Haar Graph Pooling

Yu Guang Wang [*1,2] **Ming Li** [*3] **Zheng Ma** [*4] **Guido Montúfar** [*5,2] **Xiaosheng Zhuang** [6] **Yanan Fan** [1]

Abstract

Deep Graph Neural Networks (GNNs) are useful models for graph classification and graph-based regression tasks. In these tasks, graph pooling is a critical ingredient by which GNNs adapt to input graphs of varying size and structure. We propose a new graph pooling operation based on compressive Haar transforms — *HaarPooling*. HaarPooling implements a cascade of pooling operations; it is computed by following a sequence of clusterings of the input graph. A HaarPooling layer transforms a given input graph to an output graph with a smaller node number and the same feature dimension; the compressive Haar transform filters out fine detail information in the Haar wavelet domain. In this way, all the HaarPooling layers together synthesize the features of any given input graph into a feature vector of uniform size. Such transforms provide a sparse characterization of the data and preserve the structure information of the input graph. GNNs implemented with standard graph convolution layers and HaarPooling layers achieve state of the art performance on diverse graph classification and regression problems.

1. Introduction

Graph Neural Networks (GNNs) have demonstrated excellent performance in node classification tasks and are very promising in graph classification and regression (Bronstein et al., 2017; Battaglia et al., 2018; Zhang et al., 2018b; Zhou et al., 2018; Wu et al., 2019). In node classification, the input is a single graph with missing node labels that are to be predicted from the known node labels. In this problem, GNNs with appropriate graph convolutions can be trained based on a single input graph, and achieve state-of-the-art performance (Defferrard et al., 2016; Kipf & Welling, 2017; Ma et al., 2019b). Different from node classification, graph classification is a task where the label of any given graph-structured sample is to be predicted based on a training set of labeled graph-structured samples. This is similar to the image classification task tackled by traditional deep convolutional neural networks. The significant difference is that here each input sample may have an arbitrary adjacency structure instead of the fixed, regular grids that are used in standard pixel images. This raises two crucial challenges: 1) How can GNNs exploit the graph structure information of the input data? 2) How can GNNs handle input graphs with varying number of nodes and connectivity structures?

These problems have motivated the design of proper *graph convolution* and *graph pooling* to allow GNNs to capture the geometric information of each data sample (Zhang et al., 2018a; Ying et al., 2018; Cangea et al., 2018; Gao & Ji, 2019; Knyazev et al., 2019; Ma et al., 2019a; Lee et al., 2019). Graph convolution plays an important role, especially in question 1).

The following is a widely utilized type of graph convolution layer, proposed by (Kipf & Welling, 2017):

$$X^{\text{out}} = \widehat{A} X^{\text{in}} W. \tag{1}$$

Here $\widehat{A} = \widetilde{D}^{-1/2}(A+I)\widetilde{D}^{-1/2} \in \mathbb{R}^{N\times N}$ is a normalized version of the adjacency matrix A of the input graph, where I is the identity matrix and $\widetilde{D}$ is the degree matrix for $A+I$. Further, $X^{\text{in}} \in \mathbb{R}^{N\times d}$ is the array of d-dimensional features on the N nodes of the graph, and $W \in \mathbb{R}^{d\times m}$ is the filter parameter matrix. We call the graph convolution of Kipf & Welling (2017) in Equation (1) the *GCN convolution*.

The graph convolution in Equation (1) captures the structural information of the input in terms of A (or $\widehat{A}$), and W transforms the feature dimension from d to m. As the filter size $d \times m$ is independent of the graph size, it allows a fixed network architecture to process input graphs of varying sizes. The GCN convolution preserves the number of nodes, and hence the output dimension of the network is

[*]Equal contribution [1]School of Mathematics and Statistics, University of New South Wales, Sydney, Australia [2]Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany [3]Department of Educational Technology, Zhejiang Normal University, Jinhua, China [4]Department of Physics, Princeton University, New Jersey, USA [5]Department of Mathematics and Department of Statistics, University of California, Los Angeles [6]Department of Mathematics, City University of Hong Kong, Hong Kong. Correspondence to: Ming Li <mingli@zjnu.edu.cn>, Yu Guang Wang <yuguang.wang@unsw.edu.au>, Guido Montúfar <montufar@math.ucla.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

not unique. Graph pooling provides an effective way to overcome this obstacle. Some existing approaches, EigenPooling (Ma et al., 2019a), for example, incorporate both features and graph structure, which gives very good performance on graph classification.

In this paper, we propose a new graph pooling strategy based on a sparse Haar representation of the data, which we call *Haar Graph Pooling*, or simply *HaarPooling*. It is built on the *Haar basis* (Wang & Zhuang, 2019; 2020; Li et al., 2019), which is a localized wavelet-like basis on a graph. We define HaarPooling by the compressive Haar transform of the graph data, which is a nonlinear transform that operates in the Haar wavelet domain by using the Haar basis. For an input data sample, which consists of a graph $\mathcal{G}$ and the feature on the nodes $X^{\text{in}} \in \mathbb{R}^{N\times d}$, the compressive Haar transform is determined by the Haar basis vectors on $\mathcal{G}$ and maps X^{in} into a matrix X^{out} of dimension $d \times N_1$. The pooled feature X^{out} is in the Haar wavelet domain and extracts the coarser feature of the input. Thus, HaarPooling can provide sparse representations of graph data that distill structural graph information.

The Haar basis and its compressive transform can be used to define cascading pooling layers, i.e., for each layer, we define an orthonormal Haar basis and its compressive Haar transform. Each HaarPooling layer pools the graph input from the previous layer to output with a smaller node number and the same feature dimension. In this way, all the HaarPooling layers together synthesize the features of all graph input samples into feature vectors with the same size. We then obtain an output of a fixed dimension, regardless of the size of the input.

The algorithm of HaarPooling is simple to implement as the Haar basis and the compressive Haar transforms can be computed by the explicit formula. The computation of HaarPooling is cheap, with nearly linear time complexity. HaarPooling can connect to any graph convolution. The GNNs with HaarPooling can handle multiple tasks. Experiments in Section 5 demonstrate that the GNN with HaarPooling achieves state of the art performance on various graph classification and regression tasks.

2. Related Work

Graph pooling is a vital step when building a GNN model for graph classification and regression, as one needs a unified graph-level rather than node-level representation for graph-structured inputs of which size and topology are changing. The most direct pooling method, as provided by the graph convolutional layer (Duvenaud et al., 2015), takes the global mean and sum of the features of the nodes as a simple graph-level representation. This pooling operation treats all the nodes equally and uses the global geometry of the graph. ChebNet (Defferrard et al., 2016) used a graph coarsening procedure to build the pooling module, for which one needs a graph clustering algorithm to obtain subgraphs. One drawback of this topology-based strategy is that it does not combine the node features in the pooling. The global pooling method considers the information about node embeddings, which can achieve the entire graph representation. As a general framework for graph classification and regression problems, MPNN (Gilmer et al., 2017) used the Set2Set method (Vinyals et al., 2015) that would obtain a graph-level representation of the graph input samples. The SortPool (Zhang et al., 2018a) proposed a method that could rank and select the nodes by sorting their feature representation and then feed them into a traditional 1-D convolutional or dense layer. These global pooling methods did not utilize the hierarchical structure of the graph, which may carry useful geometric information of data.

One notable recent argument is to build a differentiable and data-dependent pooling layer with learnable operations or parameters, which has brought a substantial improvement in graph classification tasks. The DiffPool (Ying et al., 2018) proposed a differentiable pooling layer that learns a cluster assignment matrix over the nodes relating to the output of a GNN model. One difficulty of DiffPool is its vast storage complexity, which is due to the computation of the soft clustering. The TopKPooling (Cangea et al., 2018; Gao & Ji, 2019; Knyazev et al., 2019) proposed a pooling method that samples a subset of essential nodes by manipulating a trainable projection vector. The Self-Attention Graph Pooling (SAGPool) (Lee et al., 2019) proposed an analogous pooling that applied the GCN module to compute the node scores instead of the projection vector in the TopKPooling. These hierarchical pooling methods technically still employ mean/max pooling procedures to aggregate the feature representation of super-nodes. To preserve more edge information of the graph, EdgePool (Diehl et al., 2019) proposed to incorporate edge contraction. The StructPool (Yuan & Ji, 2020) proposed a graph pooling that employed conditional random fields to represent the relation of different nodes.

The spectral-based pooling method suggests another design, which operates the graph pooling in the frequency domain, for example, the Fourier domain or the wavelet domain. By its nature, the spectral-based approach can combine the graph structure and the node features. The Laplacian Pooling (LaPool) (Noutahi et al., 2019) proposed a pooling method that dynamically selected the centroid nodes and their corresponding follower nodes by using a graph Laplacian-based attention mechanism. EigenPool (Ma et al., 2019a) introduced a graph pooling that used the local graph Fourier transform to extract subgraph information. Its potential drawback lies in the inherent computing bottleneck for the Laplacian-based graph Fourier transform, given the high computational cost for the eigendecomposition of the graph

Laplacian. Our HaarPooling is a spectral-based method that applies the Haar basis system in the node feature representation, as we now introduce.

3. Haar Graph Pooling

In this section, we give an overview of the proposed HaarPooling framework. First we define the pooling architecture in terms of a coarse-grained chain, i.e., a sequence of graphs $(\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_K)$, where the nodes of the $(j+1)$th graph $\mathcal{G}_{j+1}$ correspond to the clusters of nodes of the jth graph $\mathcal{G}_j$ for each $j = 0, \ldots, K-1$. Based on the chain, we construct the Haar basis and the compressive Haar transform. The latter then defines the HaarPooling operation. Each layer in the chain determines which sets of nodes the network pools, and the compressive Haar transform synthesizes the information from the graph and node feature for pooling.

Chain of coarse-grained graphs for pooling Graph pooling amounts to defining a sequence of coarse-grained graphs. In our chain, each graph is an induced graph that arises from grouping (clustering) certain subsets of nodes from the previous graph. We use clustering algorithms to generate the groupings of nodes. There are many good candidates, such as spectral clustering (Shi & Malik, 2000), k-means clustering (Pakhira, 2014), DBSCAN (Ester et al., 1996), OPTICS (Ankerst et al., 1999) and METIS (Karypis & Kumar, 1998). Any of these will work with HaarPooling. Figure 1 shows an example of a chain with 3 levels, for an input graph $\mathcal{G}_0$.

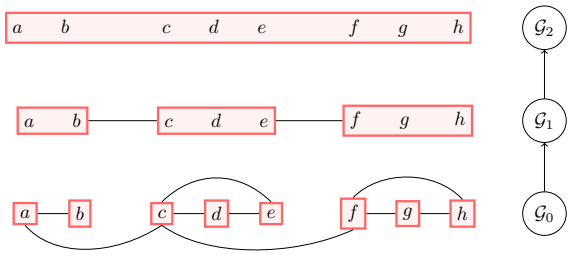


Figure 1. A coarse-grained chain of graphs. The input has 8 nodes; the second and top levels have 3 and single nodes.

Compressive Haar transforms on chain For each layer of the chain, we will have a feature representation. We define these in terms of the Haar basis. The Haar basis represents graph-structured data by low- and high-frequency Haar coefficients in the frequency domain. The low-frequency coefficients contain the coarse information of the original data, while the high-frequency coefficients contain the fine details. In HaarPooing, the data is pooled (or compressed) by discarding the fine detail information.

The Haar basis can be compressed in each layer. Consider a chain. The two subsequent graphs have N_{j+1} and N_j nodes, $N_{j+1} < N_j$. We select N_j elements from the $(j+1)$th layer for the jth layer, each of which is a vector of size N_{j+1}. These N_j vectors form a matrix Φ_j of size $N_{j+1} \times N_j$. We call Φ_j the *compressive Haar basis matrix* for this particular jth layer. This then defines the *compressive Haar transform* $\Phi_j^T X^{\text{in}}$ for feature X^{in} of size $N_j \times d$.

Computational strategy of HaarPooling By compressive Haar transform, we can define the HaarPooling.

Definition 1 (HaarPooling)**.** *The HaarPooling for a graph neural network with K pooling layers is defined as*

$$X_j^{\text{out}} = \Phi_j^T X_j^{\text{in}}, \quad j = 0, 1, \ldots, K-1,$$

where Φ_j is the $N_j \times N_{j+1}$ compressive Haar basis matrix for the jth layer, $X_j^{\text{in}} \in \mathbb{R}^{N_j \times d_j}$ is the input feature array, and $X_j^{\text{out}} \in \mathbb{R}^{N_{j+1} \times d_j}$ is the output feature array, for some $N_j > N_{j+1}$, $j = 0, 1, \ldots, K-1$, and $N_K = 1$. For each j, the corresponding layer is called the jth HaarPooling layer. More explicitly, we also write Φ_j as $\Phi^{(j)}_{N_j \times N_{j+1}}$.

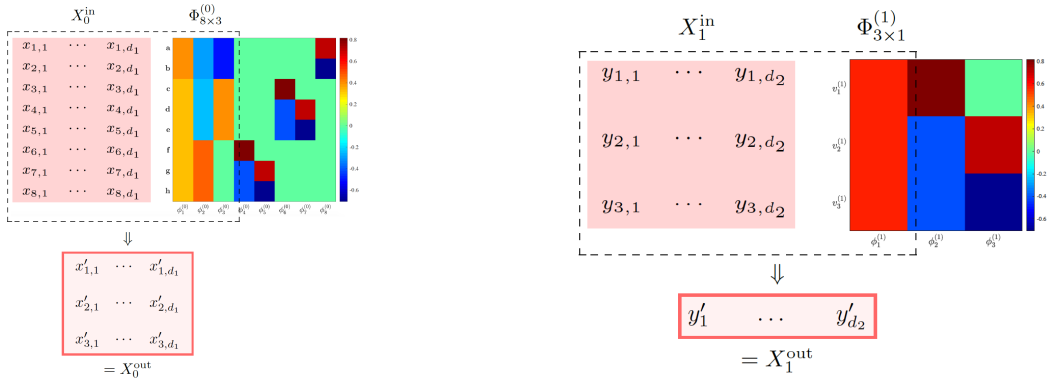
HaarPooling has the following fundamental properties.

- First, the HaarPooling is a hierarchically structured algorithm. The coarse-grained chain determines the hierarchical relation in different HaarPooling layers. The node number of each HaarPooling layer is equal to the number of nodes of the subgraph of the corresponding layer of the chain. As the top-level of the chain can have one node, the HaarPooling finally reduces the number of nodes to one, thus producing a fixed dimensional output in the last HaarPooling layer.

- The HaarPooling uses the sparse Haar representation on chain structure. In each HaarPooling layer, the representation then combines the features of input X_j^{in} with the geometric information of the graphs of the jth and $(j+1)$th layers of the chain.

- By the property of the Haar basis, the HaarPooling only drops the high-frequency information of the input data. The X_j^{out} mirrors the low-frequency information in the Haar wavelet representation of X_j^{in}. Thus, HaarPooling preserves the essential information of the graph input, and the network has small information loss in pooling.

Example Figure 2 shows the computational details of the HaarPooling associated with the chain from Figure 1. There are two HaarPooling layers. In the first layer, the input X_1^{in} of size $8 \times d_1$ is transformed by the compressive Haar basis matrix $\Phi^{(0)}_{8\times 3}$ which consists of the first three column vectors of the full Haar basis $\Phi^{(0)}_{8\times 8}$ in (a), and the output is a $3 \times d_1$ matrix X_1^{out}. In the second layer, the input X_2^{in}



(a) First HaarPooling Layer for $\mathcal{G}_0 \to \mathcal{G}_1$.

(b) Second HaarPooling Layer for $\mathcal{G}_1 \to \mathcal{G}_2$.

Figure 2. Computational strategy of HaarPooling. We use the chain in Figure 1, and then the network has two HaarPooling layers: $\mathcal{G}_0 \to \mathcal{G}_1$ and $\mathcal{G}_1 \to \mathcal{G}_2$. The input of each layer is pooled by the compressive Haar transform for that layer: in the first layer, the input $X_1^{\text{in}} = (x_{i,j}) \in \mathbb{R}^{8\times d_1}$ is transformed by the compressive Haar basis matrix $\Phi_{8\times 3}^{(0)}$ of size 8×3 formed by the first three column vectors of the original Haar basis, and the output is a feature array of size $3 \times d_1$; in the second layer, $X_2^{\text{in}} = (y_{i,j}) \in \mathbb{R}^{3\times d_2}$ is transformed by the first column vector $\Phi_{3\times 1}^{(1)}$ and the output is a feature vector of size $1 \times d_2$. In the plots of the Haar basis matrix, the colors indicate the value of the entries of the Haar basis matrix.

of size $3 \times d_2$ (usually X_1^{out} followed by convolution) is transformed by the compressive Haar matrix $\Phi_{3\times 1}^{(1)}$, which is the first column vector of the full Haar basis matrix $\Phi_{3\times 3}^{(1)}$ in (b). By the construction of the Haar basis in relation to the chain (see Section 4), each of the first three column vectors $\phi_1^{(0)}, \phi_2^{(0)}$ and $\phi_3^{(0)}$ of $\Phi_{8\times 3}^{(0)}$ has only up to three different values. This bound is precisely the number of nodes of $\mathcal{G}_1$. For each column of $\phi_\ell^{(0)}$, all nodes with the same parent take the same value. Similarly, the 3×1 vector $\phi_1^{(1)}$ is constant. This example shows that the HaarPooling amalgamates the node feature by adding the same weight to the nodes that are in the same cluster of the coarser layer, and in this way, pools the feature using the graph clustering information.

4. Compressive Haar Transforms

Chain of graphs by clustering For a graph $\mathcal{G} = (V, E, w)$, where V, E, w are the vertices, edges, and weights on edges, a graph $\mathcal{G}^{\text{cg}} = (V^{\text{cg}}, E^{\text{cg}}, w^{\text{cg}})$ is a *coarse-grained graph* of $\mathcal{G}$ if $|V^{\text{cg}}| \le |V|$ and each node of $\mathcal{G}$ has only one parent node in $\mathcal{G}^{\text{cg}}$ associated with it. Each node of $\mathcal{G}^{\text{cg}}$ is called a *cluster* of $\mathcal{G}$. For integers $J > 0$, a *coarse-grained chain* for $\mathcal{G}$ is a sequence of graphs $\mathcal{G}_{0\to J} := (\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_J)$ with $\mathcal{G}_0 = \mathcal{G}$ and such that $\mathcal{G}_{j+1}$ is a coarse-grained graph of $\mathcal{G}_j$ for each $j = 0, 1, \ldots, J-1$, and $\mathcal{G}_J$ has only one node. Here, we call the graph $\mathcal{G}_J$ the *top level* or the *coarsest level* and $\mathcal{G}_0$ the *bottom level* or the *finest level*. The chain $\mathcal{G}_{0\to J}$ hierarchically coarsens graph $\mathcal{G}$. We use the notation $J + 1$ for the number of layers of the chain, to distinguish it from the number K of layers for pooling. For details about graphs and chains, we refer the reader to the examples by (Chung & Graham, 1997; Hammond et al., 2011; Chui et al., 2015; 2018; Wang & Zhuang, 2019; 2020).

Haar (1910) first introduced Haar basis on the real axis. It is a particular example of the more general Daubechies wavelets (Daubechies, 1992). Haar basis was later constructed on graphs by (Belkin et al., 2006), and also (Chui et al., 2015; Wang & Zhuang, 2019; 2020; Li et al., 2019).

Construction of Haar basis The Haar bases $\{\phi_\ell^{(j)}\}_{\ell=1}^{N_j}$, $j = 0, \ldots, J$, is a sequence of collections of vectors. Each Haar basis is associated with a single layer of the chain $\mathcal{G}_{0\to J}$ of a graph $\mathcal{G}$. For $j = 0, \ldots, J$, we let the matrix $\widetilde{\Phi}_j = (\phi_1^{(j)}, \ldots, \phi_{N_j}^{(j)}) \in \mathbb{R}^{N_j \times N_j}$ and call the matrix $\widetilde{\Phi}_j$ *Haar transform matrix* for the jth layer. In the following, we detail the construction of the Haar basis based on the coarse-grained chain of a graph, as discussed in (Wang & Zhuang, 2019; 2020; Li et al., 2019). We attach the algorithmic pseudo-codes for generating the Haar basis on the graph in the supplementary material.

Step 1. Let $\mathcal{G}^{\text{cg}} = (V^{\text{cg}}, E^{\text{cg}}, w^{\text{cg}})$ be a coarse-grained graph of $\mathcal{G} = (V, E, w)$ with $N^{\text{cg}} := |V^{\text{cg}}|$. Here we use the sub-index "cg" to indicate the symbol is for the coarse-grained graph. Each vertex $v^{\text{cg}} \in V^{\text{cg}}$ is a cluster $v^{\text{cg}} = \{v \in V \,|\, v \text{ has parent } v^{\text{cg}}\}$ of $\mathcal{G}$. Order V^{cg}, e.g., by degrees of vertices or weights of vertices, as $V^{\text{cg}} = \{v_1^{\text{cg}}, \ldots, v_{N^{\text{cg}}}^{\text{cg}}\}$. We define N^{cg} vectors ϕ_ℓ^{cg} on $\mathcal{G}^{\text{cg}}$ by

$$\phi_1^{\text{cg}}(v^{\text{cg}}) := \frac{1}{\sqrt{N^{\text{cg}}}}, \quad v^{\text{cg}} \in V^{\text{cg}}, \tag{2}$$

and for $\ell = 2, \ldots, N^{\mathrm{cg}}$,

$$\phi_\ell^{\mathrm{cg}} := \sqrt{\frac{N^{\mathrm{cg}} - \ell + 1}{N^{\mathrm{cg}} - \ell + 2}} \left(\chi_{\ell-1}^{\mathrm{cg}} - \frac{\sum_{j=\ell}^{N^{\mathrm{cg}}} \chi_j^{\mathrm{cg}}}{N^{\mathrm{cg}} - \ell + 1} \right), \quad (3)$$

where χ_j^{cg} is the indicator function for the jth vertex $v_j^{\mathrm{cg}} \in V^{\mathrm{cg}}$ on $\mathcal{G}$ given by

$$\chi_j^{\mathrm{cg}}(v^{\mathrm{cg}}) := \begin{cases} 1, & v^{\mathrm{cg}} = v_j^{\mathrm{cg}}, \\ 0, & v^{\mathrm{cg}} \in V^{\mathrm{cg}} \backslash \{v_j^{\mathrm{cg}}\}. \end{cases}$$

Then, the $\{\phi_\ell^{\mathrm{cg}}\}_{\ell=1}^{N^{\mathrm{cg}}}$ forms an orthonormal basis for $l_2(\mathcal{G}^{\mathrm{cg}})$. Each $v \in V$ belongs to exactly one cluster $v^{\mathrm{cg}} \in V^{\mathrm{cg}}$. In view of this, for each $\ell = 1, \ldots, N^{\mathrm{cg}}$, we can extend the vector ϕ_ℓ^{cg} on $\mathcal{G}^{\mathrm{cg}}$ to a vector $\phi_{\ell,1}$ on $\mathcal{G}$ by

$$\phi_{\ell,1}(v) := \frac{\phi_\ell^{\mathrm{cg}}(v^{\mathrm{cg}})}{\sqrt{|v^{\mathrm{cg}}|}}, \quad v \in v^{\mathrm{cg}},$$

here $|v^{\mathrm{cg}}| := k_\ell$ is the size of the cluster v^{cg}, i.e., the number of vertices in $\mathcal{G}$ whose common parent is v^{cg}. We order the cluster v_ℓ^{cg}, e.g., by degrees of vertices, as

$$v_\ell^{\mathrm{cg}} = \{v_{\ell,1}, \ldots, v_{\ell,k_\ell}\} \subseteq V.$$

For $k = 2, \ldots, k_\ell$, similar to Equation (3), define

$$\phi_{\ell,k} = \sqrt{\frac{k_\ell - k + 1}{k_\ell - k + 2}} \left(\chi_{\ell,k-1} - \frac{\sum_{j=k}^{k_\ell} \chi_{\ell,j}}{k_\ell - k + 1} \right),$$

where for $j = 1, \ldots, k_\ell$, $\chi_{\ell,j}$ is given by

$$\chi_{\ell,j}(v) := \begin{cases} 1, & v = v_{\ell,j}, \\ 0, & v \in V \backslash \{v_{\ell,j}\}. \end{cases}$$

Then, the resulting $\{\phi_{\ell,k} : \ell = 1, \ldots, N^{\mathrm{cg}}, k = 1, \ldots, k_\ell\}$ is an orthonormal basis for $l_2(\mathcal{G})$.

Step 2. Let $\mathcal{G}_{0\to J}$ be a coarse-grained chain for the graph $\mathcal{G}$. An orthonormal basis $\{\phi_\ell^{(J)}\}_{\ell=1}^{N_J}$ for $l_2(\mathcal{G}_J)$ is generated using Equations (2) and (3). We then repeatedly use Step 1: for $j = 0, \ldots, J-1$, generate an orthonormal basis $\{\phi_\ell^{(j)}\}_{\ell=1}^{N_j}$ for $l_2(\mathcal{G}_j)$ from the orthonormal basis $\{\phi_\ell^{(j+1)}\}_{\ell=1}^{N_{j+1}}$ for the coarse-grained graph $\mathcal{G}_{j+1}$ that was derived in the previous steps. We call the sequence $\{\phi_\ell := \phi_\ell^{(0)}\}_{\ell=1}^{N_0}$ of vectors at the finest level, the *Haar global orthonormal basis*, or simply the *Haar basis*, for $\mathcal{G}$ associated with the chain $\mathcal{G}_{0\to J}$. The orthonormal basis $\{\phi_\ell^{(j)}\}_{\ell=1}^{N_j}$ for $l_2(\mathcal{G}_j)$, $j = 1, \ldots, J$ is called the *Haar basis* for the jth layer.

Compressive Haar basis Suppose we have constructed the (full) Haar basis $\{\phi_\ell^{(j)}\}_{\ell=0}^{N_j}$ for each layer $\mathcal{G}_j$ of the chain $\mathcal{G}_{0\to K}$. The *compressive Haar basis* for layer j is $\{\phi_\ell^{(j)}\}_{\ell=0}^{N_{j+1}}$. We will use the transforms of this basis to define HaarPooling.

Orthogonality For each level $j = 0, \ldots, J$, the sequence $\{\phi_\ell^{(j)}\}_{\ell=1}^{N_j}$, with $N_j := |V_j|$, is an orthonormal basis for the space $l_2(\mathcal{G}_j)$ of square-summable sequences on the graph $\mathcal{G}_j$, so that $(\phi_\ell^{(j)})^T \phi_{\ell'}^{(j)} = \delta_{\ell,\ell'}$. For each j, $\{\phi_\ell^{(j)}\}_{\ell=1}^{N_j}$ is the Haar basis system for the chain $\mathcal{G}_{j\to J}$.

Locality Let $\mathcal{G}_{0\to J}$ be a coarse-grained chain for $\mathcal{G}$. If each parent of level $\mathcal{G}_j$, $j = 1, \ldots, J$, contains at least two children, the number of different scalar values of the components of the Haar basis vector $\phi_\ell^{(j)}$, $\ell = 1, \ldots, N_j$, is bounded by a constant independent of j.

In Figure 2, the Haar basis is generated based on the coarse-grained chain $\mathcal{G}_{0\to 2} := (\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2)$, where $\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2$ are graphs with $8, 3, 1$ nodes. The two colorful matrices show the two Haar bases for the layers 0 and 1 in the chain $\mathcal{G}_{0\to 2}$. There are in total 8 vectors of the Haar basis for $\mathcal{G}_0$ each with length 8, and 3 vectors of the Haar basis for $\mathcal{G}_1$ each with length 3. Haar basis matrix for each level of the chain has up to 3 different values in each column, as indicated by colors in each matrix. For $j = 0, 1$, each node of $\mathcal{G}_j$ is a cluster of nodes in $\mathcal{G}_{j+1}$. Each column of the matrix is a member of the Haar basis on the individual layer of the chain. The first three column vectors of $\widetilde{\Phi}_1$ can be reduced to an orthonormal basis of $\mathcal{G}_1$ and the first column vector of $\mathcal{G}_1$ to the constant basis for $\mathcal{G}_2$. This connection ensures that the compressive Haar transforms for HaarPooling is also computationally feasible.

Adjoint and forward Haar transforms We utilize adjoint and forward Haar transforms to compute HaarPooling. Due to the sparsity of the Haar basis matrix, the transforms are computationally feasible. The *adjoint Haar transform* for the signal f on $\mathcal{G}_j$ is

$$(\widetilde{\Phi}_j)^T f = \left(\sum_{v\in V} \phi_1^{(j)}(v) f(v), \ldots, \sum_{v\in V} \phi_{N_j}^{(j)}(v) f(v) \right) \in \mathbb{R}^{N_j}, \quad (4)$$

and the *forward Haar transform* for (coefficients) vector $c := (c_1, \ldots, c_{N_j}) \in \mathbb{R}^{N_j}$ is

$$(\widetilde{\Phi}_j c)(v) = \sum_{\ell=1}^{N_j} \phi_\ell^{(j)}(v) c_\ell, \quad v \in V_j. \quad (5)$$

We call the components of $(\widetilde{\Phi}_j)^T f$ the *Haar (wavelet) coefficients* for f. The adjoint Haar transform represents the signal in the Haar wavelet domain by computing the Haar coefficients for graph signal, and the forward transform sends back the Haar coefficients to the time domain. Here, the adjoint and forward Haar transforms can be extended to a feature data with size $N_j \times d_j$ by replacing the column vector f with the feature array.

Proposition 2. *The adjoint and forward Haar Transforms are invertible in that for $j = 0, \ldots, J$ and vector f on graph*

$\mathcal{G}_j$,

$$f = \widetilde{\Phi}_j (\widetilde{\Phi}_j)^T f.$$

Proposition 2 shows that the forward Haar transform can recover the graph signal f from the adjoint Haar transform $(\widetilde{\Phi}_j)^T f$, which means that adjoint and forward Haar transforms have zero-loss in graph signal transmission.

Compressive Haar transforms Now for a graph neural network, suppose we want to use K pooling layers for $K \geq 1$. We associate the chain $\mathcal{G}_{0\to K}$ of an input graph with the pooling by linking the jth layer of pooling with the jth layer of the chain. Then, we can use the Haar basis system on the chain to define the pooling operation. By the property of Haar basis, in the Haar transforms for layer j, $0 \leq j \leq K-1$, of the N_j Haar coefficients, the first N_{j+1} coefficients are the low-frequency coefficients, which reflect the approximation to the original data, and the remaining $(N_j - N_{j+1})$ coefficients are in high frequency, which contains fine details of the Haar wavelet decomposition. To define pooling, we remove the high-frequency coefficients in the Haar wavelet representation and then obtain the *compressive Haar transforms* for the feature X_j^{in} at layers $j = 0, \ldots, K-1$, which then gives the HaarPooling in Definition 1.

As shown in the following formula, the compressive Haar transform incorporates the neighborhood information of the graph signal as compared to the full Haar transform. Thus, the HaarPooling can take the average information of the data f over nodes in the same cluster.

$$\begin{aligned} \left\|\Phi_j^T X_j^{\text{in}}\right\|^2 &= \sum_{p\in\mathcal{G}_{j+1}} \frac{1}{|Pa(v)|} \Big| \sum_{p=Pa(v)} X_j^{\text{in}}(v) \Big|^2 \\ \left\|\widetilde{\Phi}_j^T X_j^{\text{in}}\right\|^2 &= \sum_{p\in\mathcal{G}_{j+1}} \sum_{p=Pa(v)} \left| X_j^{\text{in}}(v) \right|^2, \end{aligned} \tag{6}$$

where $\widetilde{\Phi}_j$ is the full Haar basis matrix at the jth layer and $|Pa_{\mathcal{G}}(v)|$ is the number of nodes in the cluster which the node v lies in. Here, $1/\sqrt{|Pa_{\mathcal{G}}(v)|}$ can be taken out of summation as $Pa(v)$ is in fact a set of nodes. We show the derivation of formula in Equation (6) in the supplementary.

In HaarPooling, the compression or pooling occurs in the Haar wavelet domain. It transforms the features on the nodes to the Haar wavelet domain. It then discards the high-frequency coefficients in the sparse Haar wavelet representation. See Figure 2 for a two-layer HaarPooling example.

5. Experiments

In this section, we present the test results of HaarPooling on various datasets in graph classification and regression tasks. We show a performance comparison of the HaarPooling with existing graph pooling methods. All the experiments use PyTorch Geometric (Fey & Lenssen, 2019) and were run in Google Cloud using 4 Nvidia Telsa T4 with 2560 CUDA cores, compute 7.5, 16GB GDDR6 VRAM.

5.1. HaarPooling on Classification Benchmarks

Datasets and baseline methods To verify whether the proposed framework can hierarchically learn good graph representations for classification, we evaluate *HaarPooling* on five widely used benchmark datasets for graph classification (Kersting et al., 2016), including one protein graph dataset **PROTEINS** (Borgwardt et al., 2005; Dobson & Doig, 2003); two mutagen datasets **MUTAG** (Debnath et al., 1991; Kriege & Mutzel, 2012) and **MUTAGEN** (Riesen & Bunke, 2008; Kazius et al., 2005) (full name Mutagenicity); and two datasets that consist of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, **NCI1** and **NCI109** (Wale et al., 2008). We include datasets from different domains, samples, and graph sizes to give a comprehensive understanding of how the HaarPooling performs with datasets in various scenarios. Table 1 summarizes some statistical information of the datasets: each dataset containing graphs with different sizes and structures, the number of data samples ranges from 188 to 4,337, the average number of nodes is from 17.93 to 39.06, and the average number of edges is from 19.79 to 72.82. We compare **HaarPool** with **SortPool** (Zhang et al., 2018a), **DiffPool** (Ying et al., 2018), **gPool** (Gao & Ji, 2019), **SAGPool** (Lee et al., 2019), **EigenPool** (Ma et al., 2019a), **CSM** (Kriege & Mutzel, 2012) and **GIN** (Xu et al., 2019) on the above datasets.

Training In the experiment, we use a GNN with at most 3 GCN (Kipf & Welling, 2017) convolutional layers plus one HaarPooling layer, followed by three fully connected layers. The hyperparameters of the network are adjusted case by case. We use spectral clustering to generate a chain with the number of layers given. Spectral clustering, which exploits the eigenvalues of the graph Laplacian, has proved excellent performance in coarsening a variety of data patterns and can handle isolated nodes.

We apply random shuffling for the dataset. We split the whole dataset into the training, validation, and test sets with percentages 80%, 10%, and 10%, respectively. We use the Adam optimizer (Kingma & Ba, 2015), early stopping criterion, and patience, and give the specific values in the supplementary. Here, the early stopping criterion was that the validation loss does not improve for 50 epochs, with a maximum of 150 epochs, as suggested by Shchur et al. (2018).

The architecture of GNN is identified by the layer type and the number of hidden nodes at each layer. For example,

Table 1. Summary statistics of the graph classification datasets.

Dataset	MUTAG	PROTEINS	NCI1	NCI109	MUTAGEN
max #nodes	28	620	111	111	417
min #nodes	10	4	3	4	4
avg #nodes	17.93	39.06	29.87	29.68	30.32
avg #edges	19.79	72.82	32.30	32.13	30.77
#graphs	188	1,113	4,110	4,127	4,337
#classes	2	2	2	2	2

we denote 3GC256-HP-2FC256-FC128 to represent a GNN architecture with 3 GCNConv layers, each with 256 hidden nodes, plus one HaarPooling layer followed by 2 fully connected layers, each with 256 hidden nodes, and by one fully connected layer with 128 hidden nodes. Table 3 shows the GNN architecture for each dataset.

Results Table 2 reports the classification test accuracy. GNNs with HaarPooling have excellent performance on all datasets. In 4 out of 5 datasets, it achieves top accuracy. It shows that HaarPooling, with an appropriate graph convolution, can achieve top performance on a variety of graph classification tasks, and in some cases, improve state of the art by a few percentage points.

5.2. HaarPooling on Triangles Classification

We test GNN with HaarPooling on the graph dataset **Triangles** (Knyazev et al., 2019). **Triangles** is a 10 class classification problem with 45,000 graphs. The average numbers of nodes and edges of the graphs are 20.85 and 32.74, respectively. In the experiment, the network utilizes GIN convolution (Xu et al., 2019) as graph convolution and either HaarPooling or SAGPooling (Lee et al., 2019). For SAGPooling, the network applies two combined layers of GIN convolution and SAGPooling, which is followed by the combined layers of GIN convolution and *global max pooling*. We write its architecture as GIN-SP-GIN-SP-GIN-MP, where SP means the SAGPooling and MP is the global max pooling. For HaarPooling, we examine two architectures: GIN-HP-GIN-HP-GIN-MP and GIN-HP-GIN-GIN-MP, where HP stands for HaarPooling. We split the data into training, validation, and test sets of size 35,000, 5,000, and 10,000. The number of nodes in the convolutional layers are all set to 64; the batch size is 60; the learning rate is 0.001.

Table 4 shows the training, validation, and test accuracy of the three networks. It shows that both networks with HaarPooling outperform that with SAGPooling.

5.3. HaarPooling for Quantum Chemistry Regression

QM7 In this part, we test the performance of the GNN model equipped with the HaarPooling layer on the QM7 dataset. People have recently used the QM7 to measure the efficacy of machine-learning methods for quantum chemistry (Blum & Reymond, 2009; Rupp et al., 2012). The QM7 dataset contains 7,165 molecules, each of which is represented by the Coulomb (energy) matrix and labeled with the atomization energy. Each molecule contains up to 23 atoms. We treat each molecule as a weighted graph: atoms as nodes and the Coulomb matrix of the molecule as the adjacency matrix. Since the node (atom) itself does not have feature information, we set the node feature to a constant vector (i.e., the vector with components all 1), so that features here are uninformative, and only the molecule structure is concerned in learning. The task is to predict the atomization energy value of each molecule graph, which boils down to a standard graph regression problem.

Methods in comparison We use the same GNN architecture to test HaarPool and SAGPool (Lee et al., 2019): one GCN layer, one graph pooling layer, plus one 3-layer MLP. We compare the performance (test MAE) of the GCN-HaarPool against the GCN-SAGPool and other methods including Random Forest (RF) (Breiman, 2001), Multitask Networks (Multitask) (Ramsundar et al., 2015), Kernel Ridge Regression (KRR) (Cortes & Vapnik, 1995), Graph Convolutional models (GC) (Altae-Tran et al., 2017).

Experimental setting In the experiment, we normalize the label value by subtracting the mean and scaling the standard deviation (Std Dev) to 1. We then need to convert the predicted output to the original label domain (by re-scaling and adding the mean back). Following Gilmer et al. (2017), we use mean squared error (MSE) as the loss for training and mean absolute error (MAE) as the evaluation metric for validation and test. Similar to the graph classification tasks studied above, we use PyTorch Geometric (Fey & Lenssen, 2019) to implement the models of GCN-HaarPool and GCN-SAGPool, and run the experiment under the GPU computing environment in the Google Cloud AI Platform. Here, the splitting percentages for training, validation, and test are 80%, 10%, and 10%, respectively. We set the hidden dimension of the GCN layer as 64, the Adam for optimization with the learning rate 5.0e-4, and the maximal epoch 50 with no early stop. We do not use dropout as it would slightly lower the performance. For better comparison, we repeat all experiments ten times with different random seeds.

Table 2. Performance comparison for graph classification tasks (test accuracy in percent, showing the standard deviation over ten repetitions of the experiment).

Method	MUTAG	PROTEINS	NCI1	NCI109	MUTAGEN
CSM	85.4	–	–	–	–
GIN	89.4	76.2	**82.7**	–	–
SortPool	85.8	75.5	74.4	72.3*	78.8*
DiffPool	–	76.3	76.0*	74.1*	80.6*
gPool	–	77.7	–	–	–
SAGPool	–	72.1	74.2	74.1	–
EigenPool	–	76.6	77.0	74.9	79.5
HaarPool (ours)	**90.0±3.6**	**80.4±1.8**	78.6±0.5	**75.6±1.2**	**80.9±1.5**

'*' indicates records retrieved from EigenPool (Ma et al., 2019a), '–' means that there are no public records for the method on the dataset, and bold font is used to highlight the best performance in the list.

Table 3. Network architecture.

Dataset	**Layers and #Hidden Nodes**
MUTAG	GC60-HP-FC60-FC180-FC60
PROTEINS	2GC128-HP-2GC128-HP-2GC128-HP-GC128-2FC128-FC64
NCI1	2GC256-HP-FC256-FC1024-FC2048
NCI109	3GC256-HP-2FC256-FC128
MUTAGEN	3GC256-HP-2FC256-FC128

Table 4. Results on the Triangles dataset.

Architecture	Accuracy (%)		
	Train	Val	Test
GIN-SP-GIN-SP-GIN-MP	45.6	45.3	44.0
GIN-HP-GIN-HP-GIN-MP (ours)	47.5	46.3	46.1
GIN-HP-GIN-GIN-MP (ours)	47.3	45.8	45.5

Table 5 shows the results for GCN-HaarPool and GCN-SAGPool, together with the public results of the other methods from Wu et al. (2018). Compared to the GCN-SAGPool, the GCN-HaarPool has a lower average test MAE and a smaller Std Dev and ranks the top in the table. Given the simple architecture of our GCN-HaarPool model, we can interpret the GCN-HaarPool an effective method, although its prediction result does not rank the top in Table 9 reported in Wu et al. (2018). To further demonstrate that HaarPooling can benefit graph representation learning, we present in Figure 3 the mean and Std Dev of the training MSE loss (for normalized input) and the validation MAE (which is in the original label domain) versus the epoch. It illustrates that the learning and generalization capabilities of the GCN-HaarPool are better than those of the GCN-SAGPool; in this aspect, HaarPooling provides a more efficient graph pooling for GNN in this graph regression task.

6. Computational Complexity

In the supplementary material, we show the time complexity comparison of HaarPooling and other existing pooling methods. HaarPool is the only algorithm in this table which has near-linear time complexity to the node number. HaarPooling can be even faster in practice, as the cost of the compressive Haar transform is dependent on the sparsity of the Haar basis matrix. The sparsity of the compressive Haar basis matrix is mainly reliant on the chain/tree for the graph. From our construction, the compressive Haar basis matrix is always highly sparse. Thus, the computational cost does not skyrocket as the size of the graph increases.

Table 5. Test mean absolute error (MAE) comparison on QM7, with the standard deviation over ten repetitions of the experiments.

Method	Test MAE
RF	122.7 ± 4.2
Multitask	123.7 ± 15.6
KRR	110.3 ± 4.7
GC	77.9 ± 2.1
GCN-SAGPool	43.3 ± 1.6
GCN-HaarPool (ours)	$\mathbf{42.9 \pm 1.2}$

For empirical comparison, we computed the GPU time for HaarPool and TopKPool on a sequence of datasets of random graphs, as shown in Figure 4. For each run, we fix the number of edges of the graphs. For different runs, the number of the edges ranges from 4,000 to 121,000. The sparsity of the adjacency matrix of the random graph is set to 10%. The following table shows the average GPU time (in seconds) for pooling a minibatch of 50 graphs. For both pooling methods, we use the same network architecture and one pooling layer, and same network hyperparameters, and run under the same GPU computing environment.

Figure 4 shows that the cost of HaarPool does not change much as the edge number increases, while the cost of TopKPool increases rapidly. When the edge number is at most 25000, TopKPool runs slightly faster than HaarPool, but when the number exceeds 25000, the GPU time of TopKPool is longer.

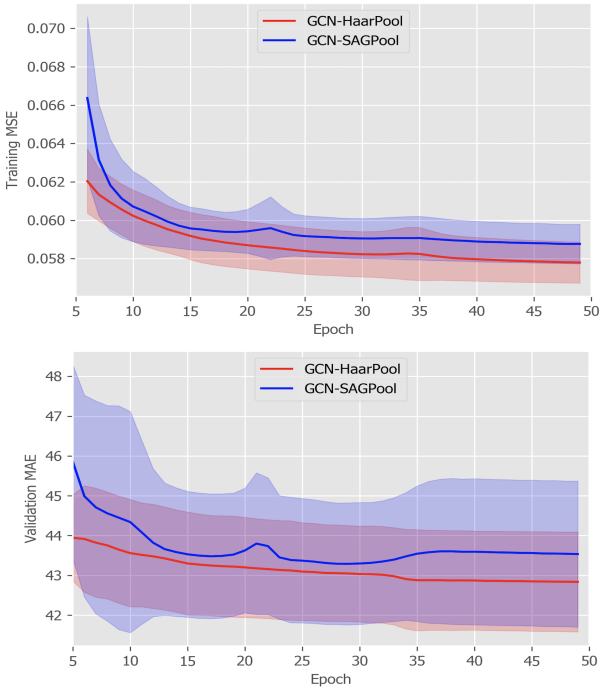


Figure 3. Visualization of the training MSE loss (top) and validation MAE (bottom) for GCN-HaarPool and GCN-SAGPool.

The computational cost of clustered tree generation depends on the clustering algorithms one uses. In the paper, we take spectral clustering as a typical example. Spectral clustering has good performance on various datasets, and its time complexity, though not linear, is smaller than the k-means clustering. The METIS is also a good candidate that has a fast implementation. As the Haar basis generation can be pre-computed, the time complexity of clustering has no impact on the complexity of pooling.

By our test, the number of layers of the chain for HaarPool has a substantial impact on the performance of GNNs, and the randomness in the clustering algorithm has little effect on the stability of GNNs.

7. Conclusion

We introduced a new graph pooling method called HaarPooling. It has a mathematical formalism derived from compressive Haar transforms. Unlike existing graph pooling methods, HaarPooling takes into account both the graph structure and the features over the nodes, to compute a coarsened representation. The implementation of HaarPooling is simple as the Haar basis and its transforms can be computed directly by the explicit formula. The time and space complexities of HaarPooling are cheap, $\mathcal{O}(|V|)$ and $\mathcal{O}(|V|^2\epsilon)$ for sparsity ϵ of Haar basis, respectively. As an individual unit, HaarPooling can be applied in conjunction with any graph convolution in GNNs. We show in experiments that

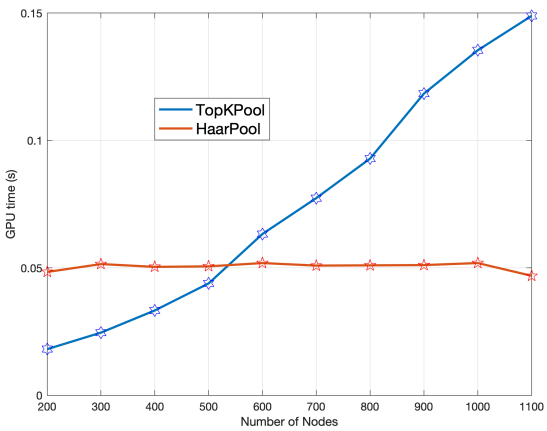


Figure 4. GPU time comparison of HaarPool with TopKPool for random graphs with up to 1,100 nodes; the Y-axis is the mean GPU time(in seconds) for pooling a minibatch of 50 graphs.

HaarPooling reaches and in several cases surpasses state of the art performance in multiple graph classification and regression tasks.

ACKNOWLEDGMENTS

Ming Li acknowledges support from the National Natural Science Foundation of China (No. 61802132 and 61877020). Yu Guang Wang acknowledges support from the Australian Research Council under Discovery Project DP180100506. Guido Montúfar has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement n° 757983). Xiaosheng Zhuang acknowledges support in part from Research Grants Council of Hong Kong (Project No. CityU 11301419). This material is based upon work supported by the National Science Foundation under Grant No. DMS-1439786 while Zheng Ma, Guido Montúfar and Yu Guang Wang were in residence at the Institute for Computational and Experimental Research in Mathematics in Providence, RI, during Collaborate@ICERM on "Geometry of Data and Networks". Part of this research was performed while Guido Montúfar and Yu Guang Wang were at the Institute for Pure and Applied Mathematics (IPAM), which is supported by the National Science Foundation (Grant No. DMS-1440415).

References

Altae-Tran, H., Ramsundar, B., Pappu, A. S., and Pande, V. Low data drug discovery with one-shot learning. *ACS Central Science*, 3(4):283–293, 2017.

Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J. Optics: ordering points to identify the clustering structure. In *ACM Sigmod Record*, volume 28, pp. 49–60. ACM, 1999.

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.

Belkin, M., Niyogi, P., and Sindhwani, V. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7(Nov):2399–2434, 2006.

Blum, L. C. and Reymond, J.-L. 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *Journal of the American Chemical Society*, 131:8732, 2009.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1): i47–i56, 2005.

Breiman, L. Random forests. *Machine Learning*, 45(1): 5–32, 2001.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42, 2017.

Cangea, C., Veličković, P., Jovanović, N., Kipf, T., and Liò, P. Towards sparse hierarchical graph classifiers. In *Workshop on Relational Representation Learning, NeurIPS*, 2018.

Chui, C., Filbir, F., and Mhaskar, H. Representation of functions on big data: graphs and trees. *Applied and Computational Harmonic Analysis*, 38(3):489–509, 2015.

Chui, C. K., Mhaskar, H., and Zhuang, X. Representation of functions on big data associated with directed graphs. *Applied and Computational Harmonic Analysis*, 44(1): 165–188, 2018. ISSN 1063-5203. doi: https://doi.org/10.1016/j.acha.2016.12.005.

Chung, F. R. and Graham, F. C. *Spectral graph theory*. American Mathematical Society, 1997.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

Daubechies, I. *Ten lectures on wavelets*. SIAM, 1992.

Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34 (2):786–797, 1991. doi: 10.1021/jm00106a046.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pp. 3844–3852, 2016.

Diehl, F., Brunner, T., Le, M. T., and Knoll, A. Towards graph pooling by edge contraction. In *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Representation*, 2019.

Dobson, P. D. and Doig, A. J. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, pp. 2224–2232, 2015.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pp. 226–231, 1996.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. In *Workshop on Representation Learning on Graphs and Manifolds, ICLR*, 2019.

Gao, H. and Ji, S. Graph U-Nets. *ICML*, pp. 2083–2092, 2019.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *ICML*, pp. 1263–1272, 2017.

Haar, A. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69(3):331–371, 1910.

Hammond, D. K., Vandergheynst, P., and Gribonval, R. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

Kazius, J., McGuire, R., and Bursi, R. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of Medicinal Chemistry*, 48(1):312–320, 2005.

Kersting, K., Kriege, N. M., Morris, C., Mutzel, P., and Neumann, M. Benchmark data sets for graph kernels, 2016. URL http://graphkernels.cs.tu-dortmund.de.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Knyazev, B., Taylor, G. W., and Amer, M. R. Understanding attention and generalization in graph neural networks. In *NeurIPS*, 2019.

Kriege, N. and Mutzel, P. Subgraph matching kernels for attributed graphs. In *ICML*, pp. 291–298, 2012.

Lee, J., Lee, I., and Kang, J. Self-attention graph pooling. In *ICML*, pp. 3734–3743, 2019.

Li, M., Ma, Z., Wang, Y. G., and Zhuang, X. Fast Haar transforms for graph neural networks. *arXiv preprint arXiv:1907.04786*, 2019.

Ma, Y., Wang, S., Aggarwal, C. C., and Tang, J. Graph convolutional networks with EigenPooling. In *KDD*, pp. 723–731, 2019a.

Ma, Z., Li, M., and Wang, Y. G. PAN: Path integral based convolution for deep graph neural networks. In *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Representation*, 2019b.

Noutahi, E., Beani, D., Horwood, J., and Tossou, P. Towards interpretable sparse graph representation learning with Laplacian pooling. *arXiv preprint arXiv:1905.11577*, 2019.

Pakhira, M. K. A linear time-complexity k-means algorithm using cluster shifting. In *2014 International Conference on Computational Intelligence and Communication Networks*, pp. 1047–1051, 2014. doi: 10.1109/CICN.2014.220.

Ramsundar, B., Kearnes, S., Riley, P., Webster, D., Konerding, D., and Pande, V. Massively multitask networks for drug discovery. *arXiv preprint arXiv:1502.02072*, 2015.

Riesen, K. and Bunke, H. IAM graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pp. 287–297. Springer, 2008.

Rupp, M., Tkatchenko, A., Müller, K.-R., and von Lilienfeld, O. A. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical Review Letters*, 108:058301, 2012.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. In *Workshop on Relational Representation Learning, NeurIPS*, 2018.

Shi, J. and Malik, J. Normalized cuts and image segmentation. *Departmental Papers (CIS)*, pp. 107, 2000.

Vinyals, O., Bengio, S., and Kudlur, M. Order matters: Sequence to sequence for sets. In *ICLR*, 2015.

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

Wang, Y. G. and Zhuang, X. Tight framelets on graphs for multiscale analysis. In *Wavelets and Sparsity XVIII, SPIE Proc.*, pp. 11138–11, 2019.

Wang, Y. G. and Zhuang, X. Tight framelets and fast framelet filter bank transforms on manifolds. *Applied and Computational Harmonic Analysis*, 48(1):64–95, 2020.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. MoleculeNet: a benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530, 2018.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *ICLR*, 2019.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, pp. 4800–4810, 2018.

Yuan, H. and Ji, S. Structpool: Structured graph pooling via conditional random fields. In *ICLR*, 2020.

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

Zhang, Z., Cui, P., and Zhu, W. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202*, 2018b.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.