF [27], and WKPI [28]. Graph embedding also works on graph-related data structures *i.e.* Ordered Neurons [14] on trees and Lyu *et al.* [29] on point clouds. In our paper, we follow the graph embedding family and embed the graphs to image space via a graph-tree-image projection.

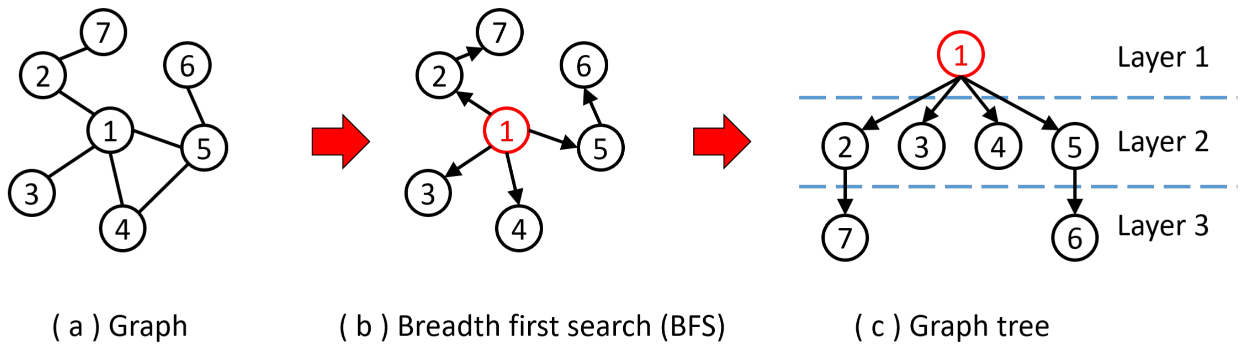


Figure 2. Illustration of function f_1 : Tree construction from a given graph using breath first search. (a) a given graph; (b) apply breadth first search to the graph from its center node; (c) a constructed tree from the graph.

III. System Overview

In this paper, We focus on the problem of graph classification. Generally, let $\mathcal{G}\{V, E, X, Z\}$ denotes a graph where V denotes the set of graph nodes, $E \subseteq (V \times V)$ denotes the set of graph edges, $X \in \mathcal{R}^{|V| \times S_V}$ denotes the set of node features with feature size S_V, and $Z \in \mathcal{R}^{|E| \times S_E}$ denotes the set of edge features with feature size S_E.

In the traditional graph learning setting, we learn a projection function $f : \mathcal{G} \rightarrow \mathcal{Y} \in \mathcal{F}$ that map a graph to one of the semantic labels. In our case, we take three steps to achieve the goal. Firstly, we propose a projection function $f_1 : \mathcal{G} \rightarrow \mathcal{T}$ where $\mathcal{T}$ denotes a tree. The function is aimed to transfer the graph to tree space where the nodes are ordered by directed edges. Secondly, we proposed projection function $f_2 : \mathcal{T} \rightarrow \mathcal{I}$ that further transfer the graph from the tree space $\mathcal{T}$ to image space $\mathcal{I}$, in which each graph node is mapped to the one or multiple pixels. Thirdly and lastly, we learn a neural network classifier $f_3 : \mathcal{I} \rightarrow \mathcal{Y} \in \mathcal{F}_3$ that predicts the graph labels by classifying the graph-tree-images. The neural network classifier learns to minimizing certain loss function $\ell = \ell(f_3(\mathcal{I}), \mathcal{Y})$. The pipeline is illustrated in Figure 1. Please note that f_1 and f_2 are non-trainable functions, and f_3 is learned from its parameter space $\mathcal{F}_3$.

IV. Projection from Graph to Tree and Image Space

A. Tree Construction from Graphs

Tree construction from graphs has been well studied in graph theory. In our work, a tree constructed from a graph denotes a rooted directed acyclic graph (DAG) that contains all graph nodes and a subset of graph edges. A tree representation of a graph have two advantages: (1) it is rooted and directed, which contributes to the context feature extraction by order-sensitive operators such as convolutional kernels and recurrent units; (2) it has no cycles so that every node is visited only once along the tree, which helps eliminate confusion to the graph structure during feature extraction. DeepWalk [13] also generates rooted, and directed subgraphs, however, it allows cycles and even self-loops that results in repeated visits to a node, which makes the feature extractor confused to learn the global features, *i.e. how many nodes are there in the graph?* Figure 2 illustrates the steps to construct a tree from the input graph.

In our work, we employ the breath first search (BFS) to accomplish the first projection function $f_1 : \mathcal{G} \rightarrow \mathcal{T}$. Comparing to the other tree construction methods such as depth first search (DFS) and minimum spanning tree (MST), the BFS traverses the graph within the least depth from the select depth. To minimize the tree depth constructed from the graph, we set the root node to the one with shortest length to its farthest node. The tree construction scheme is described in Algorithm 1. Dijkstra in the algorithm denotes the Dijkstra Algorithm [30] that calculates the distance between each node pairs in the graph.

Algorithm 1 $f_1 : \mathcal{G} \rightarrow \mathcal{T}$ Tree Construction from Graph

```
Input: Graph $G \in \mathcal{G}$
Output: Tree representation $T \in \mathcal{T}$

  $A \leftarrow adjacency\_matrix(G)$;
  $H \leftarrow dijkstra(A)$;
  $root \leftarrow argmin_x(\max_y(H(x, y)))$;
  $T \leftarrow BFS(G, root)$;
  Return $T$
```

B. Projection from Trees to Images

The projection function f_1 constructs a directed and acyclic tree from a general graph. However, the tree structure is still non-feasible for conventional neural network processing. Hence, another projection function $f2 : \mathcal{T} \rightarrow \mathcal{I}$ is proposed to further transfer the tree to an image-like array, which is feasible for network processing.

Given a set of tree T with maximum node size $|V|_{max}$ and maximum depth D_{max}, the projection function f_2 is aimed to map all the nodes in each tree to a fixed-sized image space while preserving its topology. Specifically, there are two topological features we expect to keep: (1) child nodes connected to the same parent node are expected to be connected after projection to image space, and (2) each node is also expected to keep their adjacency to its parent node in the target image space to avoid confusion during network processing, those two adjacency should distinguish to each other. Considering the memory efficiency, we also want to limit the image space to a suitable size.

In our work, inspired by the block view projection in the DeepWalk [13], we propose a similar projection f_2 from tree space to image space. The projection is illustrated in Figure 3. As we see, graph nodes in each layer of the tree occupy a row in the image and each node covers pixels as many as its descendant node size plus one representing itself. Child nodes connected to the same parent node are connected to each other in the same row, while each child node is next to its root node in the same column, which satisfies the expectations we proposed before.

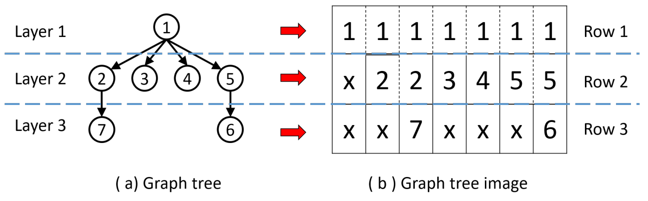


Figure 3. Illustration of f_2 : projection of a graph from tree space to image space. (a) a given graph-tree from function f_1; (b) an image representation projected from the graph tree, Each pixel of the image contains the **node features** of the projected graph node and the **edge features** connecting the node and its parent.

To determine the required image size to store the projected trees, we have to find the maximum rows and columns in need to project the graph set $\mathcal{T}$. According to f_2, the minimum number of rows $N_{row}(T)$ required for input $\mathcal{T}$ equals its tree depth $D(\mathcal{T})$. For the required number of columns $N_{col}(T)$, we introduce the following proposition to determine.

Proposition 1. *Given a tree $\mathcal{T}$ defined in Section IV-A, its required columns $N_{col}(T)$ is not less than to its node size $|V|$, meaning $N_{col}(T) \geq |V(T)|$.*

Proof. Given a tree $\mathcal{T}$, its required columns for the first row (root layer) $n_1 = n(root)$ equals the sum of required columns for its child nodes plus itself, meaning

$$n_1 = n(root) = \sum_{v \in leaf(root)} n(v) + 1. \tag{1}$$

While the set of leaf nodes of root node is exactly the set of nodes in the second row, and root is the only root in the first layer, meaning

$$n_1 = \sum_{v \in leaf(root)} n(v) + 1 = n_2 + |V|_{L_1}. \tag{2}$$

Let us extend the Eqn. 2 to other rows, we have

$$n_i = n_{i+1} + |V|_{L_i}, \tag{3}$$

and for the last row,

$$n_{D(T)} = |V|_{L_{D(T)}}. \tag{4}$$

From Eqn. 3 we conclude:

$$n_i = n_{i+1} + |V|_{L_i} \geq n_{i+1}. \tag{5}$$

Combine Eqn. 2 to 5, we have

$$N_{col}(T) \geq |V|_{L_1} + |V|_{L_2} + \cdots + |V|_{L_{D(T)}} = |V(T)|. \tag{6}$$

□

From Prop. 1 we conclude that the required image size $|\mathcal{I}|$ for graph set $\mathcal{G}$ equals $|V|_{max} \times D_{max}$. A detailed projection function is described in Algorithm 2.

Algorithm 2 $f_2 : \mathcal{T} \rightarrow \mathcal{I}$ Projection from Trees to Images

Input: Tree $T \in \mathcal{T}$, Image space $|V|_{max} \times D_{max}$
Output: Image representation $I \in \mathcal{I}$

```
$L_1 = [root(T)]$
forall $i \in 1, 2, \ldots, D(T)$ do
    $i_{col} = 1$
    $L_{i+1} \leftarrow \varnothing$
    forall $v \in L_i$ do
        if $v \neq \varnothing$ then
            $size_v \leftarrow ChildSize(v)$
            $I(i, i_{col} : i_{col} + size_v) \leftarrow v$
            $i_{col} \leftarrow i_{col} + size_v$
            $L_{i+1} \leftarrow L_{i+1} \cup \{\varnothing, leaf(v)\}$
        else
            $I(i, i_{col} : i_{col}) \leftarrow \varnothing$
            $i_{col} \leftarrow i_{col} + 1$
            $L_{i+1} \leftarrow L_{i+1} \cup \{\varnothing\}$
return $I$
```

V. TreeRNN: a 2D RNN Network on Graph-Tree-Image

The project function f_2 successfully transfers a graph from tree space to image space. In this section, we propose $f_3 : \mathcal{I} \rightarrow \mathcal{Y}$ that extracts the topological features from the graph images and classifies their categories.

There exist several approaches to f_3. We can choose widely used image classifiers such as ResNet [31] and GoogleNet [32] that are powerful, robust to extract features from images and estimate their categories. However, they are designed for images captured by cameras without considering the tree structure implied in the graph-tree-images. In ordered neurons [14] an RNN named ON-LSTM is proposed to learn these context feature by processing the graph image column by column. With the help of its customized activation function, the recurrent unit resets its neurons before it starts the next segments. This RNN structure is specifically designed for tree images and achieved impressive results in the experiments. Unfortunately, the ON-LSTM is designed for a tree within 3 layers, which is difficult to work on deep trees.

TreeRNN. In our work, we propose TreeRNN, a 2D RNN architecture that is optimized for graph-tree-images. The TreeRNN takes existing RNN unit as kernels but works on rows and columns in turns on a 2D feature map. By working along with the rows, the RNN unit goes along with the tree layers to integrates the graph nodes within the same layer; by working along with the columns, it goes across the layers to integrate the nodes with their parents. Algorithm 3 presents the scheme of the proposed TreeRNN. Figure 4 also illustrates

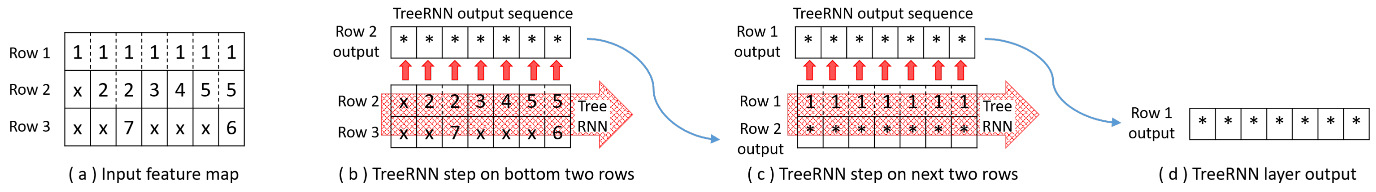


Figure 4. Illustration of TreeRNN operation in f_3 : network architecture. (a) a given image-like feature map from the previous network layer; (b) TreeRNN's first step on the feature map, which integrates pixels in the last two rows and return output sequence; (c) TreeRNN's next step on the feature map, which integrates pixels in the output of the last TreeRNN step and the next row of the feature map; (d) TreeRNN keeps integrating the output row from the last step and the input row from the feature map until it goes through all the rows in the input feature map, and eventually returns an output sequence.

how the RNN unit recurrently process the pixels by rows and columns.

Algorithm 3 f_3_RNN : 2D RNN working scheme

```
Input: Input tensor $In(u,v,c)$ with size $H \times W \times C$, RNN
Output: Output tensor $Out(u,v,c1)$ with size $H \times W \times C1$

$S \leftarrow RNN\,(In(1:2,\ :\ ,\ :))$
forall $u \in 3,4,\ldots,H$ do
    $S \leftarrow RNN\,(Concat(S, In(u,\ :\ ,\ :)))$
$Out \leftarrow S$
return $Out$
```

During processing, the RNN unit has to identify the "brother" nodes sharing the same parent node in the tree and separate them from with "cousin" nodes that share the same grand parent or great-grand parent node. In Ordered Neurons [14] a master activation and a customized LSTM layer are proposed to force clear states after integrating a segment of "brother" nodes. In our work, we write the identification inside the graph-tree-images. As mentioned in Section IV-B, in each row, we reserve a pixel before placing a segment of "brother" nodes, which separates this segment with others. Additionally, the gap between the segments exceeds one pixel if they are "distant relatives". Hence, the relationship between the graph nodes is clearly embedded in the graph images so that it can be easily learned by RNN.

VI. Experiments

A. Experimental Setup

Datasets. We evaluate our method *i.e.* tree construction + image representation + network classifier, on three medium-size datasets for graph classification, namely MUTAG, PTC-MR and NCI1. Table I summarizes some statistics of each dataset.

TABLE I
Statistics of benchmark datasets for graph classification.

Dataset	Num. of Graph	Num. of Class	Avg. Node	Avg. Edge
MUTAG	188	2	17.93	19.79
PTC-MR	344	2	14.29	14.69
NCI1	4110	2	29.87	32.30

Implementation. By default, we design a simple network for f_3 to demonstrate the success of our graph-tree-image projection and TreeRNN. The network is "MLP $\rightarrow$ TreeRNN $\rightarrow$ MaxPool $\rightarrow$ FC", where MLP denotes a point-wised multi-layer perceptron and FC denotes a fully-connected layer. By default, we utilize a point-wised single-layer perceptron with 64 neurons and relu activation for the MLP block, a vanilla RNN unit with 64 neurons for the TreeRNN operator, and FC layer is set to have a softmax activation and an output size of $|\mathcal{Y}|$, the size of categories in the graph set $\mathcal{G}$.

We implement the scheme in a GPU machine with an Intel Core i5-6500 CPU and an NVidia GTX1060 GPU. The implement environment include the following key packages: Python 3.7, Networkx 2.4 [33], and Tensorflow 2.2 [34].

Experimental Scheme. By default, during the experiment on each dataset, we separate the dataset into 10 folds, in which the samples within each category are evenly distributed. At each time we train within 9 folds and test within the last fold. During training, we use Adam [35] with learning rate $lr = 10^{-4}$ as the network optimizer. We then record the best accuracy of each fold, and calculate the mean accuracy as well as the standard deviation.

B. Graph Classification

To do a fair comparison for graph classification, we follow the standard routine, *i.e.* 10-fold cross-validation with a random split. In each fold, we have the same number of samples in each graph category. In this experiment, we apply an end-to-end training scheme that consistently inputs graph samples, generates graph-tree-images with data augmentation and feeds into network training steps. For each fold, we train the network 500 epochs and record the best testing result. In Table II we compare our method with several existing works on graph leaning.

The results show that our method achieves the second best accuracy in all three datasets. To emphasize , our work results in the best accuracy on MUTAG and PTC-MR dataset among the graph embedding solutions. The small variances indicate the stability of our method. In summary, such results demonstrate the success of our method on graph classification.

C. Ablation Study

Effects of Data Augmentation using Grid Layouts on Classification. In order to train the deep classifiers well, the

TABLE II

GRAPH CLASSIFICATION RESULTS (%) IN MUTAG, PTC-MR AND NCI1. NUMBERS IN RED ARE THE BEST IN THE COLUMN, AND NUMBERS IN BLUE ARE THE SECOND BEST.

Category	Method	MUTAG	PTC-MR	NCI1
Graph Convolution	GraphConv [7]	86.1	-	76.2
	GINConv [6]	95.00 ± 4.61	72.94 ± 6.28	80.32 ± 1.73
	ECConv [3]	89.44	-	83.80
	DGCNN [4]	85.83 ± 1.66	58.59 ± 2.47	74.44 ± 0.47
	GIC [9]	94.44 ± 4.30	77.64 ± 6.98	84.08 ±1.07
Graph Embedding	PSCN [12]	88.95 ± 4.37	62.29 ± 5.68	76.34 ± 1.68
	DDGK [11]	91.58 ± 6.74	63.14 ± 6.57	68.10 ± 2.30
	WKPI [28]	85.8 ± 2.5	62.7 ± 2.7	87.5 ± 0.5
	Ours	94.74 ± 5.55	74.69 ± 5.78	84.96 ± 4.81

amount of training data is crucial. In this paper we add two data augmentation: (1) in Algorithm 1 we randomly cut graph cycles during breadth first search (BFS) as illustrated in Figure 5, and (2) in Algorithm 2 we shuffle the $leaf(v)$ so that the leaf nodes are projected to the image in a different order. In Table III, we demonstrate the test performance of network models trained 50 epochs on MUTAG dataset using $1\times$, $6\times$, and $11\times$ data augmentation. In Table III, we observe that the data augmentation significantly increases the classification accuracy on MUTAG. Similar observations have been made for the other datasets.

TABLE III

EFFECTS OF DATA AUGMENTATION ON MUTAG ACCURACY

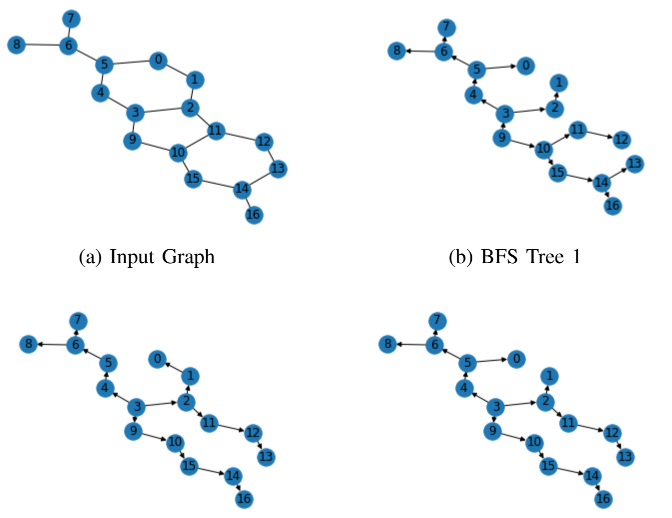
Augmentation (times)	1×	6×	11×
Accuracy (%)	84.62	89.42	92.54

Effects of Graph-Tree-Image Projection. To understand the effectiveness of our proposed projection, we compare the classification results with the DeepWalk [13] image projection in MUTAG dataset using the same network classifier. In this experiment we do not apply any data augmentation. The result shows that our projection gets 84.21% ± 5.12% in accuracy with our network classifier, while the DeepWalk projection with the same image size results in 78.32 % ± 9.51 % using the same classifier. The result indicates that our project better encodes the topological features in the graphs.

TABLE IV

COMPARISON OF MUTAG ACCURACY BETWEEN TREERNN AND OTHER NEURAL NETWORK OPERATORS.

Network Operator	Accuracy (%)	Parameters
MLP	77.81	6,024
2DConv	81.55	38,792
RNN	82.51	42,888
D-RNN	80.91	10,120
TreeRNN	**84.62**	14,216



(a) Input Graph (b) BFS Tree 1

(b) BFS Tree 2 (b) BFS Tree 3

Figure 5. Illustration of several trees constructed using breath first search (BFS). (a) a graph sample from MUTAG dataset. (b,c,d) three BFS trees corresponding to the graph sample.

Effects of TreeRNN. To understand the effectiveness of our proposed TreeRNN as a feature extractor, we compare it with (1) a multi-layer perceptron (MLP), (2) a 2D convolution layer with 3×3 kernel (2DConv), (3) a conventional RNN layer (RNN), and (4) a distributed RNN layer (D-RNN) as described in [36]. All feature extractors are implemented in the same classification network with the same feature size. In this experiment we do not apply any data augmentation. Table IV presents the comparison of various feature extractors in the MUTAG graph classification dataset. We observe that our TreeRNN achieves the best result in accuracy.

VII. CONCLUSION

In this paper, we answer the question positively that graphs can be ordered to trees and further projected to image space so that order-sensitive network operators can benefit from its order and regularity. To this end, we propose a novel graph-tree-image projection, that projects a graph to image space while preserving its topological features between graph nodes. In addition, we propose TreeRNN, a 2D RNN scheme, that integrates the graph images simultaneously along with the tree layers and across the tree layers. In the experiment, we demonstrate that our work has comparable performance to the start-of-the-art in three datasets. As future work, we are interested in applying this method to real-world problems such as point cloud classification and segmentation.

Acknowledgment

Ming Li, Xinming Huang, and Ziming Zhang were supported in part by NSF CCF-2006738.

References

[1] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[3] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3693–3702.

[4] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[6] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[7] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4602–4609.

[8] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1263–1272.

[9] J. Jiang, Z. Cui, C. Xu, and J. Yang, "Gaussian-induced convolution for graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4007–4014.

[10] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[11] R. Al-Rfou, B. Perozzi, and D. Zelle, "Ddgk: Learning graph representations for deep divergence graph kernels," in *The World Wide Web Conference*, 2019, pp. 37–48.

[12] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016, pp. 2014–2023.

[13] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[14] Y. Shen, S. Tan, A. Sordoni, and A. Courville, "Ordered neurons: Integrating tree structures into recurrent neural networks," in *Proceedings of the Eighth International Conference on Learning Representations*, 2019.

[15] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[16] T. Kamada, S. Kawai *et al.*, "An algorithm for drawing general undirected graphs," *Information processing letters*, vol. 31, no. 1, pp. 7–15, 1989.

[17] T. M. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[18] H. Meyerhenke, M. Nöllenburg, and C. Schulz, "Drawing large graphs by multilevel maxent-stress optimization," in *International Symposium on Graph Drawing*. Springer, 2015, pp. 30–43.

[19] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[20] J. Eisner, "State-of-the-art algorithms for minimum spanning trees," *A Tutorial Discussion, University of Pennsylvania*, 1997.

[21] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[22] A. Bundy and L. Wallen, "Breadth-first search," in *Catalogue of artificial intelligence tools*. Springer, 1984, pp. 13–13.

[23] A. Zelikovsky and D. Lozevanu, "Minimal and bounded trees," *Tezele Cong. XVIII Acad. Romano-Americane, Kishinev*, pp. 25–26, 1993.

[24] G. M. Adel'son-Vel'skii and E. M. Landis, "An algorithm for organization of information," in *Doklady Akademii Nauk*, vol. 146, no. 2. Russian Academy of Sciences, 1962, pp. 263–266.

[25] R. Bayer and E. McCreight, "Organization and maintenance of large ordered indexes," in *Software pioneers*. Springer, 2002, pp. 245–262.

[26] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[27] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[28] Q. Zhao and Y. Wang, "Learning metrics for persistence-based summaries and applications for graph classification," in *Advances in Neural Information Processing Systems*, 2019, pp. 9855–9866.

[29] Y. Lyu, X. Huang, and Z. Zhang, "Learning to segment 3d point clouds in 2d image space," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[30] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[33] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[34] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[35] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of machine learning research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[36] Y. Lyu, L. Bai, and X. Huang, "Road segmentation using cnn and distributed lstm," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.