Network Science **9** (S1): S23–S60, 2021
doi:10.1017/nws.2020.35

CAMBRIDGE UNIVERSITY PRESS

ORIGINAL ARTICLE

Learning to count: A deep learning framework for graphlet count estimation

Xutong Liu[1]*, Yu-Zhen Janice Chen[2], John C. S. Lui[1] and Konstantin Avrachenkov[3]

[1]The Chinese University of Hong Kong, Shatin, NT, Hong Kong (e-mail: cslui@cse.cuhk.edu.hk), [2]University of Massachusetts Amherst, MA 01002, USA (e-mail: yuzhenchen@cs.umass.edu) and [3]INRIA Sophia Antipolis, 06902 Valbonne, France (e-mail: k.avrachenkov@inria.fr)
*Corresponding author. Email: liuxt@cse.cuhk.edu.hk

Special Issue Editor: Hocine Cherifi

Abstract

Graphlet counting is a widely explored problem in network analysis and has been successfully applied to a variety of applications in many domains, most notatbly bioinformatics, social science, and infrastructure network studies. Efficiently computing graphlet counts remains challenging due to the combinatorial explosion, where a naive enumeration algorithm needs $O(N^k)$ time for k-node graphlets in a network of size N. Recently, many works introduced carefully designed combinatorial and sampling methods with encouraging results. However, the existing methods ignore the fact that graphlet counts and the graph structural information are correlated. They always consider a graph as a new input and repeat the tedious counting procedure on a regular basis even if it is similar or exactly isomorphic to previously studied graphs. This provides an opportunity to speed up the graphlet count estimation procedure by exploiting this correlation via learning methods. In this paper, we raise a novel graphlet count learning (GCL) problem: given a set of historical graphs with known graphlet counts, how to learn to estimate/predict graphlet count for unseen graphs coming from the same (or similar) underlying distribution. We develop a deep learning framework which contains two *convolutional neural network* models and a series of data *preprocessing techniques* to solve the GCL problem. Extensive experiments are conducted on three types of synthetic random graphs and three types of real-world graphs for all 3-, 4-, and 5-node graphlets to demonstrate the accuracy, efficiency, and generalizability of our framework. Compared with state-of-the-art exact/sampling methods, our framework shows great potential, which can offer up to two orders of magnitude speedup on synthetic graphs and achieve on par speed on real-world graphs with competitive accuracy.

Keywords: Graphlet count estimation; Convolutional neural networks; Deep learning on graph; Network analysis.

1. Introduction

Graphlets are defined as k-node connected induced subgraph patterns. For an undirected graph, 3-node graphlets include open triangle () and closed triangle (). There are six different types of graphlets for $k = 4$ and 21 types of graphlets for $k = 5$, where Figure 3 and Table 1 depict all possible 3-, 4-, and 5-node graphlets. The number of each graphlet, called *graphlet count*, is a signature which characterizes the *local network structure* of a given graph. Global features, such as diameter, degree distribution, in contrast, are less representative, because graphs with similar global features can have totally different local structures. Therefore, graphlet count is widely used and plays a prominent role in network analysis of many fields, such as bioinformatics (Pržulj, 2007), social science (Holland & Leinhardt, 1976), and infrastructure network studies (Akoglu et al., 2015). In bioinformatics domain, graphlet counts were applied for protein classification

© The Author(s), 2020. Published by Cambridge University Press.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

and function prediction (Shervashidze et al., 2009), disease gene identification (Milenković et al., 2010), and biological network comparison (Pržulj, 2007). In social science discipline, researchers utilized graphlet counts for analyzing social network structure (Ugander et al., 2013) and interpersonal relations (Holland & Leinhardt, 1976). In infrastructure network studies, graphlet counts were adopted for anomaly detection (Akoglu et al., 2015) and spam detection (Becchetti et al., 2008).

While graphlet count is valuable for network analysis, computing exact graphlet count is inherently difficult and computationally expensive. The main challenge is the combinatorial explosion, where the count of k-node graphlets in a network grows k-th order polynomially large as the network size grows. For example, a moderate size arXiv collaboration network with 12K nodes can have more than 3.3 million triangles (△) and 0.2 billion 4-paths (⁞). Experiments show that even the state-of-the-art exact counting method (Ahmed et al., 2015) cannot finish within a week calculating 4-node graphlet counts for a Twitter graph with 21.3 million nodes (Chen & Lui, 2018).

To deal with this difficulty, many frameworks and algorithms were proposed to achieve a more practical goal: estimate or approximate graphlet count with acceptable error in exchange for computational efficiency. These methods (Chiba & Nishizeki, 1985; Rahman et al., 2014; Seshadhri et al., 2013; Jha et al., 2015; Chen et al., 2016; Bressan et al., 2017; Jain & Seshadhri, 2017) leverage on sampling techniques to estimate graphlet count with bounded error and significantly outperforms exact counting methods in space and time.

In the real world, many graph instances have some underlying structures or characteristics and can be thought as coming from a specific distribution/domain. For example, social networks are often scale-free networks whose degrees follow power law distributions (Newman et al., 2016); brain networks have been identified as small-world networks (Sporns et al., 2009); subgraphs or snapshots of a given network and streaming graphs can also be viewed as coming from some distribution. Recent work (Rossi & Ahmed, 2019) confirmed this fact and build classifiers to predict (with 95.7% accuracy) the category of a given network, with only simple features including density, average degree, and assortativity.

Since graphlets act as fingerprints of a given graph, graphs structural information can be naturally reflected in their graphlet patterns as well as their graphlet counts. In other words, graph structural information and graphlet counts are correlated. Moreover, this correlation could be further enhanced for graphs from the same (or similar) distribution. For instance, graphs from some biochemistry datasets, such as MUTAG, NCI1, and NCI109, only have tree-like graphlets. Nevertheless, conventional graphlet count frameworks, including both exact counting and sampling methods, do not leverage on this fact and always repeat the onerous counting procedures even if a new graph is similar to or exactly isomorphic to previously studied graphs. They neglect the connection between the structural information and the graphlet count and merely treat them as inputs and outputs separately in the counting procedure. Ideally, this connection can be modeled as a complicated function mapping from graph structures to their graphlet counts. For instance, given a graph's adjacency matrix A and all its eigenvalues which contains important structural information (Butler, 2008), the triangle count can be written as one-sixth of the sum of cubes of the eigenvalues (Tsourakakis, 2008). But note that, for any other graphlets, there is no literature showed the explicit form of the function. Intuitively, if we can learn to approximate this function, graphlet counts can be efficiently estimated by applying the learned function to any given graph. Essentially, we need to solve the *graphlet count learning (GCL) problem*: assume we have a set of networks with known graphlet counts, how can we learn to estimate/predict graphlet counts in new-coming networks from the same (or similar) underlying distribution. Note that we assume the historical graphs and new-coming graphs are coming from the same (or similar) distribution because the function we want to learn could be related to the underlying graph distributions and has totally different properties for different distributions.

Learning paradigm has been applied in other graph problems, for example, graph classification, community detection. Yet, frameworks for learning graphlet count cannot be found in literature. The GCL problem lies in the intersection of graphlet count problem and learning paradigm. To

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

the best of our knowledge, the problem described above has never been studied in both graphlet counting and computational learning literature. We are the *first* to study this problem and our research goal is to develop a learning framework to solve the GCL problem.

Proposed work. To achieve this goal, we adopt a powerful model in deep learning literature as the core component of our learning framework, namely the convolutional neural network (CNN). CNN is a class of deep feedforward artificial neural networks which has become an important tool in addressing a wide variety of machine learning problems, including image recognition (Krizhevsky et al., 2012), video recognition (Karpathy et al., 2014), and natural language processing (Johnson & Zhang, 2015). CNN turns out to be very good at discovering intricate structures and offers attractive features, including significant representation power, great learning capacity, constant testing speed, and efficiency in space. However, since CNN is natively developed for regular, low-dimensional grid-structured data such as texts, images, and videos, applying CNN to graph data for graphlet count estimation is far from trivial. In this work, we first propose a straightforward 2D CNN model which applies vanilla CNN on the adjacency matrices of input graphs. Then, we point out and analyze some drawbacks of the 2D CNN and propose a 1D CNN model. The latter largely improves the learning ability and reduces the number of parameters to be trained by tailoring the receptive fields of filters. Experiments show that 1D CNN reduces the relative error (RE) up to 50% with less training and testing time compared to 2D CNN.

Moreover, we propose four preprocessing techniques to further enhance the performance and generalization ability of the CNN model. *Adjacency matrix zero padding* standardize the input adjacency matrices and make them validate for training. *Swapping augmentation* and *node ordering* leverage on graph isomorphism (GI) to solve overfitting and underfitting problems. *Random node sketching* (RNS) solves the memory restriction problem for large graphs with provable error bound. Extensive experiments on synthetic random graphs and real-world graphs show the effectiveness of these preprocessing techniques.

1.1 Contributions

- **Novel framework.** To the best of our knowledge, we are the first to formulate the GCL problem, which bridges the gap between classic graphlet count problem and the learning paradigm. We are also the first to propose a deep learning-based 2D CNN model and its improved version, 1D CNN model, to solve the GCL problem for all 3-, 4-, and 5-node graphlets. Together with four novel preprocessing techniques, we show our framework's potential to compete with the state-of-the-art methods and is accurate, efficient, and general given different graph topological structures for different k-node graphlets.
- **Extensive experiments.** We conduct extensive experiments on synthetic random graphs, real-world biochemistry networks, arXiv collaboration networks, and Facebook social networks. We also compare our framework with a series of state-of-the-art graphlet counting frameworks to validate the effectiveness of our framework.
- **Accuracy and efficiency.** Our framework can offer up to two orders of magnitude speedup on synthetic graphs and achieve on par speed for real-world graphs with highly competitive accuracy compared to state-of-the-art exact/sampling methods.
- **Generalizability.** Our framework gives accurate estimation given different graph structures or underlying distributions (e.g., sparse/dense graphs, small/large graphs, graphs from different graph models). Moreover, our framework can be easily extended to estimate other k-node graphlet counts (e.g., 6-node graphlets) without changing the learning model.

The rest of the paper is organized as follows. The preliminary for graphlet counting, CNNs, and problem formulation are given in Section 2. The proposed CNN-based GCL framework is described in Section 3. The framework evaluation experiment results are presented in Section 4. Related works are discussed in Section 5. Finally, Section 6 concludes the paper and points out the future direction.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

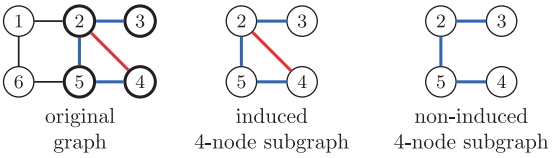


Figure 1. An example of induced/non-induced subgraph.

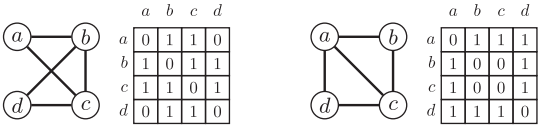
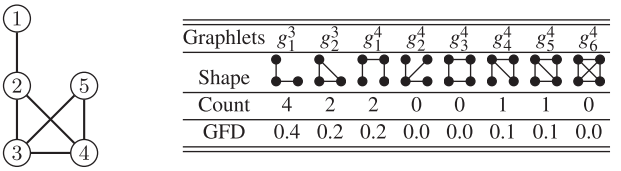


Figure 2. An example of isomorphic graphs.



Graphlets	g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
Shape								
Count	4	2	2	0	0	1	1	0
GFD	0.4	0.2	0.2	0.0	0.0	0.1	0.1	0.0

Figure 3. An example of graphlet counts for All 3-node and 4-node graphlets.

2. Preliminary

2.1 Notations and definitions

Our input network can be modeled as graph $G = (V, E)$, where V is the set of nodes and E is the set of edges of this graph. In this work, we study undirected and unweighted graph that has no self-loop or multiple edges between any two nodes, but we do not require the graph to be connected.

- **Induced subgraph.** A subgraph can be either an *induced subgraph* or a *non-induced subgraph*. A k-node-induced subgraph $G_k = (V_k, E_k)$ has k nodes in V_k and contains all the edges in the original graph whose both endpoints are in V_k. Formally, an induced subgraph in a graph $G = (V, E)$ can be written as $G_k = (V_k, E_k)$, where $V_k \subset V$, $|V_k| = k$, and $E_k = \{(u, v) : \forall\ u, v \in V_k \wedge (u, v) \in E\}$. In contrast, a non-induced subgraph does not require that all edges connecting nodes in V_k are also present in E_k. For example, in Figure 1, when $V_4 = \{2, 3, 4, 5\}$, edge set $\{(2, 3), (2, 5), (4, 5)\}$ form a non-induced subgraph because it does not contain the edge $(2, 4)$, while the same vertex set V_4 with edge set $E_4 = \{(2, 3), (2, 4), (2, 5), (4, 5)\}$ represents an induced subgraph.
- **Isomorphism.** For two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, GI is defined as a bijection mapping $\psi: V_1 \to V_2$ and $(u, v) \in E_1 \Leftrightarrow (\psi(u), \psi(v)) \in E_2$, $\forall u, v \in V_1$. If there exists such a mapping between G_1 and G_2, these two graphs are isomorphic. For instance, the two graphs in Figure 2 are isomorphic to each other, where the mapping is $\psi(a) = d$, $\psi(b) = a$, $\psi(c) = c$, $\psi(d) = b$. The GI problem is the computational problem of determining whether two finite graphs are isomorphic.
- **Graphlet and graphlet count.** A graphlet is a connected induced subgraph in a graph G. We denote a family of k-node graphlets as $\mathcal{G}^k = \{g_1^k, \cdots, g_m^k\}$, $k = 3, 4, 5$. For example, there are two different 3-node graphlets and six different 4-node graphlets. Figure 3 depicts all the 3-node and 4-node graphlets, and Table 1 depicts all 5-node graphlets .

  Graphlet count $c_i^k \in \mathbb{Z}_{\geq 0}$ is defined as the number of induced subgraphs that are isomorphic to graphlet g_i^k in a graph G. Figure 3 also gives instances of graphlet counts. In this example graph, $c_1^4 = 2$ because there are two induced subgraphs isomorphic to 4-path g_1^4 (⁘). One is induced by nodes $\{1, 2, 3, 5\}$ and the other is induced by nodes $\{1, 2, 4, 5\}$.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 1. All 5-node graphlets

Graphlets	g_1^5	g_2^5	g_3^5	g_4^5	g_5^5	g_6^5	g_7^5	g_8^5	g_9^5	g_{10}^5	g_{11}^5
Shape											
Graphlets	g_{12}^5	g_{13}^5	g_{14}^5	g_{15}^5	g_{16}^5	g_{17}^5	g_{18}^5	g_{19}^5	g_{20}^5	g_{21}^5	
Shape											

- **Graphlet frequency distribution (GFD).** Given a set of graphlets $S = \{g_1, \cdots, g_m\}$, graphlet frequency distribution (GFD) can be denoted as a vector $\boldsymbol{f} = [f_1, \cdots, f_m]$, where f_i represents the frequency of the graphlet g_i, and $f_i = \frac{f_i}{\Sigma_{i=1}^m f_i}$. For the example in Figure 3, we calculate the frequency of each graphlet among all 3-node and 4-node graphlets.
- **Relationship between GFD and graphlet count.** If we have the graphlet counts $\{c_1, \cdots, c_m\}$ for graphlets $\{g_1, \cdots, g_m\}$, we can obtain GFD as $f_i = \frac{c_i}{\Sigma_{i=1}^m c_i}$. On the other hand, if we have GFD, we can also derive the graphlet counts with the help of node degree information d_v (Chen & Lui, 2018; Jha et al., 2015), where degree d_v is the number of neighbor nodes of node v. For example, $\Sigma_{v\in V}\binom{d_v}{2} = c_1^3 + 3 \cdot c_2^3$, $\Sigma_{v\in V}\binom{d_v}{3} = c_2^4 + c_4^4 + 2 \cdot c_5^4 + 4 \cdot c_6^4$, $\Sigma_{v\in V}\binom{d_v}{3} = c_2^4 + c_4^4 + 2 \cdot c_5^4 + 4 \cdot c_6^4$ and $\Sigma_{v\in V}\binom{d_v}{4} = c_1^5 + c_4^5 + c_9^5 + c_{11}^5 + 2c_{14}^5 + c_{15}^5 + c_{16}^5 + c_{18}^5 + 2c_{19}^5 + 3c_{20}^5 + 5c_{21}^5$. For every family of k-node graphlets $\mathcal{G}^k$, we can find a linear relationship as such. As an example, in Figure 3, we have $\Sigma_{v\in V}\binom{d_v}{2} = 10$ and $f_{g_1^3} = 0.4$, $f_{g_2^3} = 0.2$. We can derive the open-triangle ( ) count and the closed triangle ( ) count by letting $c_1^3 = f_{g_1^3} \cdot x$ and $c_2^3 = f_{g_2^3} \cdot x$ and calculating x via the equation $\Sigma_{v\in V}\binom{d_v}{2} = f_{g_1^3} \cdot x + 3 \cdot f_{g_2^3} \cdot x$.
- **Computational complexity.** Exact computation of graphlet counts is inheretly expensive as the graphlet size and/or the graph size grows. First, the number of non-isomorphic k-node graphlets grows exponentially large as k increases. To illustrate, the number of different types of 4- and 5-node graphlets are 6 and 21; for 6-node graphlets, it becomes $|\mathcal{G}^6| = 112$; $|\mathcal{G}^7| = 853$. Second, it is critical to decide an induced subgraph is isomorphic to which k-node graphlet, which can be reduced to GI problem. Unfortunately, this problem is not known to have polynomial solutions (Schöning, 1988). These two facts make graphlet counting highly challenging when graphlet size k is large. However, even k is small (i.e., $k = 3, 4, 5$), a specific k-node graphlet count can be up to $\binom{N}{k}$, which is k-th order polynomial or superlinear with respect to the graph size N. This means, counting graphlets using naive enumeration method is computationally intensive. For example, a moderate size arXiv collaboration network with 12K nodes can have more than 3.3 million triangles and 0.2 billion 4-paths. Straightforward exhaustive enumeration approach has to count each graphlet at least once, which has prohibitively high cost. Experiments show that even the state-of-the-art exact counting method (Ahmed et al., 2015) can not finish within a week to calculate 4-node graphlet counts for a Twitter graph with 21.3 million nodes (Chen & Lui, 2018).

2.2 Convolutional neural network

CNN is a type of feedforward artificial neural network, which was initially developed for modeling biological neural systems (Hubel & Wiesel, 1968). CNN has been widely used on learning regular grid data and has led to breakthroughs in image recognition (Krizhevsky et al., 2012), video recognition (Karpathy et al., 2014), and natural language processing (Johnson & Zhang, 2015) problems.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

In a CNN architecture, multiple neural network layers are stacked in sequence. Typically, a CNN consists of two main types of layers: convolutional layer and fully connected layer (LeCun et al., 2015). In the context of image task, an image of height 32 pixels, width 32 pixels, and three color channels, R, B, G, will be presented as a 3D tensor of shape (32, 32, 3). Taking such a tensor as the input, a convolutional layer computes output values by performing dot products between its learnable weight filters and local regions, or receptive fields, of the input tensor. For example, from left to right, from top to down throughout every location in the input tensor, one weight filter of size (5, 5, 3) can have at most $28 \times 28 \times 1$ different dot products by sliding the weight filter by (1, 1) each time. Each dot product usually follows by a nonlinear activation (e.g., $\mathrm{ReLU}(x) = \max(0, x)$). Hence, in this case, if the first convolutional layer applies 64 weight filters, the output tensor would be of shape (28, 28, 64). And the next neural network layer will take it as an input. In a fully connected layer, each output value is a weighted sum of all elements of the layer's input. Unlike convolutional layer which only focuses on one small local region at a time, fully connected layer connects to the whole input tensor. For instance, suppose the next layer is a fully connected layer with 16 output values. To produce one output value, fully connected layer takes the $28 \times 28 \times 64$ elements as the input, sum them up with $28 \times 28 \times 64$ learnable weights and a bias and applies a nonlinear activation on the sum. This activation layer produces a size (1, 1, 16) output.

In general, a CNN architecture serves as a transformation function which maps input tensors to values or labels, and this function is parameterized by nonlinear activations and learnable weights and biases. Parameters such as weights and biases will be trained in an iterative manner, that is, via backpropagation refinement.

2.3 Problem formulation

In this work, we are solving the following GCL problem. Suppose we are given a particular type of k-node graphlet g_i^k, $k = 3, 4, 5$, and a dataset $\mathbb{G}$ consisting of m undirected unweighted graphs from the same underlying distribution $\mathcal{D}$, that is, $\mathbb{G} = \{G_1, ..., G_m\}$ and $\mathbb{G} \sim \mathcal{D}$. We assume the dataset can be divided into two parts $\mathbb{G}_1 = \{G_1, ..., G_{m_1}\}$ and $\mathbb{G}_2 = \{G_{m_1+1}, ..., G_m\}$, where graphlet counts c_i^k are known in $\mathbb{G}_1$ and unknown in $\mathbb{G}_2$. Our goal is to develop a framework that can learn from $\mathbb{G}_1$ and estimate unknown graphlet count c_i^k for every graph in $\mathbb{G}_2$. Mathematically, for any graphlet g_i^k, we want to learn a function $y_j = f_i^k(G_j|\mathcal{D})$ mapping from the graph structure G_j to the graphlet count y_j for $j \in \{m_1 + 1, ..., m\}$, where the function value $f_i^k(G_j|\mathcal{D})$, $j = 1, ..., m_1$ are given.

Note that we assume our data are coming from the same (or similar) distribution because the function we want to learn could be related to the underlying graph distributions and has totally different properties for different distributions. Conventional graphlet count frameworks can directly estimate or compute exact graphlet counts in $\mathbb{G}_2$, but they fail to utilize the ground truth $\mathbb{G}_1$, which will suffer loss in accuracy or efficiency or even both.

3. Methodology

In this section, we introduce the CNN-based framework to address the GCL problem. This framework consists of two parts: the core CNN architecture and the preprocessing techniques.

In Section 3.1, we propose the basic 2D CNN model and its improved version, 1D CNN model, and discuss the main idea and detailed architecture of both CNN models. In Section 3.2, four preprocessing techniques are designed to (a) standardize the dataset to create valid inputs for CNN models, (b) enhance the learning power and improve the model accuracy, and (c) solve the memory restriction when dealing with large graphs. In Section 3.3, we assemble above models and techniques as a whole and show the pipeline of our framework.

3.1 CNN models

The basic idea of CNN models is to learn a transformation function which maps the input graph to its graphlet counts. This function is parameterized by learnable weights and biases which are

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

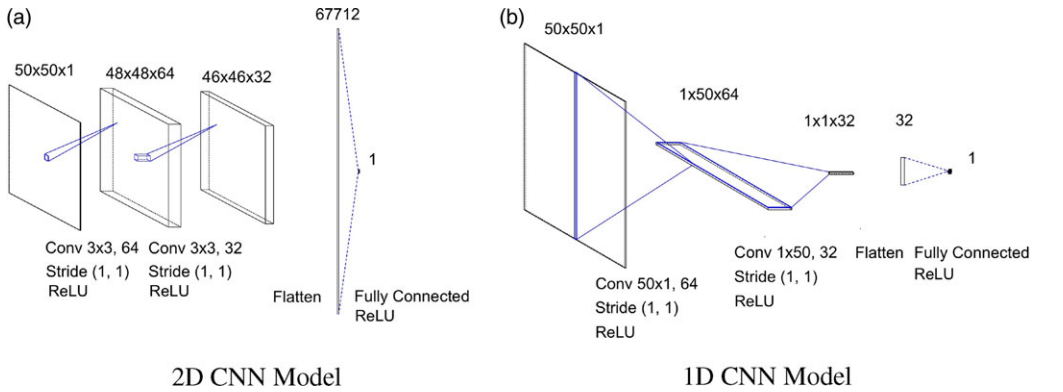


Figure 4. CNN model architectures.

associated with different layers in CNN, from shallow to deep. These parameters carry local structural information in the shallow layers, stack, and refine themselves through deeper layers by some nonlinear activation functions. In the final layer, the CNN model combines results from previous layers and produces an estimated graphlet count.

In our work, both 2D CNN model and 1D CNN model are developed for taking a graph adjacency matrix ($N \times N$ matrix, where N is the number of nodes) and/or its variants as input and giving a graphlet count (non-negative integer number) as output. And the five data preprocessing methods are also for handling graph adjacency matrices. Note that we choose the adjacency matrix as our input data structure because the CNN model heavily relies on matrix(tensor) operations, for example, matrix(tensor) addition and multiplication. Other graph representations, such as edge sequences and adjacency lists, are not meaningful or well defined under these operations. However, one major drawback of the adjacency matrix is that it requires $O(N^2)$ space for a graph with N nodes and exhausts memory resources when N is large. We will talk about how to solve this problem in Section 3.2.

3.1.1 2D CNN model

The main idea of 2D CNN model is to think of the adjacency matrix as an "image" and apply vanilla CNN to it. Intuitively, each entry in the adjacency matrix forms a pixel-like base element in this "image". The graphlets, such as triangles and paths, are analogous to higher-order patterns, for example, textures or pattern-related objects in an image. Once our first few shallow layers of 2D CNN can recognize the graphlet patterns, the deep layers can sweep the previous output of shallow layers and combine them to estimate the graphlet count of the adjacency matrix.

Based on this idea, we design a 3-layer 2D CNN model whose architecture is illustrated in Figure 4(a). For an input of shape $(N, N, 1)$, this 2D CNN model deploys two convolution activation layers, both with filters of shape (3, 3), followed by a fully connected layer. In the first layer, 64 filters are employed in order to capture more variety of feature patterns. At the second layer, the number of filter decreases to 32. Shape (3, 3) filter has been shown more effective in many deep learning researches, and same for our case. We choose the ReLU function (i.e., $\mathrm{ReLU}(x) = \max(0, x)$) for all the activations in this model because the graphlet count estimation is a regression task for which we expect the output to be a real number in $[0, \infty)$.

Let us define some notations for our CNN models. Let $\boldsymbol{O}^{(l)} \in \mathbb{R}^{N^{(l)} \times N^{(l)} \times C^{(l)}}$ be the output tensor at layer l, where $l = 0, 1, 2, 3$, $N^{(l)}$ denotes the width (and height) along each channel and $C^{(l)}$ denotes the channel size. Let $\boldsymbol{O}^{(l)}_{i,j,t}$ be the (i, j)th element along the tth channel. We assign $\boldsymbol{O}^{(0)}$ as the graph adjacency matrix. Mathematically, our 2D CNN structure can be described as follows:

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

$$\boldsymbol{O}_{i,j,t}^{(l)} = \text{ReLU}\left(\boldsymbol{W}_t^{(l)} \cdot \boldsymbol{O}^{(l-1)}[i{:}\, i + H_1^{(l)} - 1,\ j{:}\, j + H_2^{(l)} - 1,\ :] + b_t^{(l)}\right), \tag{1}$$

$$\boldsymbol{O}^{(3)} = \text{ReLU}\left(\text{Flatten}(\boldsymbol{O}^{(2)})^T \cdot \boldsymbol{W}^{(3)} + b^{(3)}\right). \tag{2}$$

Equation (1) corresponds to two convolution layers, $l = 1, 2$. Each layer applies $C^{(l+1)}$ filters over the input feature map $\boldsymbol{O}^{(l-1)}$, and the tth filter is parameterized by a trainable 3D weight tensor $\boldsymbol{W}_t^{(l)} \in \mathbb{R}^{H_1^{(l)} \times H_2^{(l)} \times C^{(l)}}$, where $H_1^{(l)}$, $H_2^{(l)}$ denotes the width and height of the filter. $[a{:}\, b, c{:}\, d, :]$ is a slicing function which extracts subset of elements indexing from a to b (include both ends) in width, c to d (include both ends) in height and all in channel to form a new tensor. The operator $\cdot$ is the sum of element-wise product of two tensors. After adding bias term $b_t^{(l)}$, we apply ReLU ($\max(0, x)$) as the activation function to obtain the output feature map $\boldsymbol{O}^{(l)}$. Equation (2) is associated with the fully connected layer. It flattens the output $\boldsymbol{O}^{(2)}$ into a column vector, applies $\boldsymbol{W}^{(3)}$, $b^{(3)}$ and ReLU to obtain the estimated graphlet count. Our 2D CNN is trained with backpropagation and mean squared error as the loss function. Finally, we choose the shape of filters as $(H_1^{(1)}, H_2^{(1)}) = (3, 3)$ and $(H_1^{(2)}, H_2^{(2)}) = (3, 3)$. The channel size is set to be $C^{(0)} = 1$, $C^{(1)} = 64$, $C^{(2)} = 32$, $C^{(3)} = 1$.

3.1.2 1D CNN model

The straightforward 2D CNN offers an efficient architecture to extract meaningful statistical patterns. In this section, we adapt the structure of the 2D CNN and propose a 1D CNN model, which is inspired by Johnson and Zhang's work (Johnson & Zhang, 2015) of applying the CNN for 1D text data for region embedding. The 1D CNN improves the efficiency by reducing the number of trainable parameters and enhance the learning performance by feeding the *complete* 1-hop neighborhood connection information into the first-layer filters.

Concretely, we consider the input graph with N nodes and model its adjacency matrix as a tensor of shape $(N, N, 1)$. Our 1D CNN model also deploys two convolution activation layers and one fully connected layer and adopts the ReLU activation function. The difference is the shapes of the filters. We apply 64 $(N, 1)$-shaped filters instead of $(3, 3)$-shaped filters at the first convolutional layer. Also, 32 $(1, N)$-shaped filters is used at the second layer rather than another 32 $(3, 3)$-shaped filters. The Figure 4(b) depicts this structure when $N = 50$.

Under such a paradigm, the first convolution layer cover all 1-hop neighbors for each node, and the function of this neural network layer is analogous to computing an embedding for each node. At the second layer, the $(1, N)$-shaped filters learn the higher level latent features based on the first-level embedding of the nodes. Mathematically, our 1D CNN structure can be described using Equations (3), (4), and (5). It differs from the 2D CNN model by setting the shape of filter $(H_1^{(1)}, H_2^{(1)}) = (N, 1)$ and $(H_1^{(2)}, H_2^{(2)}) = (1, N)$ in Equations (1) and (2).

$$\boldsymbol{O}_{i,j,t}^{(1)} = \text{ReLU}\left(\boldsymbol{W}_t^{(1)} \cdot \boldsymbol{O}^{(0)}[0{:}\, N-1,\ j{:}\, j,\ :] + b_t^{(1)}\right),\ \ i = 0, j = 0, ..., N-1, t = 0, ..., 63, \tag{3}$$

$$\boldsymbol{O}_{i,j,t}^{(2)} = \text{ReLU}\left(\boldsymbol{W}_t^{(l)} \cdot \boldsymbol{O}^{(l-1)}[0{:}\, 0,\ 0{:}\, N-1,\ :] + b_t^{(l)}\right),\ \ i = 0, j = 0, t = 0, ..., 31 \tag{4}$$

$$\boldsymbol{O}^{(3)} = \text{ReLU}\left(\text{Flatten}\left(\boldsymbol{O}^{(2)}\right)^T \cdot \boldsymbol{W}^{(3)} + b^{(3)}\right). \tag{5}$$

Compared with the 2D CNN model which only applies small sized (i.e., (3, 3)-shaped) filters, the 1D CNN allows the filters to take in more local connection information. Moreover, the total number of trainable parameters is only $O(N)$, whereas 2D CNN has $O(N^2)$ in the last fully connected layer.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

3.2 Preprocessing techniques

Given the core CNN architecture, we further develop five preprocessing techniques to improve the performance and generalization ability of our framework. In Section 3.2.1, we propose adjacency matrix zero padding technique to standardize the size of input adjacency matrices and make them valid for training. In Sections 3.2.2 and 3.2.3, we utilize GI property to enhance the learning ability of CNN model by node ordering and swapping augmentation. In Section 3.2.4, we develop the RNS to solve the memory restriction when dealing with large graphs and provide a theoretical guarantee for the accuracy of this approach.

3.2.1 Adjacency matrix zero padding

Due to the constraint of the fixed fully connected units in CNN model, the size of input data has to be the same. However, in many real-world graph datasets, graph samples not necessarily have the same size (same number of nodes). In the context of learning images, common solutions are to crop images or pad the images with a solid color so that they will have the same sizes. In our situation, cropping is not appropriate as it causes the loss of edge connectivity information that is essential for graphlet counting.

To preserve edge connectivity information of all training graphs, we consider the largest graph in the training set and use its dimension (say N) as the dimension of the input adjacency matrix ($N \times N$). For other graphs in the training set, we take each adjacency matrix and pad it with zero till we have an input matrix of dimension $N \times N$. This solves the varying input size problem. Moreover, the physical meaning of padding adjacency matrix with zero is analogous to adding isolated nodes to a graph, which do not form any extra graphlet.

3.2.2 Node ordering

In many cases, the nodes in a graph are not precisely labeled and strictly ordered; hence, an adjacency matrix is actually not a unique representation of a given graph. The very same graph can be represented by adjacency matrices that, on the face of it, look totally different. This is unfavorable for learning as it introduces more uncertainty/variance to the distribution of the dataset. If a CNN model is not powerful enough, this property makes it harder for neural network to learn the graphlet count estimation.

The node ordering technique works as follows. First, we label each node $v \in V$ in the input graph $G(V, E)$ by a specific labeling strategy and denoted the label as $l(v)$. In this work, we propose four different labeling strategies: (1) degree centrality: $l(v) = \deg(v)$, where $\deg(v)$ is the degree of v (2) closeness centrality: $l(v) = 1/\sum_{u \in V} d(u, v)$, where $d(u, v)$ is the distance between node u and v. (3) betweenness centrality: $l(v) = \sum_{s \neq v \neq t \in V} \sigma_s t(v)/\sigma_s t$, where σ_{st} is total number of shortest paths from node s to t and $\sigma_{st}(v)$ is the number of those paths that pass through v. (4) Weisfeiler–Lehman algorithm: $l(v) = \mathrm{WL}(v)$, where $\mathrm{WL}(v)$ is the label given by Weisfeiler–Lehman algorithm (Zhang & Chen, 2017). Then, we reorder the nodes according to the label value $l(v)$ in decreasing (increasing order) and settle down its adjacency matrix representation accordingly. This transformation is applied to all input adjacency matrices in the dataset. By node ordering, we obtain more consistent representations (canonical representations) of graphs to reduce the variance of the dataset.

3.2.3 Swapping augmentation

As we mentioned in previous subsection, an unattributed graph which has no node labels can have many different representatives in adjacency matrices (as isomorphic graphs). This may be harmful if a CNN model is not powerful enough to fit the data. However, when the dataset contains too few samples or the model tends to overfit the data, we can utilize this isomorphism property to acquire sufficient data for training.

The approach is straightforward: we randomly pick two indices i and j and then swap the ith row with jth row j and swap the ith column with jth column. We can repeat the swapping operation for each graph m times to create m more training data. Analogous to flipping or rotation

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

of images, via augmenting the dataset, we can improve the generalization ability of CNN models and, thereby, improve the accuracy.

3.2.4 Random node sketching

Real-world graphs can be extremely large. For instance, social networks like Sina Weibo can produce up to 21 million nodes, with 261 million edges dataset (Rossi et al., 2018). In practice, our Graphics Processing Unit (GPU) and Central Processing Unit resources are limited. Faced with the memory constraint, it is inefficient and impractical to build a CNN model which takes in the whole adjacency matrix as the input. Consequently, we save all our adjacency matrices using the sparse matrix format (i.e., COOrdinate format[1]) and unpack them when necessary. Moreover, we propose the RNS approach to further improve our scalability. Intuitively, we are training our model on randomly sampled subgraphs (i.e., submatrices) instead of the whole (sparse) adjacency matrix. Note that there are other sketching/sampling methods, such as *Random Edge Sketching* (RES) which samples a subgraph by preserving each edge uniformly at random. Though RES may give more accurate estimation (especially for some of sparse 3-node and 4-node graphlets, e.g., g_1^3, g_1^4, g_2^4), we choose RNS over RES for the following reasons. First, RNS is more space efficient given the same node/edge sampling probability p, that is, the expected edges preserved is p^2 for RNS and p for RES. Second, RNS requires less trainable parameters, because our 1D CNN model has $O(N)$ trainable parameters for a network of size N, and RNS reduces the number of nodes to pN while RES preserves almost all the nodes. Take the social network socfb-FSU53 (Rossi & Ahmed, 2015) as an example, when RES samples 2% edges, 75% nodes are preserved, and RNS only preserves 2% nodes. Third, as to be explained below, for any graphlet, the count in the sampled RNS graphs has one-to-one correspondence from graphlet counts in RNS samples to the original large graphs. However, RES needs all other graphlet counts (e.g., graphlet counts for all six types of 4-node graphlets) in RES samples to infer the graphlet counts for the original graph (Wang et al., 2016), so the accuracy may decrease due to the inaccurate estimation of other graphlet counts.

The basic idea of RNS is to randomly generate a set of small subgraphs which can easily satisfy the memory constraint for each large graph. After that, we apply our CNN model to these subgraphs, combine the results together, and estimate the graphlet count of the original large graph. We also provide a theoretical guarantee that this process will not amplify the RE.

To be concrete, each RNS graph is generated as follows. For each large graph $G(V, E)$ in our dataset, we choose a node preserving probability p and determine m as the total number of RNS graphs to be generated for G. Then, we select a set of $p \cdot |V|$ nodes from G uniformly at random and denote them as V'. One RNS graph $G'(V', E')$ is exactly the subgraph induced from G using this node set V'. Finally, We repeat this process m times to generate m RNS graphs $\mathbb{G}' = \{G'_1, G'_2, ..., G'_m\}$.

After generating the RNS set $\mathbb{G}'$, we establish an equation between the expected graphlet count on G' and the truth count on G, with one-to-one correspondence. Let c and c' denote the graphlet count of a specific k-node graphlet g for the large graph G and any RNS graph G', respectively. Let the function $\mathbb{I}(x)$ be the indicator function and $g(j)$ be the jth k-node graphlet instance of large graph G, where $j \in \{1, ..., c\}$. We have

$$\begin{aligned} E[c'] &= E\left[\sum_{j=1}^{c} \mathbb{I}(\text{Every node of } g(j) \text{ is preserved})\right] \\ &= \sum_{j=1}^{c} E[\mathbb{I}(\text{Every node of } g(j) \text{ is preserved})] \\ &= \sum_{j=1}^{c} Pr(\text{Every node of } g(j) \text{ is preserved}) \\ &= p^k c \end{aligned}$$

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

We introduce a RNS-based estimator of c:

$$\hat{c} = p^{-k} \sum_{i=1}^{m} \tilde{c}(i)/m, \tag{6}$$

where $\tilde{c}(i)$ is the estimated graphlet count by applying the CNN model on the ith RNS graph G'_i. We define the RE for our CNN model on each RNS graph G'_i to be $e(i) = |\tilde{c}(i) - c(i)|/c(i)$ and the RE for the RNS estimator of large graph G as $e = |\hat{c} - c|/c$. Note that the RNS-based estimator $\hat{c}$ is not unbiased because we cannot guarantee our CNN model is unbiased. We have the following theorem to bound the RE with $(1 - \delta)$ probability:

Theorem 3.1. *Given any graph G, its any k-node graphlet count c, its RNS set $\mathbb{G}' = \{G'_1, ..., G'_m\}$ with node preserving probability p and any parameters $\alpha, \delta > 0$, if the RE of CNN is at most ε, that is, $e(i) \leq \varepsilon$ for each $i = \{1, ..., m\}$, then the RE on G is at most $\varepsilon + \alpha(1 + \varepsilon)$ with $(1 - \delta)$ probability, that is, $e \leq \varepsilon$, when $m \geq \frac{1}{2}\alpha^{-2}p^{-2k} \log \frac{2}{\delta}$.*

Proof of Theorem 3.1. We first use the Hoeffding's inequality (Hoeffding, 1963) to give a high-probability bound of the RE $|c(i) - \mathbb{E}[c(i)]|/\mathbb{E}[c(i)]$, given $m \geq \frac{1}{2}\alpha^{-2}p^{-2k} \log \frac{2}{\delta}$. Because $c(i)$ are independent RNS samples and bounded by $[0, c]$, we define $\bar{c} = \sum_{i=1}^{m} c(i)/m$ and use the Hoeffding's inequality,

$$\begin{aligned} Pr(|\bar{c} - \mathbb{E}[\bar{c}]| \leq \alpha\mathbb{E}[\bar{c}]) &\geq 1 - 2\exp\left(-\frac{2m^2\alpha^2\mathbb{E}[\bar{c}]^2}{mc^2}\right) \\ &= 1 - 2\exp\left(-\frac{2m^2\alpha^2p^{2k}c^2}{mc^2}\right) \\ &\geq 1 - \delta \end{aligned} \tag{7}$$

The RE on the large graph G is then bounded with probability of at least $(1 - \delta)$:

$$\begin{aligned} e &= \left|\hat{c} - c\right|/c \\ &= \left|p^{-k} \sum_{i=1}^{m} \tilde{c}(i)/m - c\right| \Big/ c \\ &\leq \left|p^{-k} \sum_{i=1}^{m} (1 \pm \varepsilon)c(i)/m - c\right| \Big/ c \\ &\leq \left|p^{-k}p^k c(1 \pm \varepsilon)(1 \pm \alpha) - c\right|/c \\ &\leq \varepsilon + (1 + \varepsilon)\alpha \end{aligned} \tag{8}$$

The first and second equalities hold by definition. The first inequality holds because the RE of CNN for each instance i is at most ε. The second inequality holds because of the inequality (7) and the fact $\mathbb{E}[\bar{c}] = p^k c$. □

Theorem 3.1 can be viewed as the sensitivity analysis of RNS method. In the high-probability error bound (inequality (8)), the first part of the error term ε is brought by the inaccurate estimation of the CNN model and the second part $(1 + \varepsilon)\alpha$ is brought by inadequate RNS samples. When we have sufficient number of RNS graphs, that is, $m \to \infty$, the RE $e = \varepsilon$, which is no larger than the RE of CNN. This shows that RNS method is effective in leveraging CNN to estimate graphlet counts on large graphs without amplifying the RE. Note that this theoretical high-probability bound is conservative, whereas fewer RNS samples (or smaller preserving probability p) are needed to achieve accurate estimation in practice.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

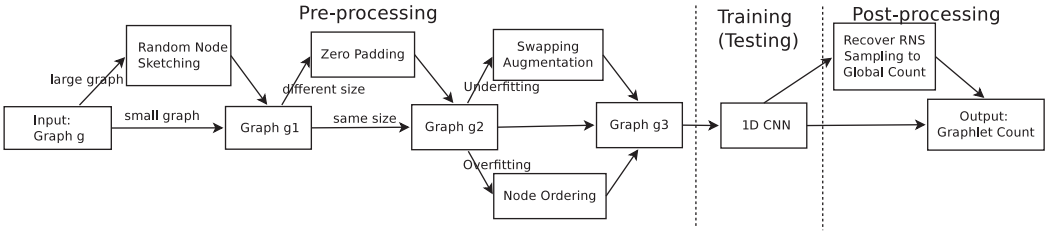


Figure 5. Framework architecture diagram.

3.3 Overall CNN-based framework

Figure 5 presents the overall architecture of the proposed framework. Given a set of graphs with known graphlet count, some preprocessing techniques are applied before inputing these graphs into CNN for training. First, we determine if the size of graphs can fit into the memory, if not, we use RNS to obtain some sample graphs. Second, we apply *zero padding* if the sizes of graphs in the dataset are not consistent. Third, we use *swapping augmentation* to enlarge the training dataset if the graph samples are insufficient. On the contrary, if the data are adequate, we use *node ordering* to enhance the accuracy of count estimation Finally, we use the preprocessed data to train a CNN model which learns a mapping from graph structures to graphlet counts. For large graphs, we recover the global graphlet count from the graphlet count estimations for RNS samples in the end.

4. Experiments

In this section, we present extensive experiments to evaluate the performance of our framework. We aim to answer the following questions:

- **Q1 accuracy:** Is the graphlet count estimation of our framework accurate? Is 1D CNN always more accurate than 2D CNN?
- **Q2 efficiency:** Is our framework faster than other existing methods?
- **Q3 generality:** Does our framework give accurate estimation on any graph structures (e.g., sparse vs. dense graphs, graphs of different random graph models)?
- **Q4 scalability:** Can our framework handle large graph?
- **Q5 practicality:** Is our framework effective on real-world networks?

4.1 Experimental setup

4.1.1 Performance evaluation metrics

We consider the following statistics and metrics, which provide a comprehensive picture of the graph dataset property, graphlet count estimation errors, and running speed, for evaluating the performance of the proposed framework. All results are averaged over multiple runs (i.e., 10 runs) to reduce the random effects in the experiments.

- **Mean and standard deviation (STD) of the ground truth.** Given a graph dataset with S graph samples, let c_i be the ground truth graphlet count of sample graph i, $i = 1, \ldots, S$. We calculate the mean of ground truth counts as $\mu = \Sigma_{i=1}^{S} c_i/S$ and the standard deviation (STD) of ground truth counts as $\delta = \Sigma_{i=1}^{S}(c_i - \mu)^2/S$. These two statistics provide information about the underlying distribution of the dataset. The former shows the average count of each graphlet. The later measures the heterogeneity/diversity of a dataset by showing the derivation from the average graphlet count.
- **Mean of absolute error (MAE).** Given a graph dataset with S graph samples, let c_i be the ground truth graphlet count and c_i' be the estimated graphlet count of sample graph i,

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

$i = 1, \ldots, S$. We compute the mean of absolute error as $\text{MAE} = \Sigma_{i=1}^{S} |c_i' - c_i|/S$. This metric measures how accurate our estimation is.

- **RE.** Although MAE measures the accuracy of estimation, it does not show us if such error is acceptable for the given dataset. For example, MAE = 10 may be a good result if the mean of ground truth over the dataset is 1,000; on the contrary, if the mean of ground truth is 20, this error is less acceptable. In order to measure the effectiveness of the estimation, we compute the ratio between MAE and mean of ground truth. We take RE as $e = \text{MAE}/\mu$.
- **Running time.** In order to evaluate the efficiency of our methods, we choose the running time as the evaluation metric. For our CNN framework, we provide both training time and testing time. The training time records the total running time to load the dataset into the memory, preprocess the dataset (e.g., node ordering, RNS sampling), and train the model parameters. The testing time records the total running time to predict the testing graphs in the dataset. For baseline methods, we provide the total running time to count/estimate testing graphs after they have been preprocessed and loaded into the memory. While training time is an important criteria to evaluate the efficiency, we care more about the testing time and use testing time to compete with other methods. The reason is that our CNN method has the once-for-all property. Concretely, once we have trained our model, we do not need to retrain it for all the new-coming data from the same distribution $\mathcal{D}$. Since the training time can be amortized over each testing graph, it can be neglected when we have considerable number of testing graphs.

  Moreover, we separately report the testing/testing time for each graphlet, because the user may want to separately estimate the count for a specific graphlet. Notice that our CNN models are fully decoupled for different graphlets and can be highly paralleled. Therefore, the time for each graphlet can also represent the total time for counting all graphlets. As for competing methods, they depend on counts of other graphlets to count a specific graphlet, and we use total time for each graphlet unless they provide separate modules to count a specific graphlet.

4.1.2 Dataset

We synthesize datasets with three random graph models: Barabási-Albert (BA) graph, Erdős-Rényi (ER) graph, and random geometric graph (RGG). The size of synthetic graphs in experiments, if not specified otherwise, is 50-node. In each dataset, we have 3,000 training graphs, 300 validation graphs, and 300 testing graphs. The ground truth of graphlet counts is calculated by Parallel Parameterized Graphlet Decomposition (PGD) library (Ahmed et al., 2017).

- **BA graph.** BA model generates random graphs with the preferential attachment mechanism. Specifically, starting from the base connected graph, every newly added node connects to m existing nodes, and the probability of connecting to a particular existing node is proportional to its degree. Hence, the degrees in a BA random graph are exponentially distributed.
- **ER graph.** In ER random graph, the edge between every two nodes exists with probability q. The degree distribution in ER random graph is mostly evenly distributed.
- **RGG.** A RGG is constructed by placing nodes uniformly at random in a unit cube and connecting two nodes by an edge if and only if their distance is within a given radius r. Different from BA graph or ER graph, RGG has more clear community properties in its structure.

We also evaluate our proposed framework on three categories of real-world networks: biochemistry graph datasets, collaboration networks, and social networks.

- **Biochemistry datasets.** We conduct experiments on three biochemistry datasets: the MUTAG dataset (Vishwanathan et al., 2010) which contains the graphs of mutagenic aromatic and heteroaromatic compounds, and the NCI1 and NCI109 datasets (Wale et al., 2008)

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 2. Biochemistry dataset details

Dataset	MUTAG	NCI1	NCI109
Maximum # of nodes	28	111	111
Average # of nodes	17.93	29.87	29.86
Number of graphs	188	4,110	4,127
# of training set graphs	160	3,800	3,800
# of validation set graphs	8	100	100
# of testing set graphs	20	210	227

Table 3. Collaboration and social network dataset details

Graph	ca-hep-ph	ca-hep-th	socfb-MSU24	socfb-FSU53
# of node	28k	23k	32k	28k
# of edge	3.1M	2.4M	1.1M	1.0M
Triangle count	195.7M	191.3M	6.5M	7.9M

which contain the graphs of chemical compounds tested on lung cancer cells and ovarian cancer cells, respectively. We divide each dataset into three parts: one for training, one for validation, and one for testing. Details are listed in Table 2.

- **Collaboration networks.** We evaluate our framework with two collaboration graphs: ca-hep-ph and ca-hep-th (Leskovec et al., 2007). These two graphs are the author networks of arXiv where a node denotes an author and an edge denotes a common publication. ca-hep-ph and ca-hep-th are the networks of the High Energy Physics Phenomenology section and the High Energy Physics Theory section, respectively. Details of these two graphs are listed in Table 3.
- **Social networks.** We also evaluate our framework with two sparse social networks: socfb-MSU24 and socfb-FSU53 (Rossi & Ahmed, 2015). These two graphs are social friendship networks extracted from Facebook consisting of people as nodes and friendship ties as edges. The detailed information can be found in Table 3

4.1.3 CNN training setup

All adjacency matrices are saved using the sparse matrix format to save space and unpacked when necessary. When space are limited, we use sparse matrix multiplication to train (or test) our model, instead of unpacking them during training (or testing). We train our CNN models on training datasets using the mean squared error loss function and the Adam optimizer (Kingma & Ba, 2014). Hyperparameters are tuned with validation datasets: the batch size was selected from {16, 32, 64} examples and the learning rate is selected from {0.0001, 0.0005, 0.001, 0.005, 0.01}. Our models are trained for at most 600 epochs (with early stopping), which usually takes up to 20 minutes (depends on the total number and the size of training graphs) on one 12GB memory GeForce GTX TITAN X GPU. Finally, we test our model on testing datasets using best hyperparameters. Moreover, in all our experiments, we find that different hyperparameters do not affect our results very much and it is always safe to choose the learning rate as 0.001 and batch size as 32. For some rare cases, the 0.001 learning rate is too large and we then use 0.0001 instead to get stable results. For each set of the experiment, we repeat 10 times, present the average of relative errors $e_1, \ldots, e_{10}$ as final result, and use the STD of these 10 REs as the error bar in our figures.

4.2 1D CNN vs. 2D CNN

In Section 3, we propose two CNN models and discuss the physical meaning of 1D and 2D convolutions. In this subsection, we validate our hypothesis by comparing the accuracies of 1D CNN and 2D CNN on estimating 3- and 4-node graphlets in three kinds of random graphs. In the experiments presented in Figure 6, 1D CNN is always more accurate than 2D CNN. Although the

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

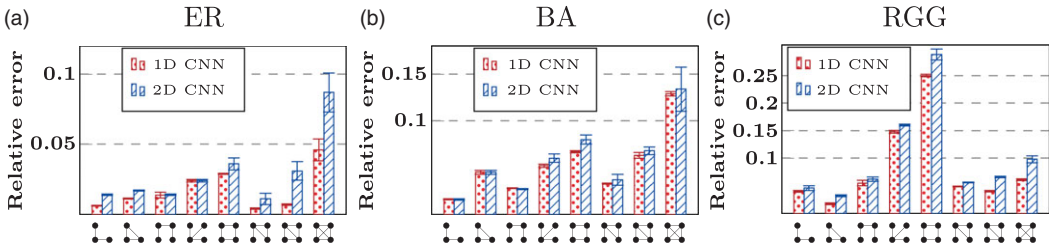


Figure 6. 1D CNN vs. 2D CNN performance.

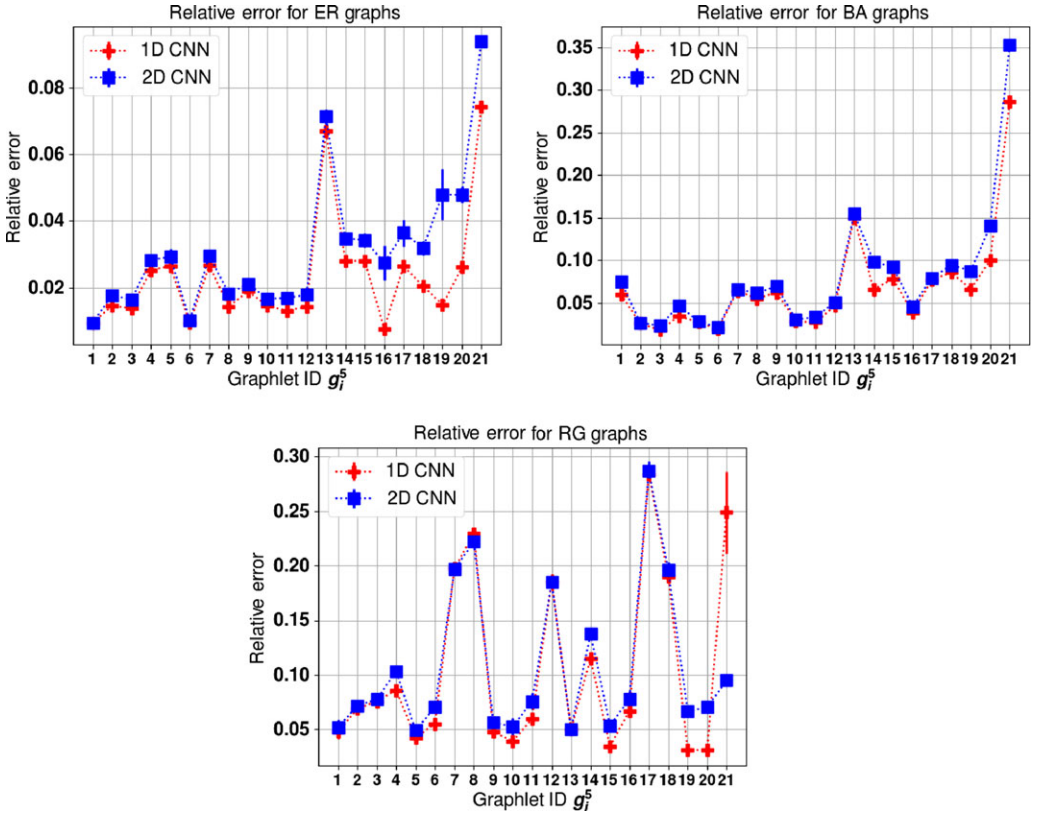


Figure 7. 1D CNN vs. 2D CNN for 5-node graphlets.

accuracy improvements on estimating tree-like graphlets, such as 4-path (g_1^4) and 3-star (g_2^4), are slight, the accuracy improvements on approximating dense graphlets, such as 4-chordal cycle (g_5^4) and 4-clique (g_6^4), go up to 50%. For 5-node graphlets, Figure 7 gives the similar result, which shows that 1D CNN outperforms 2D CNN for all random graphs. Another advantage of 1D CNN is that it reduces the number of weights to be trained compared with the 2D CNN as we mentioned in Section 3.1.

4.3 Comparison with competing methods

We empirically compare the speed of our framework with that of state-of-the-art graphlet count or GFD estimation algorithms. To compare with GFD estimation works, we derive graphlet counts from the estimated GFD using the relationship as introduced in Section 2. In this experiment, we

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

evaluate our model and previous works by estimating all eight types of 3-, 4-, and 5-node graphlets on three synthetic datasets: 50-node ER random graphs with edge existing probability $q = 0.5$, 50-node BA graphs with $m = 6$ and 50-node RGGs with $r = 0.45$. Each dataset contains 3,000 training graphs, 300 validation graphs, and 300 testing graphs. We benchmark with six sampling frameworks: GRAFT (Rahman et al., 2014), CC2 (Bressan et al., 2017), GUISE (Bhuiyan et al., 2012), Localized graphlet estimation (LGE) (Rossi et al., 2018), ApproxG (Mawhirter et al., 2018), and MOSS-5 (Wang et al., 2017). We also provide the running time for exact counting method PGD EXACT (Ahmed et al., 2015) and ESCAPE (Pinar et al., 2017) for reference. Following is a brief synopsis of these works:

- **GRAFT** (Rahman et al., 2014). GRAFT first samples a set of edges uniformly randomly. Then for each sampled edge, it counts the number of occurrence of each graphlet that uses this edge. In the end, the obtained sample graphlet counts are scaled up to approximate the global 3-, 4-, and 5-node graphlet counts. Note that GRAFT is hard-wired to compute statistics for all 3-, 4-, and 5-node graphlets and do not provide separate modules to compute 3- and 4-node graphlets.[2]
- **CC2** (Bressan et al., 2017). Color coding is a general approach that prunes the enumeration search tree by sampling. Yet, general color coding was proposed only for estimating treelet counts. (Alon et al., 1995) This work adopts the general color coding technique and further extends it to estimate the frequency distribution of all kinds of graphlets of up to 7-node.[3]
- **GUISE** (Bhuiyan et al., 2012). GUISE employs a Markov Chain Monte Carlo (MCMC) method to sample 3-, 4-, and 5-node graphlets uniformly and construct the approximate GFD of a graph. Similar to GRAFT, its implementation is hard-wired to compute statistics for all 3-, 4-, and 5-node graphlets.[4]
- **Localized graphlet estimation** (Rossi et al., 2018). The LGE framework derives 3- and 4-node global graphlet count estimation by computing graphlets in a set of sampled localized neighborhoods (egonet). Here, we stick to the settings in Rossi et al. (Rossi et al., 2018), sampling edge neighborhoods and applying the PGD (Ahmed et al., 2015) framework for the edge-centric graphlet counting.[5]
- **ApproxG** (Mawhirter et al., 2018). ApproxG applies edge sampling and parallel computing to reduce the 3- and 4-node graphlet counting runtime.[6]
- **MOSS-5** (Wang et al., 2017). MOSS-5 is a 5-node graphlets counting method based on subgraph sampling. In their implementation, they separately provide MOSS-4 to count 4-node graphlets, which are used for estimating 4-node graphlet counts in our experiments.[7]
- **PGD EXACT** (Ahmed et al., 2017). PGD EXACT represents the PGD framework which counts 3- and 4-node graphlets by leveraging on the combinatorial relationships among these graphlets.
- **ESCAPE**. (Pinar et al., 2017). ESCAPE is the state-of-the-art algorithmic framework that can be used to exactly count any 5-node graphlets. Similar to MOSS-5, ESCAPE provides a separate module for 4-node graphlets, which are used when counting the 4-node graphlets.[8]

Our comparison results are summarized in Table 4 and Figure 8 for 3- and 4-node graphlets, and Table 5 and Figure 9 for 5-node graphlets. For a fair comparison, we tune the number of iterations for all benchmarking sampling methods, so that they obtain as close REs to that of 1D CNN as possible. Take counting 4-clique (g_6^4 ⊠) on the ER graphs as an example. Our 1D CNN makes 2.8% RE. In order to obtain similar accuracy, GRAFT samples 60% of edges, GUISE runs random walk for 400K iterations, and LGE samples 13% of edges (and their neighborhoods) for 450 runs. For MOSS-5 and ApproxG, the accuracy is controllable and we can directly set the error bound. The REs are shown in the Figure 8, where 1D CNN and other competing methods achieve similar REs. For 3–4 node graphlets, Table 4 (a) shows that the running time (for total 300 testing graphs) used by estimation methods, that is, GRAFT, CC2, GUISE, LGE, MOSS-5, and APPROXG, on random

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 4. Running time of 1D CNN and competing methods for 3- and 4-node graphlets

		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
(a) ER graph									
Training (in minutes)	CNN (ours)	1.55	2.73	3.92	3.22	3.47	3.36	4.05	2.96
Testing (in seconds)	CNN (ours)	**0.012**	**0.02**	**0.01**	**0.015**	**0.019**	**0.02**	**0.025**	**0.02**
	GRAFT	3,186	3,036	2,818	2,471	1,763	2,632	1,751	2,106
	CC2	-	-	-	-	-	-	-	-
	GUISE	2,978	2,978	3,349	2,709	3,349	2,632	2,709	4,579
	LGE	8.58	8.58	8.58	8.58	8.58	8.58	8.58	8.58
	ApproxG	8.30	8.30	9.94	9.81	8.99	8.97	9.94	9.02
	MOSS-5	4.98	5.02	5.66	6.30	5.65	5.34	3.25	5.59
	PGD EXACT	10.20	10.20	10.20	10.20	10.20	10.20	10.20	10.20
	ESCAPE	2.82	2.82	2.82	2.82	2.82	2.82	2.82	2.82
(b) BA graph									
		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
Time (in minutes)	CNN (ours)	0.49	0.36	0.60	1.82	0.35	1.40	0.99	0.38
Testing (in seconds)	CNN (ours)	**0.012**	**0.02**	**0.01**	**0.015**	**0.019**	**0.02**	**0.025**	**0.02**
	GRAFT	471	413	513	291	196	401	196	179
	CC2	-	-	-	-	-	-	-	-
	GUISE	1,620	1,620	2,077	58	854	112	221	1,620
	LGE	22.65	24.23	19.76	16.51	26.89	21.46	19.76	22.65
	ApproxG	9.10	9.09	9.20	9.18	9.18	9.20	8.99	8.99
	MOSS-5	6.19	6.19	6.39	6.39	6.21	6.27	6.27	6.39
	PGD EXACT	8.62	8.62	8.62	8.62	8.62	8.62	8.62	8.62
	ESCAPE	2.61	2.61	2.61	2.61	2.61	2.61	2.61	2.61
(c) RGG graph									
		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
Time (in minutes)	CNN (ours)	1.75	2.51	2.56	1.00	1.72	2.56	2.53	2.51
Testing (in seconds)	CNN (ours)	**0.02**	**0.015**	**0.015**	**0.03**	**0.015**	**0.02**	**0.02**	**0.015**
	GRAFT	605	1,052	605	202	138	605	605	400
	CC2	-	-	-	-	-	-	-	-
	GUISE	1,037	1,214	896	339	691	691	529	206
	LGE	7.05	7.01	6.91	22.85	7.46	22.85	12.98	3.22
	ApproxG	22.56	22.56	23.50	22.51	22.56	22.56	23.15	22.75
	MOSS-5	17.79	17.79	19.59	17.44	17.41	19.98	17.22	19.97
	PGD EXACT	9.28	9.28	9.28	9.28	9.28	9.28	9.28	9.28
	ESCAPE	2.91	2.91	2.91	2.91	2.91	2.91	2.91	2.91

Note: a hyphen (-) indicates that the method did not terminate within 6 hours. Bold faces are the best results.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

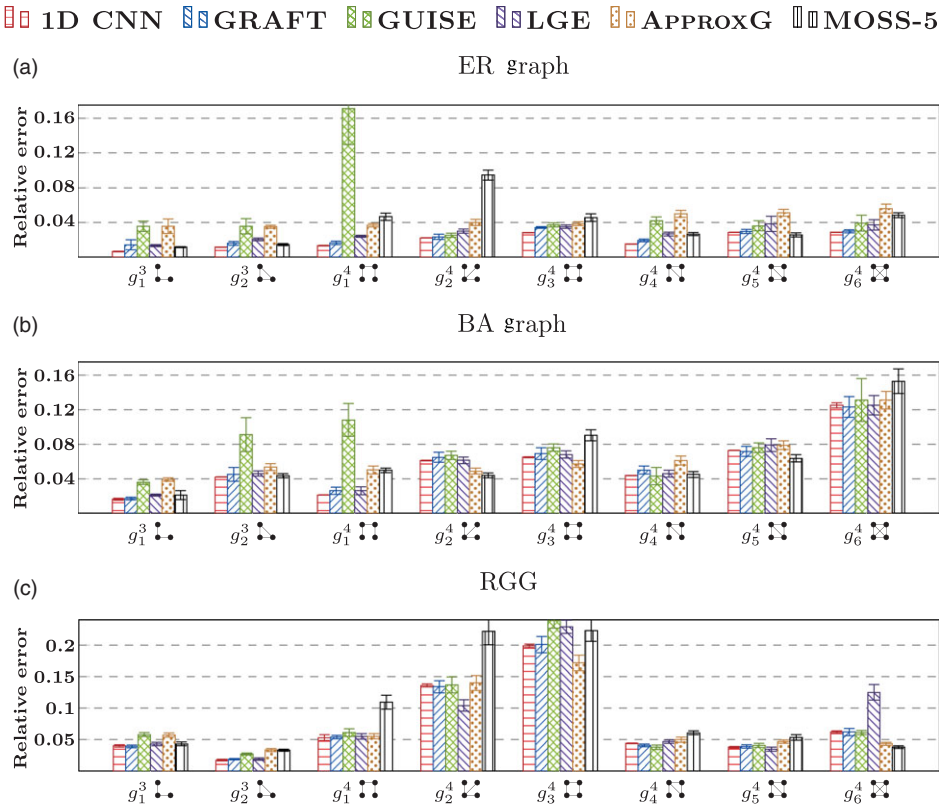


Figure 8. Relative errors of 1D CNN and competing methods for 4-node graphlets.

graphs is significantly more than that used by 1D CNN. The fastest sampling method (MOSS-5) still uses up to two orders of magnitudes longer than ours. Comparing to exact counting methods, 1D CNN also shows better running time and provides at least two orders of magnitudes speedup. The BA graphs and RGG show the similar results, which are presented in Table 4 (b) and (c).

For 5-node graphlets, we do not show the results for CC2 and GUISE because they cannot produce accurate results within reasonable time. The running time of 1D CNN is almost the same as the 4-node graphlets, while competing methods require longer running time. Thus, our 1D CNN provides more speedup for 5-node graphlets. These results demonstrate that our CNN-based graphlet count estimation approach offers remarkable speedup on predicting graphlet counts on synthetic graphs while still maintaining high accuracy.

4.4 Estimation on different graph structures

We evaluate the effectiveness of our model on different graph structures by testing on three different random graphs, that is, BA graph, ER graph, and RGG, for 3- and 4-node graphlets. For studying the performance of our model on sparse graphs and dense graphs, we experiment on different graph densities. To show that our model can tolerate heterogeneity in dataset, we also test on datasets with different ranges of graph densities.

4.4.1 Experiments on different graph densities

We train models for different graph densities: for BA graph, we alter the *edge attachment rate m* from 3 to 7; for ER graph we test on graphs with the *edge existing probability q* from 0.1 to 0.7; for RG graph, we experiment with the *radius r* ranging from 0.25 to 0.5.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 5. Running time of 1D CNN and competing methods for 5-node graphlets

	g_1^5	g_2^5	g_3^5	g_4^5	g_5^5	g_6^5	g_7^5	g_8^5	g_9^5	g_{10}^5	g_{11}^5
(a) ER graph											
CNN train (ours)	4.582	5.725	5.722	5.687	5.716	5.699	5.774	5.546	5.744	5.752	5.715
CNN test (ours)	**0.022**	**0.027**	**0.024**	**0.024**	**0.024**	**0.024**	**0.024**	**0.025**	**0.026**	**0.026**	**0.023**
MOSS-5	965.4	414.2	414.2	216.5	73.7	414.2	73.7	414.2	216.5	73.7	73.7
GRAFT	888.5	2,668.4	2,668.4	2,074.6	1,780.7	2,819.2	2,074.6	2,668.4	2,369.7	2,369.7	2,668.4
ESCAPE	89.34	89.34	89.34	89.34	89.34	89.34	89.34	89.34	89.34	89.34	89.34
	g_{12}^5	g_{13}^5	g_{14}^5	g_{15}^5	g_{16}^5	g_{17}^5	g_{18}^5	g_{19}^5	g_{20}^5	g_{21}^5	
CNN train (ours)	5.749	5.609	4.987	5.601	5.675	5.680	5.374	5.692	5.600	4.504	
CNN test (ours)	**0.027**	**0.026**	**0.025**	**0.023**	**0.025**	**0.023**	**0.026**	**0.022**	**0.026**	**0.024**	
MOSS-5	73.7	16.9	73.7	73.7	73.7	16.9	73.7	73.7	16.9	16.9	
GRAFT	2,369.7	888.5	1,780.7	1,780.7	2,819.2	1,780.7	2,369.7	2,668.4	2,074.6	596.6	
ESCAPE	89.34	89.34	89.34	89.34	89.34	89.34	89.34	89.34	89.34	89.34	
(b) BA graph											
	g_1^5	g_2^5	g_3^5	g_4^5	g_5^5	g_6^5	g_7^5	g_8^5	g_9^5	g_{10}^5	g_{11}^5
CNN train (ours)	4.876	6.219	6.253	5.892	3.248	5.444	5.613	3.907	3.607	4.550	4.568
CNN test (ours)	**0.023**	**0.024**	**0.022**	**0.024**	**0.024**	**0.022**	**0.023**	**0.026**	**0.025**	**0.025**	**0.022**
MOSS-5	14.1	25.5	57.6	25.5	25.5	25.5	0.6	25.5	14.1	14.1	14.1
GRAFT	207.1	308.5	308.5	274.1	274.1	308.5	142.6	175.5	207.1	274.1	274.1
ESCAPE	136.3	136.3	136.3	136.3	136.3	136.3	136.3	136.3	136.3	136.3	136.3
	g_{12}^5	g_{13}^5	g_{14}^5	g_{15}^5	g_{16}^5	g_{17}^5	g_{18}^5	g_{19}^5	g_{20}^5	g_{21}^5	
CNN train (ours)	4.516	3.525	4.004	3.814	3.986	3.402	3.153	3.937	3.349	2.874	
CNN test (ours)	**0.026**	**0.025**	**0.024**	**0.024**	**0.024**	**0.023**	**0.023**	**0.024**	**0.023**	**0.024**	
MOSS-5	5.6	3.0	5.6	3.0	5.6	5.6	5.6	0.6	3.0	5.6	
GRAFT	207.1	75.1	175.5	207.1	274.1	175.5	175.5	240.1	240.1	75.1	
ESCAPE	136.3	136.3	136.3	136.3	136.3	136.3	136.3	136.3	136.3	136.3	
(c) RGG graph											
	g_1^5	g_2^5	g_3^5	g_4^5	g_5^5	g_6^5	g_7^5	g_8^5	g_9^5	g_{10}^5	g_{11}^5
CNN train (ours)	3.127	6.049	6.238	4.665	6.000	6.255	4.034	3.381	6.121	6.211	5.798
CNN test (ours)	**0.026**	**0.027**	**0.024**	**0.026**	**0.026**	**0.025**	**0.028**	**0.025**	**0.025**	**0.025**	**0.024**
MOSS-5	4,858.6	336.4	226.8	336.4	15.1	15.1	336.4	226.8	15.1	7.1	15.1
GRAFT	1,485.7	993.6	500.4	500.4	833.7	668.3	500.4	169.1	833.7	833.7	500.4
ESCAPE	78.31	78.31	78.31	78.31	78.31	78.31	78.31	78.31	78.31	78.31	78.31

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 5. Continued

	g_{12}^5	g_{13}^5	g_{14}^5	g_{15}^5	g_{16}^5	g_{17}^5	g_{18}^5	g_{19}^5	g_{20}^5	g_{21}^5
CNN train (ours)	3.997	1.607	4.267	6.082	6.125	3.160	3.557	6.115	5.742	4.364
CNN test (ours)	**0.026**	**0.028**	**0.025**	**0.026**	**0.026**	**0.026**	**0.027**	**0.024**	**0.026**	**0.030**
MOSS-5	7.1	4,858.6	15.1	7.1	7.1	336.4	15.1	7.1	7.1	7.1
GRAFT	169.1	169.1	336.4	993.6	500.4	500.4	336.4	993.6	1321.6	169.1
ESCAPE	78.31	78.31	78.31	78.31	78.31	78.31	78.31	78.31	78.31	78.31

Note: Bold faces are the best results. Time for training are in minutes, others are in seconds.

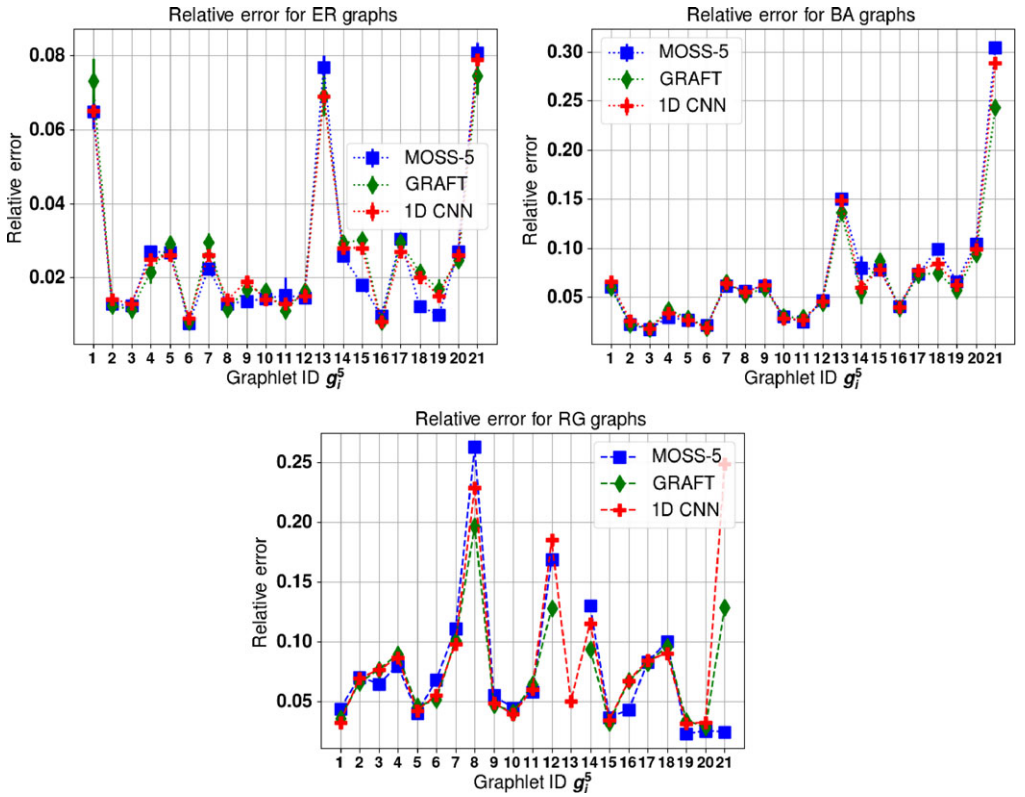


Figure 9. Relative errors of 1D CNN and competing methods for 5-node graphlets.

We record results for all 3- and 4-node graphlets in Table 6. The results are summarized as follows. In Table 6, our model's MAEs of estimating 3- and 4-node graphlet count on three kinds of random graphs all maintain in a small magnitude, while the graph densities increase and the ground truth graphlet counts grow rapidly. Take estimating open-triangle count (g_1^3 ∴) on ER graph as an example. As indicated in Table (a), the mean of truth count grows from 524.15 to 7328.57 as the edge existing probability grows from 0.1 to 0.5. Nevertheless, our MAE rises from 10.31 to only 46.77. Looking at the results for 5-node graphlets in Figure 10, we have similar trends for all random graphs. In our experiments, we find existing graphlet counting methods generally take more time counting denser graphs than sparse graphs of the same size. This experiment not only demonstrates that our framework learns on different graph structures but also shows that our model has advantage on dense graphs (with smaller RE).

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 6. Estimation on graphs of different random graph models with different densities

(a) ER graph									
Configuration		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
0.1	RE(%)	1.9±0.1	17.6±1.0	3.6±0.2	5.2±0.3	11.0±1.2	13.4±0.7	36.3±1.9	100.0±0.0
	MEAN	524.15	19.23	1,990.21	662.95	54.21	218.60	12.09	0.22
0.3	RE(%)	0.8±0.0	2.0±0.1	1.0±0.0	2.6±0.1	3.5±0.4	1.1±0.1	2.4±0.1	10.7±1.1
	MEAN	3,731.11	535.22	25,818.33	8,591.37	2,787.63	11,104.68	2,392.26	170.21
0.5	RE(%)	0.6±0.03	0.7±0.03	0.4±0.01	2.7±0.16	2.7±0.30	0.3±0.02	0.5±0.03	2.8±0.33
	MEAN	7,328.57	2,433.45	43,140.88	14,377.56	10,734.33	42,973.66	21,378.36	3,554.31
(b) BA graph									
Configuration		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
3	RE(%)	1.3±0.1	6.7±0.4	3.0±0.2	4.6±0.2	12.0±0.7	5.3±0.2	9.9±0.6	35.0±3.6
	MEAN	875.33	55.07	3,345.27	2,464.36	123.19	1,022.29	114.44	4.26
5	RE(%)	1.0±0.1	3.1±0.2	1.9±0.1	3.1±0.1	7.2±0.4	2.2±0.1	4.0±0.2	12.0±1.2
	MEAN	1,821.71	200.81	8,884.61	5,300.15	555.42	3,968.82	732.81	58.06
7	RE(%)	0.8±0.0	1.7±0.1	1.4±0.1	2.3±0.1	5.3±0.3	1.2±0.1	1.9±0.1	6.0±0.7
	MEAN	2,841.96	442.12	15,129.29	8,534.06	1,303.88	8,865.07	2,219.75	244.66
(c) RGG graph									
Configuration		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
0.25	RE(%)	8.2±0.4	3.7±0.2	15.6±0.9	22.6±2.3	42.1±4.1	8.8±0.4	5.6±0.2	11.6±1.2
	MEAN	550.49	319.85	1,701.14	186.89	15.09	1,452.20	551.01	326.45
0.35	RE(%)	3.9±0.2	1.9±0.1	6.2±0.3	10.6±1.2	25.8±2.4	3.7±0.2	3.8±0.2	5.3±0.5
	MEAN	1,626.34	1,057.27	7,982.69	925.16	84.86	7,857.52	3,210.44	2,051.50
0.45	RE(%)	2.5±0.1	1.0±0.1	4.0±0.2	7.2±0.9	19.3±2.2	2.1±0.1	3.1±0.2	2.8±0.3
	MEAN	3,291.14	2,549.46	18,538.42	2,446.09	277.45	24,321.43	11,258.74	8,125.60

4.4.2 Experiments on mixed graph configurations

To evaluate the generality of our models, we trained 1D CNNs with mixed graph configuration (density) in the dataset: for BA graph, we adjust the mixing *edge attachment rate m* range from 7 ± 1 to 7 ± 5; for ER graph, we alter the mixing *edge existing probability q* range from 0.4 ± 0 to 0.4 ± 0.2; for RGG, we experiment with the mixing *radius r* range from 0.4 ± 0.05 to 0.4 ± 0.25.

Table 7 records the results for all 3, 4-node graphlets .Our model's MAEs of estimating 3, 4-node graphlet counts on three kinds of random graphs all maintain in a small magnitude, while the graph diversity increase. For example, for estimating triangle count in BA graph, our model's MAE raises from 9.26 to 9.66 only, while the STD of ground truth counts in the testing dataset raises from 119.89 to 452.62. Similar results can be found for 5-node graphlets in Figure 11. This means our framework is accurate and has excellent generalization ability on mixed graph configurations.

4.4.3 Experiments on mixed graph distributions

In Section 4.4.2, we train three separate 1D CNNs for three synthetic datasets with mixed graph configurations. In this section, we study whether training a single 1D CNN with these three

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

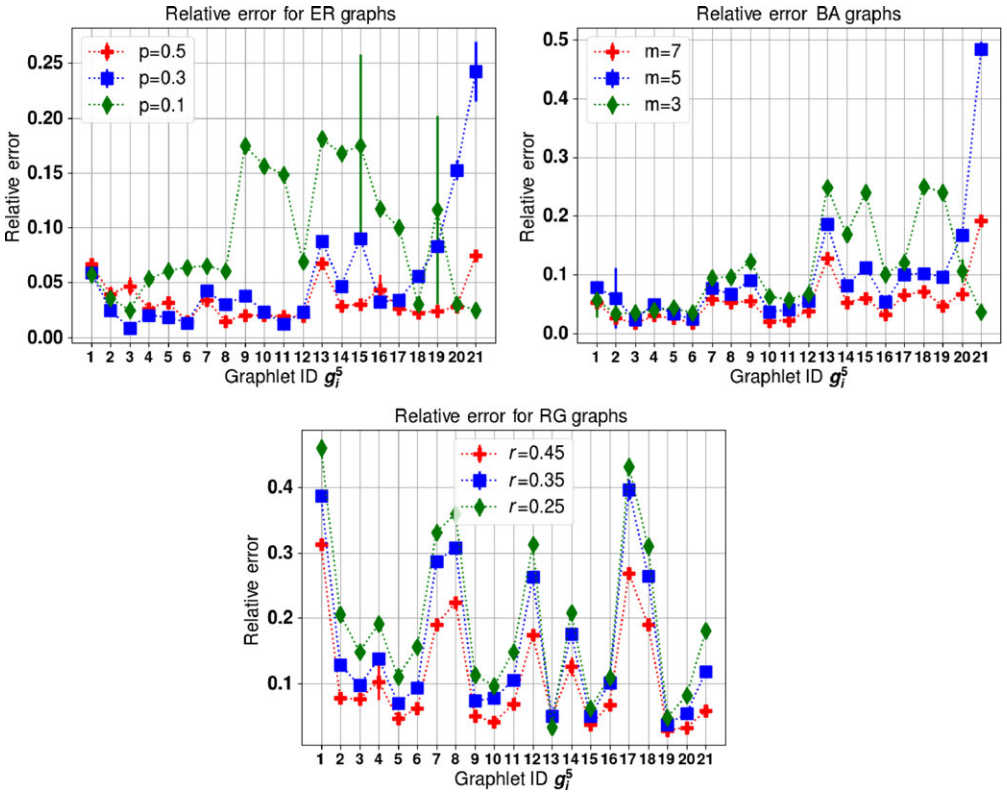


Figure 10. Relative errors for 5-node graphlets with different densities.

datasets mixed together can get accurate predictions for graphs generated from this mixed graph distribution. We randomly generate 3,000 graphs as training data, including 1,000 ER graphs with p ranges from 0.05 to 0.7, 1,000 BA graphs with m ranges from 1 to 13, and 1,000 RGG with r ranges from 0.15 to 0.65. We randomly generate another 450 graphs from the above mixed distribution for testing. As shown in the Table 8, our model can adapt itself to this mixed distribution setting and predict unseen graphs generated from this mixed distribution with fairly acceptable accuracy (all below 10% except for 4-cycle). As for the 5-node graphlets, Figure 12 shows more than half of the REs are below 10% and 18/21 are below 15%. This shows our model has the potential to be applicable to different underlying distributions with a single trained model.

4.5 Estimation on large random graphs

To demonstrate the scalability of our framework and how our CNN with RNS behave on large graphs, we do experiments on 10K-node random graphs. We randomly generate two 10k-node random ER graphs with edge existing probability $q = 0.1$, two 10k-node random BA graphs with edge attachment rate $m = 200$, and two 10k-node random RGG graphs with radius $r = 0.15$. For ER graphs, we apply RNS with node preserving probability of $p = 0.015$ for one graph to generate 3,000 training graphs and 3,000 for the other graph with the same p for testing. For BA graphs, we apply RNS with node preserving probability $p = 0.035$ for one graph to generate 5,000 training graphs and 2,000 testing graphs with the same p for the other graph. For RGG, we apply RNS with $p = 0.015$ to generate 3,000 graphs for training and 10,000 graphs for testing. Figure 13 shows the results of our experiment for 3, 4-node graphlets, where blue lines denote the ground truth graphlet counts (note that there is no blue bars as the RE of ground truth is 0); green lines

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 7. Estimation on datasets with different density ranges

(a) ER graph									
Configuration		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
0.4 ± 0.0	RE(%)	0.7±0.0	1.1±0.0	0.6±0.0	2.2±0.1	2.8±0.2	0.4±0.0	1.0±0.1	4.4±0.3
	MEAN	5,636.8	1,252.9	38,143.9	12,701.1	6,348.3	25,433.5	8,472.6	944.1
	STD	265.70	134.46	1,467.43	573.45	631.39	2,379.13	1,322.36	214.73
0.4 ± 0.1	RE(%)	0.9±0.0	1.1±0.1	3.2±0.1	3.7±0.2	3.0±0.2	0.5±0.0	0.9±0.0	4.4±0.2
	MEAN	5,472.2	1,259.0	36,314.1	12,110.9	6,185.0	24,798.5	8,891.1	1,111.3
	STD	1,050.80	534.27	5,476.89	1,888.30	2,323.41	9,210.34	5,348.80	926.91
0.4 ± 0.2	RE(%)	0.9±0.1	0.8±0.0	10.5±0.6	3.3±0.2	3.0±0.1	1.1±0.1	0.6±0.0	2.6±0.1
	MEAN	5,641.7	1,671.9	33,977.3	11,313.1	7,153.2	28,745.5	14,100.9	2,570.7
	STD	2,026.38	1,225.85	9,717.21	3,227.68	4,411.23	17,768.77	12,955.35	3,052.00
(b) BA graph									
Configuration		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
7+-1	RE(%)	0.1±0.0	2.0±0.1	1.5±0.1	2.9±0.1	5.2±0.2	1.6±0.1	2.6±0.1	7.1±0.4
	MEAN	2,879.2	462.5	15,333.0	8,598.3	1,362.4	9,222.7	2,391.3	277.6
	STD	414.86	119.89	2,615.75	1,532.10	387.68	2,371.10	846.62	129.45
7+-3	RE(%)	1.0±0.1	2.2±0.1	2.2±0.1	2.9±0.2	6.1±0.3	1.5±0.1	3.0±0.1	6.4±0.3
	MEAN	2,708.4	449.7	14,107.8	8,129.6	1,303.2	8,864.6	2,468.4	316.9
	STD	951.73	270.15	5,632.95	3,061.57	819.18	5,243.69	1,940.83	311.72
7+-5	RE(%)	0.8±0.0	1.8±0.1	2.8±0.2	3.0±0.1	4.9±0.3	1.2±0.1	2.2±0.1	6.8±0.4
	MEAN	2,776.9	543.2	14,375.8	8,274.2	1,594.3	10,404.1	3,455.0	522.8
	STD	1,569.41	452.62	8,864.47	4,857.48	1,345.40	8,452.05	3,441.41	603.70
(b) RGG graph									
Configuration		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
0.4 ± 0.05	RE(%)	3.8±0.2	1.4±0.1	5.7±0.3	8.4±0.4	20.8±2.3	3.3±0.1	3.4±0.2	4.0±0.4
	MEAN	2,433.7	1,753.8	13,282.8	1,643.5	173.1	15,323.7	6,758.0	4,668.1
	STD	665.18	592.16	4,119.02	690.95	92.10	6,520.78	3,403.68	2,717.33
0.4 ± 0.15	RE(%)	3.6±0.2	1.3±0.1	5.1±0.3	9.4±0.4	22.3±2.0	2.8±0.2	3.4±0.2	3.1±0.3
	MEAN	2,403.3	1,853.2	12,356.5	1,690.6	196.8	16,797.0	8,076.9	5,982.6
	STD	1,387.38	1,393.33	7,556.02	1,285.88	192.68	14,378.81	8,281.93	6,999.81
0.4 ± 0.25	RE(%)	0.3±0.0	0.9±0.0	3.9±0.2	9.0±0.4	20.6±2.1	2.4±0.1	2.7±0.1	2.2±0.2
	MEAN	2,568.6	2,310.7	12,084.4	1,781.3	284.8	20,793.9	11,645.3	9,695.2
	STD	1987.53	2257.41	9347.08	1593.88	332.00	20840.74	13613.36	12868.71

present the graphlet count estimation for large random graphs by RNS with exact counting on RNS samples and green bars denote the RE incurred by RNS; red lines and bars represent the result by applying RNS with our CNN framework on RNS samples. In Figure 13, our estimations are accurate as they (red lines) almost coincide with the ground truths (blue lines) and the REs of RNS with CNN are close to RNS with exact counting on all random graphs. We also provide the

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 8. Estimation on mixed graph distributions

	g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
Relative error (%)	6.98	2.82	6.47	9.58	12.90	2.87	3.34	7.27
STD of the relative error (%)	0.67	0.25	0.55	0.11	1.72	0.17	0.10	0.08

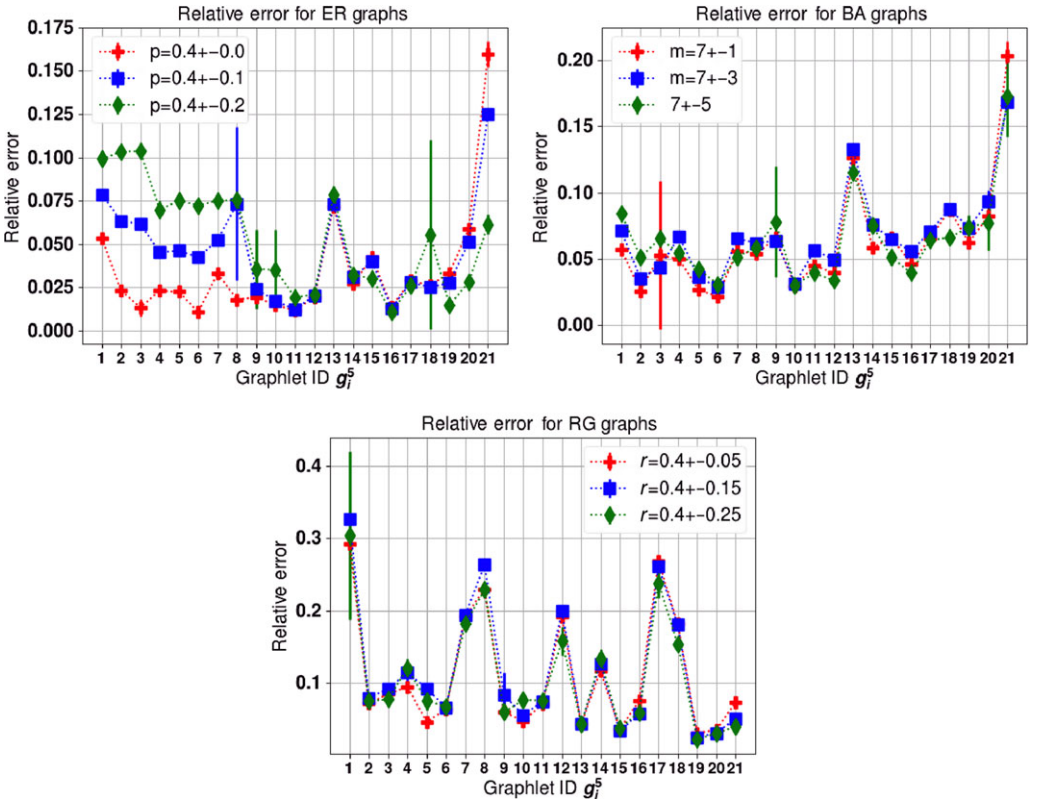


Figure 11. Relative errors for 5-node graphlets with different density ranges.

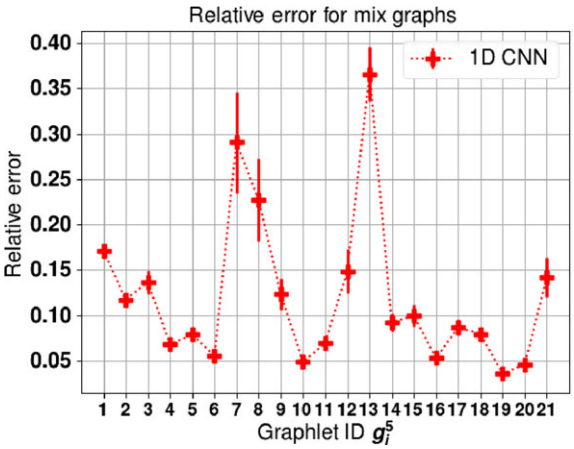


Figure 12. Relative errors for 5-node graphlets for mixed random graphs.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

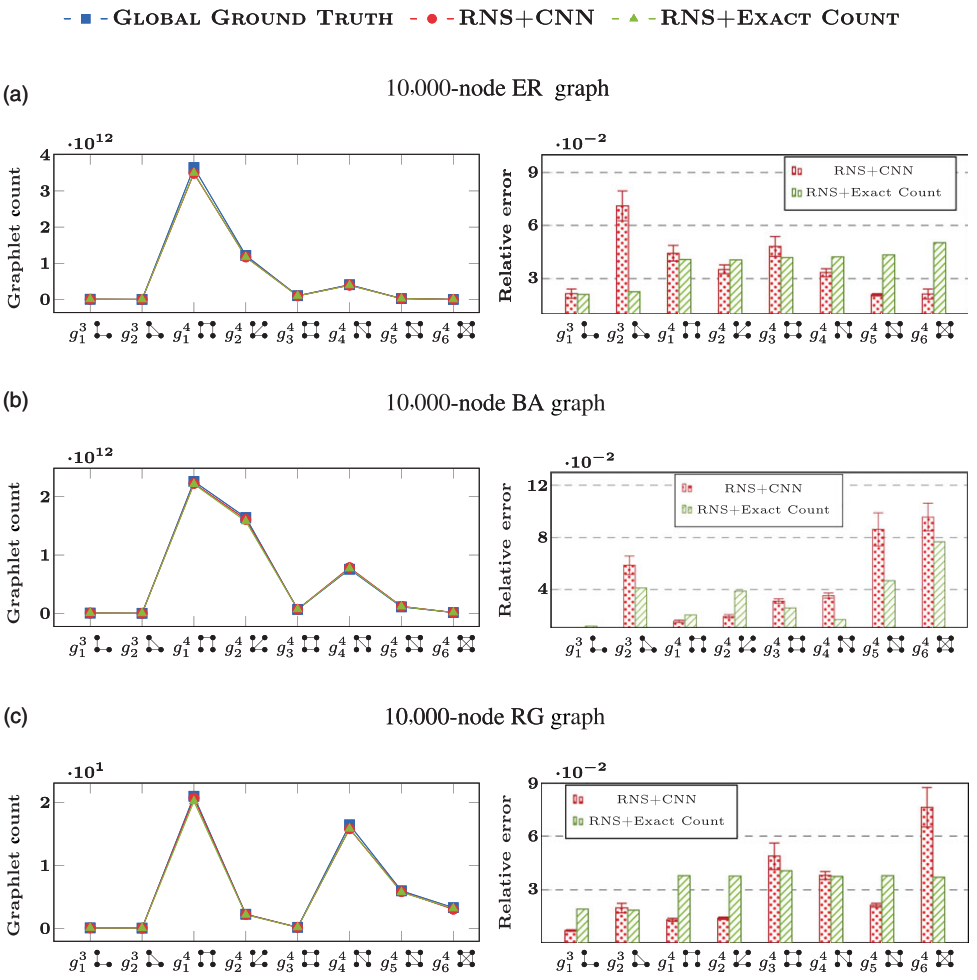


Figure 13. 3-4 node graphlet counts on large random graphs.

training time and testing time for RNS + CNN, the running time for RNS + Exact Count, and the running time for PGD exact to calculate the ground truth of all 10k graphs in Table 9. For ER and BA graphs, we can finish testing in less than 0.62 seconds. For RGG, we spend more time (less than 1.21 seconds) because we sample more RNS samples than ER and BA graphs. We show results for 5-node graphlets on the RE in Figure 14. Again, RNS+CNN shows consistent accuracy (around 6–8% RE) for most 5-node graphlets as 4-node graphlets, except for g_{18}^5-g_{21}^5 for BA graphs. For these 5-node graphlets, the large error suggests we may need more RNS samples to produce more accurate results. For the running time shown in Table 10, our RNS+CNN shows clear advantage over the ESCAPE methods and can finish within 1.2 seconds.

4.6 Preprocessing techniques evaluation

We conduct experiments to analyze the effects of our proposed preprocessing techniques.

4.6.1 Node ordering

We evaluate four kinds of node ordering methods: betweenness centrality, closeness centrality, Weisfeiler–Lehman algorithm (Weisfeiler & Lehman, 1968), and degree centrality on all 3,4-node

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 9. Running time of 3,4-node graphlets for large random graphs

		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
(a) ER graph									
Training (in minutes)	CNN	1.25	1.63	6.35	4.20	1.72	2.48	1.28	0.55
Testing (in seconds)	RNS + CNN	0.285	0.355	0.355	0.349	0.349	0.269	0.294	0.424
	RNS + EXACT	180.2	180.2	180.2	180.2	180.2	180.2	180.2	180.2
	PGD EXACT	456.9	456.9	456.9	456.9	456.9	456.9	456.9	456.9
	ESCAPE	881.01	881.01	881.01	881.01	881.01	881.01	881.01	881.01
(b) BA graph									
		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
Training (in minutes)	CNN	6.93	6.28	12.13	12.03	5.86	32.23	17.45	9.55
Testing (in seconds)	RNS + CNN	0.770	0.759	0.620	0.630	0.770	0.640	0.640	0.629
	RNS + EXACT	180.2	180.2	180.2	180.2	180.2	180.2	180.2	180.2
	PGD EXACT	131.9	131.9	131.9	131.9	131.9	131.9	131.9	131.9
	ESCAPE	310.89	310.89	310.89	310.89	310.89	310.89	310.89	310.89
(b) RGG									
		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
Training (in minutes)	CNN	1.50	1.26	1.80	1.03	0.78	1.73	1.85	1.80
Testing (in seconds)	RNS + CNN	0.934	0.890	1.170	1.165	0.894	1.164	1.195	1.210
	RNS + EXACT	259.4	259.4	259.4	259.4	259.4	259.4	259.4	259.4
	PGD EXACT	228.7	228.7	228.7	228.7	228.7	228.7	228.7	228.7
	ESCAPE	2,372.20	2,372.20	2,372.20	2,372.20	2,372.20	2,372.20	2,372.20	2,372.20

graphlets. We show the results in Table 11 and mark the most accurate results with bold faces. In our experiments, we observe that betweenness centrality and closeness centrality are not very effective. Among all the evaluated ordering methods, degree centrality provides the biggest improvements on estimating almost all 3 and 4-node graphlet counts on ER graphs and BA graphs. For estimating 4-clique count on RGGs, the degree centrality ordering reduces 5.73% on the RE. The Weisfeiler–Lehman ordering achieves the best performance on RGGs and is also shown to be effective on ER graphs and BA graphs. However, given that the complexity of Weisfeiler–Lehman ordering is much higher than degree centrality, it is more suitable to use degree centrality as the first choice. We also validate the above observation for 5-node graphlets. Figure 15 shows that degree centrality almost always outperforms other node orderings in all graphs.

4.7 Estimation on real-world graph

In this subsection, we demonstrate that our framework can be applied to real-world scenarios by experimenting on three categories of real-world graphs: biochemistry graphs, collaboration graphs, and social network graphs.

4.7.1 Biochemistry dataset

We train our 1D CNN models for estimating nonzero graphlet counts, including, g_1^4 ( ), g_2^4 ( ), g_1^5 ( ), g_2^5 ( ), on three biochemistry datasets: MUTAG, NCI1, and NCI109. Table 2 gives the detailed information of these datasets.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

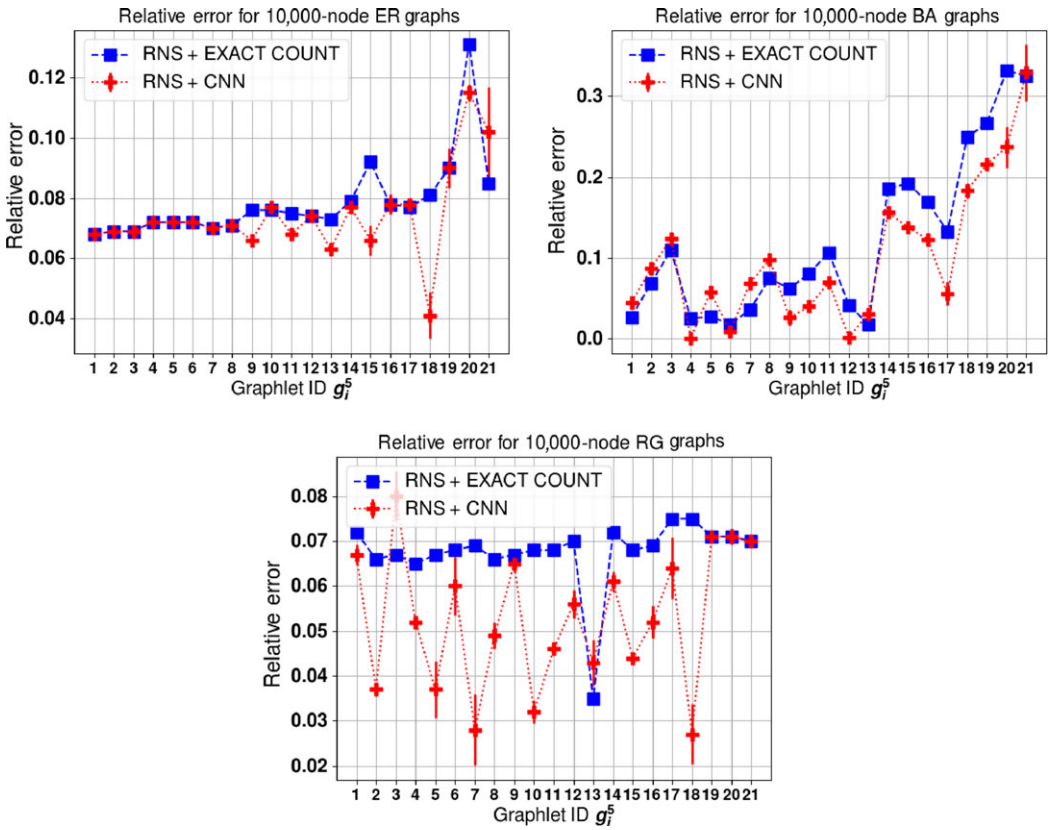


Figure 14. Relative errors for 5-node graphlets for large random graphs.

Because the graph samples in these real-world datasets have different number of nodes, we preprocess the input graph adjacency matrices with *adjacency matrix zero padding* technique as stated in Section 3.2.1. For NCI1 and NCI109, since the size of their graph samples vary over a large range, for example, the average number of nodes in a graph is around 30 but the biggest graph has 111 nodes, padding the graph adjacency matrices to size (111, 111) adds many useless information to the inputs. Hence, we further apply *node ordering* to obtain more consistent representation of the graph samples. We summarize our results in Table 12 as follows. Our model performs well on MUTAG dataset. The RE (MAE over mean) of all estimations are below 6.3%, and for estimating open triangle (g_1^3), our model achieves 0.9% RE. On NCI1 and NCI109, all the REs of our estimation are less than 7.7%. Our model performs better on estimating open triangle (g_1^3) on both NCI1 and NCI109, making only 0.09% and 1% RE, respectively. This experiment shows that our framework is effective on real-world biochemistry datasets.

4.7.2 Collaboration dataset

To demonstrate our framework's capability of handling large real-world graph, we evaluate our framework using two collaboration networks: ca-hep-ph and ca-hep-th. Table 3 gives details about these two graphs. Following the problem formulation in Section 2.3 to generate a dataset $\mathbb{G}$, we use the ca-hep-ph graph to generate 4,000 training/validation graphs as $\mathbb{G}_1$ to train our model. Then we generate 900 (unseen) testing graphs from the ca-hep-th as $\mathbb{G}_2$. Specifically, we apply the RNS approach introduced in Section 3.2.4 to sample 4,000 graphs from ca-hep-ph with the *node preserving probability* $p = 0.01$ to form a training set with size 4,000 (among them 400 graphs

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 10. Running time for large random graphs for 5-node graphlets

	g_1^5	g_2^5	g_3^5	g_4^5	g_5^5	g_6^5	g_7^5	g_8^5	g_9^5	g_{10}^5	g_{11}^5
(a) ER graph											
CNN train (ours)	3.441	3.889	3.981	3.888	3.795	3.700	3.574	4.951	3.668	4.042	3.581
RNS + CNN (ours)	**0.170**	**0.185**	**0.170**	**0.190**	**0.175**	**0.190**	**0.170**	**0.195**	**0.210**	**0.195**	**0.165**
RNS + EXACT	298.04	298.04	298.04	298.04	298.04	298.04	298.04	298.04	298.04	298.04	298.04
ESCAPE	242,820	242,820	242,820	242,820	242,820	242,820	242,820	242,820	242,820	242,820	242,820
	g_{12}^5	g_{13}^5	g_{14}^5	g_{15}^5	g_{16}^5	g_{17}^5	g_{18}^5	g_{19}^5	g_{20}^5	g_{21}^5	
CNN train (ours)	3.744	4.140	3.513	3.782	3.734	3.605	3.500	4.295	3.797	2.188	
RNS + CNN (ours)	**0.165**	**0.175**	**0.185**	**0.180**	**0.170**	**0.200**	**0.195**	**0.180**	**0.200**	**0.180**	
RNS + EXACT	298.04	298.04	298.04	298.04	298.04	298.04	298.04	298.04	298.04	298.04	
ESCAPE	242,820	242,820	242,820	242,820	242,820	242,820	242,820	242,820	242,820	242,820	
(b) BA graph											
	g_1^5	g_2^5	g_3^5	g_4^5	g_5^5	g_6^5	g_7^5	g_8^5	g_9^5	g_{10}^5	g_{11}^5
CNN train (ours)	6.390	7.229	6.970	6.766	5.393	6.033	6.419	6.116	5.654	6.194	6.263
RNS + CNN (ours)	**0.615**	**0.640**	**0.670**	**0.655**	**0.575**	**0.620**	**0.595**	**0.645**	**0.630**	**0.710**	**0.670**
RNS + EXACT	258.69	258.69	258.69	258.69	258.69	258.69	258.69	258.69	258.69	258.69	258.69
ESCAPE	36,255	36,255	36,255	36,255	36,255	36,255	36,255	36,255	36,255	36,255	36,255
	g_{12}^5	g_{13}^5	g_{14}^5	g_{15}^5	g_{16}^5	g_{17}^5	g_{18}^5	g_{19}^5	g_{20}^5	g_{21}^5	
CNN train (ours)	5.715	5.559	5.966	5.949	5.644	5.998	5.496	6.073	6.022	5.446	
RNS + CNN (ours)	**0.595**	**0.575**	**0.665**	**0.645**	**0.610**	**0.680**	**0.530**	**0.700**	**0.705**	**0.575**	
RNS + EXACT	258.69	258.69	258.69	258.69	258.69	258.69	258.69	258.69	258.69	258.69	
ESCAPE	36,255	36,255	36,255	36,255	36,255	36,255	36,255	36,255	36,255	36,255	
(c) RGG graph											
	g_1^5	g_2^5	g_3^5	g_4^5	g_5^5	g_6^5	g_7^5	g_8^5	g_9^5	g_{10}^5	g_{11}^5
CNN train (ours)	2.833	1.593	2.612	2.481	1.581	2.550	2.581	2.438	2.414	2.605	2.402
RNS + CNN (ours)	**0.875**	**1.120**	**1.120**	**1.125**	**1.140**	**0.960**	**1.040**	**1.050**	**0.875**	**0.915**	**0.920**
RNS + EXACT	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4
ESCAPE	273,810	273,810	273,810	273,810	273,810	273,810	273,810	273,810	273,810	273,810	273,810
	g_{12}^5	g_{13}^5	g_{14}^5	g_{15}^5	g_{16}^5	g_{17}^5	g_{18}^5	g_{19}^5	g_{20}^5	g_{21}^5	
CNN train (ours)	2.607	1.554	3.068	2.626	1.555	2.557	2.523	2.551	2.756	2.520	
RNS + CNN (ours)	**1.110**	**1.124**	**1.110**	**0.925**	**1.124**	**1.035**	**1.110**	**0.925**	**0.875**	**0.875**	
RNS + EXACT	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	1,289.4	
ESCAPE	273,810	273,810	273,810	273,810	273,810	273,810	273,810	273,810	273,810	273,810	

Note: Bold faces are the best results. Time for training are in minutes, others are in seconds.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 11. Relative error for different node orderings

Relative error (%)	g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
(a) ER graph								
No ordering	1.42	3.51	2.98	4.88	5.84	**1.59**	4.00	13.21
Betweenness	1.41	3.68	2.82	5.21	5.97	1.74	3.99	13.34
Closeness	1.44	3.46	2.82	4.94	5.77	1.69	4.22	12.27
WL	1.43	3.65	2.45	4.61	5.20	1.65	4.29	11.52
Degree	**1.35**	**3.27**	**2.33**	**4.24**	**5.19**	1.65	**3.57**	**11.01**
(b) BA graph								
No ordering	1.53	3.15	5.05	8.19	10.63	**1.59**	4.74	14.02
Betweenness	1.52	2.86	4.86	8.63	10.70	1.64	3.74	11.29
Closeness	1.66	2.82	4.50	8.00	10.00	1.97	3.66	11.45
WL	1.41	2.54	3.20	4.73	9.02	1.65	4.07	10.14
Degree	**1.31**	**2.31**	**2.36**	**3.26**	**7.13**	1.83	**2.71**	**8.29**
(c) RGG								
No ordering	9.35	5.80	18.90	29.00	54.90	11.56	7.68	19.25
Betweenness	9.76	5.74	18.92	30.89	50.62	11.09	7.62	18.65
Closeness	9.50	5.69	19.17	29.73	**50.14**	10.60	7.66	19.35
WL	**8.17**	**4.33**	**16.47**	**27.19**	50.24	**9.08**	7.81	**13.80**
Degree	8.91	4.53	17.71	28.03	51.16	9.61	**6.88**	14.85

Note: Bold faces are the best results.

are reserved for validation) and sample 900 graphs for testing from ca-hep-th with the node preserving probability $p = 0.009$. Since the sizes of the sampled small graphs vary, we further apply *adjacency matrix zero padding* to pad adjacency matrices to the same size, that is, (280, 280) for both training graphs and testing graphs. Also, we apply degree centrality-based node ordering technique to preprocess the training and testing graphs. Finally, as introduced in Section 3.2.4, we first take the CNN model's estimation on 900 testing graphs as inputs and reconstruct the global graphlet count estimation for ca-hep-th using the RNS estimator described by Equation (6). We denote the results as "RNS + CNN" (red bars in Figure 16(a)).

For 3,4-node graphlets, we first calculate the results by reconstructing the global count estimation using the ground truth count on 900 testing graphs, which is denoted as "RNS + Exact Count" (green bars) . Also, we use LGE with 660 edges (including its 2-hop egonets) for 7 runs and denote the result as "LGE" (purple bars). We also run MOSS-5 and ApproxG with controlled accuracy close to the RNS+CNN, which are denoted as "MOSS-5" (black bars) and "AprroxG" (orange bars). Note that we tried GRAFT, GUISE, and CC2 on ca-hep-th graph as well, but we do not present them because they did not terminate within 72 hours even with very low sampling ratio.

In Figure 16(a), we have larger error on 3-star (g_2^4) count, we analyze the reason as follows. First, as the results recovered from sample ground truth (green bars in Figure 16(a)) also have rather big error on 3-star, the 900 small testing graphs we sample seem to be less statistically representative for 3-star count in ca-hep-th. Second, we observe, in the testing set, that the variance

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

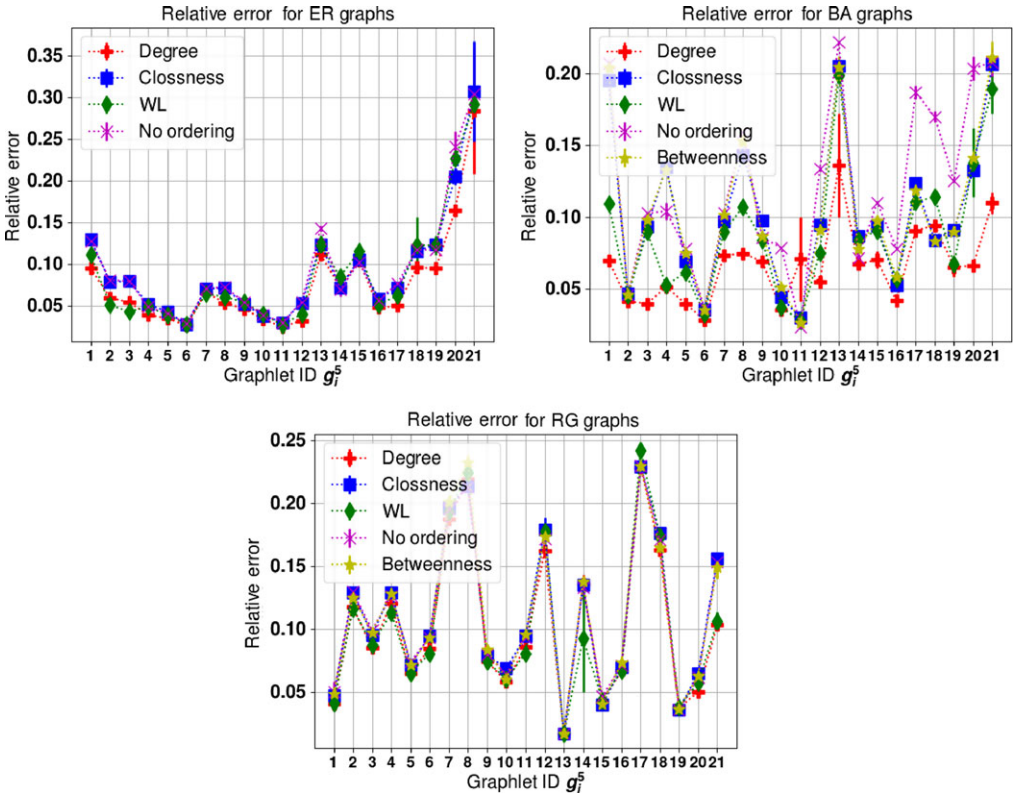


Figure 15. Relative errors for 5-node graphlets with different node orderings.

of 3-star graphlet count is large, that is, a small number of graph samples with significantly larger 3-star counts than most of the graph samples in the set. The error mainly comes from the underestimation on these outlying samples with large count. This is also the reason why LGE does not perform well in estimation of 3-star graphlet count. Except for 3-star, Figure 16(a) shows that, for all other 3-node and 4-node graphlets, the RE of our estimations is all below 8.7%. These results are close to the best results we can achieve theoretically (the green line and bars) given that we only sample 900 graphs. The results can be further improved by taking more samples from ca-hep-th. Compared with competing methods, RNS+CNN is superior in the accuracy for graphlet g_2^3 (), g_1^4 (), g_5^4 (), and g_6^4 () and on par or slightly worse for g_1^3 (), g_3^4 () and g_4^4 (). As for the running time shown in Table 13 (a), RNS+CNN spends less than 0.31 seconds for all 3 and 4-node graphlets and is the fastest among competing methods. MOSS-5 is the closest to ours and the PGD exact method even consumes 199.92 seconds, and ours is about 318 times faster than theirs.

For 5-node graphlets, we compare our 1D CNN with MOSS-5 and ESCAPE. Note that results for GRAFT, CC2, and GUISE are not shown because they cannot finish within reasonable time. Look at the RE shown on the left in Figure 17, most 5-node graphlets have RE less than 7.5%. As for the running time shown in Table 14 (a), our RNS+CNN achieves at least 3.4 times speedup compared with MOSS-5, and five orders of magnitude faster than ESCAPE. More importantly, the running time remains almost the same as the 3,4-node graphlets', while exact counting methods (i.e., ESCAPE) grows exponentially with the graphlet size. This experiment demonstrates that our framework is effective in real-world scenario for large collaboration graphs.

4.7.3 Social network dataset

We also evaluate our framework using two sparse social networks: socfb-MSU24 and socfb-FSU53. Similar to Section 4.7.2, we first use RNS to generate 3,300 graphs (3,000 for training,

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 12. Graphlet count estimation results on real-world biochemistry datasets

	g_1^3	g_1^4	g_2^4	g_1^5	g_2^5
(a) MUTAG					
MAE	0.29	1.37	0.41	2.62	2.05
Mean	30.75	43.25	7.8	58.7	32.2
STD	9.62	16.10	2.73	27.49	13.92
Relative error	0.009±0.0012	0.031±0.0030	0.052±0.0028	0.045±0.0017	0.063±0.0014
(b) NCI1					
MAE	0.51	2.27	0.37	6.97	4.20
Mean	51.26	70.27	13.84	90.94	54.32
STD	24.46	35.53	8.53	50.76	36.10
Relative error	0.009±0.0006	0.032±0.0010	0.027±0.0029	0.076±0.0038	0.077±0.0018
(c) NCI109					
MAE	0.48	2.17	0.42	6.51	3.84
Mean	48.04	65.56	13.01	85.01	50.81
STD	25.08	36.48	8.39	51.66	34.48
Relative error	0.010±0.00097	0.032±0.00048	0.032±0.0016	0.076±0.0032	0.075±0.0031

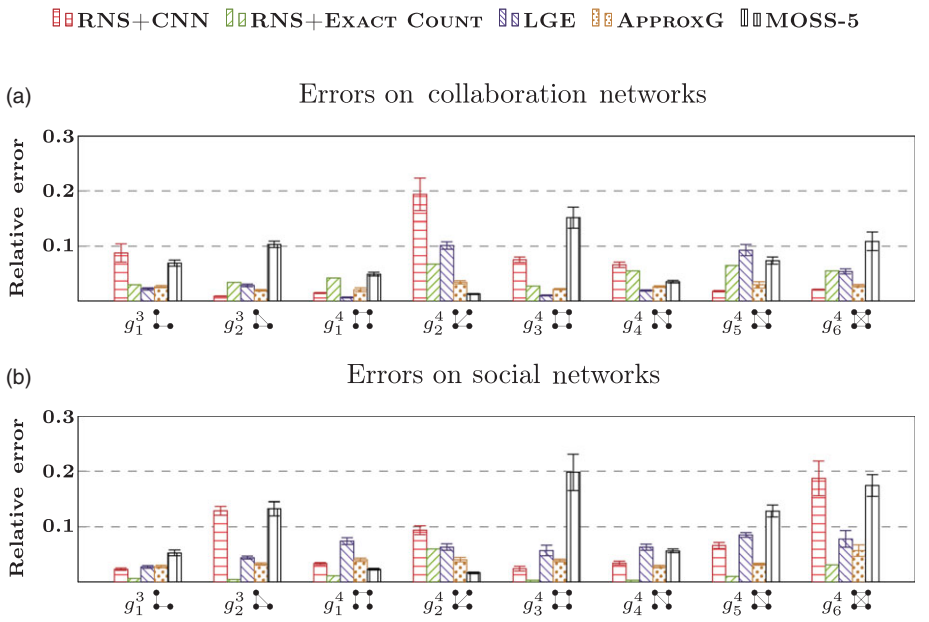


Figure 16. 3,4-node graphlet count estimation results on real world graphs.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 13. Running time for 3,4-node graphlet count estimation on real-world graphs

		(a) Collaboration networks							
		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
Training (in minutes)	RNS+CNN (ours)	3.46	5.20	2.98	5.93	2.23	6.73	3.86	2.75
Testing (in seconds)	RNS+CNN (ours)	**0.23**	**0.31**	**0.31**	**0.25**	**0.31**	**0.31**	**0.23**	**0.28**
	RNS + EXACT	167.7	167.7	167.7	167.7	167.7	167.7	167.7	167.7
	GRAFT	-	-	-	-	-	-	-	-
	CC2	-	-	-	-	-	-	-	-
	GUISE	-	-	-	-	-	-	-	-
	LGE	12.53	11.75	12.72	13.55	12.72	12.72	12.72	12.72
	ApproxG	0.98	0.98	1.02	1.02	1.25	1.02	1.25	1.02
	MOSS-5	0.55	0.55	0.63	0.62	0.78	0.62	0.62	0.88
	PGD EXACT	199.92	199.92	199.92	199.92	199.92	199.92	199.92	199.92
	ESCAPE	872.92	872.92	872.92	872.92	872.92	872.92	872.92	872.92
		(b) Social networks							
		g_1^3	g_2^3	g_1^4	g_2^4	g_3^4	g_4^4	g_5^4	g_6^4
Time (in minutes)	RNS+CNN (ours)	3.96	2.72	4.13	2.88	5.20	6.07	3.51	4.45
Testing (in seconds)	RNS+CNN (ours)	**0.487**	**0.476**	**0.440**	**0.456**	0.564	**0.310**	**0.445**	**0.483**
	RNS + EXACT	18.25	18.25	18.25	18.25	18.25	18.25	18.25	18.25
	GRAFT	-	-	-	-	-	-	-	-
	CC2	-	-	-	-	-	-	-	-
	GUISE	-	-	-	-	-	-	-	-
	LGE	0.865	0.827	0.865	0.865	0.865	0.865	0.827	0.865
	ApproxG	0.49	0.49	0.56	0.56	**0.49**	0.56	0.49	0.49
	MOSS-5	2.19	2.23	2.60	2.22	5.21	2.75	5.35	4.99
	PGD EXACT	3.42	3.42	3.42	3.42	3.42	3.42	3.42	3.42
	ESCAPE	10.82	10.82	10.82	10.82	10.82	10.82	10.82	10.82

Note: a hyphen (-) indicates that the method did not terminate within 4 hours. Bold faces are the best results.

300 for validation) with node preserving probability $p = 0.02$ from socfb-MSU24. Then 500 testing graphs are generated using socfb-FSU53 with the same node preserving probability. Next, we pad training, validation, and testing graphs to (560, 560) and apply the degree centrality-based node ordering technique. Finally, we use Equation (6) to calculate the estimated graphlet counts for socfb-FSU53.

For 3,4-node graphlets, we show the RE results for our method and competing methods in Figure 16(b). The RNS+Exact count method achieves the best accuracy among competing methods and our RNS+CNN method gets fairly accurate estimations, expect for 3-triangle (g_2^3) and 4-clique (g_6^4). The reason is that 3-triangle and 4-clique counts are much less than other graphlets in sparse graphs. With inadequate samples, our CNN model can not learn well on structural patterns of the 3-triangle and 4-clique. For the running time (shown in Table 13(b)), our method is still the fastest for most 3, 4-node graphlets. But this time, we provide less than 10 times speedup compared to the PGD exact method, and the ApproxG, MOSS-5 are also very close to RNS+CNN. In our experiments, we find most existing methods are more efficient for sparser graphs than

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

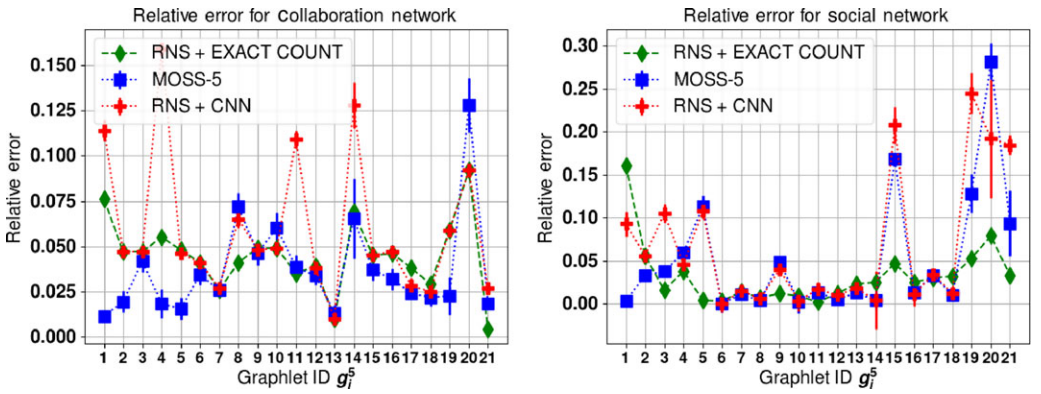


Figure 17. 5-node graphlet count estimation results on real-world graphs.

denser ones given their graph sizes do not vary too much. So for sparse graphs, the competing methods are more competitive.

For 5-node graphlets, we show the RE in the right-hand side of Figure 17. We have less than 5% error for half of the graphlets, and the largest error is clique-like graphlets, that is, g_{19}^5, g_{20}^5 and g_{21}^5. The reason is also due to the inadequate samples for these graphlets. As for the running time, we show the results in Table 14. For most graphlets, our RNS+CNN methods are superior than the competing methods. Again, for the running time, this experiment shows our method is effective and competitive against state-of-the-art methods for sparse social networks.

Interestingly, we find the RE curve of the social network is very similar to large random BA graphs in Figure 14, and the collaboration network is very similar to large RG graphs. This further demonstrates that the performance of our model is related to the underlying distribution. Moreover, we find the results for real-world networks are slightly worse than random graphs shown in Figure 14. We believe that the reason is related to the generalizability of our CNN model. For random graphs, we generate training and testing graphs from exactly the same distribution, that is, using the same or similar configurations. For real-world networks, the training and testing graph do not come from the same distribution, so our CNN model has larger error. To solve this problem, we can train a mixture model using more networks, as what we do in the Section 4.4.3, to enhance the generalizability of our model.

5. Related work

5.1 Existing graphlet counting framework

Existing techniques can be roughly categorized into two groups. One aims at obtaining the exact graphlet counts and the other utilizes sampling techniques and focuses more on GFD.

For exact graphlet counts, the most straightforward approach is to traverse through the graph, pick any k nodes, check the edge connectivity between them, and count every occurrence of a specific graphlet. Many existing works propose to accelerate this exhaustive enumeration process by leveraging on parallelism, distributed systems, and combinatorial relationships among graphlets. For example, Schank et al.'s work (Schank & Wagner, 2005) gives a MapReduce-based algorithm to count triangle on distributed systems. Ahmed et al. (Ahmed et al., 2017) propose the PGD framework which counts 3-node graphlets, 4-clique, and 4-cycle first, and then derive the counts of the rest of 4-node graphlets directly from the combinatorial relationships between 3-node and 4-node graphlets. Pinar et al. (Pinar et al., 2017) build upon Ahmed et al's idea and propose the ESCAPE framework for counting all 5-node graphlets.

The second group of works aim at a less ambitious but more practical goal: estimate graphlet count or GFD with high accuracy. They utilize various sampling techniques and provide fast and accurate algorithms with statistically provable error bounds. For example, Rahman et al. propose

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Table 14. Running time for 5-node graphlet count estimation on real-world graphs

	g_1^5	g_2^5	g_3^5	g_4^5	g_5^5	g_6^5	g_7^5	g_8^5	g_9^5	g_{10}^5	g_{11}^5
(a) Collaboration networks											
CNN train (ours)	3.675	3.694	3.450	3.714	3.435	3.739	3.679	3.262	3.729	3.712	3.449
RNS + CNN (ours)	**0.232**	**0.248**	**0.190**	**0.243**	**0.210**	**0.237**	**0.248**	**0.258**	**0.238**	**0.240**	**0.207**
RNS + EXACT	153.3	153.3	153.3	153.3	153.3	153.3	153.3	153.3	153.3	153.3	153.3
MOSS-5	0.800	0.790	0.660	0.800	0.800	0.940	1.030	1.030	1.030	0.670	0.800
ESCAPE	130,733	130,733	130,733	130,733	130,733	130,733	130,733	130,733	130,733	130,733	130,733
	g_{12}^5	g_{13}^5	g_{14}^5	g_{15}^5	g_{16}^5	g_{17}^5	g_{18}^5	g_{19}^5	g_{20}^5	g_{21}^5	
CNN train (ours)	3.427	3.285	3.456	3.730	3.748	3.390	3.158	3.495	3.705	3.465	
RNS + CNN (ours)	**0.243**	**0.245**	**0.203**	**0.235**	**0.242**	**0.240**	**0.195**	**0.205**	**0.240**	**0.202**	
RNS + EXACT	153.3	153.3	153.3	153.3	153.3	153.3	153.3	153.3	153.3	153.3	
MOSS-5	0.940	1.750	0.800	0.670	0.660	1.750	0.790	0.670	0.800	0.940	
ESCAPE	130,733	130,733	130,733	130,733	130,733	130,733	130,733	130,733	130,733	130,733	
(b) Social networks											
	g_1^5	g_2^5	g_3^5	g_4^5	g_5^5	g_6^5	g_7^5	g_8^5	g_9^5	g_{10}^5	g_{11}^5
CNN train (ours)	6.269	5.513	5.250	6.279	8.470	5.311	6.288	10.416	5.286	10.556	5.342
RNS + CNN (ours)	**0.525**	0.435	0.475	**0.550**	0.805	**1.160**	**0.548**	**1.138**	**0.480**	**0.780**	**1.202**
RNS + EXACT	125.2	125.2	125.2	125.2	125.2	125.2	125.2	125.2	125.2	125.2	125.2
MOSS-5	1.370	**0.330**	**0.310**	0.610	**0.310**	13.740	1.730	13.740	1.370	13.740	1.370
ESCAPE	584.5	584.5	584.5	584.5	584.5	584.5	584.5	584.5	584.5	584.5	584.5
	g_{12}^5	g_{13}^5	g_{14}^5	g_{15}^5	g_{16}^5	g_{17}^5	g_{18}^5	g_{19}^5	g_{20}^5	g_{21}^5	
CNN train (ours)	5.371	9.912	6.296	10.614	6.375	5.309	6.485	6.579	6.683	6.803	
RNS + CNN (ours)	**0.518**	**0.817**	**0.625**	**0.457**	**0.565**	**0.463**	**0.527**	**0.605**	**0.558**	**0.418**	
RNS + EXACT	125.2	125.2	125.2	125.2	125.2	125.2	125.2	125.2	125.2	125.2	
MOSS-5	1.370	4.620	55.790	0.620	1.370	0.610	55.790	0.610	0.620	0.520	
ESCAPE	584.5	584.5	584.5	584.5	584.5	584.5	584.5	584.5	584.5	584.5	

Note: Bold faces are the best results. Time for training are in minutes, others are in seconds.

an edge sampling-based framework called GRAFT (Rahman et al., 2014). Seshadhri et al. (2013) propose a wedge sampling technique and Jha et al. (2015) extend this idea by introducing path sampling with a special pruning schema to estimate all 4-node graphlets. Rossi et al. (2018) extend the PGD framework and develop a LGE framework by computing graphlets on sampled localized neighborhoods. Mawhirter et al. (2018) propose the distributed ApproxG framework to count 3, 4-node graphlets by the cost-aware task sampling with controllable accuracy. For estimating k-node graphlets (for any k), Bhuiyan et al. (2012) first apply the MCMC sampling method, and Chen et al. (2016) adapt the MCMC sampling and estimate the ratio between the occurrence of different k-node graphlets. Wang et al. (2017) divide 5-node graphlets into two groups, propose T-5 and Path-5 sampling methods for each group and develop the MOSS-5 framework to estimate 3, 4, 5-node graphlets. Bressan et al. (2017) point out the long mixing time of MCMC methods

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

and propose a color coding-based method to estimate up to 7-node graphlet counts. For special graphlets, such as k-cliques, Jain and Seshadhri's work (Jain & Seshadhri, 2017) incorporates Chiba and Nishizaki's subgraph listing algorithm (Chiba & Nishizeki, 1985) and Turan's theorem for approximating the number of k-clique (for any k).

However, existing exact counting and sampling methods only take graphs as input and mechanically output their graphlet counts, while ignoring the information from the results they just computed for previous graphs. As we discussed in Section 2.3, failing to utilize the information of historical graphs, these methods suffer loss in accuracy or efficiency or even both when estimating graphlet counts in new-coming graphs.

5.2 Learning on graphs

Graphs are natural representations for a variety of real-world applications, including social analysis (Backstrom & Leskovec, 2011), traffic prediction (Zhang et al., 2012), recommendation systems (Zhang et al., 2010), and computer vision (Monti et al., 2017). By representing data as graphs, on one hand, we can encode relationships among entities and give more insights in the underlying data. On the other hand, the complex structures of graphs pose a challenge to learning the true insights and extracting useful information (Boccaletti et al., 2006). Many traditional techniques such as graph kernel (Vishwanathan et al., 2010; Shervashidze et al., 2009) and graph embedding (Grover & Leskovec, 2016; Perozzi et al., 2014) methods are proposed to deal with the learning problem on graphs. Graph kernel method, in general, can be used to measure similarity among graphs by computing shortest paths, subtrees or cycles (Shervashidze et al., 2009). However, it is often hard to priorly know which graph kernel captures key structural property better (Niepert et al., 2016). Graph embedding method, which learns a graph representation in a low-dimensional space, is more task-specific. This technique has been adopted to address graph learning problems such as link prediction and node classification (Grover & Leskovec, 2016). But its learning ability is always limited for discovering intricate network patterns because of its shallow learning mechanism.

Lately, CNNs have arisen in many fields and yielded promising empirical results. CNNs have demonstrated their significant representation ability, impressive learning capacity, and efficiency in space compared with traditional kernel or embedding approaches. However, most prevailing CNN models are originally developed for regular 1D, 2D, and 3D grid data such as text, image, or video, while graph is generally of high dimensional and irregular. CNN's key components such as convolutions and filtering cannot be generalized to the graph domain in a trivial manner. Thus, a variety of recent works were proposed to solve this problem. These works can be further divided into two categories: the spectral models and the spatial models. Spectral methods (Bruna et al., 2013; Henaff et al., 2015; Defferrard et al., 2016; Kipf & Welling, 2016) are based on the spectral graph theory: they define parameter filters and convolution operators by the graph Fourier transform. As the spectral method relies on specific eigenfunctions of Laplacian matrix, the learned parameters on one graph are difficult to be transferred to another graph with different eigenvalues. The spatial models (Hamilton et al., 2017; Veličković et al., 2017; Monti et al., 2017), which focus on generalizing CNN concepts on spatial domain of graphs, are more flexible. For example, Hamilton et al. (2017) propose the GraphSAGE framework to learn node embeddings by aggregating feature information from nodes' local neighborhoods. Velickovic et al. (2017) follow Hamilton et al.'s idea and propose a novel attention layer which gives different learnable importance weights on aggregating local neighbors of a node. Besides, Monti et al. (2017) also introduce a generic graph convolution network, Monet, by designing a general patch operator which integrates the neighborhood information.

The objective of our work is different from the existing graph learning frameworks. Existing frameworks focus on learning representations of data points, that is, node embeddings and network embeddings. For these representation learning frameworks, node or edge attributes are their

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

main learning subjects; the underlying graph structure is viewed as an auxiliary information. For our work, however, the underlying graph structure is the key to solve the GCL problem. We aim to apply learning model to learn the mapping from underlying graph structure to the structural attributes, that is, graphlet counts.

6. Conclusion

In this paper, we point out that, for existing graphlet counting and estimation framework, the cost for repeated computation in testing can be further reduced, if we learn from previously studied graphs and leverage on the correlation between the graph structural information and graphlet counts. We abstract the above finding and formulate it as a novel GCL problem, which bridges the graphlet counting and the learning paradigm. We then propose a CNN-based framework with a series of data preprocessing methods to solve the GCL problem. To show the effectiveness of our CNN framework, we conduct extensive experiments on three kinds of random graphs and three categories of real-world graphs, namely, biochemistry graphs, collaboration graphs, and social network graphs for 3, 4, 5-node graphlet counting. The results demonstrates that our framework has great potential to compete with state-of-the-art methods and is accurate, efficient, general and can achieve consistent results to handle any sized graphlets (i.e., 6-node graphlets) without changing the model.

For the future work, extending the framework to local graphlet count on directed graph is a natural next step. Also, it will be interesting to further improve the accuracy and scalability of our method to handle larger graphs (e.g., graphs with millions of nodes and edges). Finally, generalizability is an important issue for learning methods. Our GCL problem assumes graph data comes from the same (or similar) distribution. Later extensions can eventually relax this assumption and learn a more general model which allows us to handle graphs from totally different distributions.

Acknowledgments. The work of John C.S. Lui is supported in part by the GRF 14201819.

Conflict of interest. None.

Notes

1 https://docs.scipy.org/doc/scipy/reference/sparse.html.
2 We use the implementation of GRAFT from http://dmgroup.cs.iupui.edu/mmrahman-GRAFT.php.
3 We use the implementation of CC2 from https://github.com/Steven–/graphlets.
4 We use the implementation of GUISE from http://dmgroup.cs.iupui.edu/mmrahman-GUISE.php.
5 We use the implementation of PGD from https://github.com/nkahmed/PGD.
6 We use the implementation of ApproxG from https://bitbucket.org/approxg/approxg/src/master/.
7 We use the implementation of MOSS-5 from http://nskeylab.xjtu.edu.cn/dataset/phwang/code/mosscode.zip.
8 Note that PGD EXACT and ESCAPE gives the exact graphlet count, and thus their REs are always 0.

References

Ahmed, N. K., Neville, J., Rossi, R. A., & Duffield, N. (2015). Efficient graphlet counting for large networks. In *2015 IEEE international conference on data mining (ICDM)* (pp. 1–10). IEEE.

Ahmed, N. K., Neville, J., Rossi, R. A., Duffield, N. G., & Willke, T. L. (2017). Graphlet decomposition: Framework, algorithms, and applications. *Knowledge and Information Systems*, *50*(3), 689–722.

Akoglu, L., Tong, H., & Koutra, D. (2015). Graph based anomaly detection and description: A survey. *Data Mining and Knowledge Discovery*, *29*(3), 626–688.

Alon, N., Yuster, R., & Zwick, U. (1995). Color-coding. *Journal of the ACM (JACM)*, *42*(4), 844–856.

Backstrom, L., & Leskovec, J. (2011). Supervised random walks: Predicting and recommending links in social networks. In *Proceedings of the Fourth ACM international conference on web search and data mining* (pp. 635–644). ACM.

Becchetti, L., Boldi, P., Castillo, C., & Gionis, A. (2008). Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 16–24). ACM.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Bhuiyan, M. A., Rahman, M., & Al Hasan, M. (2012). Guise: Uniform sampling of graphlets for large graph analysis. In *2012 IEEE 12th international conference on data mining (ICDM)* (pp. 91–100). IEEE.
Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., & Hwang, D.-U. (2006). Complex networks: Structure and dynamics. *Physics Reports*, *424*(4–5), 175–308.
Bressan, M., Chierichetti, F., Kumar, R., Leucci, S., & Panconesi, A. (2017). Counting graphlets: Space vs time. In *Proceedings of the tenth ACM international conference on web search and data mining* (pp. 557–566). ACM.
Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. arxiv preprint arxiv:1312.6203.
Butler, S. K. (2008). *Eigenvalues and structures of graphs*. Ph.D. thesis, UC San Diego.
Chen, X., & Lui, J. (2018). Mining graphlet counts in online social networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *12*(4), 41.
Chen, X., Li, Y., Wang, P., & Lui, J. (2016). A general framework for estimating graphlet statistics via random walk. *Proceedings of the VLDB Endowment*, *10*(3), 253–264.
Chiba, N., & Nishizeki, T. (1985). Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, *14*(1), 210–223.
Chung, K. L. (2001). *A course in probability theory*. Cambridge, Massachusetts, USA: Academic Press.
Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems* (pp. 3844–3852).
Grover, A., & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 855–864). ACM.
Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in neural information processing systems* (pp. 1024–1034).
Henaff, M., Bruna, J., & LeCun, Y. (2015). Deep convolutional networks on graph-structured data. arxiv preprint arxiv:1506.05163.
Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, *58*(301), 13–30.
Holland, P. W., & Leinhardt, S. (1976). Local structure in social networks. *Sociological Methodology*, *7*, 1–45.
Hubel, D. H., & Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, *195*(1), 215–243.
Jain, S., & Seshadhri, C. (2017). A fast and provable method for estimating clique counts using turán's theorem. In *Proceedings of the 26th international conference on world wide web* (pp. 441–449). International World Wide Web Conferences Steering Committee.
Jha, M., Seshadhri, C., & Pinar, A. (2015). Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the 24th international conference on world wide web* (pp. 495–505). International World Wide Web Conferences Steering Committee.
Johnson, R., & Zhang, T. (2015). Semi-supervised convolutional neural networks for text categorization via region embedding. In *Advances in neural information processing systems* (pp. 919–927).
Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1725–1732).
Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arxiv preprint arxiv:1412.6980.
Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arxiv preprint arxiv:1609.02907.
Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).
LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436.
Leskovec, J., & Faloutsos, C. (2006). Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 631–636). ACM.
Leskovec, J., Kleinberg, J., & Faloutsos, C. (2007). Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *1*(1), 2.
Liu, C., Yan, X., Yu, H., Han, J., & Yu, P. S. (2005). Mining behavior graphs for backtrace of noncrashing bugs. In *Proceedings of the 2005 SIAM international conference on data mining* (pp. 286–297). SIAM.
Mawhirter, D., Wu, B., Mehta, D., & Ai, C. (2018). ApproxG: Fast approximate parallel graphlet counting through accuracy control. In *IEEE/ACM international symposium on cluster, cloud and grid computing (CCGRID)* (pp. 533–542). IEEE.
Milenković, T., Memišević, V., Ganesan, A. K., & Pržulj, N. (2010). Systems-level cancer gene identification from protein interaction network topology applied to melanogenesis-related functional genomics data. *Journal of the Royal Society Interface*, *7*(44), 423–437.
Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2016). Pruning convolutional neural networks for resource efficient inference. arxiv preprint arxiv:1611.06440.

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., & Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proceedings of the CVPR* (p. 3), vol. 1.

Newman, M. E. J. (2005). Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, *5*, 323–351.

Niepert, M., Ahmed, M., & Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In *International conference on machine learning* (pp. 2014–2023).

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 701–710). ACM.

Pinar, A., Seshadhri, C., & Vishal, V. (2017). Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th international conference on world wide web* (pp. 1431–1440). International World Wide Web Conferences Steering Committee.

Pržulj, N. (2007). Biological network comparison using graphlet degree distribution. *Bioinformatics*, *23*(2), e177–e183.

Rahman, M., Bhuiyan, M. A., & Al Hasan, M. (2014). Graft: An efficient graphlet counting method for large graph analysis. *IEEE Transactions on Knowledge and Data Engineering*, *26*(10), 2466–2478.

Rossi, R. A., & Ahmed, N. K. (2015). The network data repository with interactive graph analytics and visualization. In *Proceedings of the twenty-ninth AAAI conference on artificial intelligence*.

Rossi, R. A., & Ahmed, N. K. (2019). Complex networks are structurally distinguishable by domain. *Social Network Analysis and Mining*, *9*, 51.

Rossi, R. A., Zhou, R., & Ahmed, N. K. (2018). Estimation of graphlet counts in massive networks. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(1), 1–14.

Schank, T., & Wagner, D. (2005). Finding, counting and listing all triangles in large graphs, an experimental study. In *International workshop on experimental and efficient algorithms* (pp. 606–609). Springer.

Schöning, U. (1988). Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, *37*(3), 312–323.

Seshadhri, C., Pinar, A., & Kolda, T. G. (2013). Triadic measures on graphs: The power of wedge sampling. In *Proceedings of the 2013 SIAM international conference on data mining* (pp. 10–18). SIAM.

Shervashidze, N., Vishwanathan, S. V. N., Petri, T., Mehlhorn, K., & Borgwardt, K. (2009). Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics* (pp. 488–495).

Sporns, O., Chialvo, D. R., Kaiser, M., & Hilgetag, C. C. (2004). Organization, development and function of complex brain networks. *Trends in Cognitive Sciences*, *8*(9), 418–425.

Tsourakakis, C. E. (2008). Fast counting of triangles in large real networks without counting: Algorithms and laws. In *Eighth IEEE international conference on data mining, 2008. ICDM'08* (pp. 608–617). IEEE.

Ugander, J., Backstrom, L., & Kleinberg, J. (2013). Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of the 22nd international conference on world wide web* (pp. 1307–1318). ACM.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2017). Graph attention networks. arxiv preprint arxiv:1710.10903.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., & Borgwardt, K. M. (2010). Graph kernels. *Journal of Machine Learning Research*, *11*, 1201–1242.

Wale, N., Watson, I. A., & Karypis, G. (2008). Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, *14*(3), 347–375.

Wang, P., Lui, J. C. S., Towsley, D., & Zhao, J. (2016). Minfer: A method of inferring motif statistics from sampled edges. In *2016 IEEE 32nd international conference on data engineering (ICDE)*.

Wang, P., Zhao, J., Zhang, X., Li, Z., Cheng, J., Lui, J. C. S., Towsley, D., Tao, J., & Guan, X. (2017). MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering*, *30*(1), 73–86.

Weisfeiler, B., & Lehman, A. A. (1968). A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-technicheskaya Informatsia*, *2*(9), 12–16.

Zhang, B., Xing, K., Cheng, X., Huang, L., & Bie, R. (2012). Traffic clustering and online traffic prediction in vehicle networks: A social influence perspective. In *INFOCOM, 2012 Proceedings IEEE* (pp. 495–503). IEEE.

Zhang, M., & Chen, Y. (2017). Weisfeiler–Lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 575–583). ACM.

Zhang, Z.-K., Zhou, T., & Zhang, Y.-C. (2010). Personalized recommendation via integrated diffusion on user–item–tag tripartite graphs. *Physica A: Statistical Mechanics and its Applications*, *389*(1), 179–186.

Cite this article: Liu X., Chen Y. J., Lui J. C. S. and Avrachenkov K. (2021). Learning to count: A deep learning framework for graphlet count estimation. *Network Science* **9**, S23–S60. https://doi.org/10.1017/nws.2020.35

https://doi.org/10.1017/nws.2020.35 Published online by Cambridge University Press