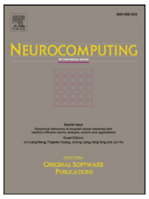
Neurocomputing 321 (2018) 346–356

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



DGCNN: Disordered graph convolutional neural network based on the Gaussian mixture model[☆]

Bo Wu[*], Yang Liu, Bo Lang[*], Lei Huang

State Key Laboratory of Software Development Environment, Beihang University, PR China

ARTICLE INFO

Article history:
Received 12 December 2017
Revised 29 July 2018
Accepted 10 September 2018
Available online 20 September 2018

Communicated by Dr Li Sheng

MSC:
00-01
99-00

Keywords:
Disordered graph convolutional neural network
The disordered graph convolutional layer
Mixed gaussian model
Node neighborhood structure
Graph classification
Graph retrieval

ABSTRACT

Convolutional neural networks (CNNs) can be applied to graph similarity matching, in which case they are called graph CNNs. Graph CNNs are attracting increasing attention due to their effectiveness and efficiency. However, the existing convolution approaches focus only on regular data forms and require the transfer of the graph or key node neighborhoods of the graph into the same fixed form. During this transfer process, structural information of the graph can be lost, and some redundant information can be incorporated. To overcome this problem, we propose the disordered graph convolutional neural network (DGCNN) based on the mixed Gaussian model, which extends the CNN by adding a preprocessing layer called the disordered graph convolutional layer (DGCL). The DGCL uses a mixed Gaussian function to realize the mapping between the convolution kernel and the nodes in the neighborhood of the graph. The output of the DGCL is the input of the CNN. We further implement a backward-propagation optimization process of the convolutional layer by which we incorporate the feature-learning model of the irregular node neighborhood structure into the network. Thereafter, the optimization of the convolution kernel becomes part of the neural network learning process. The DGCNN can accept arbitrary scaled and disordered neighborhood graph structures as the receptive fields of CNNs, which reduces information loss during graph transformation. Finally, we perform experiments on multiple standard graph datasets. The results show that the proposed method outperforms the state-of-the-art methods in graph classification and retrieval.

1. Introduction

A graph structure is a rich representational form that can describe complex structural data in the real world, such as images, biomedical data, and social networks. Many studies represent an image as an attribute graph and transform the image retrieval problem into an attribute graph search problem. The technique of chemical analysis graph searching, for example, can facilitate the study of properties of newly synthesized chemicals by referring to a database of existing chemicals with known properties. Therefore, it is important to study graph feature learning and searching.

In recent years, deep learning has been applied to many areas and has been shown to significantly outperform traditional methods. Among the available techniques, convolutional neural networks (CNNs) are widely used in image classification [1–4], semantic segmentation [5–8] and object recognition [9,10]. CNNs can learn the local structure and features of data. Because data such as images, video and sound have the same fixed-sized neighborhoods, convolution, pooling and other operations are well defined in the mathematical sense. For example, in an image, each pixel has eight neighboring nodes. However, traditional CNNs cannot be applied directly to graph data, whose neighborhoods are irregular.

To apply CNNs to graph-structured data, multiple methods have been proposed [11–18]. These methods can be divided into two categories: spatial-based methods and spectral-based methods. Spatial-based approaches [19–21] use the neighborhood information from the graph data space in convolution operations. The main strategy of these methods is to convert the convolution of the graph data into an inner product of the neighborhood information in the graph data space. However, it is challenging to find a convolution operation that is translation-invariant for irregular data. The spectral-based methods [22–25] typically use the Laplacian to transform the graph data and then use the eigenvector as the convolution operator. The purpose of this transformation is to approximate the convolution operation of the graph data as a convolution operation of the regularized data. In recent work, researchers have attempted to design a graph-CNN architecture by

☆ Fully documented templates are available in the elsarticle package on CTAN.
* Corresponding authors.
E-mail addresses: wubo@nlsde.buaa.edu.cn (B. Wu), blonster@nlsde.buaa.edu.cn (Y. Liu), langbo@nlsde.buaa.edu.cn (B. Lang), huanglei@nlsde.buaa.edu.cn (L. Huang).

https://doi.org/10.1016/j.neucom.2018.09.008
0925-2312/© 2018 Elsevier B.V. All rights reserved.

employing a graph-labelling procedure for the construction of a receptive field. Niepert et al. [26] proposed a framework for learning CNNs for graphs. To a certain extent, the methods mentioned above solve the problem of applying a CNN to graph data. However, all of these methods require the graphs to be transformed into the same neighborhoods with the same ordering. This process is called graph regularization and involves the conversion of graph data into a data format that can be processed by standard CNNs.

The local receptive field of a graph is similar to the fixed-size neighborhood of an image. However, the numbers of neighboring nodes of each node in the graph are not fixed. The standard practice is to regularize the neighborhood of the node: First, a threshold value is fixed. If the neighborhood size is less than the threshold, the neighborhood will be filled with zeros, which is equivalent to adding invalid information. When the neighborhood size is greater than the threshold, we interpret the threshold size of the node as a neighborhood, which results in the loss of some of the effective neighborhood information. However, this approach cannot reflect the neighborhood information of real nodes. This type of model can support continuous labelling with graph data but requires the graph or node neighborhoods of the graph to be transformed into fixed-sized representations to meet the processing requirements of the CNN. Therefore, this method can result in the loss of important information due to the padding and interception operations. Thus, one of the challenges in improving the effect of applying CNNs to graph data is that the neural network model can perform convolution operations directly on irregular node neighborhoods and can perform parametric learning.

To address these limitations, we present a graph convolutional neural network (g-CNN) model that can perform feature learning on graph data directly. We use a continuous mapping function (which is based on a mixed Gaussian process) between the irregular local neighborhood and the convolution weight to transform the discrete parameter learning problem into a parameter sampling problem of a continuous function. Therefore, parameter sampling becomes a function of the features in the preceding layer of the network rather than being based on manually defined parameters on the graph, as in previous studies.

The main innovation of our model is that it does not need to convert the graph or its node neighborhoods into fixed structures; instead, the model learns the irregular structural data directly and can optimize the graph convolution kernel through the neural network. Thus, the model is called the disordered graph convolutional neural network (DGCNN). We conduct experiments on multiple standard graph datasets, and the experimental results show that the proposed method outperforms the existing g-CNN methods and other types of methods in graph classification and retrieval.

The remainder of this paper is organized as follows. In Section 2, we introduce the relevant work on g-CNNs and graph kernels (g-kernels). In Section 3, we introduce the model structure. In Section 4, we introduce the DGCL. In Section 5, we describe the experiment and present the results of our method and the comparison methods. Finally, in Section 6, we discuss the results and present our conclusions, respectively.

2. Related work

Current graph processing methods can be divided into two categories: traditional kernel approaches and g-CNNs. The g-CNN approaches often apply standard CNNs to graph data feature learning, while traditional kernel approaches typically use non-linear projection to transform sample graph data into a higher-dimensional feature space, where analysis and processing are performed.

2.1. G-kernel approaches

G-kernel approaches project a graph into a feature vector space; the similarity of the two graphs is their scalar product in the space. A g-kernel often defines the similarity function for two graphs. Multiple g-kernels have been proposed, such as the random-walk (RW) kernel, the shortest-path (SP) kernel and the sub-tree kernel.

Gartnerj et al. [27] proposed an RW kernel function based on computing the RW kernel functions of common steps for two graphs and proved that this function is a positive-definite function. However, the RW g-kernel function has two disadvantages. First, for both of the g-kernels, the comparison of RW paths is of enormous computational complexity. Second, an RW path often contains multiple repeated points and edges, which influences the computational efficiency of RW g-kernel functions. Weisfeiler [28] proposed the WL sub-tree g-kernel, which is based on the one-dimensional WL isomorphism algorithm. This algorithm searches for the sub-tree structure that is shared by two graphs. However, the WL kernel supports only discrete features, and the memory consumed by the WL kernel is proportional to the number of training samples. The SP kernel [39] calculates the similarity by comparing every pair of edges in SP graphs. Shervashidze [29] proposed a graphlet count kernel (GK) function based on the sub-graph structure. A graphlet is a small-sized sub-graph that often contains 3–5 nodes. Due to the lack of an effective approach for node labelling, this GK function is not applicable to datasets that are focused on node labels.

2.2. Graph convolutional neural networks

CNNs are applied to graph data in two broad categories of research: spectral filtering methods and local filtering methods. In the field of spectral filtering methods, Henaff et al. [24] used feature vectors of graph Laplacians to perform convolution and used a weighted distance to construct the similarity matrix. Defferrard et al. [12] proposed a network model based on ChebNet, which is a spectrally defined method with space attributes. In this model, Chebyshev polynomials of the Laplacian are used to learn k-hop neighborhoods of graph data, thereby incorporating spatial information into neighborhoods. Kipf and Welling [30] derived a semi-supervised g-CNN approach by simplifying and extending the ChebNet-based model. All of these approaches require a fixed graph data structure. In the field of local filtering methods, Atwood and Towsley developed a diffusion convolutional neural network (DCNN) that performs RWs in graph data to select the neighborhood structure in the space as the input for the CNN. However, the DCNN is of complexity $O(N^2)$, which restricts the extendibility of this approach. Bruna et al. [11] proposed a multi-scale cluster-based g-CNN model in which convolution defines the weight of each non-shared attribute of each cluster. Duvenaud et al. [12] developed a local space filter that can be applied to any node and its neighborhood. Li et al. [31] used distance learning and a learning adaptive graph Laplacian matrix in training to enable spectral convolution on data of diverse graph topologies and scales. Niepert et al. [26] proposed a method that can obtain the local receptive field of graph data and apply it to a CNN, which includes three steps: 1) select a node; 2) construct the fixed-size neighborhood of this node to form a fixed-size sub-graph; and 3) regulate the neighborhood sub-graph. It is possible to obtain a one-dimensional data unit that can be processed by a standard CNN using these three steps. However, both spectrally and spatially defined methods need to transform graph data into data structures with fixed scale, and feature information loss during the transformation process is unavoidable.

Unlike previous work, the g-CNN model we propose, namely, DGCNN, is specially designed for the disordered features of node neighborhoods and can perform convolution from irregular neighborhood structures while achieving the back-propagation of graph convolution without transforming the graph data structure into a fixed regular structure. After parameter sampling based on the Gaussian mixture model (GMM), the DGCNN can perform convolution operations on irregular and disordered neighborhood structures.

3. Notation and problem formulation

A graph G with n nodes and m edges is denoted by a pair $G = (V, E)$, where V is a finite set of nodes with $|V| = n$ and $E \subset V^*V$ is a set of edges with $|E| = m$. The nodes and edges of the graph have labels. We further represent a graph with selected key nodes and their neighborhood. A neighborhood $Ng(i) = \{j; (j, i) \in E\} \cup \{i\}$ of node i is defined to contain all the adjacent nodes including i itself. Based on the characteristics of graph data, we define a graph convolutional neural network model that can perform feature learning on graph data directly. Our model computes the convolution layer of graph G by calculating the convolution value of its key node i, i.e, $X(i)$, $X(i)$ is the weighted sum of $X(j)$ where $j \in Ng(i)$. Such a commutative aggregation can solve the problem of varying neighborhood sizes and undefined node ordering and will not lose any structural information. The graph convolution operation, which is called Disordered Graph Convolution (DGC), is formalized as follows:

$$X(i) = \sum_{j\in Ng(i)} F(L(i,j))X(j) + b \\ = \sum_{j\in Ng(i)} W_i X(j) + b \tag{1}$$

where $X(i)$ is the convolution value of node i, $L(i, j)$ is the edge label value between node i and node j, b is a learnable bias, W_i is weight matrix of node i. W_i and b are model parameters updated only during training. $F()$ is the graph convolution filter function, which can be constructed using continuous functions or learning networks. The function of $F()$ is to dynamically generated weight matrix W_i for a node i, which is the core of our DGC model.

4. Model structure and preprocessing

The key step for the application of CNNs to normalize grid data is to use a window of size $k*k$ to capture the local neighborhood of the image and share the corresponding convolution kernel parameters in the window. Because of the randomness of the neighborhoods of graph nodes, the traditional window translation method is not applicable to graph data, and a g-CNN model is proposed for accommodating random node neighborhoods.

When processing an image in the framework of the standard CNN model, the local receptive field is used to implement the convolution operation on the data according to a step movement and obtain the local features of the graph, as shown in Fig. 1(a), where the convolution window size is fixed to W^*H. Due to the normalization of the pixel position of the image, the local receptive field can be moved from left to right and from top to bottom to obtain the local information of the image. As shown in Fig. 1(b), the neighborhoods of the different nodes correspond to various receptive fields of the convolution processes. The node neighborhoods of the graph have no fixed scale or order; thus, the convolution cannot be implemented directly on the graph using the fixed-sized and ordered convolution kernels.

To overcome the above problems, we propose a disordered g-CNN model that can be applied to arbitrary graph data. The network in this model can learn the parameter mapping between the random node neighborhoods and the convolution weights of the graph.

4.1. Model structure

As shown in Fig. 2, in the DGCNN, the graph data are first transformed into receptive fields that can be processed by the CNNs. Then, a convolution operation is performed over the kernel parameter matrix and the receptive field. Finally, the g-CNN takes the output of the convolutional layer (CL) as the input data of the standard overall connection layer. Our model contains the following parts:

(1) Key node selection: The selection operation is implemented on the graph data to obtain a fixed number of key nodes. To ensure that the number of neighborhoods of the nodes in each graph is consistent, the same number of key nodes is sampled for each graph.

(2) Neighborhood assembly: The nodes of the k-neighborhood are the candidates for the receptive field. Note that this time, the receptive field is disordered.

(3) Parameter sampling: The corresponding convolution kernel parameters, are sampled based on the mixed Gaussian model according to the information of the nodes in each neighborhood to implement the convolution operation for each neighboring graph with its corresponding convolution kernel parameters.

(4) Graph convolution: For each input graph G, the proposed model creates a fixed number of the receptive field, and for each receptive field, the model creates corresponding convolution kernel parameters. The convolution operation is applied in the convolution layer by using these two inputs.The convolutional layers have several filters and rectified linear units.

(5) The Fully Connected Layer (FCL) and softmax: The model integrates multiple-kernel convolution results through Fully Convolutional Networks(FCN) and softmax.

By combining the DGCL with the standard CNN and the output layers, the g-CNNs can be built and can directly learn the neighborhood of any random node.

4.2. Preprocessing

The preprocessing procedure was implemented on each input graph data, as shown in Fig. 2, which includes node sampling and node neighborhood construction:

(1) Sequence sampling of key nodes: To sort all the nodes in the graph, the method proposed in [26] was adopted, and a graph labelling function was introduced in which the set of nodes in the graph are mapped to an ordered node sequence according to the centripetal parameters (e.g., the node degree or centrad). From the sequence, w nodes are alternately selected according to a certain interval s to form the ultimate node sequences. The width w is equal to the average number of nodes in graph datasets.The stride s determines the distance, relative to the selected node sequence, between two consecutive nodes for which a receptive field is created (s = (number of all nodes)/w). As shown in Fig. 3, the nodes in the graph are sorted first, and then four nodes are alternately selected as the key node sequence according to the interval $s = 2$. The sampling operation is described in Algorithm 1.

(2) Node neighborhood construction: As shown in Fig. 3, for each node in the node sequence that was obtained in the previous step, breadth-first searching is used to find the neighboring nodes, which form the neighborhood set of the original key nodes. The node in each node neighborhood should contain the attribute (such as the weight of the edge or the similarity) between the node and the key node and the attributes of the node (such as the node category). Algorithm 2 describes the Neighborhood Assembly process. From algorithm 2, we can see that neighborhood nodes can contain 1-hop or more than 2-hop nodes.

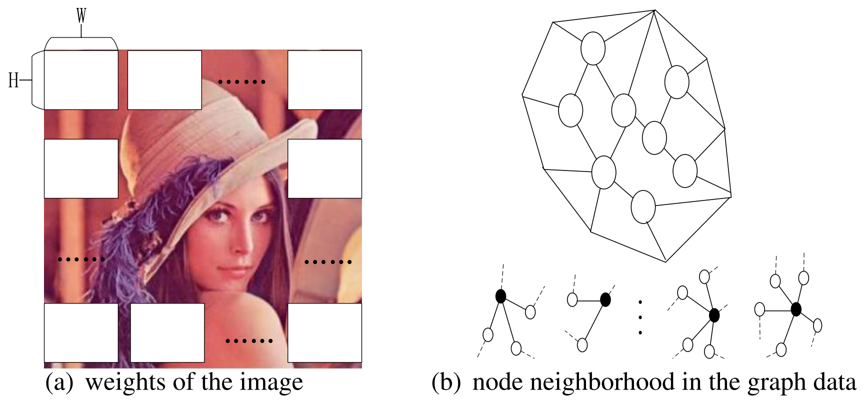


Fig. 1. Receptive fields of an image and the corresponding graph.

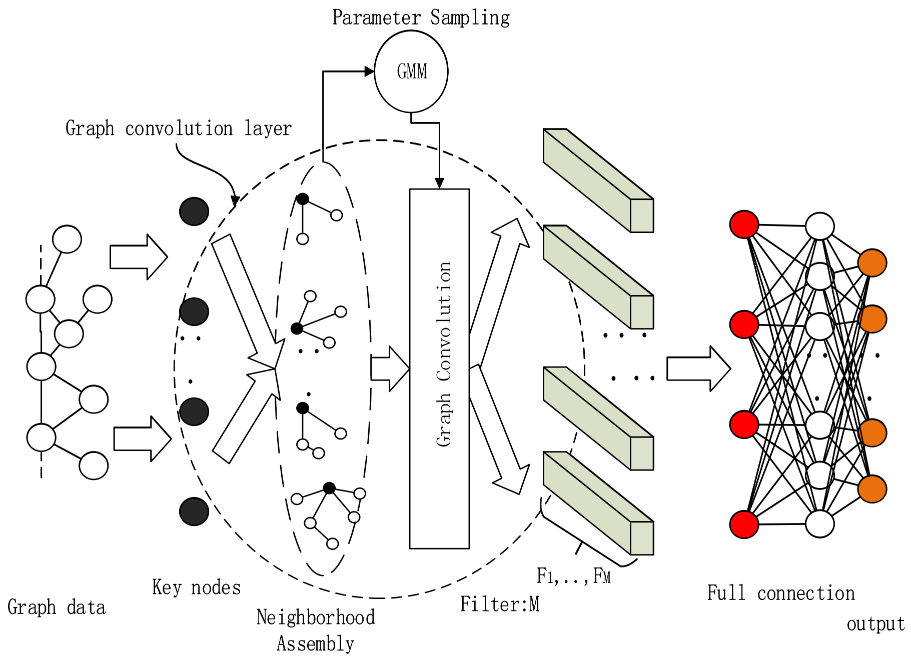


Fig. 2. Illustration of the proposed architecture.

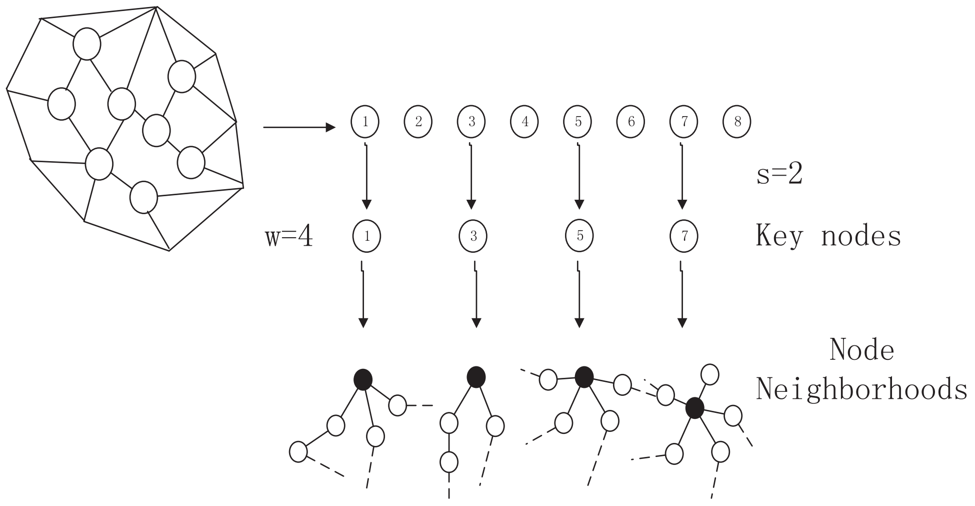


Fig. 3. Preprocessing procedure of graph data.

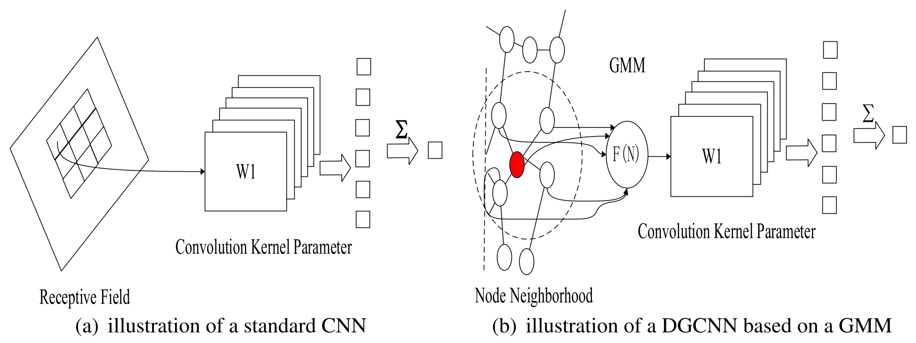

Receptive Field
W1
Convolution Kernel Parameter
Σ
(a) illustration of a standard CNN
Node Neighborhood
GMM
F (N)
W1
Convolution Kernel Parameter
Σ
(b) illustration of a DGCNN based on a GMM

Fig. 4. Convolutional process comparison between the standard image and a node neighborhood of the graph data.

Algorithm 1 Sequence sampling of key nodes.

```
input : graph $G = (V, E)$, stride $s$, number of receptive fileds $w$
output : key node Sequence $f$
$Vsort$ = the set of top $w$ elements of $V$ according to node degree
$i = 1,\ j = 1$
for $j = 1$ to $w$ do
    if $i < |Vsort|$ then
        $f[i] = Vsort[i]$
        $j = j + 1$
    else
        $f[i] = 0$
        $j = j + 1$
    end if
    $i = i + s$
end for
```

Algorithm 2 Neighborhood assembly.

```
input : vertex $v$, receptive hop $k$
output : set of neighborhood nodes $N$ for $v$
$N = [v]$
$L = [v]$
$hop = 1$
for $hop = 1$ to $k$ do
    if $L > 0$ then
        $L = \bigcup_{v \in L} N(v)$
        $N = N \bigcup L$
    else
        $L = 0$
    end if
end for
```

After the two steps of input graph data preprocessing, the input data are transformed into a random neighborhood set with a fixed size, which is similar to the local receptive field set of the graph.

5. DGCL and its learning process

In the DGCNN, a disordered graph CL(DGCL) that can receive and process any graph data is designed. A DGCL is a CL that can perform convolution operations on irregular and disordered node neighborhoods while achieving the back-propagation of the CNN.

5.1. Kernel construction

For a DGCL, the input is the node neighborhood structure that was obtained after graph preprocessing. Fig. 4(a) shows a standard CNN with a receptive field. The receptive field is ordered and fixed in size. For graph data, the neighborhood of each node is disordered and of variable size, as shown in Fig. 4(b). The node neighborhood in Fig. 4(b) contains 5 nodes in a broken circle. The red node is the key node, and the others are its neigh boring nodes in the broken circle. The convolutional kernels for these five nodes must be obtained before performing the convolution operation. An existing solution is to define a fixed-sized convolutional kernel, which requires a regularization process that may lead to information loss. In contrast, we sample the convolutional parameters of each neighborhood node on the continuous function for the similarity of the neighborhood nodes and the key node, and the number of convolutional parameters is the same as the number of key node neighborhoods. Algorithm 3 describes the procedure.

Algorithm 3 Sampling W from GMM.

```
input : neighbor graph $Ng = (V, E)$, number of gaussian component $D$, number of nodes in neighbor graph $m$, mean $\mu$, variance $\sigma$
output : Weight matrix $W$
$GMM(Ng;\ D, \mu, \sigma)$ Initialization
$i = 1$
for $i = 1$ to $m$ do
    $W[i] = GMM(Ng.E[i]; D, \mu, \sigma)$
end for
Return $W$
```

According to the central limit theorem [32], the inherent probability distribution can reasonably be approximated by a GMM since it can theoretically approximate any probability distribution. A GMM model has more parameters than a single Gaussian model; therefore, a GMM model can fit the convolution kernel better than a single Gaussian model. When the number of Gaussian components is 1, the GMM model will degenerate into a single Gauss model.

A GMM is given in Formula 2, and the convolution function for such a neighborhood and sampled kernel parameters is defined in Formula 3.

$$GMM(\theta|D, \mu, \sigma) = 1/(2\pi)^{D/2} * 1/(|\sigma|)^{1/2} * exp((-1/2) * (\theta - \mu)^T * \sigma^{-1} * (x - \mu)) \tag{2}$$

$$\begin{aligned} F(Ng, D) &= (GMM(\theta|D, \mu, \sigma), X) \\ &= \sum_{k=1}^{K} \left(\sum_{i=1}^{D} w_i G(\theta_k, \mu_i, \sigma_i), X_k \right) \\ &= \sum_{k=1}^{K} \left(\left(\sum_{i=1}^{D} w_i G(\theta_k, \mu_i, \sigma_i) \right) * X_k + B \right) \end{aligned} \tag{3}$$

where Ng denotes the neighborhood node and edge set of a key node and $Ng = < V_{set}, E_{set} >$, D is the number of Gaussian components, K is the size of V_{set}, X is the attribute value of a node according to this map, parameter X_k is the attribute of the $k-th$ node, θ

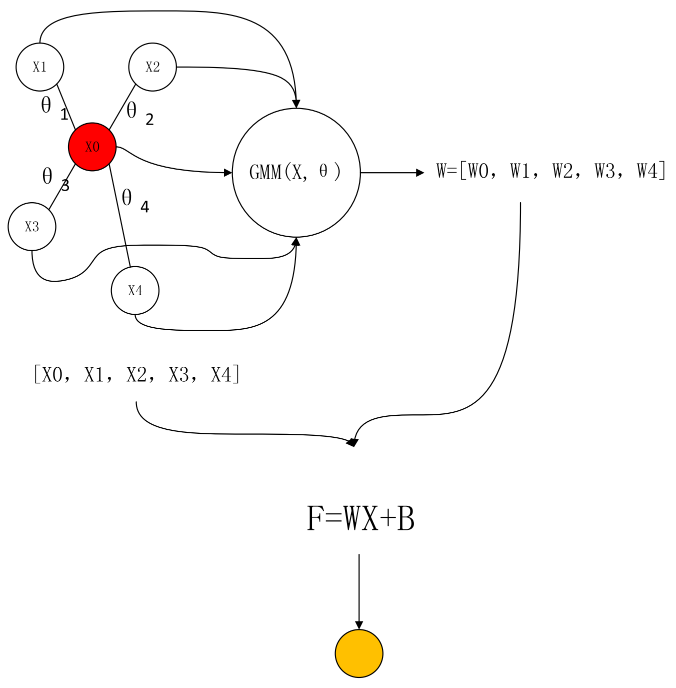


Fig. 5. Convolutional unit of a neighborhood.

is the attribute of the E_{set}, w_i is the weight of each Gaussian component, μ_i and σ_i are the mean value and variance of each Gaussian component, respectively, and $G(\cdot)$ is the Gaussian function. B denotes a vector of bias terms. Algorithm 3 lists the sampling convolution kernel from the GMM steps.

5.2. Convolution operation

After sampling the convolutional parameters for all key nodes, convolution operations are performed on the neighborhood to finish the graph data convolution in the graph CL. The convolutional processing of the graph data represents the forward propagation of the neural network, as shown in Fig. 4(b).

In Fig. 5, the red node in the input part represents the key node, the white nodes are the neighboring nodes of the key node. The yellow node is the output value. θ_0 represents the attribute values of edges between the key node and itself. Therefore, θ_0 is a constant (in this study, $\theta_0 = 0$), and $\theta_1, \theta_2, \ldots, \theta_k$ represent the attribute values of edges between neighboring nodes and the key node, X_0 represents the attribute values of the key nodes, $X_1, X_2, \ldots, X_k$ are the attribute values of the nodes. W is the convolution parameter matrix and B denotes a vector of bias terms. Although we use 1-hop nodes of the key node as an example, the DGCL also can use k-hop nodes of the key node as the receptive field.

As for the jth receptive field, we can obtain the forward output for which the graph convolutional process is performed on its receptive field.

$$f_j = \sum_{i=0}^{I} GMM(\theta_i; D, \mu_i, \sigma_i)X_i + b \tag{4}$$

where I denotes the size of the neighborhood sub-graph, X_i denotes the attribute value of the ith node, $GMM(\theta_i;\ D,\ \mu_i,\ \sigma_i)$ denotes mixed Gaussian values of the ith node, and $b \in R^E$ denotes a vector of bias terms. Algorithm 4 describes the graph convolution operation.

Algorithm 4 Graph convolution.

```
input : neighbor graph $Ng = (V, E)$, Weight matrix $W$, number of nodes $n$
output: convolution result $f$
$i = 1,\ f = 0$
for $i = 1$ to $n$ do
    $f = f + Ng.V[i] * W[i]$
end for
Return $f$
```

5.3. Graph convolution back-propagation and Gaussian parameter learning

In the DGCL based on the GMM, the parameters of each component of the GMM must be optimized. The difference between the output value of CNN and the real value is then used for back-propagation to adjust the parameters. The error function for back-propagation is defined in formula (5) [33]:

$$\begin{aligned} error(\theta) &= \frac{1}{2}\sum_{a=1}^{A}\sum_{b=1}^{B}(t_b^{(a)}(\theta) - f_b^{(a)}(\theta))^2 \\ &= \frac{1}{2}\sum_{a=1}^{A}\sum_{b=1}^{B}(\nabla f_b^a(\theta))^2 \end{aligned} \tag{5}$$

where $t_b^{(a)}$ is the ath dimension of the corresponding label of the bth graph data, $f_b^{(a)}$ is the similarity value of the ath output layer unit in response to the bth input pattern, B is the number of graph data types, and A is the number of graph data.

For the ath graph data, we can immediately compute the gradient:

$$\begin{aligned} \nabla f_\theta^a &= \frac{\partial f^a}{\partial(w_1, \mu_1, \sigma_1, \ldots, w_D, \mu_D, \sigma_D)} \\ &= \left(\frac{\partial f^a}{\partial w_1}, \frac{\partial f^a}{\partial \mu_1}, \frac{\partial f^a}{\partial \sigma_1}, \cdots, \frac{\partial f^a}{\partial w_D}, \frac{\partial f^a}{\partial \mu_D}, \frac{\partial f^a}{\partial \sigma_D}\right) \end{aligned} \tag{6}$$

where f denotes the output of forward-propagation; w, μ, σ are the weight, mean value and variance of each Gaussian component, respectively; and D is the number of Gaussian components. We need to calculate the derivative and parameters of the Gaussian component:

$$\frac{\partial f}{\partial w_i} = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}} \tag{7}$$

$$\frac{\partial f}{\partial \sigma_i} = \frac{w_i}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}} [-1 + \frac{(x-\mu_i)^2}{\sigma_i^2}] \tag{8}$$

$$\frac{\partial f}{\partial \mu_i} = -\frac{w_i * (x-\mu)}{\sqrt{2\pi}\sigma_i^3} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}} \tag{9}$$

$$w_{i+1} = \lambda * \frac{\partial f}{\partial w_i} \tag{10}$$

$$\sigma_{i+1} = \lambda * \frac{\partial f}{\partial \sigma_i} \tag{11}$$

$$\mu_{i+1} = \lambda * \frac{\partial f}{\partial \mu_i} \tag{12}$$

where $w_{i+1}, \mu_{i+1}, \sigma_{i+1}$ are the parameters that are obtained after updating w_i, μ_i, σ_i, respectively, and λ is a learning rate parameter. In practice, there is often a learning rate parameter λ for each Gaussian component.

The computation of the gradient of bias b is the same as that for the traditional CL and is explained here.

The major factor influencing the convergence of our model is the number of Gaussian components, which is proportional to the complexity of the model. Increasing the number of Gaussian components in a certain range will give our model fast convergence. In the experiment section, we examine the influence of the number of Gaussian components.

5.4. Number of parameters and computational complexity

Each weight matrix W is obtained by sampling the mixed Gaussian model, and the number of weight matrices is equal to the number of neighboring nodes of key nodes. The parameters in our method with respect to a conventional CNN are the Gaussian component weight w, mean value μ, and variance σ for each vector. Let N be the number of nodes in the graph and M be the number of Gaussian components. The total number of parameters is 3^*N^*M. Here, we ignore the bias terms, which contribute few parameters.

The complexity of the DGCNN consists of the forward and back-propagation complexities. Let k be the number of key nodes in the graph. Let E denote the average number of neighbors of each node. For the forward-propagation process, the DGCNN has a worst-case complexity of $O(k^*M^*E)$, and for the back-propagation process, the DGCNN has a worst-case complexity of $O(3^*k^*M^*E)$. Let T be the number of graphs. The total computational complexity is $O(4^*T^*k^*M^*E)$.

6. Experiment

6.1. Graph datasets

We conduct our experiments on the following popular real datasets to compare our approach with state-of-the-art g-kernels and CNNs in terms of retrieval precision:

- PTC [34]: PTC consists of 344 chemical compounds, where the classes indicate carcinogenicity for male and female rats.
- D&D [35]: D&D is a data set of 1178 protein structures classified into enzymes and non-enzymes.
- AIDS [36]: The dataset contains 4395 chemical compounds, including 423 belonging to class CA, 1081 belonging to CM, and the remaining compounds belonging to CI.
- PROTEIN [26]: PROTEINS is a graph collection in which nodes are secondary structure elements and edges indicate neighborhoods in the amino-acid sequence or in 3D space. Graphs are classified as enzymes or non-enzymes.
- COLLAB [37]: The dataset contains 12,000 graphs, each with an average of 400 nodes. The dataset contain users, movies, and the users ratings of the movies.
- CiteSeer [38]: The CiteSeer dataset consists of 3312 scientific publications classified into six classes. The citation network consists of 4732 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word in the dictionary. The dictionary consists of 3703 unique words.
- Cora [38]: The Cora dataset consists of 2708 scientific publications classified into seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word in the dictionary.
- The Oxford Buildings Dataset(referred to as OXFB): The dataset consists of 5062 images collected from Flickr by searching for particular Oxford landmarks. It has a set of 55 queries for object retrieval system evaluation. Fig. 6 shows part of the sample images in the OXFB dataset. In constructing an attributed graph (ARG) for an image, each region can be represented as a node; an edge connecting two nodes is inserted if two regions are adjacent. Each node in the ARG retains the important features of the corresponding region as its characteristic parameters, such as color, texture, and region size. For the detailed operation, refer to [26].

The properties of the datasets can be found in Table 1, which shows the variability in dataset sizes, graph sizes, classes, max node numbers, average node numbers and average node degrees.

6.2. Experimental configuration

(1) We compare the DGCNN method that is proposed herein with the following methods by experiments on graph classification and graph search:
  - G-kernel method: the SP kernel [39], the RW kernel [27], the GK kernel [40], and the Weisfeiler-Lehman subtree kernel (WL) [28].
  - g-CNNs: PATCHY-SAN [26], LMFGCN [41], and SSGCN [30].

(2) We test the influence of the number of Gaussian components on the proposed model using 5, 10, 15, 20, 25, 30 and 35 components and find the optimal number, at which the best effect is obtained;

(3) We perform efficiency analysis.

In our experiment, when calculating the Weisfeiler-Lehman fingerprint, the number of iterations is set as $h = 10$, the GK parameter is set as 7, and the factor of RW is set as $10^{-6}, 10^{-5}, \ldots, 10^{-1}$. For PATCHY-SAN, the fixed receptive field in this paper is $k = 5$. The network structure consists of two CLs, which are one-dimensional (5^*1); one dense hidden layer; and one softmax layer. The CNN uses 3^*3 filters. The network structure of SSGCN is consistent with that of PATCHY-SAN. To obtain fair comparison results, for the graph classification experiment, the network structure in the DGCNN consists of two graph convolution kernels, one standard CL, one dense hidden layer and one softmax layer. The learning rate is set to 0.05. The number of components of the GMM is set to 25. We use 2-hop nodes of the key node as the receptive field. In the graph search experiment, the network structure is the same as that used for graph classification. In this study, the output of the dense hidden layer is treated as a feature vector of the graph data.

All the experiments are performed under the following configuration: an Intel Xeon X5650, a dual-core CPU running at 2.66 GHz with 8 GB memory, and a Linux system. The methods that use CNNs are implemented using the torch framework.

6.3. Experimental results

(1) *Comparison of the graph classification results*

Graph classification is performed to determine the category of each graph datum in the graph datasets. Table 2 shows the graph classification accuracy for each dataset with eight methods: SP, RW, GK, WL, PSCH, LMFGCN, SSGCN and DGCNN. The DGCNN, the method presented here, demonstrated good accuracy for most datasets and achieved its best results for the PTC, AIDS, PROTEIN and OXFB datasets. The relatively high variance can be explained by the small size of the benchmark data sets and the fact that the g-CNNs hyperparameters were not tuned to individual data sets. Unlike traditional g-kernel approaches, the DGCNN exhibits a significant advantage on most datasets. Existing g-CNNs use the standard CNN model and predefine only a one-dimensional convolutional kernel (e.g., 1^*5) in the convolutional feature-learning process without making full use of the space information between the key nodes and their neighborhood information. The method proposed here transforms the discrete learning process into a process of projecting continuous functions and fully learns the relationships between the neighboring nodes and key nodes in the graph data. Therefore, on the most datasets, DGCNN outperforms the existing g-CNN methods and g-kernel methods.

Fig. 6. Sample images of OXFB.

Table 1
Statistical information of the datasets.

Dataset	PTC	AIDS	PROTEIN	D&D	OXFB	CiteSeer	Cora	COLLAB
Graphs	344	4395	600	1178	5062	3312	2708	12,000
Classes	2	3	2	2	11	6	7	12
Max nodes numbers	109	207	620	350	4732	5429	5513	951
Avg nodes numbers	25.56	30.15	39.06	284.32	398.12	3703.21	3781.15	400
Avg nodes degrees	12.12	15.32	16.12	60.42	30.71	80.15	83.41	35.12

Table 2
Comparison of classification precision between four graph kernel and two graph CNN methods on multiple graph datasets.

Dataset	PTC	AIDS	PROTEIN	D&D	OXFB
SP	58.53 ± 2.55	62.37 ± 1.13	65.12 ± 1.01	71.00 ± 1.11	68.23 ± 2.42
RW	57.26 ± 1.30	58.37 ± 2.21	68.11 ± 2.02	68.10 ± 0.11	67.20 ± 3.21
GK	57.32 ± 1.13	60.27 ± 1.12	61.12 ± 1.03	$\mathbf{78.45 \pm 0.26}$	63.14 ± 4.13
WL	56.97 ± 2.01	59.97 ± 1.01	62.23 ± 1.03	77.95 ± 0.70	64.42 ± 2.34
PSCH [26]	59.43 ± 3.14	60.10 ± 2.72	72.10 ± 1.72	74.58 ± 2.85	74.01 ± 3.33
LMFGCN [41]	61.32 ± 1.21	59.21 ± 2.32	67.12 ± 2.25	73.13 ± 1.13	74.21 ± 2.23
SSGCN [30]	60.21 ± 2.14	55.10 ± 1.15	65.10 ± 3.12	75.10 ± 1.02	73.10 ± 2.31
DGCNN	$\mathbf{65.43 \pm 3.14}$	$\mathbf{65.10} \pm \mathbf{1.82}$	$\mathbf{75.10 \pm 2.72}$	77.21 ± 0.85	$\mathbf{74.30 \pm 3.32}$

Table 3
Comparison of accuracy results on large graphs.

Dataset	GK	PSCH	LMFGCN	SSGCN	DGCNN
Citeseer	63.57 ± 0.21	64.24 ± 0.34	65.24 ± 0.83	67.90 ± 0.50	$\mathbf{68.34 \pm 0.13}$
Cora	70.45 ± 0.14	72.32 ± 0.45	74.13 ± 0.01	80.10 ± 0.50	$\mathbf{80.34 \pm 0.15}$
COLLAB	64.45 ± 0.12	62.32 ± 0.45	66.13 ± 0.32	63.10 ± 0.12	$\mathbf{68.34 \pm 0.13}$

Then, we test the proposed approach using different methods and large graph datasets. The DGCNN is also highly competitive for large graph data, significantly outperforming the other four methods. Table 3 lists the results of the experiments.

(2) *Comparison of the graph retrieval results*

The experiment studies a graph similarity search that retrieves all graphs from a database that approximately match the query graph under the similarity measure. The experiment compares the DGCNN with existing methods in terms of graph similarity retrieval performance for small datasets and large graph datasets. We use the output of the dense hidden layer as the feature vector of the graph data. To ensure a fair comparison, for PATCHY-SAN, we also use the output of the dense hidden layer as the feature vector of the graph data.

As indicated by the experimental results shown in Fig. 7, for the PTC, PROTEIN, COLLAB, AIDS and OXFB datasets, the DGCNN outperforms the other methods. On the D&D dataset, the DGCNN method performs similarly to the optimal GK method. The DGCNN outperforms the recently proposed g-CNN methods on most datasets because the process of regularizing node neighborhoods leads to loss of information about the node neighborhoods, whereas our convolutional kernel method, which is based on the GMM, builds a dynamic graph convolutional kernel, which eliminates the local information loss during node neighborhood regularization.

Table 4 shows the mAP results of the experiments. mAP is calculated with reference to Formula (12). The DGCNN significantly outperforms the other four methods on two of the three datasets, i.e., Citeseer and COLLAB, and performs the same as the other methods on the other datasets. For the Citeseer dataset, the proposed DGCNN method provides the second best result of 73.34.

Table 4
Comparison of the mAP results with graph retrieval methods for large graph datasets (in percent).

Dataset	GK	PSCH	LMFGCN	SSGCN	DGCNN
Citeseer	67.57	70.24	69.31	72.12	**73.34**
Cora	60.25	72.32	71.13	**73.15**	73.04
COLLAB	64.45	65.32	67.62	68.26	**68.34**

$$mAP = \int_0^1 P(R)DR \tag{13}$$

where R is recall and P is precision.

(3) *Influence of the number of Gaussian components*

The purpose of this experiment is to examine the influence of the number of Gaussian components on the DGCNN. In the experiment, the number of Gaussian components is defined as m, which is set as 5, 10, 15, 20, 25, 30 or 35. For each case, we perform the same experiment on multiple graph datasets and calculate the mean result as the final result. The experimental results are shown in Fig. 8.

For dataset PTC, the classification precision increases significantly from $m = 5$ to $m = 15$ and is stable when m is larger than 15. For the AIDS dataset, the classification precision increases significantly from $m = 5$ to $m = 20$ and is relatively stable when m is larger than 20. Similarly, for the PROTRIN and D&D datasets, the classification precision increases significantly from $m = 5$ to $m = 15$ and from $m = 5$ to $m = 25$, respectively, and is stable when m is larger. For the dataset with many graphs, namely

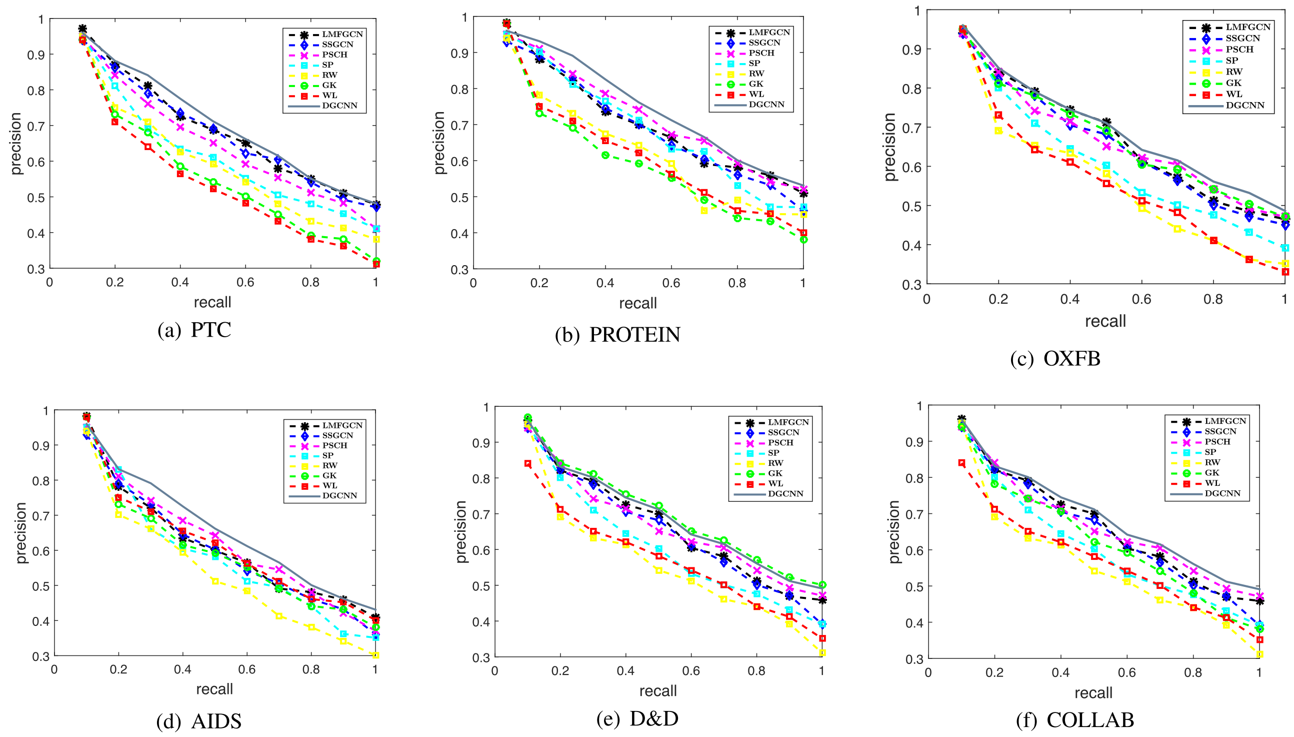


Fig. 7. Retrieval precision on five graph datasets for DGCNN, graph kernel methods and recent graph convolution networks.

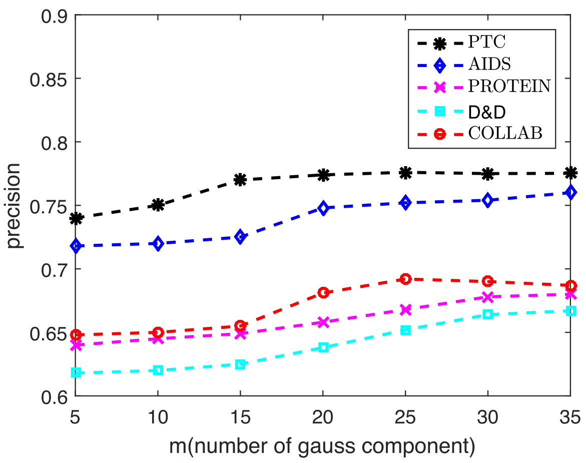


Fig. 8. Relationship between number of Gaussian components and classification precision for different datasets.

COLLAB, more parameters are needed to make the model fit the data. We found that $m = 25$ gives the best results on this graph dataset.

These experiments indicate that the optimal numbers of Gaussian components for different datasets vary and allow fast model convergence. Increasing the number above the optimal number does not promote precision.

In the next experiments, we use the optimal number of Gaussian components with the best classification result for each dataset.

(4) *Efficiency analysis*

We assess the efficiency of the DGCNN by applying it to graph datasets. For a given graph, we compute the end-to-end running time. The results in Table 5 show that the running time of the DGCNN is less than that of WL. In addition, the average values for other graph convolutional approaches on different datasets are 11.7 s (for PTC), 18 s (for AIDS), 44 s (for PROTEIN), 126 s (for D&D),

Table 5
Comparison of running time on five graph datasets (in seconds).

Dataset	PTC	AIDS	PROTEIN	D&D	COLLAB
WL [28]	30	65	143	609	245
PSCH [26]	6	10	31	154	235
LMFGCN [41]	14	23	41	72	62
SSGCN [30]	15	20	58	150	200
DGCNN	16	25	57	152	212

and 165 s (for COLLAB). The results of the DGCNN are somewhat slower (the gap is 4.3 s for PTC, 7 s for AIDS, 13 s for PROTEIN, 26 s for D&D, and 47 s for COLLAB) than the other graph convolutional approaches, probably because generating the receptive fields takes less time than sampling the kernel parameters on the GMM. The recent graph convolution approaches need to generate the receptive fields as the input of the CNN. In the CL of our approach, additional computation is required to sample the parameters on the GMM. However, the total running time of these additional computations is not significantly different and is of the same order of magnitude.

7. Discussion and conclusions

We have presented a DGCNN based on a GMM that is applicable to graph similarity matching. The main innovation of this model is that by sampling the corresponding convolutional kernel parameters from a mixed Gaussian distribution, the dynamic convolutional kernel is adapted to the size and order of the node neighborhood. Therefore, our model supports different scales of convolutional receptive fields, thereby avoiding the loss of graph information during the regularization of node neighborhoods. The key aspect of our model is the GMM-based DGCL, which performs convolutional learning on local node neighborhoods of any graph while achieving the back-propagation of graph convolution.

Graph convolutional parameters are combined into a neural network optimization process and optimized to a large degree. Fi-

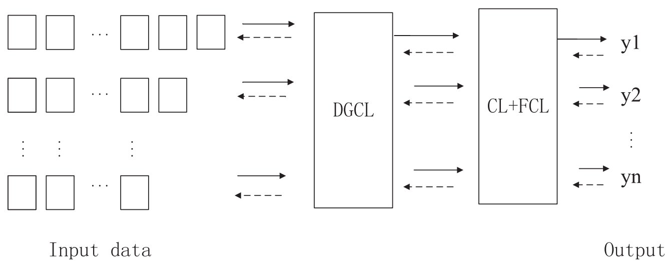


Fig. 9. The common model of the DGCL.

nally, we perform graph classification and search experiments on standard graph datasets such as PTC, PROTEIN, COLLAB and AIDS. These experiments indicate that the DGCNN outperforms existing g-kernels and g-CNNs.

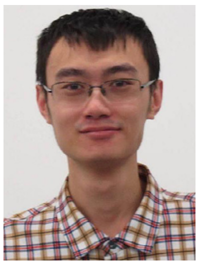
The main novelty of the DGCL is that our architecture changes the discrete parameter learning problem into a parameter sampling problem of the GMM. Therefore, the proposed approach does not rely on the format of the input data. Thus, the DGCL is also valid for feature learning on other irregular input data, such as text data and 3D shape data.

As shown in Fig. 9, we represent arbitrary data in matrix form. Then, the matrix serves as the input of the DGCL, and the output of the DGCL can serve as the input of a standard CL or a fully connected layer (FCL). Solid lines denote forward-propagation processes of networks and dotted lines represent back-propagation processes of networks. Therefore, DGCL can be combined with an arbitrary CNN to handle arbitrary regular and irregular data.

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Commun. Acm 60 (2) (2012) 2012.
[2] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, CoRRabs/1409.1556(2014). http://arxiv.org/abs/1409.1556
[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S.E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, CoRRabs/1409.4842(2014). http://arxiv.org/abs/1409.4842
[4] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, CoRRabs/1512.03385(2015). http://arxiv.org/abs/1512.03385
[5] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, CoRRabs/1411.4038(2014). http://arxiv.org/abs/1411.4038
[6] M. Mostajabi, P. Yadollahpour, G. Shakhnarovich, Feedforward semantic segmentation with zoom-out features, CoRRabs/1412.0774(2014). http://arxiv.org/abs/1412.0774
[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A.L. Yuille, Semantic image segmentation with deep convolutional nets and fully connected CRFs, CoRRabs/1412.7062(2014). http://arxiv.org/abs/1412.7062
[8] H. Noh, S. Hong, B. Han, Learning deconvolution network for semantic segmentation, IEEE International Conference on Computer Vision. IEEE (2015) 1520–1528.
[9] S. Ren, R. Girshick, R. Girshick, J. Sun, Faster r-CNN: Towards real-time object detection with region proposal networks, IEEE Trans. Pattern Anal. Mach. Intell. 39 (6) (2017) 1137–1149.
[10] J. Redmon, S.K. Divvala, R.B. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, CoRRabs/1506.02640(2015). http://arxiv.org/abs/1506.02640
[11] J. Bruna, W. Zaremba, A. Szlam, Y. Lecun, Spectral networks and locally connected networks on graphs, Comput. Sci. 51 (1) (2014) 204–212.
[12] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, Adv. Neural Inf. Process. Syst. (2016) 1–9.
[13] T. Komorowski, C. Landim, S. Olla, Fluctuations in Markov Processes, Springer Berlin Heidelberg, 2012.
[14] H.H. Lin, J.H. Chuang, T.L. Liu, Regularized background adaptation: a novel learning rate control scheme for gaussian mixture modeling, IEEE Trans. Image Process. Publ. IEEE Signal Process. Soc. 20 (3) (2011) 822–836.
[15] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, Comput. Sci. (2014) 212–226.
[16] S. Ren, et al., Faster R-CNN: towards real-time object detection with region proposal networks, International Conference on Neural Information Processing Systems, MIT Press, 2015, pp. 91–99.
[17] N. Verma, E. Boyer, J. Verbeek, Dynamic filters in graph convolutional networks, NIPS (2017) 84–95.
[18] M. Gori, G. Monfardini, F. Scarselli, A new model for learning in graph domains, in: Proceedings of the IEEE International Joint Conference on Neural Networks, IJCNN, 2005, pp. 729–734 vol. 2.
[19] F.P. Such, S. Sah, M.A. Dominguez, S. Pillai, C. Zhang, A. Michael, N.D. Cahill, R. Ptucha, Robust spatial filtering with graph convolutional neural networks, IEEE J. Sel. Top. Signal Process. PP (99) (2017) 1–11.
[20] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, Comput. Sci. 57 (2) (2016) 99–105.
[21] J. Atwood, D. Towsley, Search-convolutional neural networks, CoRRabs/1511.02136(2015). http://arxiv.org/abs/1511.02136
[22] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, CoRRabs/1312.6203(2013). http://arxiv.org/abs/1312.6203
[23] O. Rippel, J. Snoek, R.P. Adams, Spectral representations for convolutional neural networks, Adv. Neural Inf. Process. Syst. (2015) 2449–2457.
[24] M. Henaff, J. Bruna, Y. LeCun, Deep convolutional networks on graph-structured data, Comput. Sci. 54,59 (1) (2015) 92–99.
[25] D.K. Hammond, P. Vandergheynst, R. Gribonval, Wavelets on graphs via spectral graph theory, Appl. Comput. Harmonic Anal. 30 (2) (2011) 129–150.
[26] M. Niepert, M. Ahmed, K. Kutzkov, Learning convolutional neural networks for graphs, in: Proceedings of the International Conference on Machine Learning(2016) 2014–2023.
[27] T. Gaertner, P. Flach, S. Wrobel, in: On graph kernels: Hardness results and efficient alternatives, Proc. Colt. (2003) 129–143.
[28] B.Y. Weisfeiler, A.A. Leman, Reduction of a graph to a canonical form and an algebra arising during this reduction, Nauchno Tech. Inf. 9 (1975) 1015–1024.
[29] N. Shervashidze, S.V.N. Vishwanathan, T.H. Petri, K. Mehlhorn, K.M. Borgwardt, Efficient graphlet kernels for large graph comparison, Aistats 58 (1) (2009) 488–495.
[30] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, Adv. Neural Inf. Process. Syst. (2016) 1066–1075.
[31] R. Li, S. Wang, F. Zhu, J. Huang, Adaptive graph convolutional neural networks, CVPR (2018) 1011–1025.
[32] E. Cski, K. Gonchigdanzan, Almost sure limit theorem for the maximum of stationary gaussian sequences, Stat. Probab. Lett. 58 (2) (2002) 195–203.
[33] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc., 2012, pp. 1097–1105.
[34] H. Toivonen, A. Srinivasan, R.D. King, S. Kramer, C. Helma, Statistical evaluation of the predictive toxicology challenge 2000–2001, Bioinformatics 19 (10) (2003) 1183–1193.
[35] Y. Wang, Y. Tian, N. Deng, Distinguishing enzymes from non-enzymes via support vector machine, Machines in Bioinformatics Masters Thesis (2002) 166–173.
[36] I. Wallach, M. Dzamba, A. Heifets, Atomnet: A deep convolutional neural network for bioactivity prediction in structure-based drug discovery, CoRRabs/1510.02855 (2015). http://arxiv.org/abs/1510.02855
[37] Y. Koren, Factorization meets the neighborhood: a multifaceted collaborative filtering model, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 426–434.
[38] S. Prithviraj, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, T. Eliassi-Rad, Collective classification in network data articles, AI Mag. 29 (3) (2008) 93–106.
[39] K.M. Borgwardt, H.P. Kriegel, Shortest-path kernels on graphs, in: Proceedings of the IEEE International Conference on Data Mining, 2006, pp. 74–81.
[40] F. Orsini, P. Frasconi, L.D. Raedt, Graph invariant kernels, in: Proceedings of the International Conference on Artificial Intelligence, 2015, pp. 3756–3762.
[41] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, T. Hirzel, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, in: Proceedings of the International Conference on Neural Information Processing Systems, 2015, pp. 2224–2232.

Bo Wu is working toward the Ph.D. degree at the State Key Laboratory of Software Development Environment. His research interests include machine learning, computer vision.



Yang Liu received the B.S. degree in computer science from Beihang University, China, in 2009, where he is working toward the Ph.D. degree at the State Key Laboratory of Software Development Environment. His research interests include image recognition and retrieval.

Bo Lang received the Ph.D. degree from Beihang University. She is a professor in School of Computer Science and Engineering. Her research interests include information security, data management, and information retrieval. She is a member of the IEEE.

Lei Huang received the B.S. degree in computer science from Beihang University, Beijing, in 2010. He is currently a PhD student at Beihang University. His research interests include machine learning, computer vision and multimedia annotation. He is a student member of the IEEE.