Rethinking pooling in graph neural networks

Diego Mesquita[1][*], **Amauri H. Souza**[2][*], **Samuel Kaski**[1,3]
[1]Aalto University [2]Federal Institute of Ceará [3]University of Manchester
{diego.mesquita, samuel.kaski}@aalto.fi, amauriholanda@ifce.edu.br

Abstract

Graph pooling is a central component of a myriad of graph neural network (GNN) architectures. As an inheritance from traditional CNNs, most approaches formulate graph pooling as a cluster assignment problem, extending the idea of local patches in regular grids to graphs. Despite the wide adherence to this design choice, no work has rigorously evaluated its influence on the success of GNNs. In this paper, we build upon representative GNNs and introduce variants that challenge the need for locality-preserving representations, either using randomization or clustering on the complement graph. Strikingly, our experiments demonstrate that using these variants does not result in any decrease in performance. To understand this phenomenon, we study the interplay between convolutional layers and the subsequent pooling ones. We show that the convolutions play a leading role in the learned representations. In contrast to the common belief, local pooling is not responsible for the success of GNNs on relevant and widely-used benchmarks.

1 Introduction

The success of graph neural networks (GNNs) [3, 36] in many domains [9, 15, 25, 41, 46, 50] is due to their ability to extract meaningful representations from graph-structured data. Similarly to convolutional neural networks (CNNs), a typical GNN sequentially combines local filtering, non-linearity, and (possibly) pooling operations to obtain refined graph representations at each layer. Whereas the convolutional filters capture local regularities in the input graph, the interleaved pooling operations reduce the graph representation while ideally preserving important structural information.

Although strategies for graph pooling come in many flavors [26, 30, 47, 49], most GNNs follow a hierarchical scheme in which the pooling regions correspond to graph clusters that, in turn, are combined to produce a coarser graph [4, 7, 13, 21, 47, 48]. Intuitively, these clusters generalize the notion of local neighborhood exploited in traditional CNNs and allow for pooling graphs of varying sizes. The cluster assignments can be obtained via deterministic clustering algorithms [4, 7] or be learned in an end-to-end fashion [21, 47]. Also, one can leverage node embeddings [21], graph topology [8], or both [47, 48], to pool graphs. We refer to these approaches as local pooling.

Together with attention-based mechanisms [24, 26], the notion that clustering is a must-have property of graph pooling has been tremendously influential, resulting in an ever-increasing number of pooling schemes [14, 18, 21, 27, 48]. Implicit in any pooling approach is the belief that the quality of the cluster assignments is crucial for GNNs performance. Nonetheless, to the best of our knowledge, this belief has not been rigorously evaluated.

Misconceptions not only hinder new advances but also may lead to unnecessary complexity and obfuscate interpretability. This is particularly critical in graph representation learning, as we have seen a clear trend towards simplified GNNs [5, 6, 11, 31, 43].

---

[*]Equal contribution.

In this paper, we study the extent to which local pooling plays a role in GNNs. In particular, we choose representative models that are popular or claim to achieve state-of-the-art performances and simplify their pooling operators by eliminating any clustering-enforcing component. We either apply randomized cluster assignments or operate on complementary graphs. Surprisingly, the empirical results show that the non-local GNN variants exhibit comparable, if not superior, performance to the original methods in all experiments.

To understand our findings, we design new experiments to evaluate the interplay between convolutional layers and pooling; and analyze the learned embeddings. We show that graph coarsening in both the original methods and our simplifications lead to homogeneous embeddings. This is because successful GNNs usually learn low-pass filters at early convolutional stages. Consequently, the specific way in which we combine nodes for pooling becomes less relevant.

In a nutshell, the contributions of this paper are: i) we show that popular and modern representative GNNs do not perform better than simple baselines built upon randomization and non-local pooling; ii) we explain why the simplified GNNs work and analyze the conditions for this to happen; and iii) we discuss the overall impact of pooling in the design of efficient GNNs. Aware of common misleading evaluation protocols [10, 11], we use benchmarks on which GNNs have proven to beat structure-agnostic baselines. We believe this work presents a sanity-check for local pooling, suggesting that novel pooling schemes should count on more ablation studies to validate their effectiveness.

Notation. We represent a graph $\mathcal{G}$, with $n > 0$ nodes, as an ordered pair $(\boldsymbol{A}, \boldsymbol{X})$ comprising a symmetric adjacency matrix $\boldsymbol{A} \in \{0,1\}^{n \times n}$ and a matrix of node features $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. The matrix $\boldsymbol{A}$ defines the graph structure: two nodes i, j are connected if and only if $A_{ij} = 1$. We denote by $\boldsymbol{D}$ the diagonal degree matrix of $\mathcal{G}$, i.e., $D_{ii} = \sum_j A_{ij}$. We denote the complement of $\mathcal{G}$ by $\bar{\mathcal{G}} = (\bar{\boldsymbol{A}}, \boldsymbol{X})$, where $\bar{\boldsymbol{A}}$ has zeros in its diagonal and $\bar{A}_{ij} = 1 - A_{ij}$ for $i \neq j$.

2 Exposing local pooling

2.1 Experimental setup

Models. To investigate the relevance of local pooling, we study three representative models. We first consider GRACLUS [8], an efficient graph clustering algorithm that has been adopted as a pooling layer in modern GNNs [7, 34]. We combine GRACLUS with a sum-based convolutional operator [28]. Our second choice is the popular *differential pooling* model (DIFFPOOL) [47]. DIFFPOOL is the pioneering approach to learned pooling schemes and has served as inspiration to many methods [14]. Last, we look into the *graph memory network* (GMN) [21], a recently proposed model that reports state-of-the-art results on graph-level prediction tasks. Here, we focus on local pooling mechanisms and expect the results to be relevant for a large class of models whose principle is rooted in CNNs.

Tasks and datasets. We use four graph-level prediction tasks as running examples: predicting the constrained solubility of molecules (ZINC, [20]), classifying chemical compounds regarding their activity against lung cancer (NCI1, [40]); categorizing ego-networks of actors w.r.t. the genre of the movies in which they collaborated (IMDB-B, [45]); and classifying handwritten digits (Superpixels MNIST, [1, 10]). The datasets cover graphs with none, discrete, and continuous node features. For completeness, we also report results on five other broadly used datasets in Section 3.1. Statistics of the datasets are available in the supplementary material (Section A.1).

Evaluation. We split each dataset into train (80%), validation (10%) and test (10%) data. For the regression task, we use the mean absolute error (MAE) as performance metric. We report statistics of the performance metrics over 20 runs with different seeds. Similarly to the evaluation protocol in [10], we train all models with Adam [22] and apply learning rate decay, ranging from initial 10^{-3} down to 10^{-5}, with decay ratio of 0.5 and patience of 10 epochs. Also, we use early stopping based on the validation accuracy. For further details, we refer to Appendix A in the supplementary material.

Notably, we do not aim to benchmark the performance of the GNN models. Rather, we want to isolate local pooling effects. Therefore, for model selection, we follow guidelines provided by the original authors or in benchmarking papers and simply modify the pooling mechanism, keeping the remaining model structure untouched. All methods were implemented in PyTorch [12, 33] and our code is available at `https://github.com/AaltoPML/Rethinking-pooling-in-GNNs`.
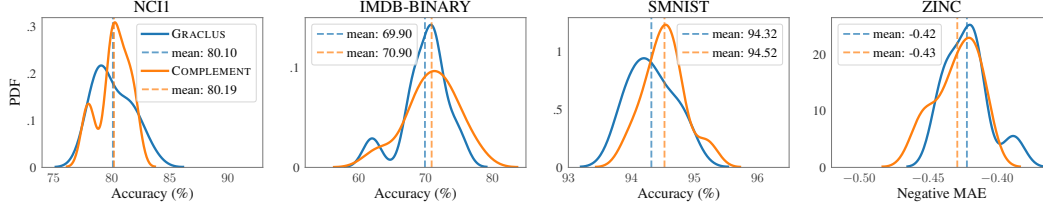
Figure 1: Comparison between GRACLUS and COMPLEMENT. The plots show the distribution of the performances over 20 runs. In all tasks, GRACLUS and COMPLEMENT achieve similar performance. The results indicate that pooling nearby nodes is not relevant to obtain a successful pooling scheme.

Case 1: Pooling with off-the-shelf graph clustering

We first consider a network design that resembles standard CNNs. Following architectures used in [7, 12, 13], we alternate graph convolutions [28] and pooling layers based on graph clustering [8]. At each layer, a neighborhood aggregation step combines each node feature vector with the features of its neighbors in the graph. The features are linearly transformed before running through a component-wise non-linear function (e.g., ReLU). In matrix form, the convolution is

$$Z^{(l)} = \mathrm{ReLU}\left(X^{(l-1)}W_1^{(l)} + A^{(l-1)}X^{(l-1)}W_2^{(l)}\right) \quad \text{with } (A^{(0)}, X^{(0)}) = (A, X), \quad (1)$$

where $W_2^{(l)}, W_1^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l}$ are model parameters, and d_l is the embedding dimension at layer l.

The next step consists of applying the GRACLUS algorithm [8] to obtain a cluster assignment matrix $S^{(l)} \in \{0, 1\}^{n_{l-1} \times n_l}$ mapping each node to its cluster index in $\{1, \dots, n_l\}$, with $n_l < n_{l-1}$ clusters. We then coarsen the features by max-pooling the nodes in the same cluster:

$$X_{kj}^{(l)} = \max_{i:S_{ik}^{(l)}=1} Z_{ij}^{(l)} \qquad k = 1, \dots, n_l \tag{2}$$

and coarsen the adjacency matrix such that $A_{ij}^{(l)} = 1$ iff clusters i and j have neighbors in $\mathcal{G}^{(l-1)}$:

$$A^{(l)} = S^{(l)\mathsf{T}}A^{(l-1)}S^{(l)} \quad \text{with } A_{kk}^{(l)} = 0 \quad k = 1, \dots, n_l. \tag{3}$$

Clustering the complement graph. The clustering step holds the idea that good pooling regions, equivalently to their CNN counterparts, should group nearby nodes. To challenge this intuition, we follow an opposite idea and set pooling regions by grouping nodes that are not connected in the graph. In particular, we compute the assignments $S^{(l)}$ by applying GRACLUS to the complement graph $\bar{\mathcal{G}}^{(l-1)}$ of $\mathcal{G}^{(l-1)}$. Note that we only employ the complement graph to compute cluster assignments $S^{(l)}$. With the assignments in hand, we apply the pooling operation (Equations 2 and 3) using the original graph structure. Henceforth, we refer to this approach as COMPLEMENT.

Results. Figure 1 shows the distribution of the performance of the standard approach (GRACLUS) and its variant that operates on the complement graph (COMPLEMENT). In all tasks, both models perform almost on par and the distributions have a similar shape.

Despite their simplicity, GRACLUS and COMPLEMENT are strong baselines (see Table 1). For instance, Errica et al. [11] report GIN [44] as the best performing model for the NCI1 dataset, achieving 80.0 ± 1.4 accuracy. This is indistinguishable from COMPLEMENT's performance (80.1 ± 1.6). We observe the same trend on IMDB-B, GraphSAGE [16] obtains 69.9 ± 4.6 and COMPLEMENT scores

Table 1: GRACLUS and COMPLEMENT performances are similar to those from the best performing isotropic GNNs (GIN and GraphSAGE) in [11] or [10]. For ZINC, lower is better.

Models	NCI1 ↑	IMDB-B ↑	SMNIST ↑	ZINC ↓
GraphSAGE	76.0 ± 1.8	69.9 ± 4.6	97.1 ± 0.02	0.41 ± 0.01
GIN	80.0 ± 1.4	66.8 ± 3.9	94.0 ± 1.30	0.41 ± 0.01
GRACLUS	80.1 ± 1.6	69.9 ± 3.5	94.3 ± 0.34	0.42 ± 0.02
COMPLEMENT	80.2 ± 1.4	70.9 ± 3.9	94.5 ± 0.33	0.43 ± 0.02

70.6 ± 5.1 — a less than 0.6% difference. Using the same data splits as [10], COMPLEMENT and GIN perform within a margin of 1% in accuracy (SMNIST) and 0.01 in absolute error (ZINC).
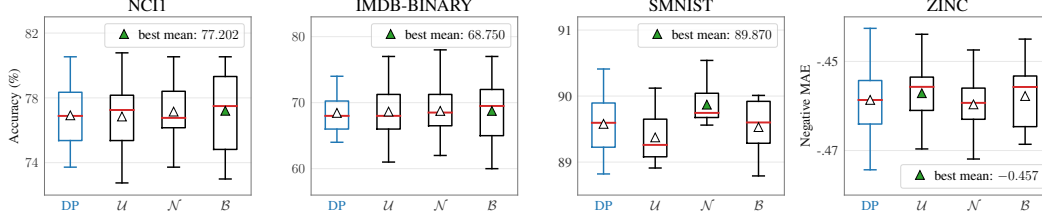
Figure 2: Boxplots for DIFFPOOL (DP) and its random variants: Uniform $\mathcal{U}(0,1)$, Normal $\mathcal{N}(0,1)$, and Bernoulli $\mathcal{B}(0.3)$. For all datasets, at least one of the random models achieves higher average accuracy than DIFFPOOL. Also, random pooling does not consistently lead to higher variance. The results show that the learned pooling assignments are not relevant for the performance of DIFFPOOL.

Case 2: Differential pooling

DIFFPOOL [47] uses a GNN to learn cluster assignments for graph pooling. At each layer l, the soft cluster assignment matrix $\boldsymbol{S}^{(l)} \in \mathbb{R}^{n_{l-1} \times n_l}$ is

$$\mathbf{S}^{(l)} = \text{softmax}\left(\text{GNN}_1^{(l)}(\boldsymbol{A}^{(l-1)}, \boldsymbol{X}^{(l-1)})\right) \quad \text{with} \quad (\boldsymbol{A}^{(0)}, \boldsymbol{X}^{(0)}) = (\boldsymbol{A}, \boldsymbol{X}). \tag{4}$$

The next step applies $\boldsymbol{S}^{(l)}$ and a second GNN to compute the graph representation at layer l:

$$\boldsymbol{X}^{(l)} = \boldsymbol{S}^{(l)\mathsf{T}} \text{GNN}_2^{(l)}(\boldsymbol{A}^{(l-1)}, \boldsymbol{X}^{(l-1)}) \quad \text{and} \quad \boldsymbol{A}^{(l)} = \boldsymbol{S}^{(l)\mathsf{T}} \boldsymbol{A}^{(l-1)} \boldsymbol{S}^{(l)}. \tag{5}$$

During training, DIFFPOOL employs a sum of three loss functions: i) a supervised loss; ii) the Frobenius norm between $\boldsymbol{A}^{(l)}$ and the Gramian of the cluster assignments, at each layer, i.e., $\sum_l \|\boldsymbol{A}^{(l)} - \boldsymbol{S}^{(l)} \boldsymbol{S}^{(l)\mathsf{T}}\|$; iii) the entropy of the cluster assignments at each layer. The second loss is referred to as the *link prediction loss* and enforces nearby nodes to be pooled together. The third loss penalizes the entropy, encouraging sharp cluster assignments.

Random assignments. To confront the influence of the learned cluster assignments, we replace $\boldsymbol{S}^{(l)}$ in Equation 4 with a normalized random matrix $\text{softmax}(\tilde{\boldsymbol{S}}^{(l)})$. We consider three distributions:

$$\text{(Uniform)} \ \tilde{S}_{ij}^{(l)} \sim \mathcal{U}(a, b) \qquad \text{(Normal)} \ \tilde{S}_{ij}^{(l)} \sim \mathcal{N}(\mu, \sigma^2) \qquad \text{(Bernoulli)} \ \tilde{S}_{ij}^{(l)} \sim \mathcal{B}(\alpha) \tag{6}$$

We sample the assignment matrix before training starts and do not propagate gradients through it.

Results. Figure 2 compares DIFFPOOL against the randomized variants. In all tasks, the highest average accuracy is due to a randomized approach. Nonetheless, there is no clear winner among all methods. Notably, the variances obtained with the random pooling schemes are not significantly higher than those from DIFFPOOL.

Figure 3: MAE obtained over random permutations of each graph in the validation set. Both Invariant and fully Random produce similar predictions throughout the training.

Remark 1. *Permutation invariance is an important property of most GNNs that assures consistency across different graph representations. However, the randomized variants break the invariance of* DIFF-POOL. *A simple fix consists of taking $\tilde{\boldsymbol{S}}^{(l)} = \boldsymbol{X}^{(l-1)} \tilde{\boldsymbol{S}}'$, where $\tilde{\boldsymbol{S}}' \in \mathbb{R}^{d_{l-1} \times n_l}$ is a random matrix. Figure 3 compares the randomized variants with and without this fix w.r.t. the validation error on artificially permuted graphs during training on the ZINC dataset. Results suggest that the variants are approximately invariant.*
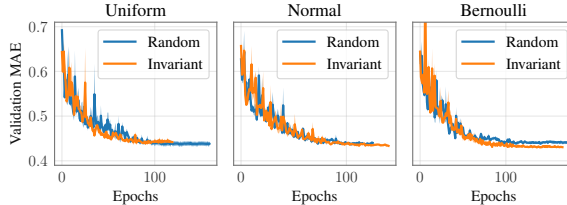
4

Case 3: Graph memory networks

Graph memory networks (GMNs) [21] consist of a sequence of memory layers stacked on top of a GNN, also known as the initial query network. We denote the output of the initial query network by $Q^{(0)}$. The first step in a memory layer computes kernel matrices between input queries $Q^{(l-1)} = [q_1^{(l-1)}, \ldots, q_{n_{l-1}}^{(l-1)}]^\intercal$ and multi-head keys $K_h^{(l)} = [k_{1h}^{(l)}, \ldots, k_{n_l h}^{(l)}]^\intercal$:

$$S_h^{(l)} : S_{ijh}^{(l)} \propto \left(1 + \|q_i^{(l-1)} - k_{jh}^{(l)}\|^2 / \tau \right)^{-\frac{\tau+1}{2}} \quad \forall h = 1 \ldots H, \tag{7}$$

where H is the number of heads and τ is the degrees of freedom of the Student's t-kernel.

We then aggregate the multi-head assignments $S_h^{(l)}$ into a single matrix $S^{(l)}$ using a 1×1 convolution followed by row-wise softmax normalization. Finally, we pool the node embeddings $Q^{(l-1)}$ according to their soft assignments $S^{(l)}$ and apply a single-layer neural net:

$$Q^{(l)} = \text{ReLU}\left(S^{(l)\intercal} Q^{(l-1)} W^{(l)}\right). \tag{8}$$

In this notation, queries $Q^{(l)}$ correspond to node embeddings $X^{(l)}$ for $l > 0$. Also, note that the memory layer does not leverage graph structure information as it is fully condensed into $Q^{(0)}$. Following [21], we use a GNN as query network. In particular, we employ a two layer network with the same convolutional operator as in Equation 1.

The loss function employed to learn GMNs consists of a convex combination of: i) a supervised loss; and ii) the Kullback-Leibler divergence between the learned assignments and their self-normalized squares. The latter aim to enforce sharp soft-assignments, similarly to the entropy loss in DIFFPOOL.

Remark 2. *Curiously, the intuition behind the loss that allegedly improves cluster purity might be misleading. For instance, uniform cluster assignments, the ones the loss was engineered to avoid, are a perfect minimizer for it. We provide more details in Appendix D.*

Simplifying GMN. The principle behind GMNs consists of grouping nodes based on their similarities to learned keys (centroids). To scrutinize this principle, we propose two variants. In the first, we replace the kernel in Equation 7 by the euclidean distance taken from fixed keys drawn from a uniform distribution. Opposite to vanilla GMNs, the resulting assignments group nodes that are farthest from a key. The second variant substitutes multi-head assignment matrices for a fixed matrix whose entries are independently sampled from a uniform distribution. We refer to these approaches, respectively, as DISTANCE and RANDOM.

Results. Figure 4 compares GMN with its simplified variants. For all datasets, DISTANCE and RANDOM perform on par with GMN, with slightly better MAE for the ZINC dataset. Also, the variants present no noticeable increase in variance.

It is worth mentioning that the simplified GMNs are naturally faster to train as they have significantly less learned parameters. In the case of DISTANCE, keys are taken as constants once sampled. Additionally, RANDOM bypasses the computation of the pairwise distances in Equation 7, which dominates the time of the forward pass in GMNs. On the largest dataset (SMNIST), DISTANCE takes up to half the time of GMN (per epoch), whereas RANDOM is up to ten times faster than GMN.
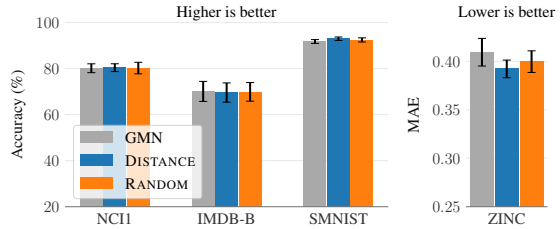

Figure 4: GMN versus its randomized variants DISTANCE and RANDOM. The plots show that pooling based on dissimilarity measure (distance) instead of similarity (kernel) does not have a negative impact in performance. The same holds true when we employ random assignments. Both variants achieve lower MAE for the regression task (ZINC).
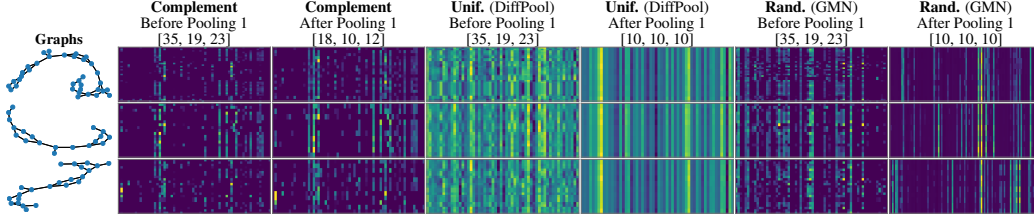
Figure 5: **Non-local pooling**: activations for three arbitrary graphs, before and after the first pooling layer. The convolutional layers learn to output similar embeddings within a same graph. As a result, methods become approximately invariant to how embeddings are pooled. Darker tones denote lower activation values. To better assess homogeneity, we normalize the color range of each embedding matrix individually. The number of nodes for each embedding matrix is given inside brackets.

3 Analysis

The results in the previous section are counter-intuitive. We now analyze the factors that led to these results. We show that the convolutions preceding the pooling layers learn representations which are approximately invariant to how nodes are grouped. In particular, GNNs learn smooth node representations at the early layers. To experimentally show this, we remove the initial convolutions that perform this early smoothing. As a result, all networks experience a significant drop in performance. Finally, we show that local pooling offers no benefit in terms of accuracy to the evaluated models.

Pooling learns approximately homogeneous node representations. The results with the random pooling methods suggest that any convex combination of the node features enables us to extract good graph representation. Intuitively, this is possible if the nodes display similar activation patterns before pooling. If we interpret convolutions as filters defined over node features/signals, this phenomenon can be explained if the initial convolutional layers extract low-frequency information across the graph input channels/embedding components. To evaluate this intuition, we compute the activations before the first pooling layer and after each of the subsequent pooling layers. Figure 5 shows activations for the random pooling variants of DIFFPOOL and GMN, and the COMPLEMENT approach on ZINC.

The plots in Figure 5 validate that the first convolutional layers produce node features which are relatively homogeneous within the same graph, specially for the randomized variants. The networks learn features that resemble vertical bars. As expected, the pooling layers accentuate this phenomenon, extracting even more similar representations. We report embeddings for other datasets in Appendix E.

Even methods based on local pooling tend to learn homogeneous representations. As one can notice from Figure 6, DIFFPOOL and GMN show smoothed patterns in the outputs of their initial pooling layer. This phenomenon explains why the performance of the randomized approaches matches those of their original counterparts. The results suggest that the loss terms designed to enforce local clustering are either not beneficial to the learning task or are obfuscated by the supervised loss. This observation does not apply to GRACLUS, as it employs a deterministic clustering algorithm, separating the possibly competing goals of learning hierarchical structures and minimizing a supervised loss.
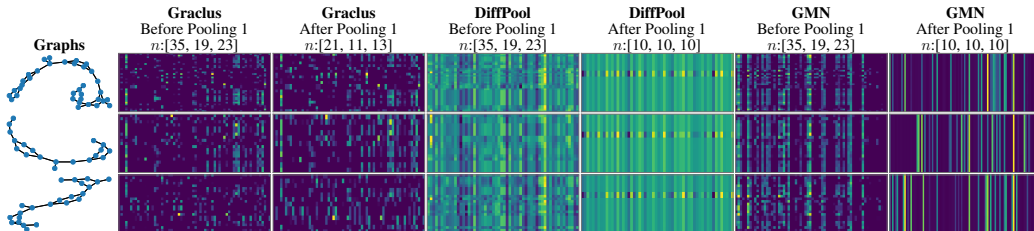

Figure 6: **Local pooling**: activations for the same graphs from Figure 5, before and after the first pooling layer. Similar to the simplified variants, their standard counterparts learn homogenous embeddings, specially for DIFFPOOL and GMN. The low variance across node embedding columns illustrates this homogeneity. Darker tones denote lower values and colors are not comparable across embedding matrices. Brackets (top) show the number of nodes after each layer for each graph.

To further gauge the impact of the unsupervised loss on the performance of these GNNs, we compare two DIFFPOOL models trained with the link prediction loss multiplied by the weighting factors $\lambda = 10^0$ and $\lambda = 10^3$. Figure 7 shows the validation curves of the supervised loss for ZINC and SMNIST. We observe that the supervised losses for models with both of the λ values converge to a similar point, at a similar rate. This validates that the un-
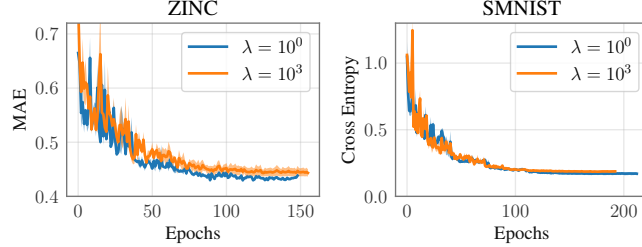

Figure 7: Supervised loss on validation set for ZINC and SMNIST datasets. Scaling up the unsupervised loss has no positive impact on the performance of DIFFPOOL.

supervised loss (link prediction) has little to no effect on the predictive performance of DIFFPOOL. We have observed a similar behavior for GMNs, which we report in the supplementary material.

Discouraging smoothness. In Figure 6, we observe homogeneous node representations even before the first pooling layer. This naturally poses a challenge for the upcoming pooling layers to learn meaningful local structures. These homogeneous embeddings correspond to low frequency signals defined on the graph nodes. In the general case, achieving such patterns is only possible with more than a single convolution. This can be explained from a spectral perspective. Since each convolutional layer corresponds to filters that act linearly in the spectral domain, a single convolution cannot filter out specific frequency bands. These ideas have already been exploited to develop simplified GNNs [31, 43] that compute fixed polynomial filters in a normalized spectrum.

Remarkably, using multiple convolutions to obtain initial embeddings is common practice [18, 26]. To evaluate its impact, we apply a single convolution before the first pooling layer in all networks. We then compare these networks against the original implementations. Table 2 displays the results. All models report a significant drop in performance with a single convolutional layer. On NCI1, the methods obtain accura-

Table 2: All networks experience a decrease in performance when implemented with a single convolution before pooling (numbers in red). For ZINC, lower is better.

# Convolutions	NCI1		IMDB-B		SMNIST		ZINC	
	$= 1$	> 1	$= 1$	> 1	$= 1$	> 1	$= 1$	> 1
GRACLUS	76.6	80.1	69.7	69.9	89.1	94.3	0.458	0.429
COMPLEMENT	74.2	80.2	69.8	70.9	81.0	94.5	0.489	0.431
DIFFPOOL	71.3	76.9	68.5	68.5	66.6	89.6	0.560	0.459
$\mathcal{U}$ DIFFPOOL	72.8	76.8	69.0	68.7	66.4	89.4	0.544	0.457
GMN	77.1	80.2	68.9	69.9	87.4	92.4	0.469	0.409
Rand. GMN	76.4	80.2	66.6	66.8	90.0	94.0	0.465	0.400

cies about 4% lower, on average. Likewise, GMN and GRACLUS report a 4% performance drop on SMNIST. With a single initial GraphSAGE convolution, the performances of Diffpool and its uniform variant become as lower as 66.4% on SMNIST. This dramatic drop is not observed only for IMDB-B, which counts on constant node features and therefore may not benefit from the additional convolutions. Note that under reduced expressiveness, pooling far away nodes seems to impose a negative inductive bias as COMPLEMENT consistently fails to rival the performance of GRACLUS. Intuitively, the number of convolutions needed to achieve smooth representations depend on the richness of the initial node features. Many popular datasets rely on one-hot features and might require a small number of initial convolutions. Extricating these effects is outside the scope of this work.

Is pooling overrated? Our experiments suggest that local pooling does not play an important role on the performance of GNNs. A natural next question is whether local pooling presents any advantage over global pooling strategies. We run a GNN with global mean pooling on top of three convolutional layers, the results are: 79.65 ± 2.07 (NCI), 71.05 ± 4.58 (IMDB-B), 95.20 ± 0.18 (SMNIST), and 0.443 ± 0.03 (ZINC). These performances are on par with our previous results obtained using GRACLUS, DIFFPOOL and GMN.

Regardless of how surprising this may sound, our findings are consistent with results reported in the literature. For instance, GraphSAGE networks outperform DIFFPOOL in a rigorous evaluation protocol [10, 11], although GraphSAGE is the convolutional component of DIFFPOOL. Likewise, but in the context of attention, Knyazev et al. [24] argue that, except under certain circumstances, general attention mechanisms are negligible or even harmful. Similar findings also hold for CNNs [35, 39].
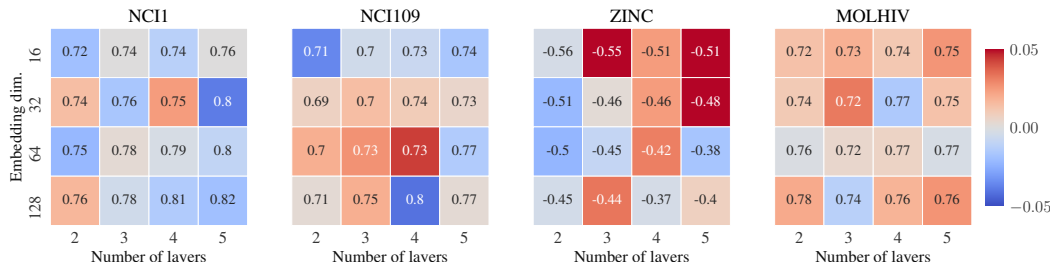
Figure 8: Performance gap between GRACLUS and COMPLEMENT for different hyperparameter settings. Numbers denote the performance of GRACLUS — negative MAE for ZINC, AUROC for MOLHIV, and accuracy for the rest. Red tones denote cases on which COMPLEMENT obtains better performance. For all datasets, the performance gap is not larger than 0.05 (or 5% in accuracy). The predominant light tones demonstrate that both models obtain overall similar results, with no clear trend toward specific hyperparameter choices.

Permutation invariance is a relevant property for GNNs as it guarantees that the network's predictions do not vary with the graph representation. For completeness, we state in Appendix C the permutation invariance of the simplified GNNs discussed in Section 2.

3.1 Additional results

More datasets. We consider four additional datasets commonly used to assess GNNs, three of which are part of the TU datasets [29]: PROTEINS, NCI109, and DD; and one from the recently proposed Open Graph Benchamark (OGB) framework [17]: MOLHIV. Since Ivanov et al. [19] showed that the IMDB-B dataset is affected by a serious isomorphism bias, we also report results on the "cleaned" version of the IMDB-B dataset, hereafter refereed to as IMDB-C.

Table 3 shows the results for the additional datasets. Similarly to what we have observed in the previous experiments, non-local pooling performs on par with local pooling. The largest performance gap occurs for the PROTEINS and MOHIV datasets, on which COMPLEMENT achieves accuracy around 3% higher than GRACLUS, on average. The performance of all methods on IMDB-C does not differ significantly from their performance on IMDB-B. However, removing the isomorphisms in IMDB-B reduces the dataset size, which clearly increases variance.

Table 3: Results for the additional datasets. Again, we observe that non-local pooling yields results as good as local pooling.

Models	IMDB-C	PROTEINS	NCI109	DD	MOLHIV
GRACLUS	70.5 ± 8.3	72.6 ± 4.1	77.7 ± 2.6	72.1 ± 5.2	74.6 ± 1.8
COMPLEMENT	70.1 ± 7.0	75.3 ± 3.1	77.0 ± 2.4	72.1 ± 3.6	76.4 ± 1.9
DIFFPOOL	70.9 ± 6.7	75.2 ± 4.0	72.2 ± 1.5	76.9 ± 4.4	75.1 ± 1.3
$\mathcal{N}$-DIFFPOOL	70.3 ± 7.5	74.6 ± 3.8	71.8 ± 2.1	78.3 ± 4.1	74.8 ± 1.3
GMN	68.4 ± 7.1	74.8 ± 3.3	77.0 ± 1.7	73.9 ± 4.6	73.7 ± 1.8
Random-GMN	70.2 ± 8.0	74.8 ± 3.9	76.9 ± 2.0	74.3 ± 3.6	73.5 ± 2.6

Another pooling method. We also provide results for MINCUTPOOL [2], a recently proposed pooling scheme based on spectral clustering. This scheme integrates an unsupervised loss which stems from a relaxation of a MINCUT objective and learns to assign clusters in a spectrum-free way. We compare MINCUTPOOL with a random variant for all datasets employed in this work. Again, we find that a random pooling mechanism achieves comparable results to its local counterpart. Details are given in Appendix B.

Sensitivity to hyperparameters. As mentioned in Section 2.1, this paper does not intend to benchmark the predictive performance of models equipped with different pooling strategies. Consequently, we did not exhaustively optimize model hyperparameters. One may wonder whether our results hold for a broader set of hyperparameter choices. Figure 8 depicts the performance gap between GRACLUS and COMPLEMENT for a varying number of pooling layers and embedding dimensionality over a single run. We observe the greatest performance gap in favor of COMPLEMENT (e.g., 5 layers and 32 dimensions on ZINC), which does not amount to an improvement greater than 0.05 in MAE. Overall, we find that the choice of hyperparameters does not significantly increase the performance gap between GRACLUS and COMPLEMENT.

4 Related works

Graph pooling usually falls into global and hierarchical approaches. Global methods aggregate the node representations either via simple flattening procedures such as summing (or averaging) the node embeddings [44] or more sophisticated set aggregation schemes [32, 49]. On the other hand, hierarchical methods [7, 14, 18, 21, 26, 27, 47] sequentially coarsen graph representations over the network's layers. Notably, Knyazev et al. [24] provides a unified view of local pooling and node attention mechanisms, and study the ability of pooling methods to generalize to larger and noisy graphs. Also, they show that the effect of attention is not crucial and sometimes can be harmful, and propose a weakly-supervised approach.

Simple GNNs. Over the last years, we have seen a surge of simplified GNNs. Wu et al. [43] show that removing the nonlinearity in GCNs [23] does not negatively impact on their performances on node classification tasks. The resulting feature extractor consists of a low-pass-type filter. In [31], the authors take this idea further and evaluate the resilience to feature noise and provide insights on GCN-based designs. For graph classification, Chen et al. [6] report that linear convolutional filters followed by nonlinear set functions achieve competitive performances against modern GNNs. Cai and Wang [5] propose a strong baseline based on local node statistics for non-attributed graph classification tasks.

Benchmarking GNNs. Errica et al. [11] demonstrate how the lack of rigorous evaluation protocols affects reproducibility and hinders new advances in the field. They found that structure-agnostic baselines outperform popular GNNs on at least three commonly used chemical datasets. At the rescue of GNNs, Dwivedi et al. [10] argue that this lack of proper evaluation comes mainly from using small datasets. To tackle this, they introduce a new benchmarking framework with datasets that are large enough for researchers to identify key design principles and assess statistically relevant differences between GNN architectures. Similar issues related to the use of small datasets are reported in [37].

Understanding pooling and attention in regular domains. In the context of CNNs for object recognition tasks, Ruderman et al. [35] evaluate the role of pooling in CNNs w.r.t. the ability to handle deformation stability. The results show that pooling is not necessary for appropriate deformation stability. The explanation lies in the network's ability to learn smooth filters across the layers. Sheng et al. [38] and Zhao et al. [51] propose random pooling as a fast alternative to conventional CNNs. Regarding attention-based models, Wiegreffe and Pinter [42] show that attention weights usually do not provide meaningful explanations for predictions. These works demonstrate the importance of proper assessment of core assumptions in deep learning.

5 Conclusion

In contrast to the ever-increasing influx of GNN architectures, very few works rigorously assess which design choices are crucial for efficient learning. Consequently, misconceptions can be widely spread, influencing the development of models drinking from flawed intuitions. In this paper, we study the role of local pooling and its impact on the performance of GNNs. We show that most GNN architectures employ convolutions that can quickly lead to smooth node representations. As a result, the pooling layers become approximately invariant to specific cluster assignments. We also found that clustering-enforcing regularization is usually innocuous. In a series of experiments on accredited benchmarks, we show that extracting local information is not a necessary principle for efficient pooling. By shedding new light onto the role of pooling, we hope to contribute to the community in at least two ways: i) providing a simple sanity-check for novel pooling strategies; ii) deconstruct misconceptions and wrong intuitions related to the benefits of graph pooling.

Acknowledgments and Disclosure of Funding

Broader impact

Graph neural networks (GNNs) have become the *de facto* learning tools in many valuable domains such as social network analysis, drug discovery, recommender systems, and natural language processing. Nonetheless, the fundamental design principles behind the success of GNNs are only partially understood. This work takes a step further in understanding local pooling, one of the core design choices in many GNN architectures. We believe this work will help researchers and practitioners better choose in which directions to employ their time and resources to build more accurate GNNs.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34 (11):2274–2282, 2012.

[2] F. M. Bianchi, D. Grattarola, and C. Alippi. Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning (ICML)*, 2020.

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

[4] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)*, 2014.

[5] C. Cai and Y. Wang. A simple yet effective baseline for non-attribute graph classification. *ArXiv:1811.03508*, 2018.

[6] T. Chen, S. Bian, and Y. Sun. Are powerful graph neural nets necessary? A dissection on graph classification. *ArXiv:1905.04579*, 2019.

[7] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[8] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

[9] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[10] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *ArXiv:2003.00982*, 2020.

[11] F. Errica, M. Podda, D. Bacciu, and A. Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations (ICLR)*, 2020.

[12] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *Workshop track of the International Conference on Representation Learning (ICLR)*, 2019.

[13] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[14] H. Gao and S. Ji. Graph U-nets. In *International Conference on Machine Learning (ICML)*, 2019.

[15] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, 2017.

[16] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[17] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[18] J. Huang, Z. Li, N. Li, S. Liu, and G. Li. Attpool: Towards hierarchical feature representation in graph convolutional networks via attention mechanism. In *International Conference on Computer Vision (ICCV)*, 2019.

[19] S. Ivanov, S. Sviridov, and E. Burnaev. Understanding isomorphism bias in graph data sets. *ArXiv:1910.12091*, 2019.

[20] W. Jin, R. Barzilay, and T. Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning (ICML)*, 2018.

[21] A. H. Khasahmadi, K. Hassani, P. Moradi, L. Lee, and Q. Morris. Memory-based graph networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[23] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[24] B. Knyazev, G. W. Taylor, and M. Amer. Understanding attention and generalization in graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[25] L. Landrieu and M. Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[26] J. Lee, I. Lee, and J. Kang. Self-attention graph pooling. In *International Conference on Machine Learning (ICML)*, 2019.

[27] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang. Graph convolutional networks with eigenpooling. In *International Conference on Knowledge Discovery & Data Mining (KDD)*, 2019.

[28] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[29] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. TU dataset: a collection of benchmark datasets for learning with graphs. In *Workshop track of the International Conference on Machine Learning (ICML)*, 2020.

[30] R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning (ICML)*, 2019.

[31] H. NT and T. Maehara. Revisiting graph neural networks: All we have is low-pass filters. *ArXiv:1905.09550*, 2019.

[32] M. Kudlur O. Vinyals, S. Bengio. Order matters: Sequence to sequence for sets. In *International Conference on Learning Representations (ICLR)*, 2015.

[33] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems (NeurIPS - Workshop)*, 2017.

[34] S. Rhee, S. Seo, and S. Kim. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. In *International Joint Conference on Artificial Intelligence, (IJCAI)*, 2018.

[35] A. Ruderman, N. C. Rabinowitz, A. S. Morcos, and D. Zoran. Pooling is neither necessary nor sufficient for appropriate deformation stability in CNNs. *ArXiv:1804.04438*, 2018.

[36] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[37] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation. *ArXiv:1811.05868*, 2018.

[38] J. Sheng, C. Chen, C. Fu, and C. J. Xue. EasyConvPooling: Random pooling with easy convolution for accelerating training and testing. *ArXiv:1806.01729*, 2018.

[39] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for Simplicity: The All Convolutional Net. In *Workshop track of the International Conference on Representation Learning (ICLR)*, 2015.

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

[41] C. Y. Wee, C. Liu, A. Lee, J. S. Poh, H. Ji, and A. Qiu. Cortical graph neural network for AD and MCI diagnosis and transfer learning across populations. *NeuroImage: Clinical*, 23, 2019.

[42] S. Wiegreffe and Y. Pinter. Attention is not not explanation. In *Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

[43] F. Wu, A. H. Souza Jr, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning (ICML)*, 2019.

[44] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

[45] P. Yanardag and S. V. N. Vishwanathan. Deep graph kernels. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2015.

[46] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *International Conference on Knowledge Discovery & Data Mining (KDD)*, 2018.

[47] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[48] H. Yuan and S. Ji. StructPool: Structured graph pooling via random fields. In *International Conference on Learning Representations (ICLR)*, 2020.

[49] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[50] Y. Zhang, P. Qi, and C. D. Manning. Graph convolution over pruned dependency trees improves relation extraction. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[51] G. Zhao, J. Wang, and Z. Zhang. Random shifting for CNN: a solution to reduce information loss in down-sampling layers. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

Rethinking pooling in graph neural networks
— Supplementary material —

A Implementation details

A.1 Datasets

Table S1 reports summary statistics of the datasets used in this paper. For SMNIST and ZINC, we use the same pre-processing steps and data splits as in [10]. For the IMDB-B dataset, we use uninformative features (vector of ones) for all nodes. NCI1 and IMDB-B are part of the TU Datasets[2], a vast collection of datasets commonly used for evaluating graph kernel methods and GNNs. Errica et al. [11] show that drawing conclusions based on some of these datasets can be problematic as structure-agnostic baselines achieve higher performance than traditional GNNs. However, in their assessment, NCI1 is the only chemical dataset on which GNNs beat baselines. Also, among the social datasets, IMDB-B produces the greatest performance gap between the baseline and DiffPool ($\approx 20\%$). Table S1 also shows statistics for PROTEINS, NCI109, DD, and MOLHIV.

Table S1: Statistics of the datasets.

Dataset	#graphs	#classes	#node feat.	#node labels	Avg #nodes	Avg #edges
NCI1	4110	2	-	37	29.87	32.30
IMDB-B	1000	2	-	-	19.77	96.53
SMNIST	70000	10	3	-	70.57	564.53
ZINC	12000	-	-	28	23.16	49.83
PROTEINS	1113	2	-	3	39.06	72.82
NCI109	4127	2	-	38	29.68	32.13
DD	1178	2	-	82	284.32	715.66
MOLHIV	41127	2	-	9	25.5	27.5

A.2 Models

We implement all models using the PyTorch Geometric Library [12]. We apply the Adam optimizer [22] with learning rate decaying from 10^{-3} to 10^{-5}. If the validation performance does not improve over 10 epochs we reduce the learning rate by half. Also, we use early stopping with patience of 50 epochs.

For the MOLHIV dataset, we propagate the edge features through a linear layer and incorporate the result in the messages of the first convolutional layer. More specifically, we add the edge embeddings to the node features and apply a ReLU function. We follow closely the procedure applied to the GIN and GCN baselines[3] for the MOLHIV dataset.

GRACLUS. Our GRACLUS model is based on the implementation available in PyTorch Geometric. For all datasets, we employ an initial convolution to extract node embeddings. Then, we interleave convolutions and pooling layers. For instance, a 3-layer GRACLUS model consists of $conv \rightarrow conv/pool \rightarrow conv/pool \rightarrow readout \rightarrow MLP$. In all experiments, we apply global mean pooling as readout layer. Also, we adopt an MLP with a single hidden layer and the same number of hidden components as the GNN layers. The Complement model sticks to exactly the same setup. We report the specific hyperparameter values for each dataset in Table S2.

---

[2]`https://chrsmrrs.github.io/datasets/`
[3]`https://github.com/snap-stanford/ogb/tree/master/examples/graphproppred/mol`

Table S2: Hyperparameters for the GRACLUS/COMPLEMENT models.

Dataset	#Layers	|Hidden dim.|	|Batch|
NCI1	3	64	8
IMDB-B	2	64	8
SMNIST	2	64	8
ZINC	3	64	8
PROTEINS	3	64	64
NCI109	3	64	64
DD	3	64	64
MOLHIV	2	128	64

DIFFPOOL.   We follow the setup in [10] and use 3 GraphSAGE convolutions before and after the pooling layers, except for the first that adopts a single convolution. The embedding and pooling GNNs consist of a 1-hop GraphSAGE convolution. We apply residual connections and use embedding dimension of 32 for NCI1, SMNIST, and IMDB-B, and 56 for ZINC (similar to [10]). We use mini-batches of size 64 for SMNIST and ZINC. We report in Table S3 the list of the main hyperparameters used in the experiments. Further details can be found in our official repository. Since the experiments with DIFFPOOL have been removed from [10] in the latest version of the paper, we report results with a different architecture in Appendix B.

Table S3: Hyperparameters for DIFFPOOL and its variants.

Dataset	#Pool. Layers	|GNN hidden dim.|	|MLP hidden dim.|	|Batch|
NCI1	1	32	32	16
IMDB-B	1	32	50	8
SMNIST	1	32	50	64
ZINC	1	56	56	64
PROTEINS	2	32	50	8
NCI109	1	32	32	16
DD	2	32	50	8
MOLHIV	2	32	50	32

GMN.   Based on the official code repository, we re-implemented the GMN using PyTorch Geometric. The initial query network consists of two basic convolutions — see Equation 1 — followed by batch normalization. We found that this basic convolution produces better results than the random walk with restart (RWR) and graph attention networks as suggested in [21]. Our design choice is also less dependent on dataset specificities, as applying RWR to graphs with multiple connected components (such as the ones on NCI1) requires additional pre-processing, for example. Table S4 shows more details on the hyperparameters used for the memory layers. We feed the output of the last memory layer to a single-hidden layer feed-forward net with fifty hidden nodes.

Table S4: Hyperparameters for the GMN models.

Dataset	#Keys	#Heads	#Layers	|Hidden dim.|	|Batch|
NCI1	[10, 1]	5	2	100	128
IMDB-B	[32, 1]	1	2	16	128
SMNIST	[32, 1]	10	2	16	128
ZINC	[10, 1]	5	2	100	128
PROTEINS	[32, 1]	5	2	16	128
NCI109	[32, 1]	5	2	16	128
DD	[32, 8, 1]	5	3	64	128
MOLHIV	[32, 1]	5	2	16	128

B Additional experiments

For completeness, in this section, we report additional results using one more pooling method. We also present the results of a second version of DIFFPOOL that employs 3 graph convolutions at each pooling layer. Furthermore, we gauge the impact of the unsupervised loss of GMNs.

MINCUTPOOL. Bianchi et al. [2] propose a pooling scheme based on a spectrum-free formulation of spectral clustering. Similarly to DIFFPOOL, MINCUTPOOL leverages node features and graph topology to learn cluster assignments. In particular, these assignments are computed using the composition of an MLP and a GNN layer. The parameters in a MINCUTPOOL layer are learned by minimizing a minCUT objective and a supervised loss. As for the code, we use the implementation available in PyTorch Geometric. In the experiments, we apply three layers of interleaved convolution and pooling operators, as originally employed in [2].

Randomized MINCUTPOOL. To evaluate the effectiveness of the MINCUTPOOL layers, we follow a randomized approach. Similarly to the DIFFPOOL variants, we simply replace the learned assignment matrix with a random one, sampled from a standard normal distribution. We only apply the supervised loss function. In the following, we refer to this approach as $\mathcal{N}$-MINCUTPOOL.

Another version of DIFFPOOL. Here we consider the architecture choice and implementation given in [11]. Each pooling and embedding GNN encompasses a 3-layer SAGE convolution. After the final pooling layer, another group of 3-layer SAGE convolutions is used before the readout layer. We evaluate a randomized variant with normal distribution, referred to as $\mathcal{N}$-DIFFPOOL v2.

Results. Table S5 displays the results for MINCUTPOOL and DIFFPOOL v2. Overall, we find that the variants and the original pooling methods obtain similar performance. Notably, SMNIST stands out as an exception, as MINCUTPOOL performs significantly better (89.3%) than its randomized version (82.0%). We argue that this is because we only use one single convolution before the first pooling layer, which is consistent with the findings in Table 2. The fact that the initial features are dense might make learning smooth features harder. Nonetheless, as we have previously seen, using as few as two initial convolutions suffices to settle this performance gap on SMNIST.

Table S5: Additional results for MINCUTPOOL and a second implementation of DIFFPOOL.

Models	NCI1 ↑	IMDB-B ↑	SMNIST ↑	ZINC ↓	PROTEINS ↑	NCI109 ↑	DD ↑	MOLHIV ↑
MINCUTPOOL	76.1 ± 1.9	68.8 ± 4.6	89.3 ± 1.0	0.47 ± 0.01	75.0 ± 3.7	74.3 ± 2.2	77.2 ± 4.1	71.9 ± 1.6
$\mathcal{N}$-MINCUTPOOL	76.6 ± 1.6	69.7 ± 4.8	82.0 ± 1.6	0.47 ± 0.01	75.2 ± 3.5	74.7 ± 2.1	77.0 ± 3.2	72.6 ± 2.2
DIFFPOOL v2	77.4 ± 2.0	67.7 ± 3.1	90.7 ± 0.3	0.50 ± 0.01	73.2 ± 3.5	75.2 ± 1.4	76.5 ± 2.6	71.8 ± 2.2
$\mathcal{N}$-DIFFPOOL v2	77.4 ± 1.4	68.2 ± 2.8	89.0 ± 0.5	0.49 ± 0.01	74.6 ± 4.0	75.2 ± 1.7	75.3 ± 3.4	74.4 ± 3.0

Unsupervised loss of GMNs. GMNs are trained by alternating between the supervised loss and the unsupervised (purity enforcing) loss. When using the unsupervised loss, we only update the model keys; and when using the supervised loss, we update all remaining parameters. In its official implementation, GMN updates its key locations once each $p = 5$ epochs. Figure S1 shows the supervised loss on the validation set, with $p \in \{2, 5\}$, for ZINC and IMDB. As is the case for DIFFPOOL, enforcing the role of the unsupervised loss does not improve the performance of GMNs.

C Permutation Invariance

Permutation invariance is an important property for GNNs since it guarantees that the network's predictions do not vary with the graph representation. Here we state the permutation invariance of some of the simplified GNNs discussed in Section 2. For completeness, we bring here definitions of invariant and equivariant functions over graphs.

Definition 1 (Permutation matrix). $\boldsymbol{P}_n \in \{0, 1\}^{n \times n}$ *is a permutation matrix of size* n *if* $\sum_i P_{i,j} = 1 \; \forall j$ *and* $\sum_j P_{i,j} = 1 \; \forall i$.

We denote by $\mathfrak{G}$ the set of undirected graphs and by $\mathfrak{G}_n$ the set of all graphs with exactly n nodes.
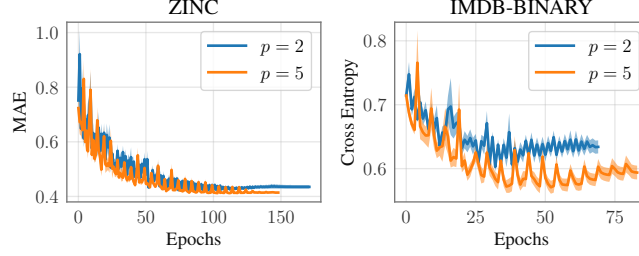
Figure S1: Supervised loss on validation set for ZINC and IMDB-BINARY datasets. Updating the keys using the unsupervised loss more frequently shows no positive impact on the performance of GMN. For IMDB-BINARY, updating the keys more frequently leads to a clearly worse minimum. Curves are averaged over twenty repetitions with different random seeds.

Definition 2 (Equivariant graph function). *A function $f : \mathfrak{G}' \subseteq \mathfrak{G} \to \mathbb{R}^{n \times d}$ is equivariant if $f(\boldsymbol{P}_n \boldsymbol{A} \boldsymbol{P}_n^\intercal, \boldsymbol{P}_n \boldsymbol{X}) = \boldsymbol{P}_n f(\boldsymbol{A}, \boldsymbol{X})$ for any $\mathcal{G} = (\boldsymbol{A}, \boldsymbol{X}) \in \mathfrak{G}'$ with n nodes and any permutation matrix $\boldsymbol{P}_n$, where n is the number of nodes in $\mathcal{G}$.*

Definition 3 (Invariant graph function). *Let $d > 0$. A function $f : \mathfrak{G}' \subseteq \mathfrak{G} \to \mathcal{I}$ is invariant if $f(\boldsymbol{P}_n \boldsymbol{A} \boldsymbol{P}_n^\intercal, \boldsymbol{P}_n \boldsymbol{X}) = f(\boldsymbol{A}, \boldsymbol{X})$ for any $\mathcal{G} = (\boldsymbol{A}, \boldsymbol{X}) \in \mathfrak{G}'$ and any permutation matrix $\boldsymbol{P}_n$, where n is the number of nodes in $\mathcal{G}$.*

C.1 GRACLUS

Remark 3 (On the invariance of GRACLUS and COMPLEMENT.). *In general, the invariance of pooling methods that rely on graph clustering depends on the invariance of the clustering algorithm itself. For any invariant clustering algorithm, COMPLEMENT (see Section 2) is also invariant. However, this is naturally not the case of GRACLUS as it employs a heuristic that can return different graph cuts depending on initialization.*

C.2 DIFFPOOL

Theorem 1 (Invariance of randomized DIFFPOOL). *The DIFFPOOL variant in Remark 1 is invariant.*

We prove this by showing that each modified pooling layer is individually invariant. Here, we treat the l-th pooling layer as a function $f_l : \mathfrak{G} \to \mathfrak{G}$. Let $\mathcal{G} = \left(\boldsymbol{X}^{(l-1)}, \boldsymbol{A}^{(l-1)} \right)$ be a graph with n_{l-1} nodes and $\boldsymbol{P}_{n_{l-1}}$ be a permutation matrix. Furthermore, we define:

$$\left(\boldsymbol{A}^{(l)}, \boldsymbol{X}^{(l)} \right) := f_l \left(\boldsymbol{A}^{(l-1)}, \boldsymbol{X}^{(l-1)} \right) \text{ and } \left(\boldsymbol{A}^{(l)\prime}, \boldsymbol{X}^{(l)\prime} \right) := f_l \left(\boldsymbol{P}_{n_{l-1}} \boldsymbol{A}^{(l-1)} \boldsymbol{P}_{n_{l-1}}^\intercal, \boldsymbol{P}_{n_{l-1}} \boldsymbol{X}^{(l-1)} \right),$$

so that our job reduces to proving $\left(\boldsymbol{A}^{(l)}, \boldsymbol{X}^{(l)} \right) = \left(\boldsymbol{A}^{(l)\prime}, \boldsymbol{X}^{(l)\prime} \right)$.

Note that modified assignment matrix for a graph with node features $\boldsymbol{X}^{(l-1)}$ is $\tilde{\boldsymbol{S}}^{(l)} = \boldsymbol{X}^{(l-1)} \tilde{\boldsymbol{S}}'$, and thus the assignment matrix for $\boldsymbol{P}_{n_{l-1}} \boldsymbol{X}^{(l-1)}$ is $\boldsymbol{P}_{n_{l-1}} \tilde{\boldsymbol{S}}^{(l)}$. Then, it follows that:

$$\begin{aligned}
\boldsymbol{X}^{(l)\prime} &= \left(\boldsymbol{P}_{n_{l-1}} \tilde{\boldsymbol{S}}^{(l)} \right)^\intercal \mathrm{GNN}_2^{(l)} (\boldsymbol{P}_{n_{l-1}} \boldsymbol{A}^{(l-1)} \boldsymbol{P}_{n_{l-1}}^\intercal, \boldsymbol{P}_{n_{l-1}} \boldsymbol{X}^{(l-1)}) \\
&= \tilde{\boldsymbol{S}}^{(l)\intercal} \boldsymbol{P}_{n_{l-1}}^\intercal \boldsymbol{P}_{n_{l-1}} \mathrm{GNN}_2^{(l)} (\boldsymbol{A}^{(l-1)}, \boldsymbol{X}^{(l-1)}) \\
&= \tilde{\boldsymbol{S}}^{(l)\intercal} \mathrm{GNN}_2^{(l)} (\boldsymbol{A}^{(l-1)}, \boldsymbol{X}^{(l-1)}) \\
&= \boldsymbol{X}^{(l)}
\end{aligned}$$

and for the coarsened adjacency matrix:

$$\begin{aligned}
\boldsymbol{A}^{(l)\prime} &= \left(\boldsymbol{P}_{n_{l-1}} \tilde{\boldsymbol{S}}^{(l)} \right)^\intercal \boldsymbol{P}_{n_{l-1}} \boldsymbol{A}^{(l-1)} \boldsymbol{P}_{n_{l-1}}^\intercal \left(\boldsymbol{P}_{n_{l-1}} \tilde{\boldsymbol{S}}^{(l)} \right) \\
&= \tilde{\boldsymbol{S}}^{(l)\intercal} \boldsymbol{A}^{(l-1)} \tilde{\boldsymbol{S}}^{(l)} \\
&= \boldsymbol{A}^{(l)}
\end{aligned}$$

16

C.3 GMN

Theorem 2 (Invariance of simplified GMN). *If the initial query network is equivariant, our simplified GMN, which employs distances rather than the Student's t-kernel, is invariant.*

We prove this by first proving each modified memory layer is invariant. The final result follows from the fact that the composition of an equivariant function with a series of invariant functions is invariant. Note that memory layer l can be seen as a function $f_l : \mathfrak{G}' \to \mathfrak{G}'$, where $\mathfrak{G}'$ denotes the set of fully connected graphs.

Let $\mathcal{G} = (\boldsymbol{Q}^{(l-1)}, \boldsymbol{1}_{n_{l-1} \times n_{l-1}}) \in \mathfrak{G}'$ and $\boldsymbol{P}_{n_{l-1}}$ be a permutation matrix. Furthermore, let

$$\left(\boldsymbol{Q}^{(l)}, \boldsymbol{1}_{n_l \times n_l}\right) := f_l(\boldsymbol{P}_n \boldsymbol{Q}^{(l-1)}, \boldsymbol{1}_{n_{l-1} \times n_{l-1}})$$

$$\left(\boldsymbol{Q}^{(l)\prime}, \boldsymbol{1}_{n_l \times n_l}\right) := f_l(\boldsymbol{Q}^{(l-1)}, \boldsymbol{1}_{n_{l-1} \times n_{l-1}})$$

To achieve our goal, it suffices to show that $\boldsymbol{Q}^{(l)}$ equals $\boldsymbol{Q}^{(l)\prime}$. Let $\boldsymbol{D}_h^{(l)}$ denote the distance matrix between the queries $\boldsymbol{Q}^{(l-1)}$ and the keys in the h-th head of the l-th memory layer, then

$$\boldsymbol{Q}^{(l)} = \text{ReLU}\left(\text{softmax}\left(\text{1x1Conv}\left(\boldsymbol{D}_1^{(l)}, \dots, \boldsymbol{D}_H^{(l)}\right)\right)^\intercal \boldsymbol{Q}^{(l-1)} \boldsymbol{W}^{(l)}\right)$$

Since the distance matrix for the permuted matrix is simply the permuted distance matrix, it follows that:

$$\begin{aligned}
\boldsymbol{Q}^{(l)\prime} &= \text{ReLU}\left(\text{softmax}\left(\text{1x1Conv}\left(\boldsymbol{P}_{n_{l-1}}\boldsymbol{D}_1^{(l)}, \dots, \boldsymbol{P}_{n_{l-1}}\boldsymbol{D}_H^{(l)}\right)\right)^\intercal \boldsymbol{P}_{n_{l-1}}\boldsymbol{Q}^{(l-1)} \boldsymbol{W}^{(l)}\right) \\
&= \text{ReLU}\left(\text{softmax}\left(\text{1x1Conv}\left(\boldsymbol{D}_1^{(l)}, \dots, \boldsymbol{D}_H^{(l)}\right)\right)^\intercal \boldsymbol{P}_{n_{l-1}}^\intercal \boldsymbol{P}_{n_{l-1}}\boldsymbol{Q}^{(l-1)} \boldsymbol{W}^{(l)}\right) \\
&= \text{ReLU}\left(\text{softmax}\left(\text{1x1Conv}\left(\boldsymbol{D}_1^{(l)}, \dots, \boldsymbol{D}_H^{(l)}\right)\right)^\intercal \boldsymbol{Q}^{(l-1)} \boldsymbol{W}^{(l)}\right) \\
&= \boldsymbol{Q}^{(l)}
\end{aligned}$$

D   Further details on Remark 2

The unsupervised loss employed to learn GMNs consists of a summation of

$$D_{\text{KL}}\left(\boldsymbol{Q}^{(l)} \| \boldsymbol{P}^{(l)}\right) = \sum_{i=1}^{n_{l-1}} \sum_{j=1}^{n_l} P_{ij}^{(l)} \log \frac{P_{ij}^{(l)}}{Q_{ij}^{(l)}}$$

over all layers, where the matrix $\boldsymbol{P}^{(l)} \in \mathbb{R}^{n_{l-1} \times n_l}$ is defined such that

$$P_{ij}^{(l)} = \frac{\left(S_{ij}^{(l)}\right)^2 / \left(\sum_i S_{ij}^{(l)}\right)}{\sum_{j'}\left[\left(S_{ij'}^{(l)}\right)^2 / \left(\sum_i S_{ij'}^{(l)}\right)\right]}.$$

The intuition here is to enforce cluster purity by pushing the probabilities $P_{i:}^{(l)}$ towards their re-normalized squares. A perhaps counter-intuitive outcome of this design is that choosing n_l identical keys, which results in totally uniform cluster assignments, perfectly minimizes the Kullback-Leibler divergence. To verify so, we set $Q_{ij} = 1/n_l$ for all $i = 1 \dots n_{l-1}$ and $j = 1 \dots n_l$, that is

$$P_{ij}^{(l)} = \frac{\frac{n_{l-1}^{-2}}{n_l n_{l-1}^{-1}}}{n_{l-1} \frac{n_{l-1}^{-2}}{n_l n_{l-1}^{-1}}} = \frac{1}{n_{l-1}} \Rightarrow D_{\text{KL}}\left(\boldsymbol{Q}^{(l)} \| \boldsymbol{P}^{(l)}\right) = 0$$

E   Additional embeddings

Figures S2-S13 illustrate the activations throughout the models' main layers, similarly to Figures 5 and 6 from the main text. For better visualization, we force the embeddings (plots) to share a common aspect ratio and include the number of nodes at each layer inside brackets. These numbers correspond to the number of rows of each embedding matrix. Also, the color scale of each embedding matrix is normalized according to its own range. Therefore, color scales are not comparable across different embedding matrices. This allows us to assess the degree of homogeneity in each embedding matrix more accurately. Figures S2-S13 reinforce our findings that GNNs learn smooth signals across features at early layers. This make the pooling strategies less relevant for learning meaningful hierarchical representations.
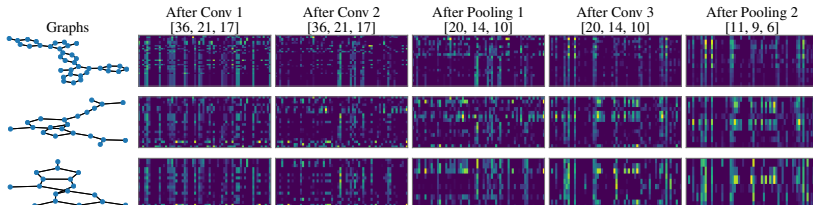
Figure S2: GRACLUS on NCI1.
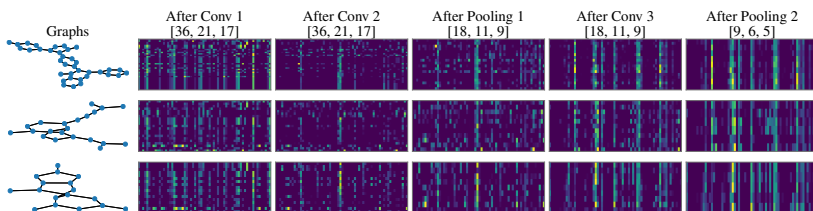
Figure S3: COMPLEMENT on NCI1.
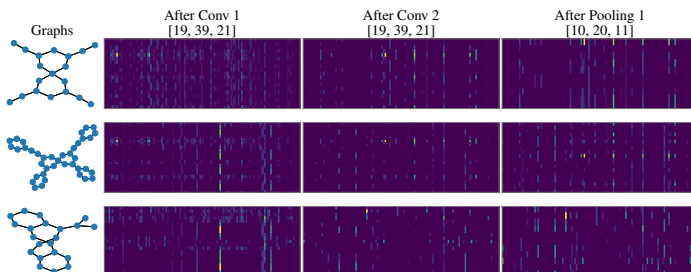
Figure S4: GRACLUS on MOLHIV.
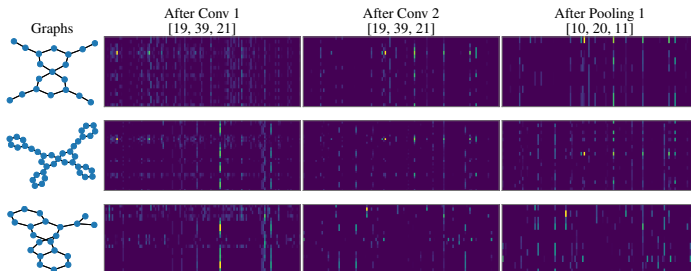
Figure S5: COMPLEMENT on MOLHIV.
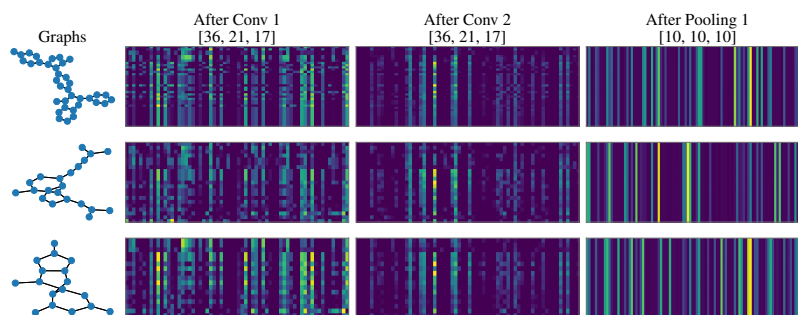
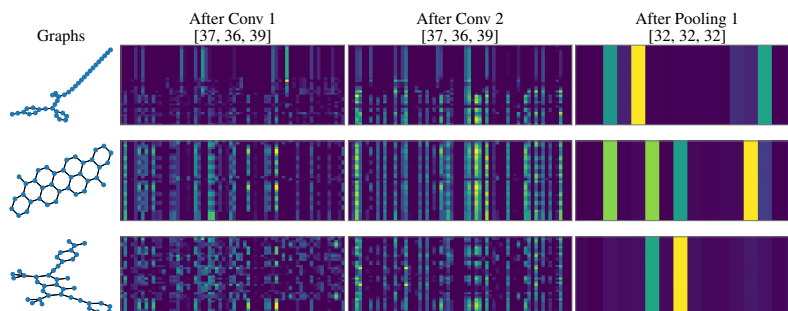Figure S6: GMN on NCI1.
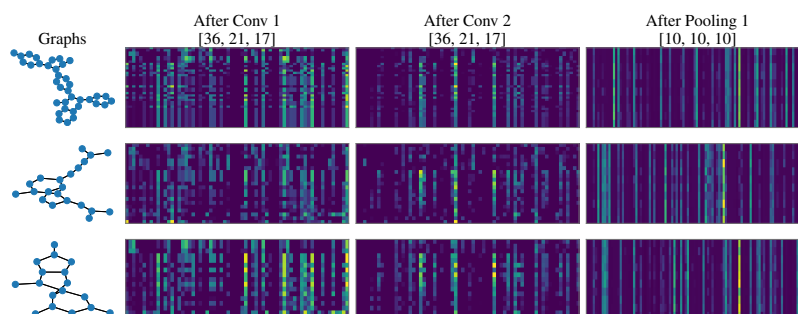
Figure S7: Random-GMN on NCI1.

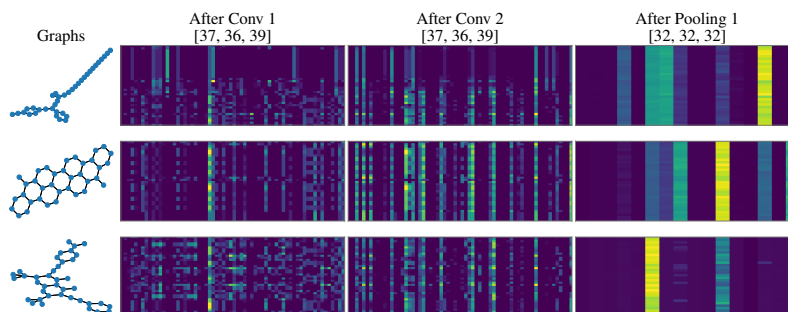
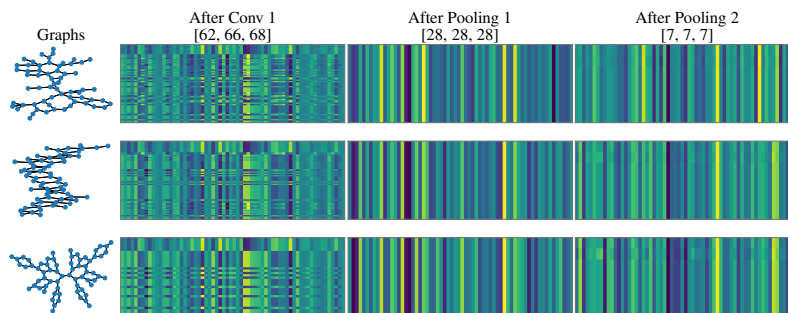Figure S8: GMN on MOLHIV.

Figure S9: Random-GMN on MOLHIV.
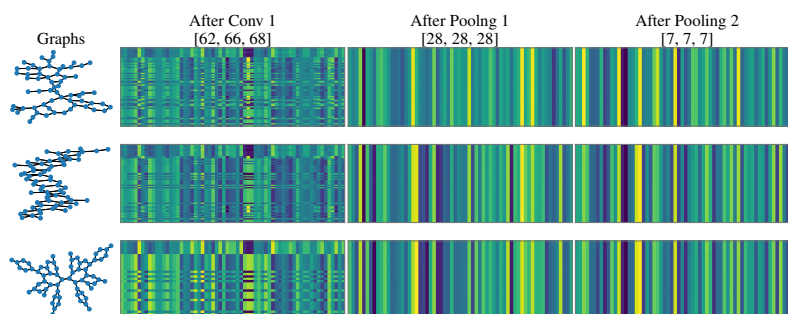
19

Figure S10: DIFFPOOL on NCI1
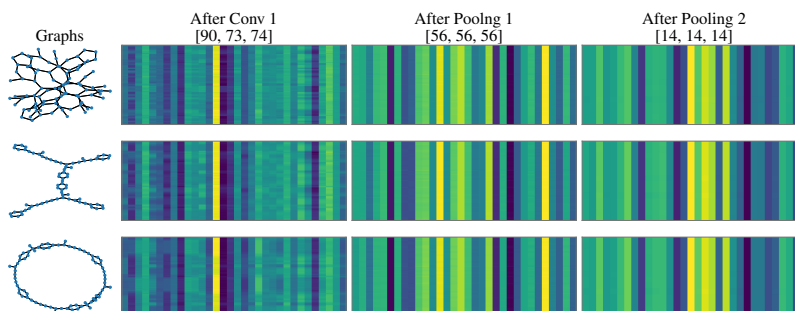

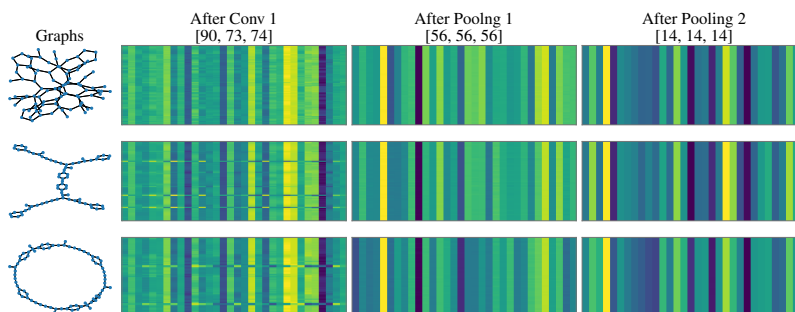Figure S11: $\mathcal{N}$-DIFFPOOL on NCI1.

Figure S12: DIFFPOOL on MOLHIV.

Figure S13: $\mathcal{N}$-DIFFPOOL on MOLHIV.