Pooling Architecture Search for Graph Classification

Lanning Wei\textsuperscript{1,2,3}, Huan Zhao\textsuperscript{3}, Quanming Yao\textsuperscript{3,4} Zhiqiang He\textsuperscript{1,5}
\textsuperscript{1}Institute of Computing Technology, Chinese Academy of Sciences, \textsuperscript{2}University of Chinese Academy of Sciences \textsuperscript{3}4Paradigm, Inc., \textsuperscript{4}Department of Electronic Engineering, Tsinghua University, \textsuperscript{5}Lenovo
Beijing, China
wei.lanning18z@ict.ac.cn; zhaohuan@4paradigm.com; qyaoaa@tsinghua.edu.cn; hezq@lenovo.com

ABSTRACT
Graph classification is an important problem with applications across many domains, like chemistry and bioinformatics, for which graph neural networks (GNNs) have been state-of-the-art (SOTA) methods. GNNs are designed to learn node-level representation based on neighborhood aggregation schemes, and to obtain graph-level representation, pooling methods are applied after the aggregation operation in existing GNN models to generate coarse-grained graphs. However, due to highly diverse applications of graph classification, and the performance of existing pooling methods vary on different graphs. In other words, it is a challenging problem to design a universal pooling architecture to perform well in most cases, leading to a demand for data-specific pooling methods in real-world applications. To address this problem, we propose to use neural architecture search (NAS) to search for adaptive pooling architectures for graph classification. Firstly we designed a unified framework consisting of four modules: Aggregation, Pooling, Readout, and Merge, which can cover existing human-designed pooling methods for graph classification. Based on this framework, a novel search space is designed by incorporating popular operations in human-designed architectures. Then to enable efficient search, a coarsening strategy is proposed to continuously relax the search space, thus a differentiable search method can be adopted. Extensive experiments on six real-world datasets from three domains are conducted, and the results demonstrate the effectiveness and efficiency of the proposed framework\textsuperscript{1}.

KEYWORDS
Graph Classification; Graph Neural Networks; Neural Architecture Search

ACM Reference Format:
Lanning Wei\textsuperscript{1,2,3}, Huan Zhao\textsuperscript{3}, Quanming Yao\textsuperscript{3,4} Zhiqiang He\textsuperscript{1,5}. 2021. Pooling Architecture Search for Graph Classification. In Proceedings of the 30th ACM Int'l Conf. on Information and Knowledge Management (CIKM '21), 1

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CIKM '21, November 1–5, 2021, Virtual Event, Australia. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3459637.3482285

1 INTRODUCTION
In recent years, graph neural networks (GNNs) \cite{kipf2016semi, velivckovic2017graph} have been the state-of-the-art (SOTA) method for graph classification, which is an important problem with applications in various domains, e.g., chemistry \cite{de2018geometric}, bioinformatics \cite{gepstein2019ml}, text categorization \cite{tan2019text}, social networks \cite{shi2019graph}, and recommendation \cite{gong2019deep, zheng2020deep}. Most GNNs, e.g., GCN \cite{kipf2016semi}, GraphSAGE \cite{hamilton2017inductive}, GAT \cite{velivckovic2017graph} and GIN \cite{xie2021group}, were designed to learn node-level representation based on neighborhood aggregation schemes \cite{de2018geometric}, i.e., the embedding of a node is updated by iteratively aggregating the embeddings of its neighbors in a graph. However, for the graph classification task, we need to generate graph-level representation from the node embeddings given a graph.

To achieve this, in the literature, various pooling methods have been proposed based on GNNs. The simplest method is directly taking the mean or summation of all node embeddings as the graph representation \cite{kipf2016semi}. However, the global pooling methods only use node features and ignore the hierarchical information as they do not exploit the graph compositional nature \cite{de2018geometric}, leading to flat graph-level representation \cite{velivckovic2017graph}. Then more advanced pooling methods are proposed to preserve the hierarchical information by aggregating messages on coarser and coarser graphs, which are generated by applying a pooling operation to reduce the size of a graph after an aggregation operation in each layer (see Figure 2(a)). These methods are dubbed hierarchical pooling, and representative ones are DiffPool \cite{lee2018learning}, SAGPool \cite{lee2019coarsening}, ASAP \cite{vendom2019asap}, Graph U-Net \cite{morris2019up}, and STRUCTPOOL \cite{yan2019graph}, etc., which assume a cluster property underlying the graph and generate coarse graph in each layer corresponding to the cluster.

Despite the success of these pooling methods, in reality, graphs are from highly diverse domains, e.g., chemistry \cite{de2018geometric}, bioinformatics \cite{gepstein2019ml}, text categorization \cite{tan2019text}, social networks \cite{shi2019graph} and recommendation \cite{gong2019deep, zheng2020deep}, leading to a challenging problem that human-designed pooling architectures cannot adapt to diverse datasets well. To verify this problem, we design an experiment based on a recent GNN benchmark GraphGym \cite{zhang2020graphgym} to compare the performance of several representative pooling methods for graph classification. From Figure 1, we can observe that the percentage of each method winning over the other three is very close (around 25%), which means that no single human-designed pooling architecture can win in all cases (420 setups). Besides, the hierarchical pooling method is not always superior to the global one, which is consistent with the two latest works \cite{lee2018learning, xie2021group}. Therefore, it is very important to design data-specific pooling architectures for the graph classification task.
To summarize, the contributions of this work are as follows:

- To the best of our knowledge, PAS is the first method to learn data-specific pooling architectures for graph classification. To apply NAS to this task, we propose a unified framework which can cover various human-designed pooling architectures, including global and hierarchical ones.
- Based on the unified framework, we design a customized and effective search space. And to enable differentiable architecture search, a coarsening strategy is designed to relax the search space into a continuous one, thus we develop an efficient search method.
- Extensive experiments on six real-world datasets from bioinformatics, chemistry, and social networks show that the searched architectures by PAS outperform various baselines for graph classification and the efficiency of PAS in terms of search cost.

Notations. We represent a graph as $G = (A, H)$ where $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix of this graph and $H \in \mathbb{R}^{N \times d}$ is the node features. $N$ is the node number. $\mathcal{N}(v) = \{v\} \cup \{u|A_{uv} \neq 0\}$ represents sets of the self-contained first-order neighbors of node $v$. Give a dataset $\mathcal{D} = \{(G_1, y_1), \cdots, (G_M, y_M)\}$, $(G_i, y_i)$ is the $i$-th graph of this dataset. $M$ is the number of total graphs, $y \in \mathcal{Y}$ is the graph label. In a $L$-layer GNN, for clear presentation, the input graph is denoted by $G^0 = (A^0, H^0)$, and the input of $L$-th layer is $G^{L-1} = (A^{L-1}, H^{L-1})$, and the output is $G^L = (A^L, H^L)$. The features of node $v$ in $L$-th layer are denoted by $H_v^L$.

2 RELATED WORK

2.1 GNN for Graph Classification

In the literature, existing GNN methods for graph classification can be roughly classified into two groups: global pooling and hierarchical pooling methods. Global pooling methods only use one global pooling function behind the final aggregation operation, and hierarchical methods use pooling operation after each aggregation in the architecture. On one hand, the global pooling methods are straightforward, which add a simple pooling operation, such as global summation of all node embeddings, however, as mentioned in [49], these global pooling methods, e.g., GIN [46] and DGCNN [53], learn flat graph embeddings, which cannot capture the potential hierarchical information in real-world graphs. On the other hand, hierarchical pooling methods are proposed to solve this problem by aggregate messages on coarser and coarser graphs, e.g., from $G^0$ to $G^L$ as shown in Figure 2(a). It is achieved by applying a pooling operation to reduce the size of a graph after an aggregation operation in each layer. For these hierarchical pooling methods, SAGPool [23], Graph U-Net [11] and ASAP [34] sample a set of nodes based on diverse node score functions and form corresponding coarse graphs; DiffPool [49] and STRUCTPOOL [52] focus on grouping nodes into clusters with different assignment functions, re-generate the edges among these clusters. Besides, GMN [20] proposes one memory layer to jointly learn node representations and coarsen graph, MinCutPool [2] and EigenPool [28] focus on learning assignment functions in the frequency domain.

However, existing methods use predefined pooling operations, which are difficult to adapt to various datasets. In this paper, by the
proposed PAS, we can obtain data-specific pooling architectures for graph classification.

2.2 Graph Neural Architecture Search

NAS methods were proposed to automatically find SOTA CNN architectures in a pre-defined search space and representative methods are [27, 33, 35, 36, 45, 60]. Very recently, researchers tried to automatically design GNN architectures by NAS. The majority of these methods focus on designing the aggregation layers in GNNs with different search algorithms. For example, GraphNAS [12], AutoGNN [59], AutoGM [50], DSS [26] and [32] learn to design aggregation layers with diverse dimensions, such as attention function, attention head number, embedding size, etc; SANE [58], SNAG [57] and AutoGraph [24] provide the extra skip connections learning; GNAS [3] and Policy-GNN [22] learn to select the best message passing layers. Apart from designing aggregation layers, RE-MPNN [19] learns adaptive global pooling functions additionally. However, these methods fail to obtain the data-specific pooling architectures because the pooling operations which are essential to graph classification are not considered. As to the search algorithm, most of the existing methods use the RL (Reinforcement Learning) and EA (Evolutionary Algorithm) based methods to select architectures from the search space. RL based algorithms, used in [12, 22, 57, 59], sample architectures with RNN controller and then updated with policy gradient; EA based algorithms, used in [19, 24, 32], select parent architecture from the search space and generate new architectures with mutation [36] and crossover [35], select parent architecture from the search space and generate new architectures with mutation [36] and crossover [35]. Bayesian optimization is utilized in AutoGM [50]. These methods need thousands of evaluations which are computationally expensive, and differentiable search algorithms are proposed to solve the efficiency problem. They construct an over-parameterized network (supernet) and optimize this supernet with gradient descent due to the continuous relaxation of the search space. For example, methods DARTS [27] and SNAS [45] use the Softmax and the Gumble-Softmax functions as the relaxation function, respectively. The differentiable search algorithms are used in SANE [58], DSS [26] and GNAS [3] to relax the aggregation dimensions. However, it is difficult to relax the pooling operations because different candidate pooling operations generate different coarse graphs consisting of diverse nodes and edges. Thus, it is a challenge to design one efficient search algorithm for learning data-specific pooling architectures.

More graph neural architecture search methods can be found in [5, 14, 43, 54–56]. Compared with existing methods in Table 1, PAS provides one search space that can cover existing pooling methods and one coarsening strategy to develop an efficient data-specific pooling architecture learning method.

3 METHOD

In this section, we elaborate on the proposed PAS, including the unified framework for graph classification, followed by the proposed search space and efficient search algorithm with the proposed coarsening strategy.

| Methods | Search Space | Search Method |
|---------|--------------|---------------|
| GNNs    | Pooling: GIN [46] | √ |
|         | Aggregation: DiffPool [49] | √ |
|         | Readout: SAGPool [23] | √ |
|         | Merge: PAS (proposed) | √ |
| NAS     | Pooling: RE-MPNN [19] | √ |
|         | Aggregation: EA |

Table 1: Comparing existing human-designed and NAS based pooling methods with PAS. We set the search algorithm of hand-designed methods as "-". A: Aggregation, P: Pooling, R: Readout, M: Merge.

3.1 The Unified Framework

We define a unified framework that consists of four key modules for learning graph-level representation derived from existing pooling architectures, Aggregation, Pooling, Readout and Merge Module, respectively. In general, one Pooling Module is placed after each Aggregation Module in each layer, and Merge Module is utilized to incorporate the intermediate graph representations produced by Readout Module. In Figure 2(b), we use a 2-layer architecture backbone as an illustrative example of the unified framework. With the input graph \(G^0\), Aggregation Module updates node embeddings and produce the graph \(G^{1a} = (A^0, H^{1a})\). Pooling Module generates the coarse graph \(G^1 = (A^1, H^1)\) behind. 3 Readout Modules used to capture the graph representations \(z\) in all layers, and Merge Module generates the final graph representation \(z^f\). Based on this framework, we can unify most existing pooling methods including global and hierarchical ones. When we remove the pooling operations in the intermediate layers and only keep one in the final layer, we obtain a global pooling method. This design can guarantee the flexibility and expressiveness of the proposed framework. In Table 1, we can see that representative pooling architectures can be covered by this framework in terms of the four modules.

Based on this unified framework, an effective search space can be naturally designed by including human-designed operations, the details of which are given in Table 2. Then different combinations of these operations can be obtained, leading to data-specific pooling architectures, by any search method. Specially, to incorporate both the global and hierarchical pooling architectures, we add the NONE operation in the Pooling Module which means no pooling operations are used. In the NAS literature, the architecture searching task is to solve the following bi-level optimization problem:

\[
\min_{\alpha \in \mathcal{A}} \mathcal{L}_{\text{val}}(W^*(\alpha), \alpha),
\]

s.t. \(W^*(\alpha) = \arg\min_W \mathcal{L}_{\text{train}}(W, \alpha),\)

where \(\mathcal{A}\) represents the search space, \(\alpha\) represents one candidate architecture in \(\mathcal{A}\), and \(W\) represents the parameters of a model from the search space, and \(W^*(\alpha)\) represents the corresponding operation parameter after training. \(\mathcal{L}_{\text{train}}\) and \(\mathcal{L}_{\text{val}}\) are the training and validation loss, respectively. Popular NAS methods use RL [12, 60], EA [15], and differentiable [27, 45, 58] search algorithms. Due to the efficiency superiority, differentiable methods are more preferable in latest NAS methods. In this work, we also adopt the differentiable search paradigm. However, the challenge is that the proposed search space cannot be directly relaxed continuously,
which is a prerequisite step for the differentiable search method, due to the fact that different candidate pooling operations in the search space generate different coarse graphs consisting of diverse nodes and edges. To address this challenge, we design a coarsening strategy to properly relax the selection of pooling operations, thus the search space, in a continuous manner. In this way, an efficient search process based on gradient descent is enabled. In Table 1, we compare PAS with existing human-designed and NAS based pooling methods in terms of the four modules (search space) and the search algorithm, and in the remaining part of this section, we introduce in detail the search space and the differentiable search algorithm.

### 3.2 The Design of the Search Space

Based on the proposed framework, we design one novel search space with a set of candidate operations as shown in Table 2. The detailed OPs are given in the following.

#### Aggregation Module.
We add five widely used GNNs: GCN [21], GAT [40], GraphSAGE [16] with mean aggregator, GIN [46] and GraphConv [30], which denoted as GCN, GAT, SAGE, GIN, and GRAPHCONV. Besides, we incorporate the operation MLP, which applies a two-layer MLP (Multilayer Perceptrons) to update node embeddings without using the graph structure.

#### Pooling Module.
The pooling operations in our search space can be unified by a computation process as

\[
S = f_S(A, H), \quad \text{idx} = \text{TOPK}_k(S),
\]

(3)

\[
A' = A(\text{idx}, \text{idx}), H' = H(\text{idx}, :).
\]

(4)

We firstly calculate a node score matrix \(S \in \mathbb{R}^{N \times 1}\) with a score function \(f_S\), which is used to evaluate the importance of nodes with different metrics, then generate the coarse graph by selecting top-\(k\) nodes \(\text{idx}\) with the function \(\text{TOPK}\), and formulating the coarse graph according to Eq. (4).

Three existing pooling operations \(\text{TOPKPOOL}\) [11], \(\text{SAGPOOL}\) [23] and \(\text{ASAP}\) [34] are incorporated in our search space. We further provide 6 score functions: \(\text{HOPPOOL}_t\) formulates the node scores

#### Table 2: The operations used in our search space.

| Module name   | Operations                  |
|---------------|-----------------------------|
| Aggregation   | GCN, GAT, SAGE, GIN, GRAPHCONV, MLP |
| Pooling       | TOPKPOOL, SAGPOOL, ASAP, HOPPOOL_1, HOPPOOL_2, HOPPOOL_3, MLPPPOOL, GCPOOL, GAPPOOL, NONE |
| Readout       | GLOBAL_SORT, GLOBAL_ATT, SET2SET, GLOBAL_MEAN, GLOBAL_MAX, GLOBAL_SUM, ZERO |
| Merge         | M_LSTM, M_CONCAT, M_MAX, M_MEAN, M_SUM |

### Pooling Module.
The pooling operations in our search space can be unified by a computation process as

\[
S = f_S(A, H), \quad \text{idx} = \text{TOPK}_k(S),
\]

(3)

\[
A' = A(\text{idx}, \text{idx}), H' = H(\text{idx}, :).
\]

(4)

where

\[
S = f_S(A, H), \quad \text{idx} = \text{TOPK}_k(S),
\]

and

\[
A' = A(\text{idx}, \text{idx}), H' = H(\text{idx}, :).
\]

(4)
based on the summation of different powers of the adjacency matrix, which is denoted as $S_u = \sum_{i=1}^{|O|} c_i o_i(x)$, where $x$ denotes the input representation of each module, and $c_i \in (0, 1)$ denotes the weight of the $i$-th operation $o_i(\cdot)$ in the set $O$. It is generated by one reparameterization trick, which is used as continuous relaxation function in NAS, as $c_i = g(\alpha, \alpha \in \mathbb{R}^{|O|}$. $o_i$ is the corresponding learnable parameter for $c_i$.

However, it is non-trivial to apply existing differentiable search algorithms in pooling architecture search. Different pooling operations produce diverse coarse graphs consisting of different nodes and edges as shown in Figure 2(c). How to relax the discrete coarsen graphs into continuous is still not solved in NAS. Here, to utilize the differentiable search algorithm, we design one coarsening strategy to address the pooling module relaxation problem.

### 3.3 Differentiable Search Algorithm

In this part, we develop a differentiable search algorithm. Specially, the designed coarsening strategy is detailed, which makes it feasible to continuously relax the selection of pooling operations.

Technically speaking, as done in existing NAS works [27, 45, 58], one needs to relax the depth of the search space to continuous one, thus the discrete selection of operations is relaxed by a weighted summation of all possible operations as

$$\bar{o}(x) = \sum_{i=1}^{|O|} c_i o_i(x),$$

where $x$ denotes the input representation of each module, and $c_i \in (0, 1)$ denotes the weight of the $i$-th operation $o_i(\cdot)$ in the set $O$. It is generated by one reparameterization trick, which is used as continuous relaxation function in NAS, as $c_i = g(\alpha, \alpha \in \mathbb{R}^{|O|}$. $o_i$ is the corresponding learnable parameter for $c_i$.

### Coarsening strategy.
For Aggregation, Readout and Merge Module in the supernetwork, we can generate the mixed results with relaxation function $g(\cdot)$ by constraining the node embedding dimension as shown in

$$H^{l_{i}} = \sum_{i=1}^{d} c_{i}^{l} o_{l_{i}}(A^{l-1}, H^{l-1}),$$
$$z^{l}_{i} = \sum_{i=1}^{d} c_{i}^{l} o_{l_{i}}(A^{l}, H^{l}),$$
$$z^{l} = \sum_{i=1}^{d} c_{i}^{l} o_{l_{i}}(z^{l}_{i}, z^{l-1}_{i}, \cdots, z^{l}_{i}),$$

where $c_{i}^{l_{i}}$ and $c_{i}^{l}$ denote the weights of $i$-th aggregation operation and global pooling operation in $l$-th layer, $c_{i}^{l}$ denotes the $i$-th operation in the Merge Module.

However, it is infeasible to directly compute the weighted summation of the output of all operations with Eq. (5) in the Pooling Module since the coarse graphs generated by different pooling operations contain diverse nodes and edges. As shown in Figure 2(c), the different node sets $\{0, 1, 4, 5\}$ and $\{0, 1, 2, 5\}$ are preserved by two pooling operations, and the features matrix $H^{l}_{i} \in \mathbb{R}^{d \times d}$ can not directly be added as the aggregation module in Eq. (6) due to the node mismatch problem, the edges are the same. Thus, it is unachievable to relax the Pooling Module, and the usage of gradient descent on this supernetwork is infeasible.

To address the challenge facing the Pooling Module in the supernetwork computation, we design a coarsening strategy to make the pooling module relaxation feasible, thus the supernetwork can be trained with gradient descent. In this coarsening strategy, we firstly calculate the node score matrix $S$ and select the top-$k$ nodes 1dks with Eq. (3) for each pooling operation. Rather than formulate the coarse graphs with Eq. (4) as in general pooling operation, as shown in Figure 2(c), we generate the result $C^{l}_{i} = (A^{l}, H^{l}_{i})$ of $i$-th pooling operation by making the coarse graph keep the “shape”, i.e., the same node numbers as the input graph $O^{l}_{i}$. By setting the unselected “part” to 0, i.e., the features of unselected nodes and weights of unselected edges, different pooling results $C^{l}_{i}$ can be added to generate the mixed results of the Pooling Module, which is shown.

### Algorithm 1 PAS - Pooling Architecture Search

**Require**: Training dataset $D_{train}$, validation dataset $D_{val}$, the epoch $T$ for search

**Ensure**: The searched architecture.

1. Randomly initialize the parameters $\alpha$ and $W$.
2. while $t = 1, \ldots, T$ do
3. for each minibatch $G_k \in D_{train}$ do
4. \hspace{1cm} Calculate the operation weights $C = g(\alpha)$.\n5. \hspace{1cm} Calculate the graph representation $z^{l}$ for each graph.
6. \hspace{1cm} Update $W$ with training loss $L_{train}$.
7. for each minibatch $G_k \in D_{val}$ do
8. \hspace{1cm} Calculate the operation weights $C = g(\alpha)$.\n9. \hspace{1cm} Calculate the graph representation $z^{l}$ for each graph.
10. Update $\alpha$ with validation loss $L_{val}$.
11. Preserve the operation with the largest weight in each module.
12. return The searched architecture.
in the following:

\[ C^l = (A^l, H^l) = \left( \sum_{i=1}^{O_i} c^l_i A^l_i, \sum_{i=1}^{O_i} c^l_i H^l_i \right) \]

where \( c^l_i \) denotes the weight of \( i \)-th pooling operation in \( l \)-th layer.

With the designed coarsening strategy, we can relax the pooling module into continuous and allow the usage of gradient descent. Therefore, it is easy to generate the graph representation \( z^l \) with Eq.(5) as in Figure 2(b).

**Optimization and deriving process.** In this paper, we choose the Gumbel-Softmax [18] as the continuous relaxation, which is designed to approximate discrete distribution in a differentiable manner and shown useful for supernet training in NAS [7, 45]. Since the training loss \( L_{\text{train}} \), validation loss \( L_{\text{val}} \) and the relaxation function are differentiable, we can optimize the parameters \( \alpha \) and \( W \) with gradient descent as shown in Alg. 1. After finishing the search process, we preserve the operations with the largest weights in each module, from which we obtain the searched architecture.

## 4 EXPERIMENTS

### 4.1 Experimental Settings

**Datasets.** In this paper, we use six datasets as shown in Table 3. D&D and PROTEINS datasets, provided by [6], are both protein graphs. In the D&D dataset, nodes represent the amino acids and two nodes are connected if the distance is less than 6 Å. In the PROTEINS dataset, nodes are secondary structure elements and edges represent nodes in an amino acid or in a close 3D space. IMDB-BINARY and IMDB-MULTI datasets, provided by [48], are movie-collaboration datasets that contain the actor/actress and genre information from IMDB. Nodes represent the actor/actress and edges mean they appear in the same movies. COX2 dataset, provided by [39], is a set of 467 cyclooxygenase-2 (COX-2) inhibitors and classify these compounds as active or inactive within vitro activities against human recombinant enzyme values. NCI109 dataset, provided in [38, 42], represents two balanced subsets of datasets of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines respectively.

| Dataset | # of Graphs | # of Feature | # of Classes | Avg.# of Nodes | Avg.# of Edges | Domain |
|---------|-------------|--------------|--------------|----------------|---------------|--------|
| D&D     | 1,178       | 89           | 2            | 384.3          | 715.7         | Bioinfo |
| PRO     | 1,113       | 3            | 2            | 39.1           | 72.8          | Bioinfo |
| IMDB-B  | 1,000       | 0            | 2            | 19.8           | 96.5          | Social |
| IMDB-M  | 1,500       | 0            | 3            | 13             | 65.9          | Social |
| COX2    | 467         | 3            | 2            | 41.2           | 43.5          | Chemistry |
| NCI109  | 4127        | 0            | 2            | 29.69          | 32.13         | Chemistry |

**Baselines.** We use 3 types of baselines: global pooling methods, hierarchical pooling methods and NAS methods for GNNs.

- **Global pooling methods**, we add a global pooling function at the end of existing node classification methods, e.g., GCN, GAT, SAGE, GIN, and combining with JK-Network [47] which add skip connections to these models, we can formulate another 4 methods which denoted as GCN-JK, GAT-JK, SAGE-JK, GIN-JK, respectively. We add the existing global pooling method DGCNN [53] in our experiment and MLP-Baseline which only contains MLPs and one global pooling function.

- For hierarchical pooling methods, we use 5 popular ones: Graph U-Net [11], DiffPool [49], SAGPool [23], ASAP [34] and MinCutPool [2].

Existing NAS methods focus on design aggregation and global pooling operations and lack the consideration of the pooling operations. To evaluate these global pooling NAS methods, we choose 4 representative methods to learn architectures based on the diverse search space and global pooling function. (a) GraphNAS [12]: a RL based method which design different aggregation operations in each layer; (b) GraphNAS-WS: GraphNAS with the weight sharing schema [33]; (c) SNAG [57]: a RL based method which design node aggregations, skip connections and layer aggregations in GNNs; (d) SNAG-WS: SNAG with the weight sharing schema. The evaluations of other global pooling NAS methods with diverse space and differentiable search algorithms is equivalent to the experiments in Section 4.3.2. Since existing methods cannot learn the data-specific pooling operations, we further provide 3 baselines for comparisons based on the proposed search space in Section 3.2. (e) EA based method (EA): update populations with mutation and crossover operations; (f) Random search (Random): samples an architecture from search space randomly; (g) Bayesian optimization (Bayesian) [1]: incorporates with prior information, use tree-structure Parzen estimator as the measurement to find a better architecture. In this respect, these 7 NAS baselines can cover the widely used search space and search algorithms, thus existing NAS methods can be fully evaluated in out experiment.

**Implementation details.** For NAS baselines and PAS, we derived the candidate GNNs from the corresponding search space in the search process. All the human-designed GNNs and the searched candidates are tuned individually with hyperparameters like embedding size, learning rate, dropout, etc. We perform 10-fold cross-validation to evaluate the performance based on the searched hyperparameters and report the averaged test accuracy and the standard deviations over 10 folds. Following the existing pooling architectures [23, 49], 2-layer backbone is chosen in this paper for all NAS baselines and PAS, and more layers can be trivially added along with the increasing number of nodes and edges on new graphs. For IMDB-MULTI dataset, which has a small node numbers, we choose the 1-layer backbone instead. The temperature is denoted 0.2 in Gumbel-Softmax. More experimental details of NAS and the experiments of layer numbers are given in Appendix.

### 4.2 Performance Comparisons

The results are given in Table 4, from which we can see that there is no absolute winner from human-designed models on all datasets. For example, GCN performs best on D&D while SAGPool performs best on IMDB-BINARY. Considering that these datasets are from three domains, it demonstrates the need for adaptive pooling architectures for graph classification. Besides, we can see that
Table 4: Performance comparisons of PAS and all baselines. We report the mean test accuracy and the standard deviation by 10-fold cross-validation. The best results in different groups of baselines are underlined, and the best result on each dataset is in boldface.

| Method    | D&D       | PROTEINS | IMDB-BINARY | IMDB-MULTI | COX2       | NCI109 |
|-----------|-----------|----------|-------------|------------|------------|--------|
| GCN       | 0.7812±0.0433 | 0.7484±0.0282 | 0.7267±0.0642 | 0.5040±0.0302 | 0.7923±0.0219 | 0.7344±0.0192 |
| GAT       | 0.7556±0.0372 | 0.7530±0.0372 | 0.7407±0.0453 | 0.4967±0.0430 | 0.8156±0.0417 | 0.7410±0.0245 |
| SAGE      | 0.7727±0.0406 | 0.7375±0.0297 | 0.7217±0.0529 | 0.4853±0.0343 | 0.8031±0.0594 | 0.7535±0.0164 |
| GIN       | 0.7540±0.0368 | 0.7448±0.0278 | 0.7167±0.0277 | 0.4980±0.0250 | 0.8309±0.0417 | 0.7456±0.0210 |

PAS consistently outperforms all baselines on all datasets, which demonstrates the effectiveness of PAS on searching for data-specific pooling architectures.

When it comes to NAS baselines, the performance gains of PAS are also significant. On one hand, compared to RL-based methods, i.e., GraphNAS and SNAG, the performance gains are mainly from the 4 modules in the designed search space, since GraphNAS and SNAG focus on designing the aggregation layers. On the other hand, compared with EA, Random and Bayesian, which use the designed search space of PAS, the performance gains are from the differentiable search algorithm on obtaining better architectures.

Further, we visualize the searched architectures for all datasets in Figure 3, from which it is clear that different operation combinations of these four modules are obtained on all datasets, i.e., data-specific architectures. Especially, a global pooling architecture in COX2 is obtained by PAS, since NONE is selected in the Pooling Module of the first layer, while a hierarchical pooling architecture is obtained on D&D dataset. This observation further indicates the flexibility of the designed search space of PAS. Besides, we show the test accuracy and model size comparisons among these methods in Figure 4. Compared with baselines, the searched architectures, which are shown in Figure 4 and denoted as “PAS (searched)” in Figure 4, can achieve the SOTA performance with moderate size in terms of the model parameters. It indicates the effectiveness of our method in finding the expressive pooling architectures. More results can be found in Appendix.

Therefore, these results demonstrate the need for data-specific methods for graph classification, and at the same time, the effectiveness of PAS on designing adaptive pooling architectures.

Table 5: Performance of PAS using different search spaces. The first column represents the corresponding module we try to evaluate by fixing it with one OP in the reduced search space.

| Aggregation | Fixed        | D&D     | IMDB-MULTI   |
|-------------|--------------|---------|--------------|
| PAS-GCN     | 0.7835±0.0407 | 0.5027±0.0409 |
| PAS-GAT     | 0.7878±0.0376 | 0.5087±0.0417 |
| Pooling     | 0.7708±0.0350 | 0.5173±0.0447 |
| Readout     | 0.7466±0.0472 | 0.5033±0.0436 |
| Merge       | 0.7682±0.0336 | 0.5047±0.0380 |
| PAS         | 0.7896±0.0368 | 0.5220±0.0373 |

4.3 Ablation Studies on the Search Space

We conduct ablation studies to show the influences of the four modules in the search space. For simplicity, we use two datasets: D&D and IMDB-MULTI, and run PAS over different variants of search space, for which the results are shown in Table 5.

4.3.1 Aggregation Module. To evaluate how the Aggregation Module affects the performance, we only search for the other three modules based on fixed aggregators GCN and GAT, which denoted as PAS-GCN and PAS-GAT, respectively. As shown in Table 5, with fixed aggregators, PAS-GCN and PAS-GAT have a performance drop compared with PAS. Besides, PAS-GAT has a better performance than PAS-GCN, which is consistent with existing works [40].

This observation demonstrates the importance of including Aggregation Module in the search space, which can also explain the superiority of PAS over human-designed hierarchical pooling methods in Table 4, e.g., DiffPool and SAGPool, both of which use fixed aggregation functions. Especially, DiffPool tries to learn the pooling...
functions. Thus, it shows that the aggregation operations should also be data-specific for graph classification.

4.3.2 Pooling Module. Due to the introduction of NONE operation in the Pooling Module in the search space, PAS can automatically emulate both hierarchical and global pooling architectures, which is one of the advantages of the proposed method. As shown in Figure 3(e), the searched architecture for COX2 correspond to a global pooling method.

Then to evaluate the usefulness of the Pooling Module in the search space, we search for the combinations of operations in the other three modules based on fixed pooling operation NONE. Thus it is equivalent to the global pooling method. As shown in Table 5, the performance of PAS-Global drops significantly compared to PAS, which demonstrates the importance of the designed Pooling Module in the search space. In other words, the hierarchical information is useful for obtaining high-quality graph-level representation, which has been demonstrated in existing works, like DiffPool or SAGPool. More interestingly, speaking of the global pooling manner, PAS-Global can be treated as one representative method of global pooling NAS methods, e.g., SANE [58], DSS [26] and GNAS [3] which design aggregation layers with the differentiable search algorithm based on one global pooling function, and RE-MPNN [19] which design aggregation layers and global pooling functions with EA. Thus, the performance drop on PAS-Global can demonstrate the superiority of PAS over those global pooling NAS methods.

4.3.3 Readout and Merge Module. In this section, we evaluate the proposed Readout and Merge Module, which are novel compared to existing pooling architectures. By fixing the global pooling function as GLOBAL_MEAN, we create the variant PAS-FR, which means that we do not search for different global pooling functions. By removing the Merge Module, we only preserve the last global pooling function, whose output is used as the graph representation $\mathbf{z}$. This variant is denoted by PAS-RM, which means the outputs of intermediate layers are not used. From Table 5, we can see that

- The performance drop of PAS-FR means that it is far from satisfying to use a simple mean function to generate fixed-size representation out of all nodes in a graph. As shown in Figure 3, complex global pooling functions like GLOBAL_ATT and SET2SET are selected on real-world datasets.
- The performance drop of PAS-RM means that the outputs of intermediate layers are important for the final representation, which have been shown in previous works [4, 47]. Thus, it demonstrates the importance of the proposed Merge Module.

Taking all results in Table 5 into consideration, we can see that it is important for graph classification to search for combinations of operations from the four essential modules by PAS, which demonstrates the contribution of the designed search space.

4.4 The Efficiency of PAS

In this section, we show the efficiency superiority of the differentiable search process of PAS, which relies on the proposed coarsening strategy, over NAS baselines.

As shown in Figure 5, we compare the search cost of all NAS methods. The search cost of PAS is the smallest among all NAS baselines, which is mainly attributed to the designed differentiable search algorithm, thus the coarsening strategy.

Compared with GraphNAS and SNAG, which focus on searching for aggregation functions and directly use global pooling methods, the search space of PAS is more expressive as analyzed in Section 4.3.2 and has a moderate size. Besides, with the designed coarsening strategy, PAS can be optimized with gradient descent and achieve the two orders of magnitude reduction of search cost.
Figure 5: (Best viewed in color) The search cost of each model on all datasets. The size of each circle represents the size of the search space each method uses. GraphNAS and SNAG have the largest and smallest search space, respectively. EA, Random, Bayesian and PAS use the same search space, thus the circles are of the same size.

On the contrast, these RL based methods need thousands of evaluations which are inefficient in particular.

Besides, from Figure 5, we can see that based on the proposed search space in Section 3.2, EA, Random, and Bayesian are much slower than PAS, especially on D&D and NCI109, the size or the number of the graphs are much larger than others, the efficiency gain of PAS is much larger. Combining with the SOTA performance in Table 4, PAS is efficient and effective in learning data-specific pooling architectures, which indicates that the coarsening strategy not only brings efficiency improvement, but also the performance gain.

To summarize, based on the unified framework and coarsening strategy, we design an effective search space and an efficient differentiable search algorithm, which can learn data-specific pooling architectures for graph classification.

4.5 More Results of GraphGym

GraphGym [51] was proposed to evaluate the design dimensions of GNN models, like aggregation functions, number of layers, etc. However, for the graph classification task, GraphGym only uses a fixed global pooling function to generate the graph representations based on node aggregation operations. It cannot evaluate different pooling methods for graph classification.

To further evaluate the existing pooling methods, we add pooling and readout layers for graph classification on top of GraphGym as shown in Figure 6. The four candidate operations in pooling layer which denoted as Global, ASAP, SAGPool and Graph U-Net in Figure 1, corresponding to the NONE, ASAP, SAGPOOL and TOPKPOOL in the proposed search space, respectively. Due to the space limit, more experimental details of GraphGym are given in Appendix.

The results are shown in Figure 7, the upper ones show the average test accuracy rank of each operation on these 420 setups and the bottom ones show the distribution of the accuracy ranking.

In pooling layer, ASAP and SAGPOOL have a lower average rank and a lower probability to rank last than NONE operation. Combine with the Figure 1, none of these architectures can outperform the other methods, hierarchical pooling methods have advantages over global methods in general. The left part of Figure 1 is created by these results. In readout layer, we use 4 global pooling functions: GLOBAL_MEAN, GLOBAL_MAX, GLOBAL_SUM and GLOBAL_ATT. GLOBAL_SUM have a slightly lower average rank over the other 3 functions, GLOBAL_ATT have a lower probability to rank last, which means there exist no general global pooling functions that can perform well on various datasets and GNN settings.

Taking into consideration these experimental results from Figure 1 and 7, it shows the need for finding the data-specific pooling architectures for graph classification, which motivates the proposal of PAS in this work.

Figure 6: The new search space for graph classification task based on GraphGym.

Figure 7: Rank analysis of the operations in pooling and readout layers. Lower is better.

5 CONCLUSION

In this paper, we propose a novel framework PAS to automatically learn data-specific pooling architectures for graph classification task. By revisiting various human-designed pooling architectures for graph classification, we design a unified framework consisting of four essential modules for graph classification. Based on this framework, an effective search space is designed by incorporating popular operations from existing human-designed architectures. To enable efficient architecture search, we develop a coarsening strategy to continuously relax the search space, thus a differentiable search method can be adopted. To demonstrate the effectiveness of PAS for graph classification, we conduct extensive experiments on
six datasets from three domains. The experimental results show that PAS can not only search SOTA data-specific pooling architectures for graph classification, but also performs very efficiently than NAS baselines.

For future work, we plan to investigate in depth the connections between the graph properties and the learned pooling architectures, which can help better understanding the graph classification task. Besides, we plan to further evaluate PAS on large-scale datasets, e.g., OGB benchmark [17].
A DETAILS OF EXPERIMENT SETTINGS

A.1 The implementation details of PAS

All models are implemented with Pytorch [31] on a GPU 2080Ti (Memory: 12GB, Cuda version: 10.2). Thus, for consistent comparisons of baseline models, we use the implementation of all GNN baselines by the popular GNN library: Pytorch Geometric (PyG) (version 1.6.1) [9], which provides a unifying code framework for various GNN models. Further, we adopt the same data-preprocessing manner by PyG and split data by means of a stratification technique with the same seed.

For all human-designed global and hierarchical pooling baselines, we search the layer numbers of this method, global pooling functions, embedding size, dropout rate, and learning rate as shown in Table 6. Following the DiffPool [49], we set the pooling rate \( \kappa = \frac{1}{L} \times N \) in Eq. (3) for all pooling operations where \( L \) is the layer number of this method and \( N \) is the node number. For each model, select 30 hyperparameters settings with Hyperopt [5], and evaluate each setting on 10-fold cross-validation data. We select settings based on mean validation accuracy then report the final test accuracy and the standard deviations. Besides, we also provide the comparisons of baseline reported results in Table 7 to show the influence of different settings.

We set training and finetuning stages to get the 10-fold cross-validation test accuracy for all NAS baselines and PAS in this paper. In the training stage, the dataset is split into 80% for training, 10% for validation and test with the same seed, and we select architectures from the supernet as shown in Alg. 1. In the finetuning stage, we tune these searched architectures over the pre-defined space as shown in Table 8 with Hyperopt based on 10-fold data, and select the final candidate with mean validation accuracy.

For all RL-based methods, GraphNAS [7], GraphNAS-WS, SNAG [8] and SNAG-WS, we set the training epoch to 200. In each training epoch, we sample 10 architectures and use the validation accuracy to update the controller parameters. After training finished, we sample 5 candidates with the controller.

For EA baseline, we follow the experiments in [15]. We set the population size to 50 and the training epoch to 40. In each training epoch, random select an architecture and mutate all operations with probability 0.1 to generate the new architecture; random select two architectures and crossed to generate one new architecture, 25 mutation operations and 25 crossover operations in each training epoch. The architecture is derived based on the validation accuracy after the training stage terminates.

For Random and Bayesian methods, we set the training epoch to 200. In each training epoch, sample one architecture and train from scratch. After training finished, we select one candidate with the validation accuracy.

For PAS, we set the training epoch to 200 as shown in Alg. 1. In each training epoch, PAS samples a set of minibatches and uses the training loss to update parameters \( \mathbf{W} \) and use the validation loss to update \( \mathbf{a} \). After search process is finished, we derive the candidate architecture from the supernet. Repeat 5 times with different seeds, we can get 5 candidates.

In the finetuning stage, each candidate architecture owns 30 hyper steps. In each hyper step, a set of hyperparameters will be sampled from Table 8 based on Hyperopt, then we generate final performance on 10-fold data. We choose the hyperparameters for each candidate with the mean validation accuracy. After that, we choose the candidate with the mean validation accuracy then report the final test accuracy and the standard deviations based on 10-fold cross-validation data.

Table 6: Hyperparameter space for human-designed baselines.

| Dimension       | Operation         |
|-----------------|-------------------|
| Layer number    | 1, 2, 3, 4, 5     |
| Global pooling function | GLOBAL_MEAN, GLOBAL_SUM |
| Embedding size  | 16, 32, 64, 128, 256 |
| Dropout rate    | 0, 0.1, 0.2, 0.5   |
| Learning rate   | [0.001, 0.025]    |

Table 7: The reported results of 3 methods (denoted as Method1) in other methods (denoted as Method2).

| Method 1       | Method 2    | D&D   | PROTEINS |
|---------------|-------------|-------|----------|
| DiffPool [49] | FAIR [8]    | 0.7500| 0.7370   |
|               | SAGPool [23]| 0.6695| 0.6820   |
|               | DiffPool [49]| 0.8064| 0.7625   |
|               | Graph U-Net [11]| 0.8064| 0.7625   |
|               | ASAP [34]    | 0.6695| 0.6820   |
|               | PAS          | 0.7775| 0.7355   |
| DGCNN [53]    | FAIR [8]    | 0.7660| 0.7290   |
|               | Graph U-Net [11]| 0.7937| 0.7554   |
|               | DiffPool [49]| 0.7937| 0.7626   |
|               | SAGPool [23]    | 0.7253| 0.6672   |
|               | ASAP [34]    | 0.7187| 0.7391   |
|               | PAS          | 0.7666| 0.7357   |
| Graph U-Net [11]| ASAP [34]   | 0.7501| 0.7110   |
|               | SAGPool [23]    | 0.7501| 0.7110   |
| GraphSAGE [16]| FAIR [8]    | 0.7290| 0.7300   |
|               | DiffPool [49]| 0.7542| 0.7048   |
|               | PAS          | 0.7727| 0.7375   |

Table 8: Hyperparameter space in the finetuning stage for NAS methods.

| Dimension       | Operation         |
|-----------------|-------------------|
| Embedding size  | 16, 32, 64, 128, 256 |
| Dropout rate    | 0, 0.1, 0.2, 0.5   |
| Learning rate   | [0.001, 0.025]    |
| Optimizer       | Adam, AdaGrad     |
| Activation function | RELU, ELU          |
A.2 The implementation details of GraphGym

In this section, we show the details of the designed experiments as mentioned in Figure 1.

Very recently, GraphGym [51] was proposed to evaluate the design dimensions of GNN models, like aggregation functions, number of layers, etc. However, for the graph classification task, GraphGym only uses a fixed global pooling function to generate the graph representations based on node aggregation operations. It cannot evaluate different pooling methods for graph classification. To further evaluate the pooling methods, we design a search space for graph classification on top of GraphGym. As shown in Figure 6, we add the pooling layer behind the 1-st, 3-rd, 5-th GNN layer, and add one Readout layer in the post-process stage. The search dimensions are shown in Table 9.

We use the 6 real-world datasets and 8 synthetic datasets mentioned in GraphGym. Each synthetic dataset chooses graph structure function from {scalefree, smallworld} and choose feature function from {clustering, const, onehot, pagerank}. 256 graphs are generated with different Average Clustering Coefficient and Average Path Length. We use 420 setups on these 14 datasets so that each dataset has 30 hits on average, and the results are shown in Figure 1 and 7.

B EXPERIMENTS

B.1 More figures

For sake of the space, we only show the test accuracy and model size comparisons on 2 datasets in Figure 4. Here, more figures are shown in Figure 9.

B.2 The influence of layers

Here we conduct experiments to show the influences of the layer number $L$ of PAS by varying $L \in \{1, 2, 3, 4, 5, 6\}$ in PAS. As introduced in Section 3.2 and Figure 2(b), each layer consists of one Aggregation and Pooling Module in PAS. The results are shown in Figure 8, from which we can see that the trend that the performance increases firstly with the increase of $L$ and then decreases.

Looking back at the statistics of the datasets in Table 3, we can get an empirical guideline for architecture design for graph classification, although it may not always be true, that the larger number of layers is preferable for graphs in the larger size (more nodes).