Star Topology Convolution for Graph Representation Learning

Chong Wu 1,1,1,1,1,2, Zhenan Feng 3, Jiangbin Zheng 3, Houwang Zhang 3, Jiawang Cao 3, and Hong YAN 3

1Department of Electrical Engineering
2Department of Electrical Engineering and Centre for Intelligent Multidimensional Data Analysis
3Affiliation not available

November 8, 2023

Abstract

We present a novel graph convolutional method called star topology convolution (STC). This method makes graph convolution more similar to conventional convolutional neural networks (CNNs) in Euclidean feature space. Unlike most existing spectral convolutional methods, this method learns subgraphs which have a star topology rather than a fixed graph. It has fewer parameters in its convolutional filter and is inductive so that it is more flexible and can be applied to large and evolving graphs. As for CNNs in Euclidean feature space, the convolutional filter is localized and maintains a good weight sharing property. By introducing deep layers, the method can learn global features like a CNN. To validate the method, STC was compared to state-of-the-art spectral convolutional and spatial convolutional methods in a supervised learning setting on three benchmark datasets: Cora, Citeseer and Pubmed. The experimental results show that STC outperforms the other methods. STC was also applied to protein identification tasks and outperformed traditional and advanced protein identification methods.
Star Topology Convolution for Graph Representation Learning

Chong Wu1*,†, Zhenan Feng1†, Jiangbin Zheng2, Houwang Zhang1, Jiawang Cao3 and Hong Yan1

1Department of Electrical Engineering and Centre for Intelligent Multidimensional Data Analysis, City University of Hong Kong, Kowloon, Hong Kong SAR.
2School of Engineering, Westlake University, Hangzhou, 310024, China.
3Academy of Engineering & Technology, Fudan University, Shanghai, 200433, China.

*Corresponding author(s). E-mail(s): chongwu2-c@my.cityu.edu.hk;
Contributing authors: zhenafeng2-c@my.cityu.edu.hk; zhengjiangbin@westlake.edu.cn; houwang.zhang@my.cityu.edu.hk;
jwcao19@fudan.edu.cn; h.yan@cityu.edu.hk;
†These authors contributed equally to this work.

Abstract

We present a novel graph convolutional method called star topology convolution (STC). This method makes graph convolution more similar to conventional convolutional neural networks (CNNs) in Euclidean feature spaces. STC learns subgraphs which have a star topology rather than learning a fixed graph like most spectral methods. Due to the properties of a star topology, STC is graph-scale free (without a fixed graph size constraint). It has fewer parameters in its convolutional filter and is inductive, so it is more flexible and can be applied to large and evolving graphs. The convolutional filter is learnable and localized, similar to CNNs in Euclidean feature spaces, and can share weights across graphs. To test the method, STC was compared with state-of-the-art graph convolutional methods in a supervised learning setting on nine node properties prediction benchmark datasets: Cora, Citeseer, Pubmed, PPI, Arxiv, MAG, ACM, DBLP, and IMDB. The
experimental results showed that STC achieved state-of-the-art performance on all these datasets and maintained good robustness. In an essential protein identification task, STC outperformed state-of-the-art essential protein identification methods. An application of using pretrained STC as the embedding for feature extraction of some downstream classification tasks was introduced. The experimental results showed that STC can share weights across different graphs and be used as the embedding to improve the performance of downstream tasks.

**Keywords:** Big Data, Graph Convolution, Graph Representation Learning, Spectral Convolution, Star Topology.

1 Introduction

Convolutional neural networks (CNNs) have been used to solve problems which have a Euclidean feature space [1], such as image classification [2] and machine translation [3]. However, most problems, such as 3D meshes, social networks, telecommunication networks, and biological networks, have a non-Euclidean nature [4], and data in the form of a graph is a typical non-Euclidean problem. This makes using CNNs to solve these problems a challenge [1]. There are three major problems in generalizing CNNs to graphs: (1) the numbers of directly connected neighbors for nodes usually differ [5]; (2) the feature dimensions for nodes may also differ; (3) the edges may have features with dimensions that differ.

The convolution operation based on the graph Fourier transform, which is called spectral convolutional neural network (Spectral CNN), was first introduced to the graph domain by [6]. Simplified versions of Spectral CNN based on polynomial expansion have been proposed, like Chebyshev network (ChebyNet) [7] and graph convolutional network (GCN) [8]. ChebyNet [7] restricts the kernel of Spectral CNN to a polynomial expansion. GCN [8], a simplification of ChebyNet to bypass the spectral transform, reduces the computational cost of the eigen decomposition of the graph Laplacian matrix and is able to get spatially localized filters [1]. Due to its good performance and faster speed of convolution computation, GCN is widely used to solve graph problems [9]. It has limitations when scaled to large graphs and is difficult to train in a minibatch setting [10].

To address the problem of scaling GCN to large graphs, layer sampling methods [11–15] and subgraph sampling methods [10, 16, 17] have been proposed. These are designed for efficient minibatch training of GCN [10]. Layer sampling methods are based on a GCN on the full training graph with nodes or edges sampled from the whole graph to form minibatches in each layer with forward and backward propagation on the sampled graph. Subgraph sampling methods perform subgraph sampling before training the GCN to reduce the
problem of large computational costs in the node/edge sampling of each layer [10, 16, 17].

However, the filter size is based on the size of the full graph or the size of the sampled subgraph, which is not flexible enough for different sizes of input. If the size changes, the model trained using the previous filter will need to be reconstructed and retrained. Although some flexible spatial methods [18] have been proposed, their aggregators are neither learnable nor convolutional. The problem of designing a flexible filter, like spatial methods, with a learnable and convolutional property, like spectral methods, still remains to be solved.

To address this problem, we propose star topology convolution (STC). All graphs are composed of subgraphs with a star topology, as Figures 1 & 2 show. A star topology has several advantages in its eigen decomposition, as its eigenvalues are all equal except for the first and last elements. This means that the Laplacian matrices of star topology graphs, even of different sizes, will have some common eigenvectors, although they may be rotated. These properties allow the design of a graph-scale free (without a fixed graph size constraint), learnable, and convolutional filter which can maintain weight sharing for subgraphs of different sizes and structures.

With such a flexible filter the spectral convolution over a subgraph with a star topology can be defined. Then, a star topology convolution (STC) can be introduced for graph representation learning to perform the spectral convolution on these subgraphs to obtain the spectral node embedding. These can be aggregated to the central node of each subgraph. Hence, this method can create localized filters in both the spectral and spatial domains. The computational complexity and memory requirements are largely reduced compared to conventional spectral methods. To further improve the performance of STC, an edge attention mechanism is defined on the star topology subgraphs. As STC is inductive and has a low memory requirement, it can be applied to large or evolving graphs. STC can share weights across different graphs like a CNN which makes it more flexible than most existing spectral methods.

The contributions of this paper can be summarized as:
1. This is the first attempt to use star topology subgraphs for convolution in graph representation learning.
2. This is a graph convolutional method which has a graph-scale free, learnable, and convolutional filter.
3. This method has a comparatively low computational complexity with high memory efficiency, based on the properties of a star topology, even though it uses a self-attention mechanism.
4. This is a graph convolutional method that is more similar to conventional CNNs than most existing spectral methods.

2 Related work

Since the success of CNNs in areas such as computer vision, natural language processing, and speech processing, researchers began to generalize CNNs to
the graph domain. The key to generalizing CNNs to graphs is to define a convolution operator on graphs [1]. Two broad methods, spectral and spatial, categorize current graph convolutional neural networks.

Spectral methods use the graph Fourier transform [19] to transfer signals from the spatial domain to the spectral domain and perform convolution on the spectral domain. Some spectral methods have been applied to node classification. Spectral convolutional neural network (Spectral CNN) [6] introduces the graph Fourier transform directly and uses spectral convolution for graph signals. However, the number of learnable parameters for the filter is large, potentially causing severe computational costs [4]. The Chebyshev network (ChebyNet) [7] restricts the kernel of Spectral CNN to a polynomial expansion. Graph convolutional network (GCN) [8] further simplifies ChebyNet to avoid the Fourier transform by reducing the computational cost of the eigen decomposition of the graph Laplacian matrix [1]. Graph wavelet neural network (GWNN) [1] introduces a graph wavelet transform to replace the Fourier basis of Spectral CNN to achieve sparse and localized filters while maintaining a good computational performance.

All these methods are difficult to scale to large graphs and train in a mini-batch setting [10]. To scale spectral methods to large graphs, versions of GCN [11–15] have been designed for efficient minibatch training, but they are based on the whole training graph [10] and iteratively sample nodes or edges from the whole graph to form the minibatches in each layer [10]. This may cause “neighbor explosion” and lead to a large computational complexity. Some heuristic based methods, which make subgraph sampling as a preprocessing step [16, 17], attempt to solve this, but may introduce non-identical node sampling probabilities and bias. To cancel the bias, a graph sampling based inductive learning method, (GraphSAINT) [10], developed a normalization technique so that feature learning does not give larger weights to nodes which are sampled more frequently. These spectral methods still have a problem with generalization: they have a fixed graph size constraint (not graph-scale free). The filter size of spectral methods is determined by the size of the full or sampled subgraph and larger sizes will result in large computational and memory costs. If the size changes, the model will need to be reconstructed and retrained.

Spatial methods define the convolution on graph geometry. Mixture model CNN (MoNet) uses a weighted average of functions defined over the neighborhood of a node as the spatial convolution operator to provide a general framework for the design of spatial methods [20]. The approach, Graphs with generative adversarial nets (GraphSGAN) [21], facilitates generalization of generative adversarial nets (GANs) to the graph domain through low-density areas by generating fake samples between subgraphs to improve the performance of semi-supervised learning on graphs. Some spatial methods focus on improving model capacity by introducing an attention mechanism to the graph domain, such as the Graph attention network (GAT), which adopts a self-attention mechanism to learn the weighting function [4]. Developments of GAT, such as Dual-primal graph convolutional network (DPGCN) [22]
generalized GAT by using convolutions on nodes and edges, giving a better performance, Temporal graph attention network (TempGAN) learns node representations from continuous-time temporal graphs [23], and Hyperbolic graph attention network learns robust node representations of graphs in hyperbolic spaces [24]. The graph sample and aggregate method (GraphSage) [18], a node-based spatial method, learns node, rather than graph, embeddings so it is graph-scale free and can be applied to large or evolving graphs. It performs a uniform node sampling, with a predefined sampling size, to neighbors in each layer iteratively. The sampling size gives an upper bound to the minibatch computational complexity. Unlike STC, proposed here, its aggregator is not learnable. A long-short term memory (LSTM) [25] version has been proposed, but is inherently symmetric.

3 Methods

3.1 Preliminary

Given an undirected graph $G = \{V, E, A\}$, where $V = (V_1, V_2, \ldots, V_n)$ is a node set ($|V| = n$), $E$ is an edge set, and $A$ is an adjacency matrix ($A_{i,j} = A_{j,i}$); its graph Laplacian matrix $L$ can be defined as $L = D - A$, where $D = \text{diag}(\sum_{i \neq j} A_{i,j})$ is a degree matrix, and $L$ is a symmetric positive-semidefinite matrix. The eigen decomposition of $L$ is:

$$L = U \Lambda U^T,$$

where, $U = (u_1, u_2, \ldots, u_n)$ are the eigenvectors, which are orthonormal, and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is the diagonal matrix of the corresponding eigenvalues, which are real and non-negative and can be interpreted as the frequencies of the graph. These eigenvectors compose the basis of the feature space in the spectral domain.

3.2 Spectral convolution

For a signal $F = (F_1, F_2, \ldots, F_n)$ on the nodes of graph $G$, its graph Fourier transform is defined as:

$$\hat{F} = U^T F,$$

(2)

Given another signal $g$, the convolution of $g$ and $F$ can be defined as:

$$F \ast g = U g_\theta U^T F,$$

(3)

where, $g_\theta$ is the graph Fourier transform of $g$. Equation (3) is the spectral convolution, which is similar to the convolution theorem defined in Euclidean feature space, and $g_\theta$ can be regarded as the convolution filter which provides a set of kernel functions.
3.3 Inductive spectral convolution

To apply spectral convolution on large or evolving graphs, we need to introduce an inductive version which can learn the local and global structural properties of each node. Inductive methods focus on obtaining inductive node embedding, rather than whole graph embedding, which provides good flexibility. Hence, the inductive spectral convolution can be defined as a node-based subgraph spectral convolution by:

$$F_{[V_i,:]} \star g = \sum_{V_j \in V_i} W \hat{F}_{[V_i,:]}$$

(4)

where, $W$ is a flexible filter for subgraphs with different topology, and $V_i$ is the set of directly connected neighbors of $V_i$. The key to this work is to find a universal $W$ to make Equation (4) hold.

3.4 Properties of a star topology graph

$W$ is related to the topology of the subgraphs. A good $W$ should maintain the weight sharing property and, at the same time, provide different kernels for different structures [26]. We need to find a common structure, lying in different subgraphs, which should be identical, or at least symmetric, to help us design filters as in a conventional CNN. We find that star topology graphs with different sizes, $n$, are symmetric in their structure and Laplacian matrix, as Figures 1 & 2 show. All graphs can be regarded as a composition of subgraphs with a star topology (Figure 2). The eigen decomposition of an $n$-dimensional star topology ($n$ neighboring nodes and one central node, $n \geq 1$) can use a universal formulation:

$$L = \begin{bmatrix} n & -1 & \cdots & -1 \\ -1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & 0 & \cdots & 1 \end{bmatrix} = U \Lambda U^T,$$

(5)

where, all elements on the diagonal of $\Lambda$ are 1, except the first and last elements. That is all eigenvectors have the same weight, except the first and last ones. The largest eigenvalue corresponds to the central node and its connections. All of the value 1 eigenvalues correspond to neighboring nodes and their connections. Since the rank of $L \in \mathbb{R}^{(n+1) \times (n+1)}$ is $n$, the last eigenvalue is 0. The eigenvalues in Equation (5) can be obtained by:

$$LU = \Lambda U,$$

(6)
Fig. 1  The structural similarity of graphs with a star topology. The red node is the central node.

Fig. 2  Subgraphs with a star topology exist in different graphs. The red node is the central node.

\[ LU = \Lambda I U, \quad (7) \]

\[ (\Lambda I - L) U = 0, \quad (8) \]

where, \( I \) is an identical matrix. According to the theorem of linear systems of equations, Equation (8) having non-zero solutions is equivalent to \( \det(\Lambda I - L) = 0 \). Using Gaussian elimination on \( \det(\Lambda I - L) = 0 \),

\[ (\lambda - n - 1)(\lambda - 1)^{n-1} \lambda = 0, \quad (9) \]

the solutions of Equation (9) are \( \Lambda \).

Eigenvectors \( U = (u_1, u_2, \ldots, u_{n+1}) \) are orthonormal. Different size star topology graphs have common parts in their eigenvectors for which the corresponding eigenvalue is 1. For an \( m \)-dim star topology graph \( G_m \in \mathbb{R}^{(m+1) \times (m+1)} \) and an \( n \)-dim star topology graph \( G_n \in \mathbb{R}^{(n+1) \times (n+1)} \), where \( m \leq n \), the eigenvectors corresponding to eigenvalues of 1 in \( G_m \) can be expressed by any \( m - 1 \) eigenvectors with corresponding eigenvalues of 1 in \( G_n \).
using the vector rotation formula:

\[ U_{G_m[2:m]} = RU_{G_n[\text{ind}]}, \quad \text{ind} = \text{randperm}(\{2 : n\}, m - 1), \quad (10) \]

where, \( R \) is an identical rotation matrix, and \( \text{ind} \) is the mask of \( m-1 \) randomly selected indices. Hence, these eigenvectors can be regarded as equivalent and are suitable to be the universal basis for different scales of subgraphs.

Referring to Equation (3), the first row of \( g_\theta U^T \) corresponds to the first row of the new feature matrix \( g_\theta U^T F \) and also to the first row of \( F \). The first row of \( F \) corresponds to the central node as in \( L \). Since we want to obtain information on neighbors of the central node, the first row of \( F \) can be padded to zero. The last eigenvalue is zero which means that the last eigenvector is the least important. By manually adding an artificial node to the original subgraph, we get a new \( L \in \mathbb{R}^{(n+2)\times(n+2)} \). Adding a zero row under the last row of \( F \) won’t affect the non-zero rows, so any number of zero rows in \( F \) can be added if needed. Then, we can design the flexible filter for \( \{1, 2, \ldots, k\} \)-dimensional \((k \text{ can be any non-zero positive integer})\) star topology subgraphs as:

\[ W = U\Theta, \quad W \in \mathbb{R}^{(k+2)\times(k+2)}, \quad (11) \]

\[ \Theta = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \theta_1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \theta_k & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}, \quad (12) \]

where, \( \Theta \) is a diagonal matrix which has \( k \) learnable parameters. The filter can provide different kernels on different star topology subgraphs and its weight sharing property is shown in Figure 3.

Decomposition of a graph into star topology subgraphs will lose some link information as Figure 4 shows. In the first convolution layer, the edges \((V_1, V_2)\) and \((V_3, V_4)\) are lost. This can be compensated by using more layers to aggregate higher order neighboring nodes. As Figure 4 shows, through introducing the second convolution layer, the lost link information \(((V_1, V_2)\) and \((V_3, V_4))\) is compensated. Hence, using more layers can reduce the link information loss. But also as Figure 4 shows, some neighboring nodes will be sampled repeatedly with more layers introduced. Hence, there is a trade-off between link information loss and repeatedly sampling problem. How to select a proper layer depth to reduce the link information loss and at the same time make the effect of repeatedly sampling not too big will be a problem, we will discuss it in the subsection 4.3.

### 3.5 Star topology convolution

The spectral convolution defined on a star topology can be obtained using the properties of the topology. The update function for the \( l + 1 \)-th layer’s
The Θ of a flexible filter \( W \) with a size 7 Graph \( G \)

(a) (b) (c) (d) (e) (f)

**Fig. 3** A toy example showing the weight sharing property of the filter on subgraphs with different star topology. The red node is the central node and the yellow nodes are neighboring nodes to be aggregated. The kernels provided by the filter differ by subgraph. It is a general filter which can deal with different subgraphs, that is, the filter is graph-scale free.

neighboring information is defined as:

\[
h^{l+1}_{[V_i,z]} = \sum_{x=1}^{p} W^{l+1}_{x,z} U^{T}_y h^l_{[V_i,x]}, \quad (z = 1, 2, \ldots, q),
\]  

(13)

where, \( y^l_{[V_i,x]} \) is the output of the \( l \)-th layer for neighboring nodes set \( V_i \), \( x \) and \( z \) are the dimension indices, \( p \) and \( q \) are the dimensions of the embedding, and \( V_i \) is the set of directly connected neighbors of the central node \( V_i \). To reduce the computational complexity of \( h^{l+1} \), we first perform spectral convolution:

\[
h^{l+1}_{[V_i,x]} = W^{l+1}_x U^{T}_x y^l_{[V_i,x]}, \quad (x = 1, 2, \ldots, p),
\]  

(14)

Information on the neighbors obtained from the spectral convolution will be aggregated to the central node using feature transformation.

\[
h^{l+1}_{[V_i,:]} = \sum_{V_j \in V_i} b^{l+1}_{V_j} h^{l+1}_{[V_j,:]},
\]  

(15)

where, \( b^{l+1}_{V_j} \) denotes the learnable weight of edge \((V_i, V_j)\) in the \( l + 1 \)-th layer, \( h^{l+1}_{[V_i,:]} \) is the convolution result of the neighboring information of the central node \( V_i \). Then \( h^{l+1}_{[V_i,:]} \) will be concatenated with the last layer output feature vector \( y^l_{[V_i,:]} \) of the central node \( V_i \). After a scaling transformation, the new feature vector will be activated using a nonlinear function as the current layer.
output of node $V_i$ as:

$$y_{[V_i,:]}^{l+1} = \sigma\left(\text{cat}\left(y_{[V_i,:]}^{l}, h_{[V_i,:]}^{l+1}\right) S^{l+1}\right),$$

(16)

where, $y_{[V_i,:]}^{l}$ is the feature vector of the central node $V_i$ obtained in $l$-th layer, $\text{cat}()$ denotes a concatenation function, $\sigma$ denotes a nonlinear activation function, and $S^{l+1} \in \mathbb{R}^{2p \times q}$ is a scaling parameter matrix for feature transformation.

Figure 5 shows the comparison of a two-layer conventional CNN and a two-layer STC. The two-layer STC is formulated as:

$$1^{st} \text{ layer} : y_{[V_i,:]}^{1} = \text{ReLU}\left(\text{cat}\left(F_{[V_i,:]}, h_{[V_i,:]}^{1}\right) S^{1}\right),$$

(17)

$$2^{nd} \text{ layer} : y_{[V_i,:]}^{2} = \text{ReLU}\left(\text{cat}\left(y_{[V_i,:]}^{1}, h_{[V_i,:]}^{2}\right) S^{2}\right),$$

where, $S^{1} \in \mathbb{R}^{2p \times q}$ and $S^{2} \in \mathbb{R}^{2q \times d}$ are two scaling parameter matrices, and $d$ is the embedding dimension. STC divides the convolution process of a conventional CNN into two cascading steps: (1) spatial search; (2) spectral convolution. Spatial search is used to find spatially localized areas. Hence, our method can directly create spatially localized filters. As with a conventional CNN, STC obtains global information through the introduction of more layers.
An undirected graph with 11 nodes. Among them, the central node needs to be learned the embedding.

Find directly first and second order connected neighbors of the central node.

STC Layer 1:
Obtain $y[:, :]$ (see Eq. 16)

Step 2: Spectral Convolution
(1) Obtain $h_{V,C}$ (see Eq. 15)
(2) STC Layer 1:
Obtain $y_{V,C}$ (see Eq. 16)

Convolution
Spatial search
Central node
Selected node
Un-selected node
Undirected edge
Selected central node

Step 2: Spectral Convolution
(3) Obtain $h_{V,C}$ (see Eq. 15)
(4) STC Layer 2:
Obtain $y_{V,C}$ (see Eq. 16)

After the spatial search, a spectral convolution on the subgraphs is obtained. In a conventional CNN, the shape of different areas selected by a window function/kernel is usually the same. In a 3*3 kernel, for example, the different areas selected are all square and have the same size (9 pixels). In a graph, star topology subgraphs have symmetry in their spatial structures which allows STC to have a convolution process, which can share weights across different graphs, similar to a conventional CNN. If a new filter is used to replace $W$, 

Fig. 5 Comparison of a conventional CNN and a star topology convolution (STC). In the convolution process, a CNN uses a grid filter to select spatially localized areas from the training image for convolution. Similarly, STC uses a universal filter $W$ to select spatially localized star topology subgraphs from the training graph for convolution. It adopts a spatial search, similar to GraphSage [18], to find neighbors. The depth of the neural network determines the order of neighbors used in the convolution. Shown is an example of a two-layer STC, hence only first and second order neighbors are selected. Spectral convolution is performed on the selected subgraphs.
a new inductive spectral convolutional method can be designed. Hence, STC provides a framework to design inductive spectral convolutional methods.

### 3.6 Node sampling

Since the filter size $k$ is unknown, it must be confirmed in an implementation. This can be done simply by assigning it as $K$, where $K$ is the maximum number of connections of a node in the graph. The node sampling strategy of Graph-Sage [18], giving a dropout function [27], is used to improve the robustness of STC. For a node $V_i$ with $|V_i|$ directly connected neighbors, when $|V_i| > k$, $k$ neighbors will be randomly selected in each training iteration.

$$y_i^l = y_{[\text{ind}]}^l, \text{ ind} = \text{randperm}(V_i, k),$$  \hspace{1cm} (18)

where, $\text{ind}$ is the mask, with its size determined by the size of the filter, $k$. For nodes with $|V_i| \leq k$ directly connected neighbors, all neighbors will be selected, and these nodes can be considered to not have enough information. Hence, it is necessary to take advantage of all their information. Through controlling the filter size, STC can achieve good robustness. The effect of the filter size and how to select it will be discussed in the ablation study in subsection 4.5.

### 3.7 Edge attention

We introduce a self-attention mechanism (edge attention) [4] to STC to further improve its performance and robustness. As mentioned in subsection 3.5, STC provides a framework to design different inductive spectral convolution methods by designing different $W$. Here we use edge attention to get the weights $\theta$ in $W$ ($W = U\Theta$) as Figure 6 shows. For a node $V_i$, the weights $\theta$ of the $l+1$-th layer in the filter for each neighboring node of it can be calculated as follows,

$$\theta_{V_j}^{l+1} = \exp \left( \sigma \left( r_{[V_j, :]}^l a_{l+1} \right) \right), \quad V_j \in V_i,$$

$$\sum_{V_j \in V_i} \exp \left( \sigma \left( r_{[V_j, :]}^l a_{l+1} \right) \right),$$  \hspace{1cm} (20)

where, $C^l \in \mathbb{R}^{p \times q}$ is a scaling parameter matrix and $a_{l+1} \in \mathbb{R}^{2q+1}$ is a scaling parameter vector.

### 3.8 Comparison with state-of-the-art methods

In vanilla GCN, the $l+1$-th layer of GCN can be defined as:

$$Y^{l+1} = \sigma(A Y^l S^{l+1}),$$  \hspace{1cm} (21)

where $Y^{l+1}$ is the hidden unit output of the $l+1$-th layer, $A$ is the adjacency matrix which is used to introduce the edge information to the learning process.
And the scaling parameter matrix $S^{l+1}$ is acting as the convolution filter, in the $l + 1$-th layer. Compared to GCN, STC uses a spectral filter $W^{l+1}$ to replace $A$. Unlike $A$, $W^{l+1}$ implicitly includes the edge information and STC has an edge attention mechanism.

For the commonly used method GraphSage (mean aggregator), its $l + 1$-th aggregator can be defined as:

$$h_{[V_i,:]}^{l+1} = \text{mean}_{V_j \in V_i} \left( y_j^l \right).$$

The embedding of the central node, $V_i$, in the $l + 1$-th layer of GraphSage can be obtained using Equation (16). GraphSage sets all weights to 1 to replace the spectral convolution filter in STC, which will lose some information in the learning process. GraphSage uses a simple mean to aggregate neighboring information to the central node $V_i$, while STC uses a learnable weighted average function.

GAT performs edge attention on the whole graph by:

$$Y_{l+1} = \sigma \left( \Omega^{l+1} Y^l S^{l+1} \right),$$

where, $\Omega^{l+1}$ is the attention weight matrix. STC performs edge attention on the star topology subgraphs, making STC more suitable than GAT for large graphs.

### 3.9 Star topology convolution aggregator for neighbor averaging over relation subgraphs

For large heterogeneous graph representation learning, Neighbor averaging over relation subgraphs (NARS) [28] and its variants have achieved leading performance. The vanilla NARS uses relational graph embeddings [29] as the features for featureless node types and then combines different relation types.
randomly to construct \( k \) (\( k \) can be seen as the filter size in STC) relation subgraphs. For a central node, in each relation subgraph of it, NARS computes its \( l \)-th hop (\( l \)-th order, \( l \) can be regarded as the layer depth like STC) neighbor averaging features using Equation (22). And then, NARS uses a simple weighted aggregator to aggregate these \( l \) hops neighbor averaging features from \( k \) relation subgraphs into a single set of features as the new features of the central node. Finally, a SIGN [30] classifier will be used to do node classification based on the new features. The experimental results show that NARS outperforms state-of-the-art heterogeneous methods on large heterogeneous graph benchmarks. In this paper, we propose an improved version of NARS called STC-NARS which uses STC as the aggregator instead of the simple weighted aggregator. For \( l \)-th hop neighbor averaging features of the relation subgraph set \( \text{Rel}_{[V_i,:]} \) of the central node \( V_i \), the aggregated features of all relation subgraphs in \( \text{Rel}_{[V_i,:]} \) can be computed as follows,

\[
h^{l}_{[V_i,:]} = \sum_{\text{Rel}_{[V_i,:]} \in \text{Rel}_{[V_i,:]}} b^{l}_{\text{Rel}_{[V_i,:]}; h^{l}_{[\text{Rel}_{[V_i,:]},:]}}.
\]

where, \( y^{l}_{[\text{Rel}_{[V_i,:]},:]} \) denotes the \( l \)-th hop neighbor averaging features of all relation subgraphs in \( \text{Rel}_{[V_i,:]} \).

3.10 Variants

We propose three variants of STC. They are named STC, STC (conv only), and STC-NARS, respectively. The differences of them are summarized as follows,

**STC** is based on Equation (16) and uses Equation (20) to replace the \( \Theta \) of Equation (12).

**STC (conv only) or vanilla STC** is based on Equation (16) to perform convolution only.

**STC-NARS** is based on Equations (24-25) to do graph representation learning on large heterogeneous graphs.

3.11 Complexity analysis

The computational complexity of STC can be considered in two parts: (1) the convolution process; (2) the edge attention process. The convolution process of STC depends on the filter size, the size of the scaling parameter matrix, and the size of the learnable weight \( b \), so its computational complexity in the \( l+1 \)-th layer is \( 2p \times q + 2k \), where \( 2p \times q \) is the size of scaling parameter matrix and \( k \) is the filter size. \( k \) is much smaller than \( p \times q \) in most cases. Compared to other spectral methods, like Spectral CNN [6] (with a complexity of \( p \times q \times |V| \), where \( |V| \) is the total number of nodes in the full graph) and GWNN [1] (with a complexity of \( p \times q + |V| \)), the complexity of STC is much smaller allowing it to be used for large or evolving graphs. STC performs edge attention on
star topology subgraphs. If the filter size is \( k \), then the maximum sampling number of directly connected neighbors of a star topology subgraph is \( k \), so its computational complexity is \( q \ast (p + 2) + k \). The computational complexity of GAT is \( q \ast (p + 2) + (|E| + |V|) \), where \(|E|\) is the total number of unique edges in the full graph. \( k \) is much smaller than \(|E| + |V|\).

4 Experiments and results

4.1 Experiment settings

**Benchmarks:** Six homogeneous graph datasets: Cora [31], Citeseer [31], Pubmed [31], PPI [18, 32], Arxiv [33], and Essential Proteins (EP) were selected to validate the ability of STC to predict the properties of nodes on homogeneous graphs. The statistics of these datasets can be seen in Table 1. What’s more, we also selected four heterogeneous graph datasets: MAG [33], ACM [34], DBLP [34], and IMDB [34] to further validate the performance of STC. Table 2 shows the statistics of these datasets. Cora, Citeseer, Pubmed, Arxiv, ACM, and MAG are six citation datasets. Arxiv and MAG are two large datasets. PPI is a dataset used to test the ability of graph convolutional methods to generalize across graphs. DBLP is a bibliography dataset of computer science and IMDB is a dataset about movies. We also collected an essential protein dataset (EP) of Yeast to test STC. Yeast was selected because its protein-protein interaction data and gene essentiality data are the most complete and reliable among various species. Essential protein identification can be regarded as a binary node classification task. The protein-protein interaction data of Yeast in EP were downloaded from the DIP database [35] updated to Oct.10, 2010. There are 4512 proteins and 22091 protein-protein interactions, excluding self and repeated interactions, in this dataset. The essential protein list in EP was selected from MIPS [36], SGD [37], DEG [38], and SGDP [39]. There are 1285 essential proteins in the list, 1031 of which are in the protein-protein interaction network. These 1031 proteins were considered essential and the other 3481 proteins were considered non-essential. The gene expression data used as the features were collected from GEO [40], sampling 36 time points during successive Yeast metabolic cycles. The division of training/validation/test sets for Cora, Citeseer, and Pubmed was made according to [41]; for PPI according to [18]; for Arxiv and MAG according to [33]; for ACM, DBLP, and IMDB according to [34]; for EP it was set to 3012:500:1000.

**Table 1** The statistics of six homogeneous benchmark datasets.

|                  | Citeseer | Cora | Pubmed | EP   | Arxiv | PPI   |
|------------------|----------|------|--------|------|-------|-------|
| Nodes            | 3327     | 2708 | 19717  | 4512 | 169343| 56944 |
| Edges            | 4732     | 5429 | 44338  | 22091| 1166243| 818716|
| Classes          | 6        | 7    | 3      | 2    | 40    | 121   |
| Features         | 3703     | 1433 | 500    | 36   | 128   | 50    |
| Training nodes   | 1827     | 1208 | 18217  | 3012 | 90941 | 44906 |
| Validation nodes | 500      | 500  | 500    | 500  | 29799 | 6514  |
| Test nodes       | 1000     | 1000 | 1000   | 1000 | 48603 | 5524  |
Star Topology Convolution for Graph Representation Learning

Table 2  The statistics of four heterogeneous benchmark datasets.

|         | ACM   | DBLP  | IMDB  | MAG   |
|---------|-------|-------|-------|-------|
| Nodes   | 10942 | 26128 | 21420 | 1939743 |
| Edges   | 547872 | 239566 | 86642 | 21111007 |
| Classes | 3     | 4     | 5     | 349   |
| Features| 1902  | 334   | 3489  | 128   |
| Training nodes | 726     | 974     | 1097   | 629571 |
| Validation nodes | 181     | 243     | 274    | 64879  |
| Test nodes   | 2118   | 2840   | 3202   | 41939  |

Baselines: To test STC, it was compared with several state-of-the-art methods: GWNN [1], GAT (sparse) [4], GCN [8], GraphSAINT (mean aggr) [10], GraphSAINT (concat aggr) [10], GraphSage (mean aggr) [18], MoNet [20], Plain Linear + C&S [42], GraphZoom (Node2vec) [43], Node2vec [44], Label Propagation [45], MLP [33], MLP+FLAG [46], CoLinkDistMLP [47], MetaPath2vec [48], R-GCN [49], SIGN [30], GraphSAINT (R-GCN aggr) [33], GraphSAINT+MetaPath2vec [50], ClusterGCN (R-GCN aggr) [33], NeighborSampling (R-GCN aggr) [33], NARS [28], R-GCN+FLAG [46], HAN [51], GTN [52], RSHN [53], HetGNN [54], MAGNN [55], and HetSANN [56]. And we also compared STC with several state-of-the-art essential protein identifica-
tion methods: deep neural network based method (DNN) [57], multi-objective optimization based method (IMAMOBH) [58], degree centrality (DC) [59], betweenness centrality (BC) [60], closeness centrality (CC) [61], eigenvector centrality (EC) [62], edge clustering coefficient centrality (NC) [63], and local average connectivity (LAC) [64] on EP. We followed the protocol of [33, 34] to run all experiments. All methods were used their default settings, run with enough epochs to converge, and used a validation process to select the model with the highest micro-F1 score on the validation set for testing in all experiments in this paper. The layer depth and hidden unit size of all methods used in this paper were set according to [33].

The optimizer for STC was Adam [65] and the learning rate was set to 0.001 for all datasets except EP (0.0001) in this paper. STC adopted a fixed number for the filter size (18) for all datasets, except Arxiv, Citeseer (Arxiv and Citeseer were used to do the ablation study of filter size) and PPI (48), and a fixed dropout rate (0.5) for all datasets, except Citeseer (0.6), Cora (0.6), PPI (0.1), MAG (0.2), and EP (0). The settings of two variants are the same as STC. We also introduced three ablation studies to clarify the effect of layer depth, the effect of scaling parameter matrix in attention process, and the effect of filter size. What’s more, since the similarity between STC and CNN, a study about applying transfer learning [66, 67] in STC has been introduced to investigate some potential applications of some CNN techniques in STC. To do node classification, a single fully connected layer was used to classify the results from two convolution layers of STC or STC (conv only).

All experiments except three ablation studies and GPU memory cost comparison were conducted on a personal computer (PC1) with Ubuntu OS 18.04, Intel (R) Xeon (R) Silver 4210 2.20 GHz 10-Core CPU, 188 GB RAM, CUDA.
version 10.2, torch version 1.8.1, and 4 NVIDIA TITAN RTX GPUs. Other experiments were run on a personal computer (PC2) with Windows OS 10 Home, AMD Ryzen 5 3600 3.59 Ghz 6-Core CPU, 16 GB RAM, CUDA version 10.2, torch version 1.8.1, and 1 NVIDIA GeForce GTX 1660 Super GPU.

4.2 Performance analysis

**Small datasets:** Tables 3-5 show the test accuracy (micro-F1) obtained by STC and state-of-the-art graph convolutional methods on small citation datasets. STC achieved the highest test accuracy (micro-F1) on Citeseer. STC also outperformed the other methods on Pubmed. GWNN achieved the highest test accuracy (micro-F1) on Cora with the performance of STC just slightly inferior. STC achieved state-of-the-art performance on small datasets. Tables 3-5 also show the number of trainable parameters for each method on these small citation datasets. Full-batch methods: GAT (sparse), GCN, and GWNN have fewer parameters than minibatch methods: STC, STC (conv only), GraphSage (mean aggr), MoNet, GraphSAINT (mean aggr), and GraphSAINT (concat aggr). But full-batch methods are difficult to scale to large graphs. What’s more, STC has fewer parameters and better performance than other minibatch methods. Compared to STC (Conv only), STC outperformed it significantly on these small citation datasets with using less parameters, which shows that the introducing of some techniques to improve the filter of vanilla STC is available.

| Method                  | Citeseeer | Param. |
|-------------------------|-----------|--------|
| STC†                    | 76.60±0.50| 1.15M  |
| STC (conv only)         | 74.55±0.83| 2.03M  |
| GraphSage (mean aggr)   | 75.23±0.60| 2.03M  |
| MoNet                   | 69.55±1.36| 2.85M  |
| GAT (sparse)            | 74.80±0.00| 0.95M  |
| GCN                     | 75.50±0.00| 0.95M  |
| GWNN                    | 74.93±0.50| 0.96M  |
| GraphSAINT (mean aggr)  | 69.52±0.28| 2.10M  |
| GraphSAINT (concat aggr)| 69.27±1.37| 2.29M  |

**Essential protein identification:** Table 6 shows the performance of STC and state-of-the-art methods on an essential protein identification task. STC and STC (conv only) outperformed other graph convolutional methods and essential protein identification methods. Graph convolutional methods outperformed traditional essential protein identification methods, even DNN, showing that graph convolutional methods have an advantage in solving traditional graph problems. On this dataset, STC outperformed other methods significantly. STC has more parameters, but a comparable number, than other
methods. Compared to STC (conv only), STC has more parameters on this dataset, but STC outperformed STC (conv only) significantly.

**Medium-sized datasets:** Table 7 shows the test accuracy (micro-F1) obtained by STC and state-of-the-art graph convolutional methods on a medium-sized dataset. Although STC has more parameters than other methods on this dataset, STC achieved substantially better performance than other methods including STC (conv only). It further proves that the introducing of some techniques to improve the filter of vanilla STC is available.

**Large datasets:** Table 8 compares STC and STC (conv only) with some leading methods on Arxiv. STC outperformed the other methods and STC (conv only) was close to STC. Compared to some methods like GraphZoom (Node2vec) and Node2vec, STC and STC (conv only) have fewer parameters. Although STC has more parameters, but a comparable number, than GCN at the same level. All in all, STC has achieved state-of-the-art performance on large homogeneous datasets.

**Small heterogeneous datasets:** To further show the superiority of STC, we applied STC on three small heterogeneous graphs: ACM, IMDB, and DBLP. Table 9 shows the performance comparison of STC and leading methods on ACM, IMDB, and DBLP. STC outperformed GCN and GAT significantly.

---

### Table 4
Comparison of test accuracy (micro-F1) and number of trainable parameters of STC and state-of-the-art methods on Cora. Best values are in bold, second best values are in italic. Note: STC† denotes STC using an untrainable constant scaling parameter matrix in attention process.

|                      | Cora   | Param. |
|----------------------|--------|--------|
| STC†                 | 87.43±0.22 | 0.57M  |
| STC (conv only)      | 86.50±0.54 | 0.87M  |
| GraphSage (mean aggr)| 86.08±0.44 | 0.87M  |
| MoNet                | 80.50±0.78 | 1.11M  |
| GAT (sparse)         | 86.20±0.00 | 0.37M  |
| GCN                  | 86.20±0.00 | 0.37M  |
| GWNN                 | 87.85±0.29 | 0.37M  |
| GraphSAINT (mean aggr)| 81.98±0.67 | 0.94M  |
| GraphSAINT (concat aggr)| 81.52±1.11 | 1.13M  |

### Table 5
Comparison of test accuracy (micro-F1) and number of trainable parameters of STC and state-of-the-art methods on Pubmed. Best values are in bold, second best values are in italic. Note: STC† denotes STC using an untrainable constant scaling parameter matrix in attention process.

|               | Pubmed       | Param. |
|---------------|--------------|--------|
| STC†          | 91.20±0.30   | 0.33M  |
| STC (conv only)| 90.77±0.38  | 0.39M  |
| GraphSage (mean aggr)| 89.95±0.34 | 0.39M  |
| MoNet         | 84.80±0.62   | 0.39M  |
| GAT (sparse)  | 87.50±0.00   | 0.13M  |
| GCN           | 89.00±0.00   | 0.13M  |
| GWNN          | 83.85±0.26   | 0.17M  |
| GraphSAINT (mean aggr)| 88.17±0.42 | 0.46M  |
| GraphSAINT (concat aggr)| 87.80±0.47 | 0.65M  |
Table 6 Comparison of test accuracy (micro-F1) and number of trainable parameters of STC and state-of-the-art methods in an essential protein identification task. Best values are in bold, second best values are in italic.

| Method                  | EP       | Param. |
|-------------------------|----------|--------|
| STC                     | 78.98±0.23 | 0.28M  |
| STC (conv only)         | 77.58±0.31 | 0.15M  |
| GCN                     | 77.50±0.00 | 0.01M  |
| GAT (sparse)            | 77.30±0.00 | 0.01M  |
| GraphSage (mean aggr)   | 77.10±0.39 | 0.15M  |
| MoNet                   | 76.27±0.92 | 0.03M  |
| DNN                     | 75.80±0.00 | 0.0003M|
| LAC                     | 75.63±0.00 | 0      |
| IMAMOBH                 | 75.33±0.13 | 0      |
| NC                      | 74.69±0.00 | 0      |
| DC                      | 73.36±0.00 | 0      |
| EC                      | 71.94±0.00 | 0      |
| CC                      | 71.51±0.00 | 0      |
| BC                      | 71.39±0.00 | 0      |

Table 7 Comparison of test accuracy (micro-F1) and number of trainable parameters of STC and state-of-the-art methods on PPI. Best values are in bold.

| Method                  | PPI      | Param. |
|-------------------------|----------|--------|
| STC                     | 98.06±0.10 | 0.29M  |
| STC (conv only)         | 69.16±0.17 | 0.19M  |
| GraphSage (mean aggr)   | 74.55±0.41 | 0.19M  |
| GCN                     | 64.69±0.00 | 0.04M  |
| GraphSAINT (mean aggr)  | 66.51±0.25 | 0.26M  |
| GraphSAINT (concat aggr)| 67.64±0.33 | 0.45M  |

Even compared with some leading heterogeneous methods on ACM, IMDB, and DBLP, STC still achieved the best overall performance which shows that STC reduces the performance gap between heterogeneous methods and homogeneous methods on small heterogeneous graphs.

**Large heterogeneous datasets:** To verify whether STC can reduce the performance gap between homogeneous methods and heterogeneous methods on large heterogeneous graphs, we applied STC on MAG. Table 10 shows the comparison of STC and some leading methods on MAG. Our STC outperformed other homogeneous graph methods significantly and also outperformed some basic heterogeneous methods like SIGN, R-GCN, and MetaPath2vec significantly. These methods are the backbone of some leading methods like NARS (based on SIGN), GraphSAINT+MetaPath2vec, GraphSAINT (R-GCN aggr), R-GCN+FLAG, and NeighborSampling (R-GCN aggr), etc. on large heterogeneous graphs. To achieve the state-of-the-art performance on large heterogeneous graphs, we also proposed an improved version of NARS called STC-NARS which uses STC as the aggregator. STC-NARS outperformed the vanilla NARS with only using half of parameters. And the number of its parameters is also the least among all heterogeneous methods in Table 10. Through introducing STC structure, the improved NARS framework
Table 8 Comparison of test accuracy (micro-F1) and number of trainable parameters of STC and state-of-the-art methods on Arxiv. Best values are in bold, second best values are in italic.

| Methods                      | Arxiv     | Param. |
|------------------------------|-----------|--------|
| STC                          | 71.85±0.04| 0.34M  |
| STC (conv only)              | 71.45±0.08| 0.21M  |
| GCN                          | 71.74±0.29| 0.11M  |
| GraphSage (mean aggr)        | 71.49±0.27| 0.22M  |
| Plain Linear + C&S           | 71.26±0.01| 0.01M  |
| GraphZoom (Node2vec)         | 71.18±0.18| 8.96M  |
| Node2vec                     | 70.07±0.13| 21.82M |
| Label Propagation            | 68.32±0.00| 0      |
| GraphSAINT (mean aggr)       | 58.86±0.51| 0.28M  |
| GraphSAINT (concat aggr)     | 58.11±0.71| 0.47M  |
| CoLinkDistMLP                | 56.38±0.16| 0.12M  |
| MLP+FLAG                     | 56.02±0.19| 0.11M  |
| MLP                          | 55.50±0.23| 0.11M  |

Table 9 Comparison of test accuracy (micro-F1) of STC and leading methods on ACM, IMDB, and DBLP. Best values are in bold.

| Methods           | ACM         | IMDB         | DBLP         |
|-------------------|-------------|--------------|--------------|
| Homogeneous methods |             |              |              |
| STC               | 93.03±0.37  | 66.60±0.32   | 93.44±0.09   |
| GCN               | 92.12±0.23  | 64.82±0.64   | 91.47±0.34   |
| GAT               | 92.19±0.93  | 64.86±0.43   | 93.39±0.30   |
| R-GCN             | 91.41±0.75  | 62.05±0.15   | 92.07±0.50   |
| HAN               | 90.79±0.43  | 64.63±0.58   | 92.05±0.62   |
| GTN               | 91.20±0.71  | 65.14±0.45   | 93.97±0.54   |
| RSHN              | 90.32±1.54  | 64.22±1.03   | 93.81±0.55   |
| HetGNN            | 86.05±0.25  | 51.16±0.65   | 92.33±0.41   |
| MAGNN             | 90.77±0.65  | 64.67±1.67   | 93.76±0.45   |
| HetSANN           | 89.91±0.37  | 57.68±0.44   | 80.56±1.50   |

can be used as a new backbone to design more powerful methods for large heterogeneous graphs.

In summary, STC achieved state-of-the-art performance on small, middle, large homogeneous graphs, and also heterogeneous graphs, demonstrating its robustness. Table 11 compares the ability of STC and STC (conv only) with state-of-the-art methods to be run on large datasets (Arxiv and MAG) using PC2 (one NVIDIA GeForce GTX 1660 Super GPU (6GB)). Compared with other minibatch methods: GraphSage (mean aggr), GraphSAINT (mean aggr), and GraphSAINT (concat aggr), our methods and GraphSage (mean aggr) can be trained on both datasets but GraphSAINT (mean aggr) and GraphSAINT (concat aggr) exceed available memory on MAG while two full-batch methods: GWNN and GAT (sparse) do this on both datasets. It shows that compared to full-batch methods, minibatch methods are more suitable for large datasets although in most cases, minibatch methods have more trainable parameters than full-batch methods as Tables 3-8, and 10 show. What’s more, GWNN
Table 10 Comparison of test accuracy (micro-F1) and number of trainable parameters of STC and state-of-the-art methods on MAG. Best values are in bold, second best values are in italic.

|                      | MAG                  | Param.   |
|----------------------|----------------------|----------|
| **Homogeneous methods** |                      |          |
| STC                  | 42.54±0.11           | 0.62M    |
| GraphSage (mean aggr)| 31.53±0.15           | 0.29M    |
| GCN                  | 30.43±0.25           | 0.12M    |
| CoLinkDistMLP        | 27.61±0.18           | 0.28M    |
| MLP                  | 26.92±0.26           | 0.19M    |
| **Heterogeneous methods** |                    |          |
| STC-NARS             | 52.20±0.04           | 2.30M    |
| NARS                 | 52.08±0.40           | 4.13M    |
| GraphSAINT+MetaPath2vec | 49.66±0.22         | 309.76M  |
| GraphSAINT (R-GCN aggr) | 47.51±0.22         | 154.37M  |
| R-GCN+FLAG           | 47.37±0.48           | 154.37M  |
| NeighborSampling (R-GCN aggr) | 46.78±0.67     | 154.37M  |
| SIGN                 | 40.46±0.12           | 3.72M    |
| R-GCN                | 39.77±0.46           | 154.37M  |
| ClusterGCN (R-GCN aggr) | 37.32±0.37         | 154.37M  |
| MetaPath2vec         | 35.44±0.36           | 94.48M   |

needs to store graph wavelets which requires $O(2*|\mathcal{V}|^2)$ memory. Take Arxiv for example, GWNN needs to store 57354M ($2*169343^2$) floating-point numbers. But the memory requirement of the storage of eigenvectors of our STC is only $O(2*k^2)$, for Arxiv, it only needs to store 648 ($2*18^2$) floating-point numbers. Besides, GAT (sparse) needs to store full graph pair features for edge attention which requires $O(|\mathcal{E}]*2P)$ memory, where $P$ is the size of node feature. While our STC does the edge attention on star topology subgraphs which only requires $O(k*2P*M)$, where $M$ is the minibatch size. Take Arxiv for example, GAT (sparse) needs to store 299M ($1166243*2*128$) floating-point numbers, while our STC only needs to store 2.30M ($18*2*128*500$) floating-point numbers (minibatch size: 500). It further shows the memory efficiency of STC on large graphs.

Table 11 Comparison of STC and state-of-the-art methods using one NVIDIA GeForce GTX 1660 Super GPU (6GB memory) on two large datasets.

|                | Arxiv   | MAG   |
|----------------|---------|-------|
| STC            | Available | Available |
| STC (conv only)| Available | Available |
| GraphSage (mean aggr) | Available | Available |
| GraphSAINT (mean aggr) | Available | Out of memory |
| GraphSAINT (concat aggr) | Available | Out of memory |
| GWNN           | Out of memory | Out of memory |
| GAT (sparse)   | Out of memory | Out of memory |
4.3 Effect of layer depth

To validate the effect of layer depth of STC on its performance, we trained STC with 1-4 layers on six homogeneous graphs and compared the test accuracy (micro-F1) of them respectively. As Figure 7 shows, with the increasing of the number of layers, the performance of STC were improved first and then reduced, while the parameter number increased steadily. Because with more layers introduced, link information loss will be reduced. But at the same time, repeatedly sampling problem arises. Hence, there is a trade-off here and to most problems layer depth of 2 will achieve the best performance as Figure 7 and Table 12 show. That’s why we select the layer depth of 2 to run all experiments.

Table 12 Comparison of test accuracy (micro-F1) of STC with using different number of layers on six homogeneous graphs. Best values are in bold.

|          | 1 layer | 2 layers | 3 layers | 4 layers |
|----------|---------|----------|----------|----------|
| Citeseer | 72.80±0.25 | 76.60±0.50 | 74.90±0.64 | 73.29±0.53 |
| Cora     | 80.80±0.14  | 87.43±0.22  | 86.37±0.29 | 85.05±0.70 |
| Pubmed   | 87.80±0.22  | 91.20±0.30  | 90.25±0.57 | 89.77±0.46 |
| PPI      | 94.86±0.15  | 98.06±0.10  | 95.02±0.18 | 90.23±1.08 |
| EP       | 78.52±0.33  | 78.98±0.23  | 78.40±0.25 | 77.73±0.75 |
| Arxiv    | 67.83±0.07  | 71.85±0.04  | 70.42±0.18 | 69.85±0.32 |

4.4 Effect of scaling parameter matrix in attention process

We found that in some small datasets, STC can get better performance using an untrainable constant scaling parameter matrix in attention process. To further validate it, we trained STC with an untrainable constant scaling parameter matrix and a trainable constant scaling parameter matrix respectively on six homogeneous graphs and compared the test accuracy (micro-F1) of them.
As Table 13 shows, in some small datasets like Citeseer, Cora, and Pubmed, using an untrainable constant scaling parameter matrix can improve the performance significantly and at the same time reduce the parameter number. But for larger graphs like PPI and Arxiv, an untrainable constant scaling parameter matrix will cause performance degradation.

| Table 13 | Comparison of test accuracy (micro-F1) and number of trainable parameters of STC and STC\(^\dagger\). Note: STC\(^\dagger\) denotes STC using an untrainable constant scaling parameter matrix in attention process. Best values are in bold. |
|----------|---------------------------------------------------------------------------------------------------------------|
|          | STC | Param. | STC\(^\dagger\) | Param. |
| Citeseer | 74.92±0.28 | 2.16M | **76.60±0.50** | 1.15M |
| Cora     | 85.72±0.52 | 1.00M | **87.43±0.22** | 0.57M |
| Pubmed   | 90.35±0.23 | 0.52M | **91.20±0.30** | 0.33M |
| PPI      | **98.06±0.10** | 0.29M | 95.71±0.07 | 0.21M |
| EP       | **78.98±0.23** | 0.28M | 78.14±0.60 | 0.21M |
| Arxiv    | **71.85±0.04** | 0.34M | 70.18±0.16 | 0.24M |

4.5 Effect of filter size

The effect of different filter sizes in each STC layer were examined on Citeseer and Arxiv. Validation and test accuracy (Figure 8) improved with increasing filter size to a peak value and then steadily reduced. A small filter size might give insufficient information to train a good model, while larger filter sizes might introduce redundant information to the trained model, leading to overfitting and a degradation in performance. Figure 8 also shows that the average batch time obtained by STC with different filter sizes on Citeseer and Arxiv were linear with respect to filter size. The filter size in STC is unrelated to the parameter number, since parameter number is determined by scaling parameter vector \(a\) and scaling parameter matrix \(C\). Their dimension is only related to the input feature dimension. The filter size could be tuned to achieve the best performance.

| Table 14 | Comparison of test accuracy (micro-F1) of STC (conv only) and Pretrained STC (conv only) on three homogeneous citation graphs. The Pretrained STC (conv only) is pretrained on the large homogeneous citation graph: Arxiv. Best values are in bold. |
|----------|---------------------------------------------------------------------------------------------------------------|
|          | STC (conv only) | Pretrained STC (conv only) |
| Citeseer | 74.55±0.83 | **76.78±0.30** |
| Cora     | 86.50±0.54 | **88.12±0.48** |
| Pubmed   | 90.77±0.38 | **91.84±0.13** |

4.6 Pretrained model

STC is more similar to conventional CNNs than most existing spectral methods. Because its filter can share weights across different graphs as Figure 3 shows, it is more flexible than some spectral methods like Spectral CNN and
GWNN. Their filters are defined on the graph Fourier transform or graph wavelet transform of the full graph, which makes them difficult to reuse the weights on different graphs. Hence, a potential application of STC is using transfer learning [66, 67] to employ pretrained STC as an embedding for feature extraction of downstream tasks. Since the filter size of STC is related to the feature dimension while the filter size of STC (conv only) is only related to filter size which has less constraints, we select STC (conv only) to do the transfer learning. We trained a STC (conv only) model on the large homogeneous citation graph, Arxiv, first. Then we extracted the two STC layers of the pretrained model and froze their parameters to work as an embedding for downstream classification tasks of other three homogeneous citation datasets: Citeseer, Cora, and Pubmed. Table 14 shows the performance comparison of STC (conv only) and Pretrained STC (conv only) on Citeseer, Cora, and Pubmed. Through introducing the pretrained STC layers as the embedding, the classification performance has been improved significantly.

5 Conclusions

In this paper, we presented a novel graph convolutional method, called star topology convolution (STC), which is a graph-scale free inductive spectral convolutional method. It learns node embedding on subgraphs within a star topology. In experiments, our method outperformed state-of-the-art graph
convolutional methods on both homogeneous graph and heterogeneous graph benchmarks, and showed better robustness and was more generalizable. STC also outperformed state-of-the-art methods in identifying essential proteins. STC can share weights across different graphs which makes it possible to be pretrained as an embedding for feature extraction to improve the performance of some downstream tasks.

Declarations

• Conflict of interest: The authors declare that they have no conflict of interest.

• Acknowledgments: This work is supported by Hong Kong Innovation and Technology Commission (InnoHK Project CIMDA), Hong Kong Research Grants Council (Project 11204821), and City University of Hong Kong (Project 9610034).

• Code and data availability: The codes and data used in this paper are available at: https://github.com/Espera-fzn/Star_Topology_Steps.

References

[1] Xu, B., Shen, H., Cao, Q., Qiu, Y., Cheng, X.: Graph wavelet neural network. In: International Conference on Learning Representations (2019). https://openreview.net/forum?id=H1ewdiR5tQ

[2] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)

[3] Gehring, J., Auli, M., Grangier, D., Dauphin, Y.N.: A convolutional encoder model for neural machine translation. CoRR abs/1611.02344 (2016) arXiv:1611.02344

[4] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representations (2018). https://openreview.net/forum?id=rJXMpikCZ

[5] Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: Going beyond Euclidean data. IEEE Signal Processing Magazine 34(4), 18–42 (2017)

[6] Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. CoRR abs/1312.6203 (2014)

[7] Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in
[8] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (2017). https://openreview.net/forum?id=SJU4ayYgl&noteId=SJU4ayYgl

[9] Shi, M., Tang, Y., Zhu, X., Liu, J.: Multi-label graph convolutional network representation learning. IEEE Transactions on Big Data, 1–1 (2020). https://doi.org/10.1109/TBDATA.2020.3019478

[10] Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.: GraphSAINT: Graph sampling based inductive learning method. In: International Conference on Learning Representations (2020). https://openreview.net/forum?id=BJe8pkHFwS

[11] Chen, J., Ma, T., Xiao, C.: FastGCN: Fast learning with graph convolutional networks via importance sampling. In: International Conference on Learning Representations (2018). https://openreview.net/forum?id=rytstxWAW

[12] Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 31, pp. 4805–4815. Curran Associates Inc., Red Hook, NY, USA (2018)

[13] Chen, J., Zhu, J., Song, L.: Stochastic training of graph convolutional networks with variance reduction. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 942–950 (2018). http://proceedings.mlr.press/v80/chen18p.html

[14] Gao, H., Wang, Z., Ji, S.: Large-scale learnable graph convolutional networks. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD ’18, pp. 1416–1424. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3219819.3219947. https://doi.org/10.1145/3219819.3219947

[15] Huang, W., Zhang, T., Rong, Y., Huang, J.: Adaptive sampling towards fast graph representation learning. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 31, pp. 4563–4572. Curran
[16] Chiang, W.L., Liu, X., Si, S., Li, Y., Bengio, S., Hsieh, C.J.: Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD ’19, pp. 257–266. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3292500.3330925

[17] Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.: Accurate, efficient and scalable graph embedding. In: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 462–471 (2019). https://doi.org/10.1109/IPDPS.2019.00056

[18] Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems vol. 30, pp. 1025–1035. Curran Associates Inc., Red Hook, NY, USA (2017)

[19] Hammond, D.K., Vanderheynst, P., Gribonval, R.: Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis 30(2), 129–150 (2011). https://doi.org/10.1016/j.acha.2010.04.005

[20] Monti, F., Boscaini, D., Masci, J., Rodol`a, E., Svoboda, J., Bronstein, M.M.: Geometric deep learning on graphs and manifolds using mixture model CNNs. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5425–5434. IEEE Computer Society, Los Alamitos, CA, USA (2017)

[21] Ding, M., Tang, J., Zhang, J.: Semi-supervised learning on graphs with generative adversarial nets. CoRR abs/1809.00130 (2018) arXiv:1809.00130

[22] Monti, F., Shchur, O., Bojchevski, A., Litany, O., Günnemann, S., Bronstein, M.M.: Dual-primal graph convolutional networks. CoRR abs/1806.00770 (2018) arXiv:1806.00770

[23] Mohan, A., Pramod, K.V.: Temporal network embedding using graph attention network. Complex & Intelligent Systems, 13–27 (2022)

[24] Zhang, Y., Wang, X., Shi, C., Jiang, X., Ye, Y.F.: Hyperbolic graph attention network. IEEE Transactions on Big Data, 1–1 (2021). https://doi.org/10.1109/TBDATA.2021.3081431

[25] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735–1780 (1997)
Star Topology Convolution for Graph Representation Learning

[26] de Haan, P., Cohen, T.S., Welling, M.: Natural graph networks. arXiv preprint arXiv:2007.08349 (2020)

[27] Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. CoRR abs/1207.0580 (2012) arXiv:1207.0580

[28] Yu, L., Shen, J., Li, J., Lerer, A.: Scalable graph neural networks for heterogeneous graphs. arXiv preprint arXiv:2011.09679 (2020)

[29] Bordes, A., Usunier, N., Garcia-Durán, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2, pp. 2787–2795. Curran Associates Inc., Red Hook, NY, USA (2013)

[30] Rossi, E., Frasca, F., Chamberlain, B., Eynard, D., Bronstein, M.M., Monti, F.: SIGN: Scalable inception graph neural networks. CoRR abs/2004.11198 (2020) 2004.11198

[31] Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., Eliassi-Rad, T.: Collective classification in network data articles. AI Magazine 29, 93–106 (2008). https://doi.org/10.1609/aimag.v29i3.2157

[32] Zitnik, M., Leskovec, J.: Predicting multicellular function through multilayer tissue networks. Bioinformatics (Oxford, England) 33(14), 190–198 (2017). https://doi.org/10.1093/bioinformatics/btx252

[33] Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., Leskovec, J.: Open graph benchmark: Datasets for machine learning on graphs. arXiv preprint arXiv:2005.00687 (2020)

[34] Lv, Q., Ding, M., Liu, Q., Chen, Y., Feng, W., He, S., Zhou, C., Jiang, J., Dong, Y., Tang, J.: Are we really making much progress? revisiting, benchmarking, and refining heterogeneous graph neural networks. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 1150–1160. Association for Computing Machinery, New York, NY, USA (2021)

[35] Xenarios, I., Salwinski, L., Duan, X.J., Higney, P., Kim, S.M., Eisenberg, D.: DIP, the database of interacting proteins: A research tool for studying cellular networks of protein interactions. Nucleic Acids Research 30(1), 303–305 (2002) https://academic.oup.com/nar/article-pdf/30/1/303/9901064/300303.pdf. https://doi.org/10.1093/nar/30.1.303
[36] Mewes, H.W., Frishman, D., Mayer, K.F.X., Münsterkötter, M., Noubibou, O., Pagel, P., Rattei, T., Oesterheld, M., Ruepp, A., Stümpflen, V.: MIPS: Analysis and annotation of proteins from whole genomes in 2005. Nucleic Acids Research 34(suppl_1), 169–172 (2006). https://doi.org/10.1093/nar/gkj148

[37] Cherry, J.M., Adler, C., Ball, C., Chervitz, S.A., Dwight, S.S., Hester, E.T., Jia, Y., Juvik, G., Roe, T., Schroeder, M., Weng, S., Botstein, D.: SGD: Saccharomyces genome database. Nucleic Acids Research 26(1), 73–79 (1998). https://doi.org/10.1093/nar/26.1.73

[38] Zhang, R., Lin, Y.: DEG 5.0, a database of essential genes in both prokaryotes and eukaryotes. Nucleic Acids Research 37(suppl_1), 455–458 (2008). https://doi.org/10.1093/nar/gkn858

[39] Saccharomyces genome deletion project. [http://yeastdeletion.stanford.edu/].1:32 2020/5/26 Accessed 20 June 2012

[40] Edgar, R., Domrachev, M., Lash, A.E.: Gene expression omnibus: NCBI gene expression and hybridization array data repository. Nucleic Acids Research 30(1), 207–210 (2002). https://doi.org/10.1093/nar/30.1.207

[41] Rong, Y., Huang, W., Xu, T., Huang, J.: DropEdge: Towards deep graph convolutional networks on node classification. In: International Conference on Learning Representations (2020). https://openreview.net/forum?id=Hkx1qkrKPr

[42] Huang, Q., He, H., Singh, A., Lim, S.N., Benson, A.R.: Combining label propagation and simple models out-performs graph neural networks. arXiv preprint arXiv:2010.13993 (2020) arXiv:2010.13993 [cs.LG]

[43] Deng, C., Zhao, Z., Wang, Y., Zhang, Z., Feng, Z.: GraphZoom: A multi-level spectral approach for accurate and scalable graph embedding. In: International Conference on Learning Representations (2020). https://openreview.net/forum?id=r1lGO0EKDH

[44] Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. CoRR abs/1607.00653 (2016) arXiv:1607.00653

[45] Zhu, X.J.: Semi-supervised learning literature survey. world 10, 10 (2005)

[46] Kong, K., Li, G., Ding, M., Wu, Z., Zhu, C., Ghanem, B., Taylor, G., Goldstein, T.: FLAG: Adversarial data augmentation for graph neural networks. arXiv preprint arXiv:2010.09891 (2020) arXiv:2010.09891 [cs.LG]
Star Topology Convolution for Graph Representation Learning

[47] Luo, Y., Chen, A., Yan, K., Tian, L.: Distilling self-knowledge from contrastive links to classify graph nodes without passing messages. arXiv preprint arXiv:2106.08541 (2021) arXiv:2106.08541 [cs.LG]

[48] Dong, Y., Chawla, N.V., Swami, A.: metapath2vec: Scalable representation learning for heterogeneous networks. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 135–144. Association for Computing Machinery, New York, NY, USA (2017)

[49] Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al.: Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261 (2018)

[50] Chi, H., Wang, Y., Hao, Q., Xia, H.: Residual network and embedding usage: New tricks of node classification with graph convolutional networks. CoRR abs/2105.08330 (2021) 2105.08330

[51] Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., Yu, P.S.: Heterogeneous graph attention network. In: The World Wide Web Conference, pp. 2022–2032. Association for Computing Machinery, New York, NY, USA (2019)

[52] Yun, S., Jeong, M., Kim, R., Kang, J., Kim, H.J.: Graph transformer networks. Advances in Neural Information Processing Systems 32, 11983–11993 (2019)

[53] Zhu, S., Zhou, C., Pan, S., Zhu, X., Wang, B.: Relation structure-aware heterogeneous graph neural network. In: 2019 IEEE International Conference on Data Mining (ICDM), pp. 1534–1539. IEEE Computer Society, Los Alamitos, CA, USA (2019). IEEE

[54] Zhang, C., Song, D., Huang, C., Swami, A., Chawla, N.V.: Heterogeneous graph neural network. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD ’19, pp. 793–803. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3292500.3330961. https://doi.org/10.1145/3292500.3330961

[55] Fu, X., Zhang, J., Meng, Z., King, I.: Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In: Proceedings of The Web Conference 2020, pp. 2331–2341. Association for Computing Machinery, New York, NY, USA (2020)

[56] Hong, H., Guo, H., Lin, Y., Yang, X., Li, Z., Ye, J.: An attention-based
graph neural network for heterogeneous structural learning. Proceedings of the AAAI Conference on Artificial Intelligence 34(04), 4132–4139 (2020). https://doi.org/10.1609/aaai.v34i04.5833

[57] Zeng, M., Li, M., Fei, Z., Wu, F., Li, Y., Pan, Y.: A deep learning framework for identifying essential proteins based on protein-protein interaction network and gene expression data. In: 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 583–588. IEEE Computer Society, Los Alamitos, CA, USA (2018)

[58] Wu, C., Zhang, H., Zhang, L., Zheng, H.: Identification of essential proteins using a novel multi-objective optimization method. In: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1329–1333 (2020)

[59] Jeong, H., Mason, S.P., Barabasi, A.L., Oltvai, Z.N.: Lethality and centrality in protein networks. Nature 411, 41–42 (2001). https://doi.org/10.1038/35075138

[60] Joy, M.P., Brock, A., Ingber, D.E., Huang, S.: High-betweenness proteins in the yeast protein interaction network. Journal of biomedicine & biotechnology 2005, 96–103 (2005). https://doi.org/10.1155/JBB.2005.96

[61] Wuchty, S., Stadler, P.F.: Centers of complex networks. Journal of theoretical biology 223(1), 45–53 (2003). https://doi.org/10.1016/S0022-5193(03)00071-7

[62] Bonacich, P.: Power and centrality: A family of measures. American Journal of Sociology 92(5), 1170–1182 (1987). https://doi.org/10.1086/228631

[63] Wang, J., Li, M., Wang, H., Pan, Y.: Identification of essential proteins based on edge clustering coefficient. IEEE/ACM Transactions on Computational Biology and Bioinformatics 9(4), 1070–1080 (2012)

[64] Li, M., Wang, J., Chen, X., Wang, H., Pan, Y.: A local average connectivity-based method for identifying essential proteins from the network level. Computational Biology and Chemistry 35(3), 143–150 (2011). https://doi.org/10.1016/j.compbiolchem.2011.04.002

[65] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[66] Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering 22(10), 1345–1359 (2010). https://doi.org/10.1109/TKDE.2009.191
[67] Valdenegro-Toro, M., Preciado-Grijalva, A., Wehbe, B.: Pre-trained models for sonar images. CoRR abs/2108.01111 (2021) 2108.01111