The expressive power of pooling in Graph Neural Networks

Filippo Maria Bianchi[1][0000−0002−7145−3846] and Veronica Lachi[2][0000−0002−6947−7304]

[1] UiT the Arctic University of Norway
[2] University of Siena

Abstract. In Graph Neural Networks (GNNs), hierarchical pooling operators generate local summaries of the data by coarsening the graph structure and the vertex features. Considerable attention has been devoted to analyzing the expressive power of message-passing (MP) layers in GNNs, while a study on how graph pooling affects the expressiveness of a GNN is still lacking. Additionally, despite the recent advances in the design of pooling operators, there is not a principled criterion to compare them. In this work, we derive sufficient conditions for a pooling operator to fully preserve the expressive power of the MP layers before it. These conditions serve as a universal and theoretically-grounded criterion for choosing among existing pooling operators or designing new ones. Based on our theoretical findings, we analyze several existing pooling operators and identify those that fail to satisfy the expressiveness conditions. Finally, we introduce an experimental setup to verify empirically the expressive power of a GNN equipped with pooling layers, in terms of its capability to perform a graph isomorphism test.

Keywords: Pooling operators · Graph Neural Networks · Expressive power of Graph Neural Networks.

1 Introduction

In recent years, there has been extensive research focused on studying the theoretical properties of Graph Neural Networks (GNNs), including generalization [24,21], explainability [2,26] and scalability [13]. In particular, significant effort has been devoted to characterizing the expressive power of GNNs in terms of their capabilities for testing graph isomorphism [33]. GNNs with appropriately formulated message-passing (MP) layers are as effective as the Weisfeiler-Lehman (WL) test in distinguish graphs [36], while higher-order GNN architectures can match the expressiveness of the k-WL test [31]. Despite the progress made in understanding the expressive power of GNNs, the results are still limited to *flat* GNNs consisting of a stack of MP layers followed by a final readout [36,6].

Inspired by pooling in convolutional neural networks, recent works introduced hierarchical pooling operators that enable GNNs to learn increasingly abstract and coarser representations of the input graphs [37,11]. By interleaving MP with

pooling layers that gradually distill global graph properties through the computation of local graph summaries, it is possible to build deep GNNs that improve the accuracy in graph classification [10,4] and node classification tasks [20,27].

It is not straightforward to evaluate the power of a graph pooling operator and the quality of the coarsened graphs it produces. The most common approach is to simply measure the performance of a GNN with pooling layers on a downstream task, such as graph classification. However, such an approach is highly empirical and provides a rather indirect evaluation that is affected by external factors. One factor is the overall GNN architecture: the pooling layers are combined with different MP layers, activation functions, normalization or dropout layers, and specific optimization algorithms, which makes it difficult to disentangle the contribution of the individual components. Another factor is the dataset at hand: some classification tasks only require isolating a specific motif in the graph [24,8], while others require considering global properties that depend on the whole graph structure [17]. Two new criteria were recently proposed to evaluate a graph pooling operator in terms of the spectral similarity between the original and the coarsened graph topology and its capability of reconstructing the features of the original graph from the coarsened one [23]. While providing valuable insights, these criteria give results that are, to some extent, contrasting and in disagreement with the traditional evaluation based on the performance of the downstream task.

To address this issue, we introduce a universal and principled criterion that quantifies the power of a pooling operator as its capability to retain the information in the graph from an expressiveness perspective. In particular, we show that when certain conditions are met in the MP layers and in the pooling operator, their combination produces an injective function between graphs. This implies that the GNN can effectively coarsen the graph to learn high-level data descriptors, without compromising its expressive power. Based on our theoretical analysis, we identify commonly used pooling operators that do not satisfy these conditions and may lead to failures in certain scenarios. Finally, we introduce a simple yet effective experimental setup for measuring, empirically, the expressive power of any GNN in terms of its capability to perform a graph isomorphism test. Besides providing a criterion for choosing among existing pooling operators and for designing new ones, our findings allow us to debunk criticism and misconceptions about graph pooling.

2 Background

2.1 Graph neural networks

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with node features $\mathcal{X}^0 = \{\boldsymbol{x}_i^0\}_{i=1}^N$, where $\{\cdot\}$ denotes a multiset and $|\mathcal{V}| = N$. The principal operations performed by GNNs are those of the MP layers [22], which implement a local computational mechanism to process graphs. Specifically, the information related to a node v that is stored in a feature vector $\mathbf{h}_v$, is updated by combining the features of neighboring nodes. After l

iterations, the vector $\mathbf{h}_v^l$ embeds both the structural information and the node content of the l–hop neighborhood of v. With enough iterations, the node feature vectors can be used to classify the nodes or the entire graph. More rigorously, the output of the l-th layer of a MP-GNN is:

$$\boldsymbol{x}_v^l = \texttt{COMBINE}^{(l)}(\boldsymbol{x}_v^{l-1}, \texttt{AGGREGATE}^{(l)}(\boldsymbol{x}_u^{l-1},\, u \in \mathcal{N}[v]\})) \tag{1}$$

where $\texttt{AGGREGATE}^{(l)}$ is a function that aggregates the node features from the neighborhood $\mathcal{N}[v]$ at the $(l-1)$–th iteration, and $\texttt{COMBINE}^{(l)}$ is a function that combines the own features with those of the neighbors.

2.2 Expressive power of graph neural networks

When analyzing the expressive power of GNNs, the primary objective is to evaluate their capacity to produce different outputs for non-isomorphic graphs. While an exact test for graph isomorphism has a combinatorial complexity [3], the WL test for graph isomorphism [35] is an effective and computationally efficient test that can distinguish a broad range of graphs, with some exceptions, such as strongly regular graphs [12]. The algorithm assigns to each graph vertex a color that depends on the multiset of labels of its neighbors and on its own color. At each iteration, the colors of the vertices are updated until convergence is reached.

There is a strong analogy between an iteration of the WL-test and the aggregation scheme implemented by MP in GNNs. In fact, it has been proved that MP-GNNs are at most as powerful as the WL test in distinguishing different graph-structured features [36,31]. Moreover, if the MP operation is injective, then the resulting MP-GNN is as powerful as the WL test [36]. The Graph Isomorphism Network (GIN) implements such injective multiset function as follows:

$$\boldsymbol{x}_v^l = \texttt{MLP}^{(l)}\left((1+\epsilon^l)\boldsymbol{x}_v^{l-1} + \sum_{u \in \mathcal{N}[v]} \boldsymbol{x}_u^{l-1}\right). \tag{2}$$

Under the condition that the nodes' features are a countable multiset, the representational power of GIN equals that of the WL test. Some GNNs can surpass the discriminative power of the WL test by using higher-order generalizations of MP operation [31], or by using a composition of invariant and equivariant functions [28], at the price of higher computational complexity. In this work, we focus on the standard MP-GNN, which remains the most widely adopted due to its computational efficiency.

2.3 Graph pooling operators

A graph pooling operator implements a function $\texttt{POOL} : \mathcal{G} \mapsto \mathcal{G}_P = (\mathcal{V}_P, \mathcal{E}_P)$ such that $|\mathcal{V}_P| = K$, with $K \leq N$ and $\mathcal{X}_P = \{\boldsymbol{x}_{P_i}\}_{i=1}^K$ is the multiset of the pooled nodes features. To formally describe the POOL function, we adopt the Select-Reduce-Connect (SRC) framework [23], that expresses a graph pooling operator through the combination of three functions: *selection*, *reduction*, and *connection*.

The selection function (SEL) clusters the nodes of the input graph into subsets called *supernodes*, namely $\texttt{SEL} : \mathcal{G} \mapsto \mathcal{S} = \{\mathcal{S}_1, \dots, \mathcal{S}_K\}$ with $\mathcal{S}_j = \left\{s_i^j\right\}_{i=1}^N$ where s_i^j is a membership score that measures how much node i contributes to supernode j. Typically, a node can be assigned to zero, one, or several supernodes, each with different scores. The reduction function (RED) creates the pooled vertex features multiset by aggregating the features of the vertices assigned to the same supernode, that is, $\texttt{RED} : (\mathcal{G}, \mathcal{S}) \mapsto \mathcal{X}_P$. Finally, the connect function (CON) generates the edges (and, potentially, the edge features) by connecting the supernodes $\mathcal{E}_P = \{\texttt{CON}(\mathcal{G}, \mathcal{S}_m, \mathcal{S}_l)\}_{m,l=1}^K$.

Most of the existing graph pooling operators can be specified by a particular implementation of the SRC functions. It is worth noting that the input and output of the SRC functions can be represented as both multisets and matrices. In Section 3.1 we use the multisets representation as it facilitates presenting our main result, while in the rest of the paper, we adopt the matrix representation.

3 Expressive power of graph pooling operators

We define the expressive power of a graph pooling operator as its capability of preserving the expressive power of the MP layers that came before it. We first present our main result, which is a formal criterion to verify the expressive power of a pooling operator. In particular, we provide three sufficient (though not necessary) conditions ensuring that if the MP and the pooling layers meet certain criteria, then the latter retains the same level of expressive power as the former. Then, we analyze several existing pooling operators and analyze their expressive power based on those criteria.

3.1 Conditions for expressiveness

Theorem 1. *Let $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ with $|\mathcal{V}_1| = N$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ with $|\mathcal{V}_2| = M$ with node features $\mathcal{X}_1$ and $\mathcal{X}_2$ respectively, such that $\mathcal{G}_1 \neq_{WL} \mathcal{G}_2$. Let $\mathcal{G}_1^L$ and $\mathcal{G}_2^L$ be the graph obtained after applying a block of L MP layers such that $\mathcal{X}_1^L = \left\{\boldsymbol{x}_i^L\right\}_{i=1}^N$ and $\mathcal{X}_2^L = \left\{\boldsymbol{y}_i^L\right\}_{i=1}^M$ are the new multisets of nodes features. Let* POOL *be a pooling operator expressed by the functions* SEL, RED, CON*, which is placed after the MP layers. Let $\mathcal{G}_{1_P} = \texttt{POOL}(\mathcal{G}_1)$ and $\mathcal{G}_{2_P} = \texttt{POOL}(\mathcal{G}_2)$ with $|\mathcal{V}_{1_P}| = |\mathcal{V}_{2_P}| = K$. Let $\mathcal{X}_{1_P} = \{\boldsymbol{x}_{P_i}\}_{j=1}^K$ and $\mathcal{X}_{2_P} = \left\{\boldsymbol{y}_{P_j}\right\}_{j=1}^K$ be the nodes features of the pooled graphs. If the following conditions hold:*

1. $\sum_i^N \boldsymbol{x}_i^L \neq \sum_i^M \boldsymbol{y}_i^L$;
2. *For each node i, the memberships generated by* SEL *satisfy $\sum_{j=1}^K s_i^j = \lambda$, with $\lambda > 0$;*
3. *The function* RED *is of type* $\texttt{RED} : \left\{\left\{(\boldsymbol{x}_i^L, s_i^j)\right\}_{i=1}^N\right\}_{j=1}^K \mapsto \mathcal{X}_P = \left\{\sum_{i=1}^N \boldsymbol{x}_i^L \cdot s_i^j\right\}_{j=1}^K$;

then $\mathcal{X}_{1_P} \neq \mathcal{X}_{2_P}$.

The proof can be found in Appendix A and a schematic summary is in Fig. 1.

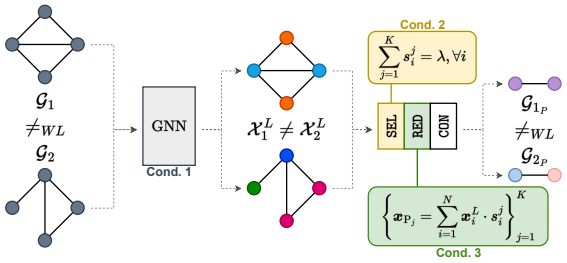

Fig. 1: A GNN with expressive MP layers (condition 1) extracts different features $\mathcal{X}_1^L$ and $\mathcal{X}_2^L$ for two graphs $\mathcal{G}_1$, $\mathcal{G}_2$ that are WL-distinguishable. A pooling layer satisfying the conditions 2 and 3 generates coarsened graphs $\mathcal{G}_{1_P}$ and $\mathcal{G}_{2_P}$ that are still WL-distinguishable.

When the three conditions are satisfied, the combination of the MP layers and pooling operator results in an injective function between graphs. When using a powerful MP layer such as GIN [36], there are theorems for functions defined on sets that guarantee condition 1 to be met. In particular, the sum over a multiset that is countable is an injective function [38]. When $\mathcal{G}_1 \neq_{\text{WL}} \mathcal{G}_2$ a GNN block with enough GIN layers produces different multisets of node features $\mathcal{X}_1^L \neq \mathcal{X}_2^L$. Note that this is true also when $\mathcal{G}_1$ and $\mathcal{G}_2$ have the same number of nodes, i.e., Th. 1 holds for $|\mathcal{V}_1| = |\mathcal{V}_2| = N$.

If the pooling operator satisfies conditions 2 and 3, it will also produce multisets of node features so that $\mathcal{X}_{1_P} \neq \mathcal{X}_{2_P}$. Due to the injectiveness of the coloring function of the WL algorithm, two graphs with different multisets of node features will be classified as non-isomorphic by the WL test and, therefore, $\mathcal{G}_{1_P} \neq_{\text{WL}} \mathcal{G}_{2_P}$. This means that the pooling operator effectively coarsens the graphs while retaining all the information necessary to differentiate between them and that the composition of GIN layers and appropriate pooling operator maps non-WL equivalent graphs $\mathcal{G}_1 \neq_{\text{WL}} \mathcal{G}_2$ into non-WL equivalent graphs $\mathcal{G}_{1_P} \neq_{\text{WL}} \mathcal{G}_{2_P}$.

Condition 2 implies that all nodes in the original graph must contribute to the supernodes. Moreover, letting the sum of the memberships s_i^j to be a constant λ (usually, $\lambda = 1$), places a restriction on the formation of the super-nodes. Condition 3 requires that the features of the supernodes $\mathcal{X}_P$ are a convex combination of the node features $\mathcal{X}^L$. It is important to note that the conditions for the expressiveness only involve `SEL` and `RED`, but not the `CON` function. Indeed, both the graph's topology and the nodes' features are embedded in the features of the supernodes by MP and pooling layers satisfying the conditions of Th. 1. Nevertheless, even if a badly-behaved `CON` function does

not affect the expressiveness of the pooling operator, it can still compromise the effectiveness of the MP layers that come afterward. This will be discussed further in Sections 3.3 and 4.

3.2 Expressiveness of existing pooling operators

The SRC framework allows building a comprehensive taxonomy of the existing pooling operators, based on the density of supernodes, the trainability of the `SEL`, `RED`, and `CON` functions, and the adaptability of the number of supernodes K [23]. The density of a pooling operator is defined as the expected value $\mathbb{E}[|\mathcal{S}_k|/N]$, which is the ratio between the cardinality of a supernode $\mathcal{S}_k$ and the number of nodes in the graph $\mathcal{G}$. A method is referred to as *dense* if the supernodes have cardinality $O(N)$, whereas a pooling operator is considered *sparse* if the supernodes generated have constant cardinality $O(1)$ [23].

Pooling methods can also be distinguished according to the number of nodes K of the pooled graph. If K is constant and independent of the input graph size, the pooling method is *fixed*. On the other hand, if the number of supernodes is a function of the input graph, the method is *adaptive*. Finally, the in some pooling operators the `SEL`, `RED`, and `CON` functions can be learned end-to-end along with the other components of the GNN architecture. In this case, the method is said to be *trainable*, meaning that the operator has parameters that are learned by optimizing a task-driven loss function. Otherwise, the methods are *non-trainable*.

Dense pooling operators Prominent methods in this class of pooling operators are DiffPool [37], MinCutPool [10], and DMoN [34]. Besides being dense, all these operators are also trainable and fixed. DiffPool, MinCutPool, and DMoN compute a cluster assignment matrix $\mathbf{S} \in \mathbb{R}^{N \times K}$ either with an MLP or an MP-layer, which are fed with the node features $\mathbf{X}^L$ and end with a `softmax`. The main difference among these methods is in how they define unsupervised auxiliary loss functions, which are used to inject a bias in how the clusters are formed. Thanks to the `softmax` normalization, the cluster assignments sum up to one, ensuring condition 2 of Th. 1 to be satisfied. Moreover, the pooled node features are computed as $\mathbf{X}_p = \mathbf{S}^{\mathrm{T}}\mathbf{X}^L$, making also condition 3 satisfied.

There are dense pooling operators that use algorithms such as non-negative matrix factorization [5] to obtain a cluster assignment matrix $\mathbf{S}$, which may not satisfy condition 2. Nonetheless, it is always possible to apply a suitable normalization to $\mathbf{S}$ to ensure that its rows sum up to one. Therefore, we claim that all dense methods preserve the expressive power of the preceding MP layers.

Non-expressive sparse pooling operators Members of this category are Top-k [20,24] ASAPool [32], SAGPool [25] and PanPool [27], which are also trainable and adaptive. These methods reduce the graph by selecting a subset of its nodes based on a ranking score and they mainly differ in how their `SEL` function computes such a score. Specifically, the Top-k method ranks nodes based on a score obtained by multiplying the node features with a trainable projection vector. A node i is kept ($s_i = 1$) if is among the top-K in the ranking and is

discarded ($s_i = 0$) otherwise. SAGPool simply replaces the projection vector Top-k with an MP layer to account for the graph's structure when scoring nodes. ASAPool, instead, examines all potential local clusters within the input graph given a fixed receptive field and it employs an attention mechanism to compute the cluster membership of the nodes. The clusters are subsequently scored using a GNN. Finally, in PanPool the scores are obtained from the diagonal entries of a maximal entropy transition matrix, which is a transition matrix that generalizes the graph Laplacian.

Regardless of how the score is computed, all these methods generate a cluster assignment matrix $\mathbf{S}$ where not all the rows sum to one. Indeed, if a node is not selected, it is not assigned to any supernode in the coarsened graph. Therefore, these methods fail to meet condition 2 of Theorem 1. Additionally, all these methods share the same `RED`, which involves multiplying the features of each selected node by its ranking score, making condition 3 also unsatisfied.

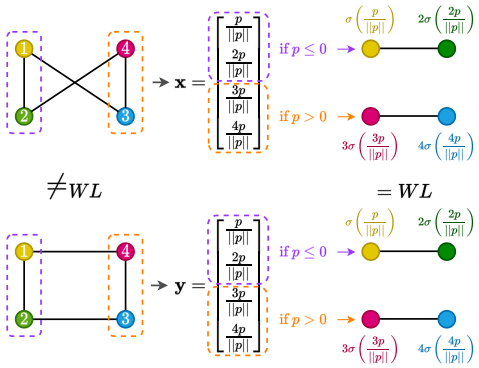

Fig. 2: Example of failure of Top-k pooling operator. Regardless of the value learned for the projector p, two WL-distinguishable graphs are mapped into the same coarsened graph.

Intuitively, these operators produce a pooled graph that is a subgraph of the original graph and discard the content of the remaining parts. This hinders the ability to retain all the necessary information for preserving the expressiveness of the preceding MP layers. The limitation of Top-k is exemplified in Fig. 2: regardless of the projector p, Top-k maps two WL-distinguishable graphs into two isomorphic graphs, meaning that it cannot preserve the partition on graphs induced by the WL test.

Expressive sparse pooling operators Not all sparse pooling operators coarsen the graph by selecting a subgraph. In fact, some of them assign each node in the original graph to exactly one supernode and, thus, satisfy condition 2 of Th. 1. In matrix form, the cluster assignment would be represented by a sparse matrix $\mathbf{S}$ that satisfies $\mathbf{S}\mathbf{1}_K = \mathbf{1}_N$ and where every row has one entry equal to one and the

others equal to zero. Within this category of sparse pooling operators, notable examples include Graclus [15], ECPool [16], and k-MISPool [4].

Graclus is a non-trainable, greedy bottom-up spectral clustering algorithm, which matches each vertex with the neighbor that is closest according to the graph connectivity [15]. When Graclus is used to perform graph pooling, the `RED` function is usually implemented as a `max_pool` operation between the vertices assigned to the same cluster [14]. In this work, to ensure condition 3 of Th. 1 to be satisfied, we use a `sum_pool` operation instead. Contrarily from Gralcus, ECPool and k-MISPool are trainable. ECPool first assigns to each edge $e_{i\to j}$ a score $r_{ij} = f(\boldsymbol{x}_i, \boldsymbol{x}_j; \boldsymbol{\Theta})$. Then, iterates over each edge $e_{i\to j}$, starting from those with higher scores, and contracts it if neither nodes i and j are attached to an already contracted edge. The endpoints of a contracted edge are merged into a new supernode $\mathcal{S}_k = r_{ij}(\boldsymbol{x}_i + \boldsymbol{x}_j)$, while the remaining nodes become supernodes themselves. Since each supernode either contains the nodes of a contracted edge or is a node from the original graph, all columns of $\mathbf{S}$ have either one or two entries equal to one, while each row sums up to one. The `RED` function can be expressed as $\mathbf{r} \odot \mathbf{S}^T\mathbf{X}^L$, where $\mathbf{r}[k] = r_{ij}$ if k is the contraction of two nodes i j, otherwise $\mathbf{r}[k] = 1$. As a result, ECPool met the expressiveness conditions of Th. 1. Finally, k-MISPool identifies the supernodes with the centroids of the maximal k-independent sets of a graph [7]. To speed up computation, the centroids are selected with a greedy approach based on a ranking vector π. Since π can be obtained from a trainable projector $\mathbf{p}$ applied to the vertex features, $\pi = \mathbf{X}^L\mathbf{p}^T$, k-MISPool is a trainable pooling operator. k-MISPool assigns each graph vertex to one of the centroids and aggregates the features of the vertex assigned to the same centroid with a `sum_pool` operation to create the features of the supernodes. Therefore, k-MISPool satisfies the expressiveness conditions of Th. 1.

A common characteristic of these methods is that the number of supernodes K cannot be directly specified. Graclus and ECPool achieve a pooling ratio of approximately 0.5 by roughly reducing each time the graph size by 50%. On the other hand, k-MISPool can control the coarsening level by computing the maximal independent set from $\mathcal{G}^k$, which is the graph where each node of $\mathcal{G}$ is connected to its k-hop neighbours. As the value of k increases, the pooling ratio decreases.

3.3 Criticism on graph pooling

Recently, the effectiveness of graph pooling has been questioned using as an argument a set of empirical results aimed at exposing the weaknesses of certain pooling operators [29]. The experiments showed that using a randomized cluster assignment matrix $\mathbf{S}$ (followed by a `softmax` normalization) gives comparable results to using the assignment matrices learned by Diffpool [37] and MinCut-Pool [10]. Similarly, applying Graclus [15] on the complementary graph would give a performance similar to using the original graph.

We identified potential pitfalls in the proposed evaluation, which considered only pooling operators that are expressive and that, even after being modified, retain their expressive power. Clearly, even if expressiveness ensures that all the

information is preserved in the pooled graph, its structure is corrupted when using a randomized $\mathbf{S}$ or a complementary graph. This hinders the effectiveness of the MP layers that come after pooling, as their inductive biases no longer match the data structure they receive. Notably, this might not affect certain classification tasks, e.g., when the goal is to detect small structures that are already captured by the MP layers before pooling.

To address these limitations, first, we propose to corrupt a pooling operator that is not expressive. In particular, we design a Top-k pooling operator where the nodes are ranked based on a score that is sampled from a Normal distribution rather than being produced by a trainable layer applied to the vertex features. Second, we evaluate all the modified pooling operators in a setting where the MP layers after pooling are essential for the task and show that the performance drop is significant.

4 Experimental Results

To empirically confirm the theoretical results presented in Section 3, we designed a synthetic dataset that is specifically tailored to evaluate the expressive power of a GNN. We considered a GNN with MP layers interleaved with 10 different pooling operators: DiffPool [37], DMoN [34], MinCut [10], ECPool [16], Graclus, k-MISPool [4], Top-k [20], PanPool [27], ASAPool [32], and SAGPool [25]. For each pooling method, we used the implementation in Pytorch Geometric [19] with the default configuration. In addition, following the setup used to criticize the effectiveness of graph pooling [29], we considered the following pooling operators: Rand-Dense, a dense pooling operator where the cluster assignment is a normalized random matrix; Rand-Sparse, a sparse operator that ranks nodes based on a score sampled from a Normal distribution; Cmp-Graclus, an operator that applies the Graclus algorithm on the complement graph.

4.1 The EXPWL1 dataset

Our experiments aim at evaluating the expressive power of MP layers when combined with pooling layers. However, existing real-world and synthetic benchmark datasets are unsuitable for this purpose as they are not specifically designed to relate the power of GNNs to that of the WL test. Recently, the EXP dataset was proposed to test the capability of special GNNs to achieve higher expressive power than the WL test [1], which, however, goes beyond the scope of our evaluation. Therefore, we introduce a modified version of EXP called EXPWL1, which comprises a collection of graphs $\{\mathcal{G}_1, \ldots, \mathcal{G}_N, \mathcal{H}_1, \ldots, \mathcal{H}_N\}$ that represent propositional formulas that can be satisfiable or unsatisfiable. Each pair $(\mathcal{G}_i, \mathcal{H}_i)$ in EXPWL1 consists of two non-isomorphic graphs distinguishable by a WL test, which encode formulas with opposite SAT outcomes. Therefore, any GNN that has an expressive power equal to the WL test can distinguish them and achieve approximately 100% classification accuracy on the dataset. Compared to the original EXP dataset, we increased the size of the dataset to a total of 3000

graphs and we also increased the size of each graph from an average of 55 nodes to 76 nodes. This was done to make it possible to apply an aggressive pooling without being left with a trivial graph structure. The EXPWL1 dataset and the code to reproduce the experimental results are publicly available [3].

4.2 Experimental procedure

To empirically evaluate which pooling operator maintains the expressive power of the MP layers preceding it, we first identified a GNN architecture without pooling layers, which achieves approximately 100% accuracy on the EXPWL1. We found that a GNN with three GIN layers followed by a `global_sum_pool` reaches the desired accuracy. Then, we inserted a pooling layer between the second and third GIN layers, which performs an aggressive pooling by using a pooling ratio of 0.1 that reduces the graph size by 90%. The details of the GNN configuration are in Appendix B.1. To ensure a fair comparison, when testing each method we shuffled the datasets and created 10 different train/validation/test splits using the same random seed. We trained each model on all splits for 500 epochs and reported the average training time and the average test accuracy obtained by the models that achieved the lowest loss on the validation set.

To validate our experimental approach, we also measured the performance of the proposed GNN architecture equipped with the different pooling layers on popular benchmark datasets for graph classification. In particular, we considered six TUD datasets [30] (NCI1, Proteins, Mutagenicity, COLLAB, Reddit-B, COLORS-3) and an additional synthetic dataset, B-Hard [9].

4.3 Experimental Results

Table 1 reports the performances of different pooling operators on EXPWL1. These results are consistent with our theoretical findings: pooling operators that satisfy the conditions of Th. 1 achieve the highest average accuracy among all the pooling operators. Despite the aggressive pooling, these operators retain all the necessary information and achieve the same performance as the GNN without a pooling layer. On the other hand, non-expressive pooling operators achieve lower accuracy as they are not able to correctly distinguish all graphs.

Table 1 also show that employing a pooling operator based on a normalized random cluster assignment matrix (Rand-dense) or the complement graph (Cmp-Graclus) gives a lower performance. First of all, this result disproves the argument that such operators are comparable to the regular ones [29]. Additionally, we notice that the reduction in performance is less significant for Rand-Dense and Cmp-Graclus than for Rand-sparse. This outcome is expected because, in terms of expressiveness, Rand-dense and Cmp-Graclus still satisfy the conditions of Th. 1. Nevertheless, their performance is still lower than the original pooling operators. The reason is that even if a badly-behaved `CON` function does not compromise the expressiveness of the pooling operator, the structure of the pooled graph

[3] https://github.com/FilippoMB/The-expressive-power-of-pooling-in-GNNs

Table 1: Classification on **EXPWL1** Dataset.

Pooling	s/epoch	GIN layers	Pool Ratio	Test Acc	Expr.
No-pool	0.33s	2+1	–	$99.3_{\pm 0.3}$	–
DiffPool	0.69s	2+1	0.1	$97.0_{\pm 2.4}$	✓
DMoN	0.75s	2+1	0.1	$99.0_{\pm 0.7}$	✓
MinCut	0.72s	2+1	0.1	$98.8_{\pm 0.4}$	✓
ECPool	4.79s	2+1	0.5	$99.5_{\pm 0.5}$	✓
Graclus	1.00s	2+1	0.1	$99.9_{\pm 0.1}$	✓
k**-MIS**	1.17s	2+1	0.1	$99.9_{\pm 0.1}$	✓
Top-k	0.47s	2+1	0.1	$67.9_{\pm 13.9}$	✗
PanPool	3.82s	2+1	0.1	$63.2_{\pm 7.7}$	✗
ASAPool	1.11s	1+1	0.1	$83.5_{\pm 2.5}$	✗
SAGPool	0.59s	1+1	0.1	$79.5_{\pm 9.6}$	✗
Rand-dense	0.41s	2+1	0.1	$91.7_{\pm 1.3}$	✓
Cmp-Graclus	7.42s	2+1	0.5	$91.0_{\pm 1.6}$	✓
Rand-sparse	0.47s	2+1	0.1	$62.8_{\pm 1.8}$	✗

is corrupted when utilizing a randomized $\mathbf{S}$ or a complementary graph. This, in return, reduces the effectiveness of the last GIN layer, which is essential to correctly classify the graphs in EXPWL1.

There are two remarks about the experimental evaluation. As discussed in Section 3.2, it is not possible to explicitly specify the pooling ratio in Graclus, ECPool, and k-MISPool. For k-MISPool, setting $k = 5$ gives a pooling ratio of approximately 0.1 on EXPWL1. However, for Graclus and ECPool, the only feasible option is to apply the pooling operator recursively until the desired pooling ratio of 0.1 is reached. Unfortunately, this approach is demanding, both in terms of computing time and memory usage. While it was possible to do this with Graclus in EXPWL1, we encountered an out-of-memory error after a few epochs when using ECPool on an RTX A6000 with 48GB of VRAM. Thus, the results for ECPool on the EXPWL1 are obtained with a single pooling layer that gives a pooling ratio of approximately 0.5 rather than 0.1. Clearly, a pooling ratio of 0.5 retains more information from the original graph, greatly simplifying the training in ECPool with respect to the other methods. Nevertheless, due to its expressiveness, we argue that ECPool would have reached approx 100% accuracy on EXPWL1 if implementing a more aggressive pooling was feasible.

The second remark is that in EXPWL1 when using too many MP layers, at least one node ends up containing enough information to accurately classify the graphs. This was demonstrated by using a model with 3 GIN layers followed by `global_max_pool`, which achieved an accuracy of $0.983_{\pm 0.006}$. It should be noted that the baseline model with 3 GIN layers equipped with the more expressive `global_sum_pool` achieves a slightly higher accuracy of $0.993_{\pm 0.003}$. In contrast, a model with only 2 GIN layers and `global_max_pool` gives a significantly lower accuracy of $0.665_{\pm 0.018}$. Therefore, to ensure that our evaluation is meaningful, no more than 2 MP layers should precede the pooling operator. Since ASAPool and SAGPool implement an additional MP operation internally, we used only 1 GIN layer before them, rather than 2 as for the other pooling methods.

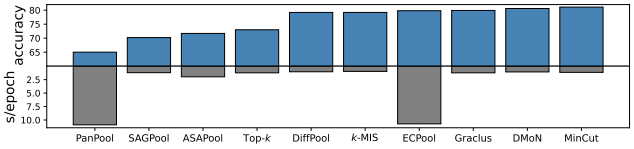

Fig. 3: Average accuracy and average runtime across the benchmark graph classification datasets.

Finally, Fig. 3 shows the average accuracy and the average run-time obtained on the seven benchmark datasets (the detailed results are in Appendix B.3). These benchmarks are not designed to test the expressive power and, thus, a GNN equipped with a non-expressive pooling operator could achieve good performance. This happens, for example, in those datasets where all the necessary information is captured by the first two GIN layers that come before pooling or in datasets where only a small part of the graph is what determines the class. Nevertheless, this second experiment serves two purposes. First, it demonstrates the soundness of the GNN architecture used in the first experiment, which achieves results comparable to those of models carefully optimized on the benchmark datasets [18]. Second, and most importantly, it shows that the performances on the benchmark datasets and EXPWL1 are aligned; this underlines the relevance of our theoretical result on the expressiveness in practical applications.

As a concluding remark, we comment on the training time of the dense and sparse pooling methods. A popular argument in favor of sparse pooling methods is their computational advantage compared to the dense ones. Our results show that this is not the case in modern deep-learning pipelines. In fact, ECPool and PanPool are approximately 10 times slower than dense pooling methods, ASAPool is twice as slow, and the only sparse method with training times lower than the dense ones is k-MIS. While it is true that the sparse methods save memory by avoiding computing intermediate dense matrices, such an advantage is relevant only for extremely large graphs that are rarely encountered in applications.

5 Conclusions

In this work, we studied for the first time the expressive power of pooling operators in GNNs. We identified the sufficient conditions that a pooling operator must satisfy to fully preserve the expressive power of the original GNN model. Based on our theoretical results, we proposed a principled approach to evaluate the expressive power of existing graph pooling operators by verifying whether they met the conditions for expressiveness.

To empirically test the expressive power of a GNN, we introduced a new dataset that allows verifying if a GNN architecture achieves the same discriminative power of the WL test. We used such a dataset to evaluate the expressiveness of a GNN equipped with different pooling operators and we found that the experimental results were consistent with our theoretical findings.

Acknowledgements We gratefully acknowledge the support of Nvidia Corporation with the donation of the RTX A6000 GPUs used in this work. We also thank Daniele Zambon, Caterina Graziani, and Antonio Longa for the useful discussions.

References

1. Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *Proceedings of the Thirtieth International Joint Conference on Artifical Intelligence (IJCAI)*, 2021.
2. Steve Azzolin, Antonio Longa, Pietro Barbiero, Pietro Liò, and Andrea Passerini. Global explainability of gnns via logic combination of learned concepts. *arXiv preprint arXiv:2210.07147*, 2022.
3. László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697, 2016.
4. Davide Bacciu, Alessio Conte, and Francesco Landolfi. Graph pooling with maximum-weight k-independent sets. In *Thirty-Seventh AAAI Conference on Artificial Intelligence*, 2023.
5. Davide Bacciu and Luigi Di Sotto. A non-negative factorization approach to node pooling in graph convolutional neural networks. In *AI* IA 2019–Advances in Artificial Intelligence: XVIIIth International Conference of the Italian Association for Artificial Intelligence, Rende, Italy, November 19–22, 2019, Proceedings 18*, pages 294–306. Springer, 2019.
6. Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.
7. Stephen T Barnard and Horst D Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and experience*, 6(2):101–117, 1994.
8. Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*, 2022.
9. Filippo Maria Bianchi, Claudio Gallicchio, and Alessio Micheli. Pyramidal reservoir graph neural network. volume 470, pages 389–404. Elsevier, 2022.
10. Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International conference on machine learning*, pages 874–883. PMLR, 2020.
11. Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Hierarchical representation learning in graph neural networks with node decimation pooling. *IEEE Transactions on Neural Networks and Learning Systems*, 33(5):2195–2207, 2020.
12. Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992.
13. Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Scalable graph neural networks via bidirectional propagation. *Advances in neural information processing systems*, 33:14556–14566, 2020.

14. Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.
15. Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11):1944–1957, 2007.
16. Frederik Diehl. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019.
17. Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. 2020.
18. Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations*, 2020.
19. Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
20. Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.
21. Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*, pages 3419–3430. PMLR, 2020.
22. Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.
23. Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.
24. Boris Knyazev, Graham W Taylor, and Mohamed Amer. Understanding attention and generalization in graph neural networks. *Advances in neural information processing systems*, 32, 2019.
25. Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International conference on machine learning*, pages 3734–3743. PMLR, 2019.
26. Antonio Longa, Steve Azzolin, Gabriele Santin, Giulia Cencetti, Pietro Liò, Bruno Lepri, and Andrea Passerini. Explaining the explainers in graph neural networks: a comparative study. *arXiv preprint arXiv:2210.15304*, 2022.
27. Zheng Ma, Junyu Xuan, Yu Guang Wang, Ming Li, and Pietro Liò. Path integral based convolution and pooling for graph neural networks. *Advances in Neural Information Processing Systems*, 33:16421–16433, 2020.
28. Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.
29. Diego Mesquita, Amauri Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. *Advances in Neural Information Processing Systems*, 33:2220–2231, 2020.
30. Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.
31. Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

32. Ekagra Ranjan, Soumya Sanyal, and Partha Talukdar. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5470–5477, 2020.
33. Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.
34. Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*, 2020.
35. Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series*, 2(9):12–16, 1968.
36. Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.
37. Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.
38. Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

A Proof of Theorem 1

Proof. Let $\texttt{SEL}(\mathcal{G}_1^L) = \left\{ \left\{ s_i^j \right\}_{i=1}^N \right\}_{j=1}^K$ and $\texttt{SEL}(\mathcal{G}_2^L) = \left\{ \left\{ t_i^j \right\}_{i=1}^M \right\}_{j=1}^K$ with $\sum_{j=1}^k s_i^j = 1 \; \forall i = 1, \dots, N$ and $\sum_{j=1}^K t_i^j = 1 \; \forall i = 1, \dots, M$.

Suppose:

$$\mathcal{X}_{1_P} = \left\{ \sum_{i=1}^N \boldsymbol{x}_i^L \cdot s_i^j \right\}_{j=1}^K = \left\{ \sum_{i=1}^M \mathbf{y}_i^L \cdot t_i^j \right\}_{j=1}^K = \mathcal{X}_{2_P}.$$

This implies that there exists a permutation $\pi : \{1, \dots, K\} \to \{1, \dots, K\}$ such that

$$\sum_{i=1}^N \boldsymbol{x}_i^L \cdot s_i^j = \sum_{i=1}^M \mathbf{y}_i^L \cdot t_i^{\pi(j)} \quad \forall j = 1, \dots, K$$

which implies

$$\sum_{j=1}^K \sum_{i=1}^N \boldsymbol{x}_i^L \cdot s_i^j = \sum_{j=1}^K \sum_{i=1}^M \mathbf{y}_i^L \cdot t_i^{\pi(j)} \Leftrightarrow \sum_{i=1}^N \boldsymbol{x}_i^L \cdot \sum_{j=1}^K s_i^j = \sum_{i=1}^M \mathbf{y}_i^L \cdot \sum_{j=1}^K t_i^{\pi(j)} \overset{2}{\Leftrightarrow} \sum_{i=1}^N \boldsymbol{x}_i^L = \sum_{i=1}^M \mathbf{y}_i^L$$

which contradicts 1.

B Experimental details

B.1 Hyperparameters of the GNN architecture

The GNN architecture used in all experiments consists of: [2 GIN layers] – [1 pooling layer with pooling ratio 0.1] – [1 GIN layer] – [`global_sum_pool`] – [dense readout].

Each GIN layer is configured with an MLP with 2 hidden layers of 64 units and ELU activation functions. The readout is a 3-layer MLP with units [64, 64, 32], ELU activations, and dropout 0.5. The GNN is trained with Adam optimizer with an initial learning rate of 1e-4 using batches with size 32. The pooling ratio is set to 0.5 for EdgePool and Cmp-Graclus. For SAGPool or ASAPool we used only one GIN layer before pooling. For PanPool we used 2 PanConv layers with filter size 2 instead of the first 2 GIN layers. The auxiliary losses in DiffPool, MinCutPool, and DMoN are added to the cross-entropy loss with weights [0.1,0.1], [0.5, 1.0], [0.3, 0.3, 0.3], respectively. For k-MIS we used $k = 5$ and we aggregated the features with the sum. For Graclus, we aggregated the node features with the sum.

B.2 Statistics of the datasets

Table 2 reports the information about the datasets used in the experimental evaluation. Since the COLLAB and REDDIT-BINARY datasets lack vertex features, we assigned a constant feature value of 1 to all vertices.

Table 2: Details of the graph classification datasets.

Dataset	#Samples	#Classes	Avg. #vertices	Avg. #edges	Vertex attr.	Vertex labels
EXPWL1	3,000	2	76.96	186.46	–	yes
NCI1	4,110	2	29.87	64.60	–	yes
Proteins	1,113	2	39.06	72.82	1	yes
COLORS-3	10500	11	61.31	91.03	4	no
Mutagenicity	4,337	2	30.32	61.54	–	yes
COLLAB	5,000	3	74.49	4,914.43	–	no
REDDIT-B	2,000	2	429.63	995.51	–	no
B-hard	1,800	3	148.32	572.32	–	yes

B.3 Detailed performance on the benchmark datasets

The average test accuracy of the GNNs configured with the different pooling operators on the graph classification benchmarks is reported in Table 3, while Table 4 reports the run-time of each model expressed in seconds per epochs. The average accuracy and average run-time computed across all datasets are presented in Table 5. For each dataset we use the same GNN configured as described in B.1, including the pooling ratio of 0.1 (except for Graclus and EdgePool, where is 0.5), as the goal is to validate the architecture used in the first experiment. Clearly, by using less aggressive pooling, carefully configuring the GNN models, and increasing their capacity it is possible to improve the results on several datasets. We refer the reader to the original papers introducing the different pooling operators for such results.

Table 3: Graph classification test accuracy on benchmark datasets.

Pooling	NCI1	PROTEINS	COLORS-3	Mutagenity	COLLAB	REDDIT-B	B-hard
DiffPool	77.8±3.9	72.8±3.3	87.6±1.0	80.0±1.9	76.6±2.5	89.9±2.8	70.2±1.5
DMoN	78.5±1.4	73.1±4.6	88.4±1.4	81.3±0.3	80.9±0.7	91.3±1.4	71.1±1.0
MinCut	80.1±2.6	76.0±3.6	88.7±1.6	81.2±1.9	79.2±1.5	91.9±1.8	71.2±1.1
ECPool	79.8±3.3	69.5±5.9	81.4±3.3	82.0±1.6	80.9±1.4	90.7±1.7	74.5±1.6
Graclus	81.2±3.4	73.0±5.9	77.6±1.2	81.9±1.6	80.4±1.5	92.9±1.7	72.3±1.3
***k*-MIS**	77.6±3.0	75.9±2.9	82.9±1.7	82.6±1.2	73.7±1.4	90.6±1.4	71.7±0.9
Top-*k*	72.6±3.1	73.2±2.7	57.4±2.5	74.4±4.7	77.9±2.1	87.4±3.5	68.1±7.7
PanPool	66.1±2.3	75.2±6.2	40.7±11.5	67.2±2.0	78.2±1.5	83.6±1.9	44.2±8.5
ASAPool	73.1±2.5	75.5±3.2	43.0±4.7	76.5±2.8	78.4±1.6	88.0±5.6	67.5±6.1
SAGPool	79.1±3.0	75.2±2.7	43.1±11.1	77.9±2.8	78.1±1.8	84.5±4.4	54.0±6.6

Table 4: Graph classification test run-time in s/epoch.

Pooling	NCI1	PROTEINS	COLORS-3	Mutagenity	COLLAB	REDDIT-B	B-hard
DiffPool	0.83s	0.23s	1.67s	0.90s	1.68s	1.74s	0.29s
DMoN	1.01s	0.28s	1.94s	1.06s	1.83s	1.04s	0.33s
MinCut	0.95s	0.28s	1.82s	1.10s	1.82s	1.78s	0.35s
ECPool	4.39s	1.97s	10.30s	4.22s	44.11s	3.17s	6.90s
Graclus	0.95s	0.27s	2.47s	0.98s	3.01s	0.75s	0.31s
***k*-MISPool**	0.88s	0.25s	2.48s	0.95s	1.38s	0.48s	0.43s
Top-*k*	1.04s	0.29s	2.78s	1.04s	2.79s	0.47s	0.30s
PanPool	2.81s	0.81s	7.16s	5.48s	7.67s	46.15s	6.27s
ASAPool	1.83s	0.52s	4.48s	1.80s	3.97s	0.79s	0.52s
SAGPool	1.09s	0.30s	2.52s	1.07s	2.81s	0.43s	0.28s

Table 5: Average run-time in seconds per epoch (first row) and average classification accuracy (second row) achieved by the different pooling methods on the benchmark datasets.

DiffPool	DMoN	MinCut	ECPool	Graclus	*k*-MIS	Top-*k*	PanPool	ASAPool	SAGPool
1.04s	1.07s	1.15s	10.72s	1.24s	0.97s	1.24s	10.90s	1.98s	1.21s
$79.2_{\pm 2.4}$	$80.6_{\pm 1.5}$	$81.1_{\pm 2.0}$	$79.8_{\pm 2.6}$	$79.9_{\pm 2.3}$	$79.2_{\pm 2.1}$	$73.0_{\pm 3.7}$	$65.0_{\pm 4.8}$	$71.7_{\pm 3.7}$	$70.2_{\pm 4.6}$