1. Additional Experiments

1.1. Graph Regression of Molecular Properties on QM9

The QM9 chemical database is a collection of ≈135k small organic molecules, associated to continuous labels describing several geometric, energetic, electronic, and thermodynamic properties.[1] Each molecule in the dataset is represented as a graph $\{\mathbf{A}_i, \mathbf{X}_i\}$, where atoms are associated to nodes, and edges represent chemical bonds. The atomic number of each atom (one-hot encoded; C, N, F, O) is taken as node feature and the type of bond (one-hot encoded; single, double, triple, aromatic) can be used as edge attribute. In this experiment, we ignore the edge attributes in order to use all pooling algorithms without modifications.

The purpose of this experiment is to compare the trainable pooling methods also on a graph regression task, but it must be intended as a proof of concept. In fact, the graphs in this dataset are extremely small (the average number of nodes is 8) and, therefore, a pooling operation is arguably not necessary. We consider a GNN with architecture *MP(32)-pool-MP(32)-GlobalAvgPool-Dense*, where *pool* is implemented by Top-K, Diffpool, or MinCutPool. The network is trained to predict a given chemical property from the input molecular graphs. Performance is evaluated with a 10-fold cross-validation, using 10% of the training set for validation in each split. The GNNs are trained for 50 epochs, using Adam with learning rate 5e-4, batch size 32, and ReLU activations. We use the mean squared error (MSE) as supervised loss.

The MSE obtained on the prediction of each property for different pooling methods is reported in Tab. 1. As expected, the flat baseline with no pooling operation (*MP(32)-MP(32)-GlobalAvgPool-Dense*) yields a lower error in most cases. Contrarily to the graph classification and the AE task, Top-K achieves better results than Diffpool in average. Once again, MinCutPool significantly outperforms the other methods on each regression task and, in one case, also the flat baseline.

Property	**Top-K**	**Diffpool**	**MinCutPool**	**Flat baseline**
mu	0.600$\pm$0.085	0.651$\pm$0.026	**<u>0.538</u>**$\pm$0.012	0.559$\pm$0.007
alpha	0.197$\pm$0.087	0.114$\pm$0.001	<u>0.078</u>$\pm$0.007	**0.065**$\pm$0.006
homo	0.698$\pm$0.102	0.712$\pm$0.015	<u>0.526</u>$\pm$0.021	**0.435**$\pm$0.013
lumo	0.601$\pm$0.050	0.646$\pm$0.013	<u>0.540</u>$\pm$0.005	**0.515**$\pm$0.007
gap	0.630$\pm$0.044	0.698$\pm$0.004	<u>0.584</u>$\pm$0.007	**0.552**$\pm$0.008
r2	0.452$\pm$0.087	0.440$\pm$0.024	<u>0.261</u>$\pm$0.006	**0.204**$\pm$0.006
zpve	0.402$\pm$0.032	0.410$\pm$0.004	<u>0.328</u>$\pm$0.005	**0.284**$\pm$0.005
u0_atom	0.308$\pm$0.055	0.245$\pm$0.006	<u>0.193</u>$\pm$0.002	**0.163**$\pm$0.001
cv	0.291$\pm$0.118	0.337$\pm$0.018	<u>0.148</u>$\pm$0.004	**0.127**$\pm$0.002

Table 1. MSE on the graph regression task. The best results with a statistical significance of $p < 0.05$ are highlighted: the best overall are in bold, the best among pooling methods are underlined.

2. Experimental Details

All GNN architectures in this work have been implemented with the Spektral library.[2] The code to reproduce all experiments is available at `https://github.com/FilippoMB/Spectral-Clustering-with-Graph-Neural-Networks-for-Graph-Pooling`. For the WL kernel, we used the implementation provided in the GraKeL library.[3] The pooling strategy based on Graclus, is taken from the ChebyNets repository.[4]

2.1. Clustering on Citation Networks

Diffpool and MinCutPoolare configured with 16 hidden neurons with linear activations in the MLP and MP layer, respectively used to compute the cluster assignment matrix $\mathbf{S}$. The MP layer used to compute the propagated node features $\mathbf{X}^{(1)}$ uses an ELU activation in both architectures. The learning rate for Adam is 5e-4, and the models are trained for 10000 iterations. The details of the citation networks dataset are reported in Tab. 2.

[1] `http://quantum-machine.org/datasets/`
[2] `https://graphneural.network`
[3] `https://ysig.github.io/GraKeL/dev/`
[4] `https://github.com/mdeff/cnn_graph`

Table 2. Details of the citation networks datasets

Dataset	Nodes	Edges	Node features	Node classes
Cora	2708	5429	1433	7
Citeseer	3327	9228	3703	6
Pubmed	19717	88651	500	3

2.2. Graph Classification

We train the GNN architectures with Adam, an L_2 penalty loss with weight 1e-4, and 16 hidden units (H) both in the MLP of MinCutPool and in the internal MP of Diffpool. *Mutagenicity*, *Proteins*, *DD*, *COLLAB*, and *Reddit-2k* are datasets representing real-world graphs and are taken from the repository of benchmark datasets for graph kernels.[5] *Bench-easy* and *Bench-hard*[6] are datasets where the node features $\mathbf{X}$ and the adjacency matrix $\mathbf{A}$ are completely uninformative if considered alone. Hence, algorithms that account only for the node features or the graph structure will fail to classify the graphs. Since *Bench-easy* and *Bench-hard* come with a train/validation/test split, the 10-fold split is not necessary to evaluate the performance. The statistics of all the datasets are reported in Tab. 3.

Table 3. Summary of statistics of the graph classification datasets

Dataset	samples	classes	avg. nodes	avg. edges	node attr.	node labels
Bench-easy	1800	3	147.82	922.66	–	yes
Bench-hard	1800	3	148.32	572.32	–	yes
Mutagenicity	4337	2	30.32	30.77	–	yes
Proteins	1113	2	39.06	72.82	1	no
DD	1178	2	284.32	715.66	–	yes
COLLAB	5000	3	74.49	2457.78	–	no
Reddit-2K	2000	2	429.63	497.75	–	no

3. Architectures Schemata

Fig. 1 depicts the GNN architecture used in the clustering and segmentation tasks; Fig. 2 depicts the GNN architecture used in the graph classification task; Fig. 3 depicts the GNN architecture used in the graph regression task; Fig. 4 depicts the graph autoencoder used in the graph signal reconstruction task.

[5] https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets
[6] https://github.com/FilippoMB/Benchmark_dataset_for_graph_classification

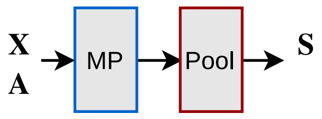

Figure 1. Architecture for clustering/segmentation.

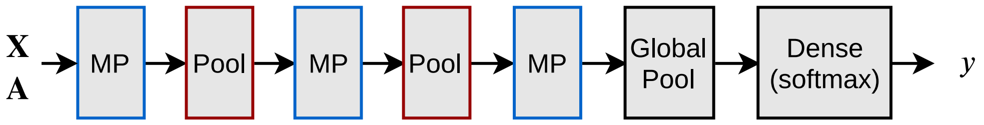

Figure 2. Architecture for graph classification.

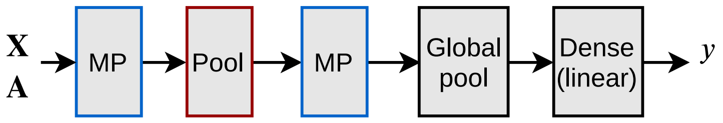

Figure 3. Architecture for graph regression.

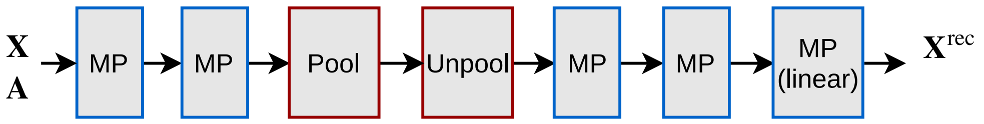

Figure 4. Architecture for the autoencoder.