Article

Hierarchical Pooling in Graph Neural Networks to Enhance Classification Performance in Large Datasets

Hai Van Pham [1]*, Dat Hoang Thanh [1] and Philip Moore [2]

[1] School of Information and Communication Technology, Hanoi University of Science and Technology, 1 Dai Co Viet, Le Dai Hanh, Hai Ba Trung, Hanoi city, 10000, Vietnam. haipv@soict.hust.edu.vn. thanhdath97@gmail.com.
[2] School of Information Science and Engineering, Lanzhou University, Feiyun Building, 222 Tianshui S Rd, Chengguan Qu, Lanzhou Shi, Lanzhou 730030, Gansu Sheng. philip.moore@outlook.com.
* Correspondence: Hai Van Pham: haipv@soict.hust.edu.vn.

Abstract: In considering knowledge graphs in a diverse range of domains of interest, graph neural networks have demonstrated significant improvements in node classification and prediction when applied to graph representation with learning node embedding to effectively represent hierarchical properties of graphs. DiffPool is a deep-learning approach using a differentiable graph pooling technique that generates hierarchical representations of graphs. In operation DiffPool is a differentiable graph pooling technique that generates hierarchical representations of graphs. DiffPool learns a differentiable soft cluster assignment for nodes at each layer of a deep graph neural network with nodes mapped sets of clusters. However, control of the learning process is difficult given the complexity and large number of parameters on an 'end-to-end' model. To address this difficulty we propose an novel approach termed FPool which is predicated on the basic approach adopted in DiffPool (where pooling is applied directly to node representations). Methods designed to enhance data classification have been developed and evaluated using a number popular and publicly available sensor data sets. Experimental results for FPool demonstrate improved classification and prediction performance when compared to alternative methods. Moreover, FPool shows an important reduction in the training time over the basic DiffPool framework.

Keywords: Knowledge graphs; hierarchical pooling; graph classification; graph neural networks; FPool; large graph sensor datasets

Citation: Hierarchical Pooling in Graph Neural Networks to Enhance Classification Performance in Large Datasets. *Preprints* **2021**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

1. Introduction

Data mining predicated on graph data has gained traction driven by the modelling capabilities of knowledge graphs in a heterogeneous domains of interest. Research has used data mining on a diverse range of graph types which include: social networking platforms and biology and chemistry networks (molecular structures, protein interaction).

Historically, deep learning (DL) has been the key to solving many machine learning problems in diverse fields which include: image processing, natural language processing, and the video games industry. The data generated can result in large datasets represented in spaces with a finite number of dimensions in both two dimensional (flat) and three dimensional spaces. Graph Neural Networks (GNN) are the deep learning techniques applied to graphs and are effective for node representation in a broad range of fields [1].

The traditional graph classification methods are based on GNN. However, such methods generally fail to learn the hierarchical representation of graphs [2] [3]. Two dimensional graphs are inherently flat and only propagate information across edges of graphs; the result is a failure to capture the hierarchical information. Lan *et al* in [4] proposed a novel complex fuzzy inference system using Knowledge Graph and extensions in decision making. Viet *et al* in [5] have introduced extended membership graphs for picture inference systems for knowledge graphs. Context and context-awareness is an important consideration; intelligent context with decision support under uncertainty has been considered in [6] and the application of rules in knowledge reasoning for inference has been addressed in [7].

DiffPool [8] is a deep-learning approach using a differentiable graph pooling technique that generates hierarchical representations of graphs predicated on a differentiable graph pooling technique that generates hierarchical representations of graphs. The reported experimental results [for DiffPool] show an "average improvement in the accuracy for graph classification in the range 5% to 10% when compared to the alternative pooling methods considered. However, control of the learning process is difficult given the complexity and large number of parameters on an 'end-to-end' model.

To address this difficulty in this paper we propose an novel approach termed *FPool* which is predicated on the basic approach adopted in *DiffPool* (where pooling is applied directly to node representations). *FPool* implements methods novel designed to enhance data normalisation. We have evaluated *FPool* using a number sensor data sets; experimental results demonstrate improved classification and prediction performance when compared to alternative methods. Moreover, *FPool* shows an important reduction in the training time over the basic *DiffPool* framework.

Our contribution can be summarised as:

- A novel approach implementing hierarchical pooling in GNN to enhance classification performance for large datasets.
- The proposed model (*FPool*) designed to reduce the number of parameters of for the GNN. Specifically: a single GNN layer will learn node representations $\left(X^{(l)}\right)$ for all nodes in the graph with the representations used to assign nodes into clusters. The new framework *FPool* is illustrated in figure 3.
- The development of an approach which improves classification performance with significant reductions in training time.

The remainder of this paper is structured as follows: related research is considered in Section 3 with materials and methods addressed in Section 4 where GNN, graph classification, and hierarchical pooling are introduced in Sections 4.1, 4.2, and 4.3 respectively. The proposed approach is presented in Section 5. The basis for experimental testing and comparative analyses is discussed in Section 6 with the evaluation results. A discussion is presented in Section 7 with concluding observations provided in Section 8.

2. Graph Classification and Graph Convolutional Networks

Graph classification is a crucial task on many domains and systems where the aim is to identify the labels for each graph in large sensor data sets. For instance, in chemistry the prediction of chemical properties [e.g. toxicity] of molecules is crucial im medical research. Moreover, graph classification is applied to biomedical networks to predict protein functions [9] where: (i) each graph represents exactly one protein and (ii) nodes indicate secondary structure elements [helices, sheets and turns]. Edges connect nodes if those are neighbors along the amino acids along with neighbors in the space within protein structure.

Formally, a GCN is a neural network that operates on graphs. Given a graph $(G = (V, E))$ G = (V, E), a GCN takes as input an input feature matrix $\left(N \times F^0\right)$ feature matrix (X) where (N) is the number of nodes and $\left(F^0\right)$ is the number of input features for each node and an $(N \times N)$ matrix representation of the graph structure such as the adjacency matrix $(AofG)$.

A hidden layer can be written as $\left(H^i = f\left(H^{i-1}, A\right)\right)$ where $\left(H^0 = X\right)$ and (f) is a propagation. Each layer $\left(H^i\right)$ corresponds to a $\left(N \times F^i\right)$ feature matrix where each row is a feature representation of a node.

At each layer, these features are aggregated to form the features for the next layer using the propagation rule (f). Features become increasingly more abstract at each consecutive layer and [with this framework] variants of GCN differ only in the choice of propagation rule (f).

Specifically, the GCN approach is inspired by the notion of *convolutional neural networks* (CNN) for image processing. CNN aggregates the adjacent pixels of the current pixel to extract local features [such as shapes and backgrounds] of an image. When considering graphs, while image processing operates on pixels, the GCN operates on node features. For each vertex on the graph, the GCN approach aggregates the features of neighbor vertices and then generates the hidden representations for that vertices.

3. Related Research

Research has investigated multiple techniques to address the demands of classification performance related to large datasets and there are a number of proposed techniques in the literature based on Graph Convolutional Networks (GCN) concept including:

- Graph Convolutional Networks (GCN) [10] (see Section 2)
- GraphSAGE [11]
- Graph Attention Networks (GAN) [12]
- DiffPool [8]
- Self-Attention Generative Adversarial Networks (SAGAN) [13]
- Self-Attention Graph Pooling (SAGPool) [14]

GraphSAGE [11] is a general inductive framework designed to utilise the feature data of nodes, such data includes text attributes. The goal for the GraphSAGE method is to efficiently generate node embeddings for previously unseen

data. The approach enables unsupervised learning on graphs and to overcome the "Out of memory" problem experienced by the GCN method. Hamilton *et al* has also released other aggregation functions including *MEAN*, *SUM*, and *Short Term Long Term Memory* (LSTM) which can potentially increase the diversity of GNN methods. A *graph neural network* (GNN) using a knowledge graph (KG) has been proposed in [15] for Recommendation Systems to enhance the classification performance accuracy.

The GAN method is designed to "operate on graph structured data" [12]. The method uses neural network architectures to leverage "masked self-attentional layers" with the aim of addressing issues [in alternative methods] based on "graph convolutions or their approximations". The approach applies layer stacking where nodes can access neighbourhood features. The reported results claim: (i) the "implicit" specification of different nodes in a neighbourhood, and (ii) without "costly matrix operation (such as inversion) or depending on knowing the graph structure upfront". The reported result claim to realise improvements in classification performance (where test graphs are unseen during training) using four established "transductive and inductive graph benchmarks": the *Cora*, *Citeseer*, and *Pubmed* citation network datasets along with a protein interaction dataset.

Ying 'textitet al in [8] have proposed the DiffPool approach to address the requirements of graph representational learning based on effective learned node embeddings. While alternative proposed methods achieve good results, the current GNN methods are "inherently flat and do not learn hierarchical representations of graphs". This limitation can be an issue where the goal is to predict the label associated with an entire graph. DiffPool [8] introduces a "differentiable graph pooling module that can generate hierarchical representations of graphs and can be combined with various graph neural network architectures in an end-to-end fashion". The DiffPool approach is a deep-learning approach which learns a differentiable soft cluster assignment for nodes at each layer of a deep graph neural network with nodes mapped to sets of clusters. There is however an issue identified in Section 1 where control of the learning process is difficult given the complexity and large number of parameters on an 'end-to-end' model.

Zhang *et al* in [13] have proposed the Self-Attention Generative Adversarial Network (SAGAN).The proposed method enables "attention-driven, long-range dependency modeling for image generation tasks. Zhang *et al* argue that "traditional convolutional GAN generate high-resolution de- tails as a function of only spatially local points in lower-resolution feature maps" and the SAGAN approach provides a basis upon which "details can be generated using cues from all feature locations" given that he discriminator can check for highly detailed features in distant portions of the image which are consistent with each other. The reported experimental results show that SAGAN improves on the best published Inception score (27.62) with a score of (52.52) along with a reduction in the *Fre'chet* Inception distance from (27.62) to (18.65) for the 'ImageNet' dataset. From a visualisation perspective the authors argue that the attention layers the generator can leverage neighbourhoods that correspond to object shapes rather than local regions of fixed shape.

Many advanced methods of applying deep learning to structured data (e.g., graphs) have been proposed which focus on generalising convolutional neural networks (CNN) to graph data, which includes redefining the convolution and the downsampling (pooling) operations for graphs. Lee *et al* in [14] propose a " Self-Attention Graph Pooling" (SAGPool) approach, a graph pooling method for GNN related to hierarchical graph pooling. Generalising the convolution operation for graphs has been shown to provide improved levels of performance and accordingly has been widely used. However, the method of applying down-sampling to graphs is remains a challenge with significant room for improvement. Lee *et al* in [14] propose the Self-Attention Graph Pooling (SAGPool) approach which enables pooling with consideration of both node features and graph topology. The SAGPool method (where the "self-attention mechanism" can distinguish between the "nodes that should be dropped and the nodes that should be retained") employs graph convolution to calculate attention scores and node features along with consideration of graph topology. The reported experimental results demonstrate that SAGPool realises improved graph classification performance on the benchmark datasets using "a reasonable number of parameters". The authors posit that SAGPool provides advantages over the alternative methods considers and is "the first method to use self-attention for graph pooling with high performance".

Rousseau *et al* in [16] has considered "text categorisation as a graph classification problem" where each document is represented as a "graph-of-words" instead of the historical "n-gram bag-of-words". By leveraging the power of graph structures, the "graph-of-words" captures the word inversion and subset matching [e.g., "article about news" *vs* "news article") while the "bag-of-words" fails to enable word inversion and subset matching.

3.1. Summary

For the classification of knowledge graphs, a number of graph classification methods have been proposed to combat the limitations and weaknesses identified in the approaches considered. The graph classification methods considered include: GraphSAGE [11], GAN [12], *DiffPool* [8], SAGAN [13], SAGPool [14], and Rousseau *et al* [16].

In this paper, to overcome the issues identified in our overview of related research, we propose our novel *FPool* method which is based on the *DiffPool* approach with hierarchical graph representation. However, the *DiffPool* approach has suffered from the over fitting problem while having to train an end-to-end model.

The study presented in this paper introduces *FPool*. In our research we have investigated adjustments in parameters along with the related training process completion. For example, the nodes of graphs are pooled to very few clusters including two clusters, leading to significant redundant clusters and/or redundant parameters. In our proposed *FPool* method the pooling process is performed directly on the node embedding which reduces the number of parameters. To avoid over fitting, we have added normalisation techniques consisting of *Felonious normalisation* (applied for node representations), *zero mean* (used for node features in the training process), and *unit variance*.

While our study has addressed many issues we have identifies open research questions (see Section 7.1) which form the basis for future directions for research.

4. Material and Methods

In this section we introduce the materials and methods used in this study namely: (i) Graph Neural Networks (GNN), Graph Classification (GC), and Hierarchical Pooling (HP). The proposed model is introduced in Section 5.

4.1. Graph Neural Networks

Let $(G(V,E))$ be a graph, each node $v \in V$ has features $()X_v \in R^d$. A GNN use the graph structure and the node features to learn a vector representation (h_v) for each node. Recent GNN methods follow the message-passing mechanism where the vector representation of each node is iteratively updated by aggregating the hidden representations of neighbor nodes [10] [14]. Following completion of the (k) iteration, the vector representation of (v) holds the information of the k-hop network where (v) is a center vertices. For instance, at layer k, GNNs perform these functions, given by Eq.(1, 2):

$$a_v^{(k)} = AGGREGATE^{(k)}(\{h_u^{(k-1)} : u \in N(v)\}) \tag{1}$$

$$h_v^{(k)} = COMBINE^{(k)}(h_u^{(k-1)}, a_v^{(k)}) \tag{2}$$

where $\left(a_v^{(k)}\right)$, $\left(h_v^{(k)}\right)$ represents vectors of $(N(v))$ and (v) at the layer (k), respectively. $(N(v))$ indicates the neighbor(s) of (v). $\left(h_v^{(0)} = X_v\right)$ is an initial value set.

There are a number of $(AGGREGATE)$ and $(COMBINE)$ functions, for example, GraphSAGE-MAX [11] use the $(AGGREGATE)$ function as given by Eq.(3):

$$a_v^{(k)} = MAX(\{ReLU(W.h_u^{(k-1)}), \forall u \in N(v)\}) \tag{3}$$

where (W) is a learning matrix parameter and (MAX) is the maximum 'element-wise' function. The $(COMBINE)$ step represents a vector concatenation or the summation 'element-wise' function followed by a mapping matrix $\left(W.[h_v^{(k-1)}, a_v^{(k)}]\right)$.

A further relevant example of a GCN where the mean 'element-wise' is implemented is shown in [10]. The $(AGGREGATE)$ and $(COMBINE)$ functions are shown in Eq.(4). Figure 1 illustrates the GNN process on a specific (red) node).

$$h_v^{(k)} = ReLU(W.MEAN\{ReLU(W.h_u^{(k-1)} : \forall u \in N(v)\}) \tag{4}$$

In the initial stage (termed the neighborhood sampling stage) a number of neighbor nodes are selected; for large graphs, neighborhood sampling is essential to address the memory consumption issue where a large number of nodes with large number of GNN layers easily leads to the "out of memory" error. Following the sampling of neighbor nodes, the $(AGGREGATE)$ and $(COMBINE)$ functions are implemented. Thus, the hidden representation of node is forwarded to downstream tasks such as node classification and clustering.

4.2. Graph Classification

In a graph there are two main classification tasks including: (i) on the node-level, and (ii) on the graph-level:

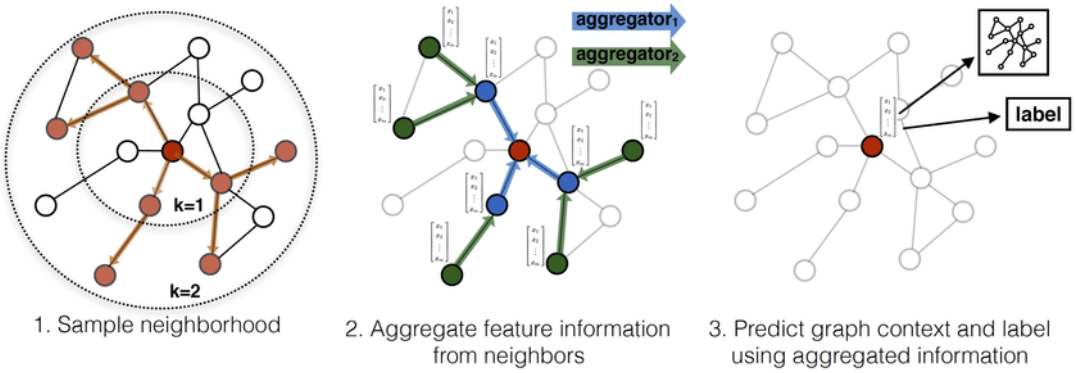

Figure 1. Illustration of a general GNN inductive framework on a specific node - (the red node): source: [11]

1. For node classification: each node (v) has an associated label (y_v) and the goal is to learn a representation vector (h_v) that could be used to predict the label (y_v) by using a function (f), $(y = f(h_v))$
2. For graph classification: given a set of graphs $(G_1, G_2, ..., G_n)$ and their labels $(y_1, y_2, ..., y_n)$, instead of learning (h_v) for each node, the model aims to learn the representation vector (h_G) for the whole graph so that (h_G) helps to predict the label of graph, $(y_G = g(h_G))$.

4.3. Hierarchical Pooling

Conventional approaches (for example see Xu *et al* [2] and Duvenaud *et al* [3]) do not capture the hierarchical properties of graphs while all the node embeddings are just globally pooled together. The embedding of graph is therefore similar to a virtual node that connect to all the nodes of the graph and such common approaches have not addressed the need to learn the natural structures of many 'real-world' graphs.

Ying *et al* [8] has proposed the *DiffPool* approach which is a differentiable graph pooling method which learns a cluster assignment matrix in an end-to-end fashion. The key motivation [for *DiffPool*] is to induce learning to enable nodes to be assigned to clusters at layer $\{l\}$ by using the embeddings generated from the GNN layer at layer $\{l-1\}$.

We have denoted (n_l) as the number of nodes at layer (l), $\left(S^{(l)} \in R^{n_l \times n_{l+1}}\right)$ denotes the assignment matrix at layer (l) and (GNN_l) represents for (K) GNN layers. To generate the assignment matrix, *DiffPool* employs the following by Eq.(5).

$$S^{(l)} = softmax(GNN_{l,pool}(A^{(l)}, X^{(l)})) \tag{5}$$

Therefore, (S_{ij}) contains the probability value of node (i) at layer $\{l\}$ assigned to cluster (j) at the next layer. Given that $\left(Z^{(l)} \in R^{n_l \times d}\right)$ is the node embeddings at layer $\{l\}$, the hidden representation of nodes $\left(X^{(l+1)}\right)$ and the adjacency matrix $\left(A^{(l+1)}\right)$ at layer $(l+1)$ is expressed by Eqs.(6, 7)

$$X^{(l+1)} = S^{(l)T} Z^{(l)} \tag{6}$$

$$A^{(l+1)} = S^{(l)T} A^{(l)} S^{(l)} \tag{7}$$

The *DiffPool* framework is illustrated at figure 2. Given an input graph $\left(G(A^{(0)}, X^{(0)})\right)$ the adjacency matrix and the node features are forwarded to two separated GNN: $\left(GNN_{0,pool}\right)$ and $(GNN_{0,embed})$. In the $(GenerateGraph)$ stage, the new graph is generated given the output of (GNN_{embed}) (denoted at $\left(Z^{(l)}\right)$) and the output of $\left(GNN_{pool}\right)$ (denoted at $\left(S^{(l)}\right)$). The two equations 6, and 7 are then implemented.

To predict the label for each graph, the last layer of the *DiffPool framework* would be the classification layer with a softmax function. However, it is difficult to train the *DiffPool framework* using only the gradient from the classification layer. Therefore, Ying *et al* in [8] proposes two alternative loss functions: (i) the *link prediction* loss, and (ii) the *entropy* loss. The link prediction loss aims to pool nearby nodes, at each layer (l) , link prediction loss that loss function is expressed by Eq.(8).

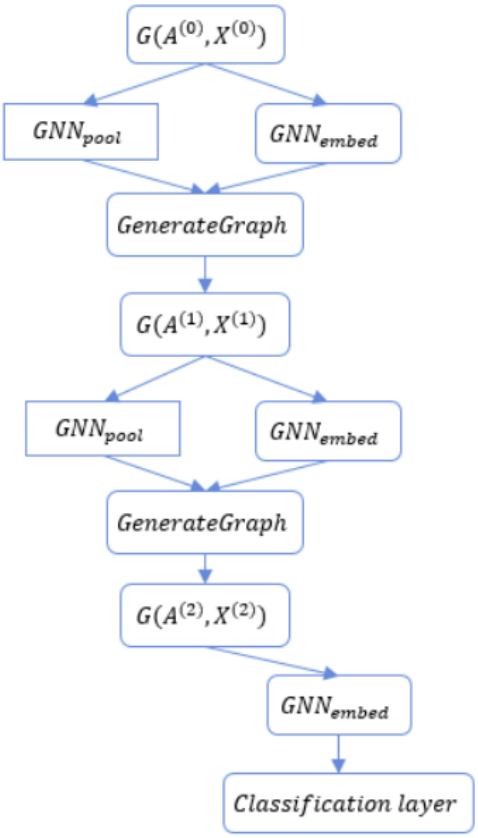

Figure 2. An overview of the *DiffPool* framework with 3 layers where the ***input*** is a graph $\left(G(A^{(0)}, X^{(0)})\right)$ and the ***output*** is the predicted label for that graph.

$$L_{LP} = \|A^{(l)}, S^{(l)} S^{(l)T}\|_F \tag{8}$$

where $(\|.\|_F)$ is the Forbenious norm (note: each node assigns completely to a cluster). Moreover, the entropy loss is assigned to a vector for each node in a 'one-hot vector'. The entropy loss uses as given by Eq.(9).

$$L_E = \frac{1}{n} \sum_{i=1}^{n} H(S_i) \tag{9}$$

where (H) denotes the entropy function and (S_i) is the assignment vector for node (i). Therefore, the whole framework is trained by using the combination of these loss functions.

5. The Proposed Model

In this section we present the proposed model (called *FPool*) which is designed to implement the improvements to the *DiffPool framework*. Figure 2 shows an integration of GNN ($\left(GNN_{pool}\right)$ and (GNN_{embed})). The proposed model has been conceived to enable the merging these models by reducing the number of parameters of for the GNN. Specifically: a single GNN layer will learn node representations $\left(X^{(l)}\right)$ for all nodes in the graph with the representations used to assign nodes into clusters. The new framework *FPool* is illustrated in figure 3.

In the *FPool* framework: to compute the node embeddings $\left(Z^{(l)} \in R^{n_l \times d}\right)$ and matrix $\left(S^{(l)} \in R^{n_l \times n_{l+1}}\right)$ at layer $\{l\}$ is given by Eqs.(10 and 11):

$$Z^{(l)} = GNN_l(A^{(l)}, X^{(l)}) \tag{10}$$

$$S^{(l)} = softmax(Z^{(l)}.W^{(l)} + B) \tag{11}$$

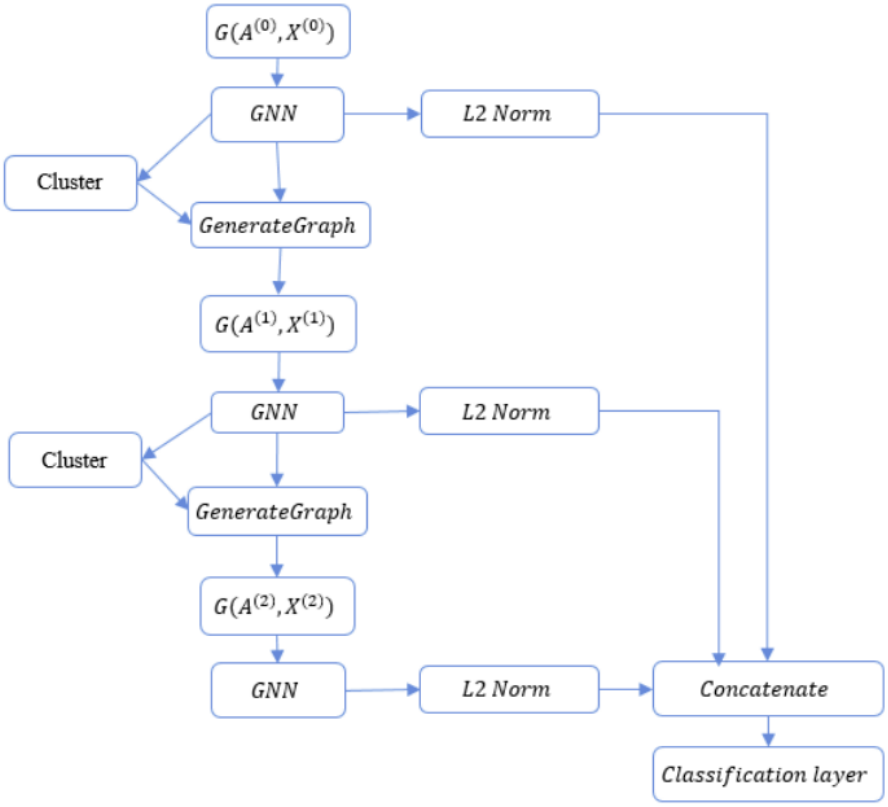

Figure 3. The FPool framework represents within 3 layers.

where $\left(W^{(l)} \in R^{d \times n_{l+1}}\right)$ is a weight matrix, $(B \in R^{n_l})$ denotes the bias matrix, and the $(softmax)$ function is applied to every row. Equations 10, 11 are equivalent to the (GNN) and $(Cluster)$ stages in the *FPool* framework. Following the generation of the node embeddings and the assignment matrix equations 10 and 11 are performed to generate the pooled graph.

By increasing the number of GNN layers may fail to capture additional information and the related methods discussed generally choose ffrom 2 to 6 [8] [2]. Therefore, instead of using only the output of the GNN from the final layer, the graph representation is the combination of all of the (L) GNN layers. Specifically, the graph embedding (h_G) is computed as shown in Eqs.(12 and 13) where $\left(Z_F^{(l)}\right)$ is the normalised representation vector for all nodes in the graph.

$$Z_F^{(l)} = Z^{(l)} / \|Z^{(l)}\|_F, \forall l < L \tag{12}$$

$$h_G = [MEAN(Z_F^{(0)}), ..., MEAN(Z_F^{(L-1)})] \tag{13}$$

By standardizing node embeddings, the training process becomes more stable. The $(MEAN)$ represents an element-wise mean function and $([.])$ denotes the vector concatenation functions. Therefore, (h_G) has the size $(d_0 + d_1 + ... + d_n)$ where $(d_0, d_1, ..., d_n)$ is the size of the node representation at layers $(0, 1, ..., n)$ respectively.

The notion of vector concatenation is inspired by the *Residual block* on the *Resnet architecture* in computer vision [17]. However, deep networks are hard to optimize and increasing network depth may not lead to better performance due to 'vanishing gradient' problem. While stacking more layers onto the network may result in 'performance saturation' with reduced performance [18] [19]; the study presented in [17] proposes "shortcut connections" (an approach which "skips" one or more layers) where the gradient from upper layers could "flow directly" to any earlier layers.

In the initialisation step: the node features matrix $\left(F \in R^{n \times d}\right)$ is normalised to have zero mean and unit variance; this step is crucial since it leads the machine learning model which converges faster and performs better. The normalisation equation for node features is represented as given by Eq.(14) where (μ_j) and (σ_j) denotes the mean and standard variance values of (F) on column (j) respectively.

Table 1: The statistical data for the benchmark datasets used in the experimental testing

	Benchmark Datasets			
Dataset	Number of Graphs	Number of Classes	Node Attribute Dimension	Contains Node Labels?
Mutag	188	2	0	Yes
Enzymes	600	6	28	Yes
IMDB-Binary	1000	2	0	No
D & D	1178	2	0	Yes

$$F_{norm,j} = \frac{F_j - \mu_j}{\sigma_j}, j = \overline{1,d} \tag{14}$$

6. Experiment Testing and Evaluation

To evaluate the performance of the proposed model we have conducted a comparative analysis where we compare the relative performance of: (i) the proposed approach, (ii) *DiffPool*,, and (iii) GIN. Implementation for all the methods evaluated has used the same dataset(s) under the same testing environments to measure the relative classification performance. In the expermental testing we have used several popular data sets for the evaluation with graph classification tasks; these sensor datasets are publicly available at http://graphkernels.cs.tu-dortmund.de [20]. The benchmark sensor data sets, the identification, and statistical information is provided in Table 1.

- ***Mutag*** [20]: the dataset consists of 188 graphs equivalent to 188 chemical compounds. These graphs are divided into two classes based on their *mutagenic* sensors effect on a bacterium.
- ***Enzymes*** [9] [21]: is a biological dataset for enzymes. The sensor dataset contains 600 enzymes with 6 associated classes which represent the characteristics of enzymes. each graph represents exactly one protein, nodes indicate the secondary structure (SSE) in protein and there exists a connection between two vertices if they are neighbors in the amino-acid or on the 3-D space.
- D&D [22]: is a sensor dataset of protein structures which includes 1178 graphs. Nodes indicate amino acids and edges denote that two nodes are close to each others on 3-D space.
- IMDB-Binary [23]: is a social networks dataset in which each graph is equivalent to an ego-network where nodes represent actors, edges denote two actors collaborating in a film. Each graph is derived from a pre-specified genre of film.

In a GNN, each vertices [in the input graph] must have an associated feature vector. Therefore, for graphs without a node feature matrix, we initialise it as a vector of constant values ($X_v = [1,1], \forall\, v \in V$). For graphs which contain node labels or node attributes (or both) the feature vector for each vertices is then a concatenation of node attribute vector and node label vector.

In our experimental testing and evaluation the training data is separated into three set: (i) a *training* set, (ii) a *validation* set, and (iii) a test set; the relative proportions are 8:1:1 (i.e., 80%, 10%, and 10%) respectively. To avoid bias, and ensure a fair comparison in the comparative analysis, we have applied the same approach to implementation for all the methods compared and all of the methods use the same training, validation, and test sets at each time of running. We also evaluate the accuracy on each data sets on 10 running iterations which means there are 10 different combinations of the training, validation, and test sets.

For the *DiffPool* and *GIN* we have implemented by Pytorch Geometric library [24] with some minor editing. Pytorch Geometric is a Python library which supports many types of GNN along with many processed datasets (including all of the data sets used in our experiments).

For *FPool* we have used 3 layers GraphSAGE-MEAN for each GNN block in figure 3 with the number of hidden units 64. The number of clusters is 25 for both the first and the second pooling layers on the *Mutag*, *Enzymes* and *IMDB-Binary*. On the D&D data set the number of clusters is larger and set 125.

For the final classification layer, both the *DiffPool* and *FPool* use the same architecture:

- Linear $(embedding\_size)$, $(hidden\_size)$
- ReLU - Linear $(hidden\_size)$, $(n\_classes)$

- Log Softmax where $(embedding\_size)$ is the size of (h_G), $(hidden\_size)$ is the number of hidden units, and $(n\_classes)$ indicates the number of classes need to classify.

6.1. Experimental Results

The results derived from our experimental testing identify the *mean value* and *standard variance* of accuracy. The accuracy is calculated as shown in Eq.(15) where $y_{true,i}$ is the actual class of graph i, $y_{pred,i}$ is the predicted class for graph i, and N denotes the number of graphs in test set.

$$Acc = \frac{1}{N}\sum_{i=1}^{N}(y_{pred,i} == y_{true,i}) \tag{15}$$

As shown in Tables 2, 3 we have calculated: (i) the statistics for the accuracy (see Table 2), and (ii) the training time for DiffPool, FPool and GIN using the Mutag, Enzymes, IMDB-Binary, and D&D datasets (see Table 3). The proposed method (FPool) has demonstrated a significant performance improvement with respect to classification as compared to DiffPool and GIN. Only for the IMDB-Binary the accuracy of GIN is slightly better than the others with a relatively small improvement in performance results in the range 0.5% to 1.2%.

While GIN outperforms on the training time experiment it suffers from an inability to capture the hierarchical structure of many 'real-world' datasets as shown in table 2 with lower accuracy. The training time of *FPool* is much faster than *DiffPool* due to the reduction in the number of parameters resulting from the merging process of (GNN_{embed}) and $\left(GNN_{pooling}\right)$ into a single process.

Table 2: Accuracy (%) of *DiffPool*, *FPool*, and *GIN*

	Comparative Analysis			
	Mutag	**Enzymes**	**IMDB-Binary**	**D & D**
DiffPool	78.42 +- 10.90	44.00 +- 7.93	67.70 +- 5.29	74.84 +- 4.89
GIN	82.63 +- 10.00	54.83 +- 4.91	**68.20 +- 2.96**	70.09 +- 4.60
FPool	**84.21 +- 6.66**	**67.50 +- 7.97**	67.00 +- 2.45	**81.60 +- 0.48**

Table 3: Training time (seconds per epoch) for *DiffPool*, *FPool*, and *GIN*

	Comparative Analysis			
	Mutag	**Enzymes**	**IMDB-Binary**	**D & D**
DiffPool	0.32 +- 0.04	0.73 +-0.01	1.24 +- 0.01	3.36 +- 0.38
GIN	**0.08 +- 0.01**	**0.17 +- 0.01**	**0.17 +- 0.00**	**0.56 +- 0.05**
FPool	0.26 +- 0.03	0.57 +- 0.01	0.98 +- 0.01	2.69 +- 0.24

6.2. Simulation results for FPool node clustering

Figure 4 illustrates the hierarchical cluster assignment of *FPool* with two pooling layers and three example graphs taken from the ENZYMES database. Node clustering colours indicate *cluster* and *edge* colours indicate edge weights; in certain cases it may be difficult to clearly recognise the different of edge colours].

In the assignment matrices, $S^{(l)} \in R^{n_l \times n_{l+1}}$ are real-values matrices; therefore, the generated graph is a complete weighted graph. Because of the entropy loss function, each node is likely assigned to only one cluster. Therefore, in this visualisation each node was allocated to the cluster which has the highest value. In figure 4, while the number of clusters is set to 25, many clusters are empty; the proposed *FPool* model has been automatically trained to allocate nodes to appropriate and meaningful clusters.

There remains an open research question, there is an issue which relates to: "how to learn the number of clusters to reduce parameters when both FPool and DiffPool complete the training successfully". The issue we have identified is the potential for a significant number of redundant parameters.

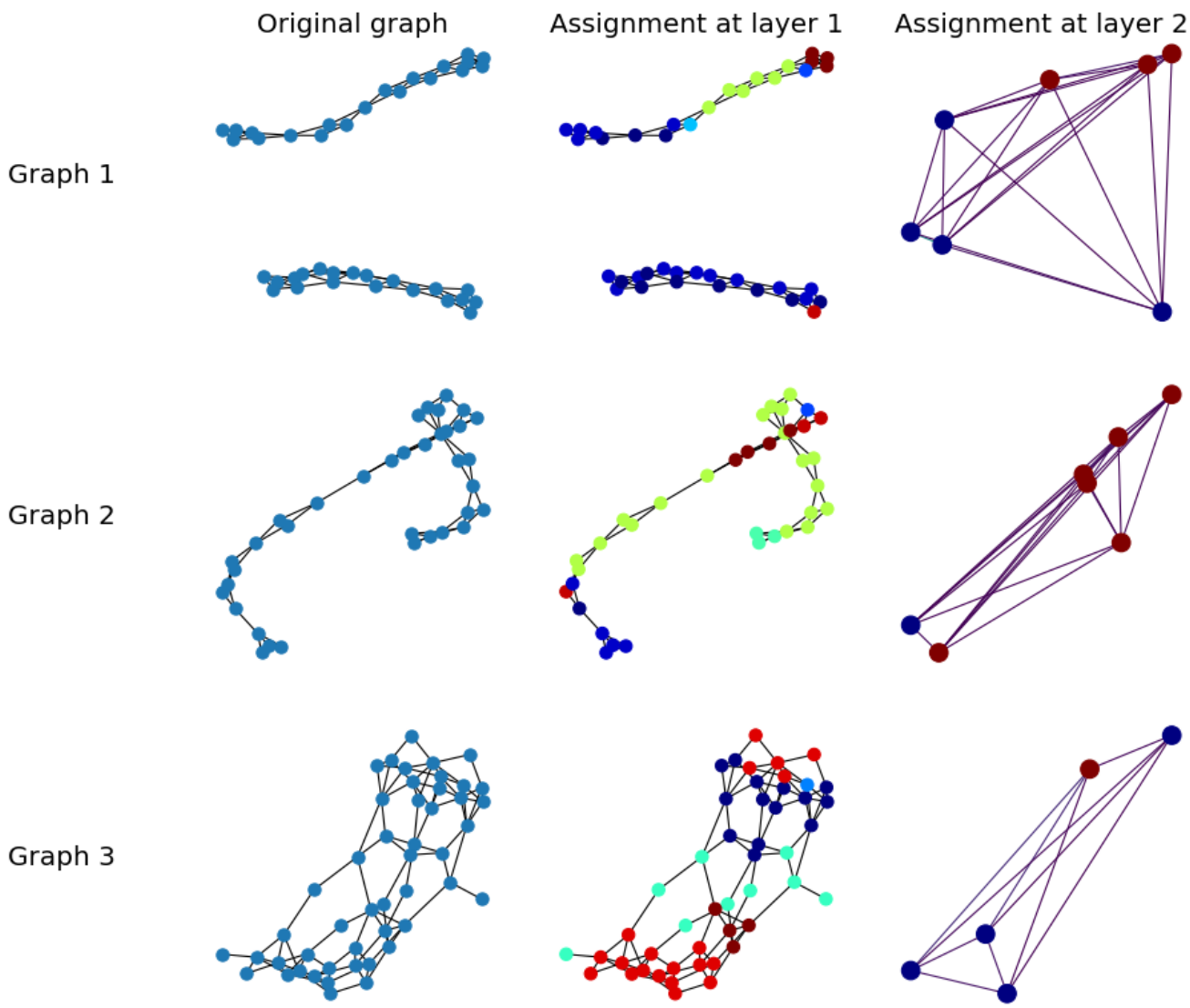

Figure 4. *FPool* visualisation of node assignment with two pooling layers on 3 example graphs from the ENZYMES dataset.

6.3. Evaluation of FPool and DiffPool hierarchical structures

In the experiments, we have evaluated the training curves for both FPool and DiffPool using the ENZYMES and MUTAG datasets; the results of the evaluation are shown in Figures 5 (the training and test accuracy on ENZYMES dataset versus training epoch) and 6 (the training and test accuracy on MUTAG dataset versus training epoch). The Figures (5 and 6) show the results for the adaptation in the training process.

Recall that in our experimental testing and evaluation the training data is separated into three set: (i) a *training* set, (ii) a *validation* set, and (iii) a test set; the relative proportions are 8:1:1 (i.e., 80%, 10%, and 10%) respectively. The test accuracy is calculated on the current best model on the *validation set*; therefore, in theory, this line is usually up-trend. The results demonstrate that our proposed novel *FPool* provides improved performance over DiffPool in terms of graph classifications.

In testing the relative accuracy of *FPool* and DiffPool, experimental results shows that *FPool* produces consistently better training accuracy results for graph classification for both the ENZYMES and MUTAG datasets. The training accuracy of *FPool* is higher than DiffPool for very early epoch(s) for both the ENZYMES and MUTAG datasets; this demonstrates that *FPool* has a faster training time than DiffPool in terms of the hierarchical structure of a graph though there remains an overfitting problem in the approaches because of less training data.

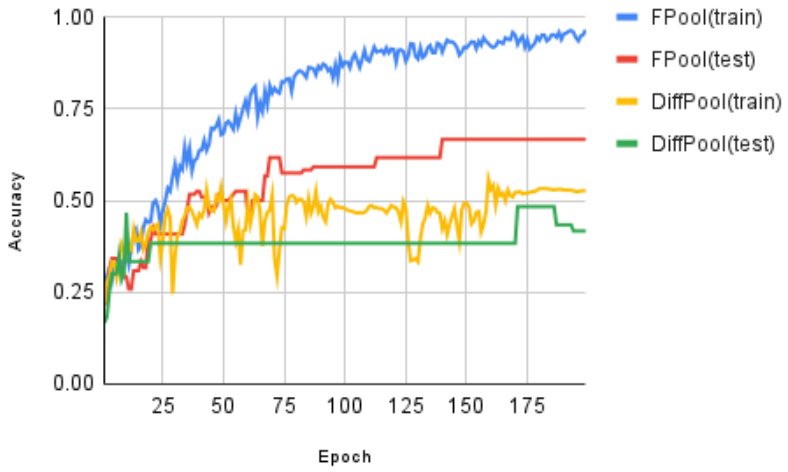

Figure 5. Training and test accuracy on ENZYMES dataset versus training epoch.

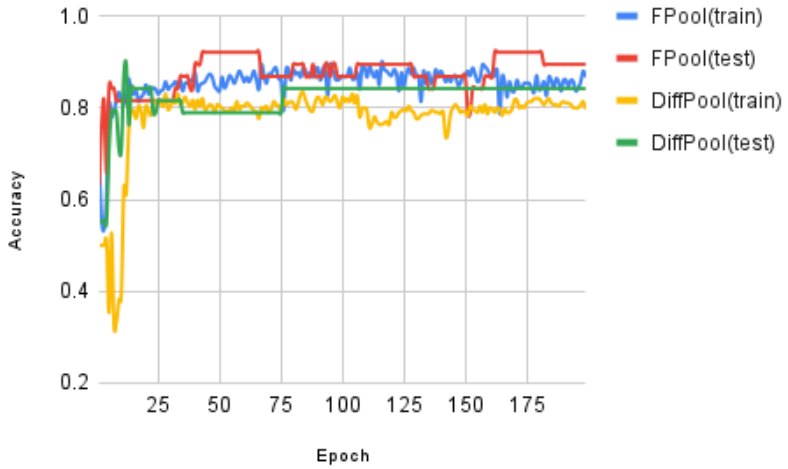

Figure 6. Training and test accuracy on MUTAG dataset versus training epoch.

7. Discussion

In this paper we have considered knowledge graphs in a diverse range of domains along with GNN which have been shown to enable improvements in node classification and prediction when applied to graph representation with learning node embedding to effectively represent hierarchical properties of graphs. Data mining predicated on graph data has gained traction driven by the modelling capabilities of knowledge graphs in a heterogeneous domains. Research has used data mining on a diverse range of graph types which include: social networking platforms and biology and chemistry networks (molecular structures, protein interaction).

Historically DL has been applied in solving many machine learning problems in heterogeneous fields including: image processing, natural language processing, and video games. However, the data generated can result in large datasets represented in spaces with a finite number of dimensions in both two dimensional (flat) and three dimensional spaces. GNN are the deep learning techniques applied to graphs and are effective for node representation in a broad range of fields.

Traditional graph classification methods are based on GNN; however, such methods generally fail to learn the hierarchical representation of graphs. Two dimensional graphs are inherently flat and only propagate information across edges of graphs; the result is a failure to capture the hierarchical information. Context and context-awareness is an important consideration along with intelligent context processing with decision support under uncertainty with the application of rules in knowledge reasoning for inference.

Current methods have used deep learning using a differentiable graph pooling technique that generates hierarchical representations of graphs; however, control of the learning process is difficult given the complexity and large number of

parameters on an 'end-to-end' model. To address this difficulty in this paper we propose an novel approach termed *FPool* which is predicated on the basic approach adopted in *DiffPool* (where pooling is applied directly to node representations). *FPool* implements methods novel designed to enhance data normalisation.

We have evaluated *FPool* using a number sensor data sets; experimental results demonstrate: (i) improved classification and prediction performance when compared to alternative methods, and (ii) significant reductions in the training time over the basic *DiffPool* framework. The evaluation and experimental results derived from the comparative analysis and experimental testing are set out in Section 6 with results presented in tabulated form in Tables 2, and 3. The comparative analysis has been conducted using four publicly available datasets: *Mutag*, *Enzymes*, *IMDB-Binary*, and *D & D*; details of the datasets can be found in Section 5 with the details of the datasets shown in tabulated form in Table 1 where the number of graphs and classes are shown with the node attribute dimension and where code labels are included.

7.1. Open Research Questions and Future Directions for Research

We have trained *FPool* and *DiffPool* using the three loss functions including the: (i) classification loss, (ii) link prediction loss, and (iii) entropy loss. Since the classification loss is required, the affect of the two other loss functions is evaluated in Section 6. However, there remain open research questions (ORQ)and potential issues for *DiffPool* framework:

1. Is the entropy loss is necessary given that a GNN could handle soft adjacency matrix which could be translated to a complete weighted graph? The question *could using additional entropy loss help improving performance*? remains an ORQ.
2. As discussed in Section 6.1, while GIN outperforms on the training time experiment it suffers from an inability to capture the hierarchical structure of many 'real-world' datasets as shown in table 2 with lower accuracy.
3. There remains an open research question: "how to learn the number of clusters to reduce parameters when both FPool and DiffPool complete the training successfully". The issue we have identified is the potential for a significant number of redundant parameters.

In future work we intend to investigate points 1 to 3 as they relate to out proposed *FPool* method with the aim of further improving: (i) the classification performance for the IMDB-Binary dataset for which GIN is slightly better than *FPool*, and (ii) the computational overhead (the training time) which as we have noted in Table 3 is slower than for GIN across all datasets.

8. Concluding Observations

The paper has presented the graph classification problem using the GNN technique which is a widely used deep learning method. In this paper we have proposed *FPool* which is a novel method for graph classification based on the notion of hierarchical pooling to provide an effective method of capturing the hierarchical structure of graphs. While *FPool* and *DiffPool* employ the same hierarchical pooling concept, our reported experimental results show that the novel *FPool* method has demonstrated improved classification performance over the alternative methods evaluated in a comparative analysis where classification performance and the training time has been evaluated using the same sensing datasets and training methodology. The training time of *FPool* is significant faster than DiffPool when dealing with sensor data sets. We posit that *FPool* provides an effective and efficient method for capturing the hierarchical structure of graphs with improved classification performance and training time.

Author Contributions: Conceptualization, H.VP and D.NT; methodology, H.VP and P.M investigation, H.VP and D.NT; writing–original draft preparation, P.M.; writing–review and editing, P.M; visualization, H.V.P. P.M. H.VP and D.NT and PM have read and agreed to the published version of the manuscript.

Funding: This research is funded by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant: 102.05-2019.316.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wang, Z.; Lv, Q.; Lan, X.; Zhang, Y. Cross-lingual knowledge graph alignment via graph convolutional networks. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 349–357.
2. Xu, K.; Hu, W.; Leskovec, J.; Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* **2018**.
3. Duvenaud, D.; Maclaurin, D.; Aguilera-Iparraguirre, J.; Gómez-Bombarelli, R.; Hirzel, T.; Aspuru-Guzik, A.; Adams, R.P. Convolutional networks on graphs for learning molecular fingerprints. *arXiv preprint arXiv:1509.09292* **2015**.
4. Lan, L.T.H.; Tuan, T.M.; Ngan, T.T.; Giang, N.L.; Ngoc, V.T.N.; Van Hai, P.; others. A New Complex Fuzzy Inference System With Fuzzy Knowledge Graph and Extensions in Decision Making. *IEEE Access* **2020**, *8*, 164899–164921.

5. Van Viet, P.; Chau, H.T.M.; Van Hai, P.; others. Some extensions of membership graphs for picture inference systems. 2015 seventh international conference on knowledge and systems engineering (KSE). IEEE, 2015, pp. 192–197.
6. Moore, P.; Pham, H.V. Intelligent context with decision support under uncertainty. Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on. IEEE, IEEE, 2012, pp. 977–982. doi:10.1109/CISIS.2012.17.
7. Moore, P.T.; Pham, H.V. On Context and the Open World Assumption. The 29th IEEE International Conference on Advanced Information Networking and Applications (AINA-2015); IEEE, IEEE: Gwangju, Korea, 2015; pp. 387–392. doi:10.1109/WAINA.2015.7.
8. Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W.L.; Leskovec, J. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804* **2018**.
9. Borgwardt, K.M.; Ong, C.S.; Schönauer, S.; Vishwanathan, S.; Smola, A.J.; Kriegel, H.P. Protein function prediction via graph kernels. *Bioinformatics* **2005**, *21*, i47–i56.
10. Kipf, T.N.; Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* **2016**.
11. Hamilton, W.L.; Ying, R.; Leskovec, J. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216* **2017**.
12. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903* **2017**.
13. Zhang, H.; Goodfellow, I.; Metaxas, D.; Odena, A. Self-attention generative adversarial networks. International conference on machine learning. PMLR, 2019, pp. 7354–7363.
14. Lee, J.; Lee, I.; Kang, J. Self-attention graph pooling. International Conference on Machine Learning. PMLR, 2019, pp. 3734–3743.
15. Tien, D.N.; Van, H.P. Graph Neural Network Combined Knowledge Graph for Recommendation System. International Conference on Computational Data and Social Networks. Springer, 2020, pp. 59–70.
16. Rousseau, F.; Kiagias, E.; Vazirgiannis, M. Text categorization as a graph classification problem. Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), 2015, pp. 1702–1712.
17. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
18. Goodfellow, I.; Warde-Farley, D.; Mirza, M.; Courville, A.; Bengio, Y. Maxout networks. International conference on machine learning. PMLR, 2013, pp. 1319–1327.
19. Srivastava, R.K.; Greff, K.; Schmidhuber, J. Highway networks. *arXiv preprint arXiv:1505.00387* **2015**.
20. Kersting, K.; Kriege, N.M.; Morris, C.; Mutzel, P.; Neumann, M. Benchmark data sets for graph kernels, 2016. *URL http://graphkernels. cs. tu-dortmund. de* **2016**, *795*.
21. Schomburg, I.; Chang, A.; Ebeling, C.; Gremse, M.; Heldt, C.; Huhn, G.; Schomburg, D. BRENDA, the enzyme database: updates and major new developments. *Nucleic acids research* **2004**, *32*, D431–D433.
22. Kriege, N.; Mutzel, P. Subgraph matching kernels for attributed graphs. *arXiv preprint arXiv:1206.6483* **2012**.
23. Shervashidze, N.; Schweitzer, P.; Van Leeuwen, E.J.; Mehlhorn, K.; Borgwardt, K.M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* **2011**, *12*.
24. Fey, M.; Lenssen, J.E. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* **2019**.