Spectral Clustering with Graph Neural Networks for Graph Pooling

Filippo Maria Bianchi [*1] **Daniele Grattarola** [*2] **Cesare Alippi** [2 3]

Abstract

Spectral clustering (SC) is a popular clustering technique to find strongly connected communities on a graph. SC can be used in Graph Neural Networks (GNNs) to implement pooling operations that aggregate nodes belonging to the same cluster. However, the eigendecomposition of the Laplacian is expensive and, since clustering results are graph-specific, pooling methods based on SC must perform a new optimization for each new sample. In this paper, we propose a graph clustering approach that addresses these limitations of SC. We formulate a continuous relaxation of the normalized `minCUT` problem and train a GNN to compute cluster assignments that minimize this objective. Our GNN-based implementation is differentiable, does not require to compute the spectral decomposition, and learns a clustering function that can be quickly evaluated on out-of-sample graphs. From the proposed clustering method, we design a graph pooling operator that overcomes some important limitations of state-of-the-art graph pooling techniques and achieves the best performance in several supervised and unsupervised tasks.

1. Introduction

State-of-the-art convolutional neural networks (CNNs) alternate convolutions, which extrapolate local features from the input signal, with pooling, which downsamples the feature maps by computing local summaries of nearby points. Pooling helps CNNs to discard information that is superfluous for the task, provides translation invariance, and keeps model complexity under control by reducing the size of the intermediate representations.

[*]Equal contribution [1]NORCE, the Norwegian Research Centre, Norway [2]Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland [3]DEIB, Politecnico di Milano, Milano, Italy. Correspondence to: Filippo Maria Bianchi <filippombianchi@gmail.com>.

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

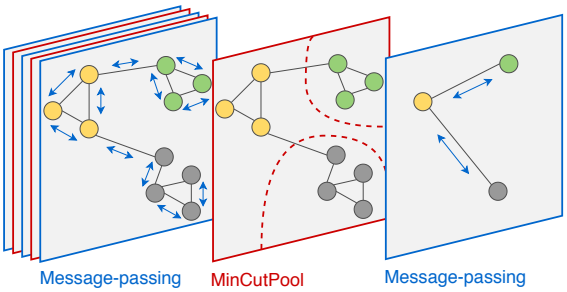


Figure 1. A deep GNN architecture where message-passing is followed by the MinCutPool layer.

Graph Neural Networks (GNNs) extend the convolution operation from regular domains to arbitrary topologies and unordered structures (Battaglia et al., 2018). As in CNNs, graph pooling is an important operation that allows a GNN to learn increasingly more abstract and coarser representations of the input graphs, by summarizing local components and discarding redundant information. The development of pooling strategies for GNNs, however, has lagged behind the design of newer and more effective message-passing (MP) operations (Gilmer et al., 2017), such as graph convolutions. The reason, is mainly due to the difficulty of defining an aggregated version of a graph that effectively supports the pooled node features.

Several approaches have been proposed in recent GNN literature, ranging from model-free methods that precompute the pooled graphs by leveraging graph-theoretical properties (Bruna et al., 2013; Defferrard et al., 2016), to model-based methods that perform pooling trough a learnable function of the node features (Ying et al., 2018; Cangea et al., 2018; Hongyang Gao, 2019). However, existing model-free methods only consider the graph topology but ignore the node features, while model-based methods are mostly based on heuristics. As a consequence, the former cannot learn how to coarsen graphs adaptively for a specific downstream task, while the latter are unstable and prone to find degenerate solutions even on simple tasks. A pooling operation that is theoretically grounded and can adapt to the data and the task is still missing.

Spectral clustering (SC) is a well-known clustering technique that leverages the Laplacian spectrum to find strongly

connected communities on a graph. SC can be used to perform pooling in GNNs by aggregating nodes belonging to the same cluster (Bruna et al., 2013; Defferrard et al., 2016), although the approaches based on this technique suffer from the aforementioned issues of model-free pooling methods. In particular, SC does not explicitly account for the node attributes and the eigendecomposition of the Laplacian is a non-differentiable and expensive operation. Additionally, pooling methods based on SC must compute the spectral decomposition even at test time, since the decomposition is unique for each new graph.

We propose a graph clustering approach that addresses the limitations that hinder the applicability of SC in GNNs. Specifically, we formulate a continuous relaxation of the normalized `minCUT` problem and train a GNN to compute cluster assignments by optimizing this objective. Our approach learns the solution found by SC while also accounting explicitly for the node features to identify clusters. At the same time, our GNN-based implementation is differentiable and does not require to compute the expensive spectral decomposition of the Laplacian, since it exploits spatially localized graph convolutions that are fast to compute. This also allows to cluster the nodes of out-of-sample graphs simply by evaluating the learned function.

From the proposed clustering method, we derive a model-based pooling operator called *MinCutPool*, which overcomes the disadvantages of both model-free and model-based pooling methods. The parameters in a MinCutPool layer are learned by minimizing the `minCUT` objective, which can be jointly optimized with a task-specific loss. In the latter case, the `minCUT` loss acts as a regularization term, which prevents degenerate solutions, and the GNN can find the optimal trade-off between task-specific and clustering objectives. Because they are fully differentiable, MinCutPool layers can be stacked at different levels of a GNN to obtain a hierarchical representation and the overall architecture can be trained end-to-end (Figure 1). We perform a comparative study on a variety of unsupervised and supervised tasks and show that MinCutPool leads to significant improvements over state-of-the-art pooling methods.

2. Background

Let a graph be represented by a tuple $G = \{\mathcal{V}, \mathcal{E}\}$, $|\mathcal{V}| = N$, with node set $\mathcal{V}$ and edge set $\mathcal{E}$. Each node is associated with a vector attribute in $\mathbb{R}^F$. A graph is characterized by its adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ and the node features $\mathbf{X} \in \mathbb{R}^{N\times F}$.

2.1. Graph Neural Networks

Several approaches have been proposed to process graphs with neural networks, including recurrent architectures (Scarselli et al., 2009; Li et al., 2016; Gallicchio & Micheli, 2020) or convolutional operations inspired by filters used in graph signal processing (Defferrard et al., 2016; Kipf & Welling, 2017; Bianchi et al., 2019a). We base our GNN architecture on a simple MP operation that combines the features of each node with its first-order neighbours. We adopt a MP implementation that does not require to modify the graph by adding self-loops (like in, e.g., (Kipf & Welling, 2017)) but accounts for the initial node features through a skip connection.

Let $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \in \mathbb{R}^{N\times N}$ be the symmetrically normalized adjacency matrix, where $\mathbf{D} = \text{diag}(\mathbf{A}\mathbf{1}_N)$ is the degree matrix. The output of the MP layer is

$$\bar{\mathbf{X}} = MP(\mathbf{X}, \tilde{\mathbf{A}}) = \text{ReLU}(\tilde{\mathbf{A}}\mathbf{X}\boldsymbol{\Theta}_m + \mathbf{X}\boldsymbol{\Theta}_s), \quad (1)$$

where $\boldsymbol{\Theta}_m$ and $\boldsymbol{\Theta}_s$ are the trainable weights of the mixing and skip components of the layer, respectively.

2.2. `minCUT` and Spectral Clustering

Given a graph $G = \{\mathcal{V}, \mathcal{E}\}$, the *$K$-way normalized* `minCUT` problem (simply referred to as `minCUT`) is the task of partitioning $\mathcal{V}$ in K disjoint subsets by removing the minimum volume of edges. The problem is equivalent to maximizing

$$\frac{1}{K}\sum_{k=1}^{K}\frac{\text{links}(\mathcal{V}_k)}{\text{degree}(\mathcal{V}_k)} = \frac{1}{K}\sum_{k=1}^{K}\frac{\sum_{i,j\in\mathcal{V}_k}\mathcal{E}_{i,j}}{\sum_{i\in\mathcal{V}_k, j\in\mathcal{V}\setminus\mathcal{V}_k}\mathcal{E}_{i,j}}, \quad (2)$$

where the numerator counts the edge volume within each cluster, and the denominator counts the edges between the nodes in a cluster and the rest of the graph (Shi & Malik, 2000). Let $\mathbf{C} \in \{0,1\}^{N\times K}$ be a *cluster assignment matrix*, so that $\mathbf{C}_{i,j} = 1$ if node i belongs to cluster j, and 0 otherwise. The `minCUT` problem can be expressed as (Dhillon et al., 2004):

$$\begin{aligned} &\text{maximize} \ \ \frac{1}{K}\sum_{k=1}^{K}\frac{\mathbf{C}_k^T\mathbf{A}\mathbf{C}_k}{\mathbf{C}_k^T\mathbf{D}\mathbf{C}_k}, \\ &\text{s.t.} \ \ \mathbf{C} \in \{0,1\}^{N\times K}, \ \ \mathbf{C}\mathbf{1}_K = \mathbf{1}_N \end{aligned} \quad (3)$$

where $\mathbf{C}_k$ is the k-th column of $\mathbf{C}$. Since problem (3) is NP-hard, it is recast in a relaxed continuous formulation that can be solved in polynomial time and guarantees a near-optimal solution (Yu & Shi, 2003):

$$\begin{aligned} &\underset{\mathbf{Q}\in\mathbb{R}^{N\times K}}{\arg\max} \ \ \frac{1}{K}\sum_{k=1}^{K}\mathbf{Q}_k^T\mathbf{A}\mathbf{Q}_k \\ &\text{s.t.} \ \mathbf{Q} = \mathbf{C}(\mathbf{C}^T\mathbf{D}\mathbf{C})^{-\frac{1}{2}}, \ \mathbf{Q}^T\mathbf{Q} = \mathbf{I}_K. \end{aligned} \quad (4)$$

While problem (4) is still non-convex, there exists an optimal solution $\mathbf{Q}^* = \mathbf{U}_K\mathbf{O}$, where $\mathbf{U}_K \in \mathbb{R}^{N\times K}$ contains the eigenvectors of $\mathbf{A}$ corresponding to the K largest

eigenvalues, and $\mathbf{O} \in \mathbb{R}^{K \times K}$ is an orthogonal transformation (Ikebe et al., 1987).

Spectral clustering (SC) obtains the cluster assignments by applying k-means to the rows of $\mathbf{Q}^*$, which are node embeddings in the Laplacian eigenspace (Von Luxburg, 2007). One of the main limitations of SC lies in the computation of the spectrum of $\mathbf{A}$ to obtain $\mathbf{Q}^*$, which has a memory complexity of $\mathcal{O}(N^2)$ and a computational complexity of $\mathcal{O}(N^3)$. This prevents its applicability to large datasets.

To deal with the scalability issues of SC, the constrained optimization in (4) can be solved by gradient descent algorithms that refine the solution by iterating operations whose individual complexity is $\mathcal{O}(N^2)$, or even $\mathcal{O}(N)$ (Han & Filippone, 2017). These algorithms search the solution on the manifold induced by the orthogonality constraint on the columns of $\mathbf{Q}$, by performing gradient updates along the geodesics (Wen & Yin, 2013; Collins et al., 2014). Alternative approaches rely on QR factorization to constrain the space of feasible solutions (Damle et al., 2016), and alleviate the cost $\mathcal{O}(N^3)$ of the factorization by ensuring that orthogonality holds only on one minibatch at a time (Shaham et al., 2018). Dhillon et al. (2007) discuss the equivalence between graph clustering objectives and the kernel k-means algorithm, and their Graclus algorithm is a popular model-free method for hierarchical pooling in GNNs (Defferrard et al., 2016).

To learn a model that finds an approximate SC solution also for out-of-sample graphs, several works propose to use neural networks. In (Tian et al., 2014), an autoencoder is trained to map the i^{th} row of the Laplacian to the i^{th} components of the first K eigenvectors. Yi et al. (2017) define an orthogonality constraint to learn spectral embeddings as a volumetric reparametrization of a precomputed Laplacian eigenbasis. Finally, Shaham et al. (2018) propose a loss function to cluster generic data and process out-of-sample data at inference time. While these approaches learn to embed data in the Laplacian eigenspace of the given graph, they rely on non-differentiable operations to compute the cluster assignments and, therefore, are not suitable to perform pooling in a GNN trained end-to-end.

3. Spectral Clustering with GNNs

We propose a GNN-based approach that addresses the aforementioned limitations of SC algorithms and that clusters the nodes according to the graph topology (nodes in the same cluster should be strongly connected) and to the node features (nodes in the same cluster should have similar features). Our method assumes that node features represent a good initialization for computing the cluster assignments. This is a realistic assumption due to the homophily property of many real-world networks (McPherson et al., 2001). Additionally, even in disassortative networks (i.e., networks where dissimilar nodes are likely to be connected (Newman, 2003)), the features of nodes in strongly connected communities tend to become similar due to the smoothing effect of MP operation.

Let $\bar{\mathbf{X}}$ be the matrix of node representations yielded by one or more MP layers. We compute a cluster assignment of the nodes using a multi-layer perceptron (MLP) with softmax on the output layer, which maps each node feature $\mathbf{x}_i$ into the i^{th} row of a soft cluster assignment matrix $\mathbf{S}$:

$$\begin{aligned} \bar{\mathbf{X}} &= \text{GNN}(\mathbf{X}, \tilde{\mathbf{A}}; \mathbf{\Theta}_{\text{GNN}}) \\ \mathbf{S} &= \text{MLP}(\bar{\mathbf{X}}; \mathbf{\Theta}_{\text{MLP}}), \end{aligned} \tag{5}$$

where $\mathbf{\Theta}_{\text{GNN}}$ and $\mathbf{\Theta}_{\text{MLP}}$ are trainable parameters. The softmax activation of the MLP guarantees that $\mathbf{s}_{ij} \in [0, 1]$ and enforces the constraints $\mathbf{S}\mathbf{1}_K = \mathbf{1}_N$ inherited from the optimization problem in (3). We note that it is possible to add a temperature parameter to the Softmax in the MLP to control how much $\mathbf{s}_i$ should be close to a one-hot vector, *i.e.*, the level of fuzziness in the cluster assignments.

The parameters $\mathbf{\Theta}_{\text{GNN}}$ and $\mathbf{\Theta}_{\text{MLP}}$ are jointly optimized by minimizing an unsupervised loss function $\mathcal{L}_u$ composed of two terms, which approximates the relaxed formulation of the `minCUT` problem:

$$\mathcal{L}_u = \mathcal{L}_c + \mathcal{L}_o = \underbrace{-\frac{Tr(\mathbf{S}^T\tilde{\mathbf{A}}\mathbf{S})}{Tr(\mathbf{S}^T\tilde{\mathbf{D}}\mathbf{S})}}_{\mathcal{L}_c} + \underbrace{\left\| \frac{\mathbf{S}^T\mathbf{S}}{\|\mathbf{S}^T\mathbf{S}\|_F} - \frac{\mathbf{I}_K}{\sqrt{K}} \right\|_F}_{\mathcal{L}_o}, \tag{6}$$

where $\| \cdot \|_F$ indicates the Frobenius norm.

The *cut loss* term, $\mathcal{L}_c$, evaluates the `minCUT` given by the soft cluster assignment $\mathbf{S}$ and is bounded by $-1 \leq \mathcal{L}_c \leq 0$. Minimizing $\mathcal{L}_c$ encourages strongly connected nodes to be clustered together, since the inner product $\langle \mathbf{s}_i, \mathbf{s}_j \rangle$ increases when $\tilde{a}_{i,j}$ is large. $\mathcal{L}_c$ has a single maximum, reached when the numerator $Tr(\mathbf{S}^T\tilde{\mathbf{A}}\mathbf{S}) = \frac{1}{K}\sum_{k=1}^{K} \mathbf{S}_k^T\tilde{\mathbf{A}}\mathbf{S}_k = 0$. This occurs if, for each pair of connected nodes (i.e., $\tilde{a}_{i,j} > 0$), the cluster assignments are orthogonal (i.e., $\langle \mathbf{s}_i, \mathbf{s}_j \rangle = 0$). $\mathcal{L}_c$ reaches its minimum, -1, when $Tr(\mathbf{S}^T\tilde{\mathbf{A}}\mathbf{S}) = Tr(\mathbf{S}^T\tilde{\mathbf{D}}\mathbf{S})$. This occurs when, in a graph with K disconnected components, the cluster assignments are equal for all the nodes in the same component and orthogonal to the cluster assignments of nodes in different components. However, $\mathcal{L}_c$ is a non-convex function and its minimization can lead to local minima or degenerate solutions. For example, given a connected graph, a trivial - yet optimal - solution is the one that assigns all nodes to the same cluster. As a consequence of the continuous relaxation, another degenerate minimum occurs when the cluster assignments are all uniform, that is, all nodes are equally assigned to all clusters. This problem is exacerbated by the MP operations, whose smoothing effect makes the node features

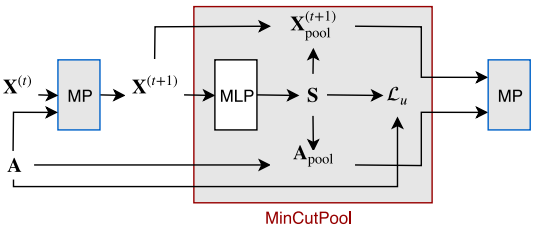


Figure 2. Schema of the MinCutPool layer.

more uniform.

To penalize the degenerate minima of $\mathcal{L}_c$, the orthogonality loss term $\mathcal{L}_o$ encourages the cluster assignments to be orthogonal and the clusters to be of similar size. Since the two matrices in $\mathcal{L}_o$ have unitary norm, it is easy to see that $0 \leq \mathcal{L}_o \leq 2$. Therefore, $\mathcal{L}_o$ is commensurable to $\mathcal{L}_c$ and the two terms can be safely summed without rescaling them (see Fig. 5 for an example). $\mathbf{I}_K$ can be interpreted as a (rescaled) clustering matrix $\mathbf{I}_K = \hat{\mathbf{S}}^T\hat{\mathbf{S}}$, where $\hat{\mathbf{S}}$ assigns exactly N/K points to each cluster. The value of the Frobenius norm between clustering matrices is not biased by differences in the size of the clusters (Law et al., 2017) and, thus, can be used to optimize intra-cluster variance.

While traditional SC requires to compute the spectral decomposition for every new sample, here the cluster assignments are computed by a neural network that learns a mapping from the *nodes feature space* to the *clusters assignment space*. Since the neural network parameters are independent of the graph size, and since the MP operations in the GNN are localized in the node space and independent from the spectrum of the Laplacian, the proposed clustering approach generalizes to unseen graphs at inference time. This also gives the opportunity of training our network on small graphs and then use it to cluster larger ones.

4. Pooling and Graph Coarsening

The methodology proposed in Section 3 is a general technique that can be used to solve clustering tasks on any data represented by graphs. In this work, we focus on using it to perform pooling in GNNs and introduce the MinCutPool layer, which exploits the cluster assignment matrix $\mathbf{S}$ in (5) to generate a coarsened version of the graph. Fig. 2 depicts a scheme of the MinCutPool layer.

The coarsened adjacency matrix and the pooled vertex features are computed, respectively, as

$$\mathbf{A}^{pool} = \mathbf{S}^T\tilde{\mathbf{A}}\mathbf{S}; \quad \mathbf{X}^{pool} = \mathbf{S}^T\mathbf{X}, \tag{7}$$

where the entry $x^{pool}_{i,j}$ in $\mathbf{X}^{pool} \in \mathbb{R}^{K\times F}$ is the sum of feature j among the elements in cluster i, weighted by the cluster assignment scores. $\mathbf{A}^{pool} \in \mathbb{R}^{K\times K}$ is a symmetric matrix, whose entries $a^{pool}_{i,i}$ indicate the weighted sum of edges between the nodes in the cluster i, while $a^{pool}_{i,j}$ is the weighted sum of edges between cluster i and j. Since $\mathbf{A}^{pool}$ corresponds to the numerator of $\mathcal{L}_c$ in (7), the trace maximization yields clusters with many internal connections and weakly connected to each other. Hence, $\mathbf{A}^{pool}$ will be a diagonal-dominant matrix that describes a graph with self-loops much stronger than any other connection. Since very strong self-loops hamper the propagation across adjacent nodes in the MP operations following the pooling layer, we compute the new adjacency matrix $\tilde{\mathbf{A}}^{pool}$ by zeroing the diagonal and applying degree normalization

$$\hat{\mathbf{A}} = \mathbf{A}^{pool} - \mathbf{I}_K\text{diag}(\mathbf{A}^{pool}); \quad \tilde{\mathbf{A}}^{pool} = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}. \tag{8}$$

Because our GNN-based implementation of SC is fully differentiable, MinCutPool layers can be used to build deep GNNs that hierarchically coarsen the graph representation. The parameters of each MinCutPool layer can then be learned end-to-end, by jointly optimizing $\mathcal{L}_u$ along with any supervised loss for a particular downstream task. Contrarily to SC methods that search for feasible solutions only within the space of orthogonal matrices, $\mathcal{L}_o$ only introduces a soft constraint that can be partially violated during the learning procedure. This allows the GNN to find the best trade-off between $\mathcal{L}_u$ and the supervised loss, and makes it possible to handle graphs with intrinsically imbalanced clusters. Since $\mathcal{L}_c$ is non-convex, the violation of the orthogonality constraint could compromise the convergence to the global optimum of the `minCUT` objective. However, we note that:

1. since MinCutPool computes the cluster assignments from node features that become similar due to MP operations, clusters are likely to contain nodes that are both strongly connected and with similar features, reducing the risk of finding a degenerate solution;
2. the degenerate minima of $\mathcal{L}_c$ lead to discarding most of the information from the input graph and, therefore, optimizing the task-specific loss encourages the GNN to avoid them;
3. since the `minCUT` objective acts mostly as a regularization term, a solution that is sub-optimal for (4) could instead be preferable for the supervised downstream task;

For these reasons, we show in Section 5 that MinCutPool never yields degenerate solutions in practice, but consistently achieves good performance on a variety of tasks.

4.1. Computational Complexity

The space complexity of the MinCutPool layer is $\mathcal{O}(NK)$, as it depends on the dimension of the cluster assignment

matrix $\mathbf{S} \in \mathbb{R}^{N \times K}$. The computational complexity is dominated by the numerator in the term $\mathcal{L}_c$, and is $\mathcal{O}(N^2K + NK^2) = \mathcal{O}(NK(K+N))$. Since $\tilde{\mathbf{A}}$ is usually sparse, we can exploit operations for sparse tensors and reduce the complexity of the first matrix multiplication to $\mathcal{O}(EK)$, where E is the number of non-zero edges in $\tilde{\mathbf{A}}$. Since the sparse multiplication yields a dense matrix, the second multiplication still costs $\mathcal{O}(NK^2)$ and the total cost is $\mathcal{O}(K(E+NK))$.

4.2. Related Work on Pooling in GNNs

In this section, we summarize some related works on graph pooling in GNNs covering the two main families of methods: model-free and model-based.

Model-Free Pooling These methods pre-compute the coarsened graphs based on the topology ($\mathbf{A}$) but do not take explicitly into consideration the original node features ($\mathbf{X}$), or the intermediate node representation produced by the GNN. During the forward pass of the GNN, the node features are aggregated with simple procedures and matched to the pre-computed graph structures. These methods are less flexible and cannot adapt to different tasks, but provide a stronger inductive bias that can help to avoid degenerate solutions (e.g., coarsened graphs collapsing in a single node).

One of the most popular model-free method is Graclus (Dhillon et al., 2007) (adopted in (Defferrard et al., 2016; Fey et al., 2018; Monti et al., 2017)), which implements an equivalent formulation of SC based on the less expensive kernel k-means algorithm. At each pooling level, Graclus identifies pairs of maximally similar nodes to be clustered together into a new vertex. During each forward pass of the GNN, max pooling is used to determine the node attribute to keep from each pair. Fake vertices are added so that the number of nodes can be halved each time, although this injects noisy information in the graph.

Node decimation pooling (NDP) is a method originally proposed in graph signal processing literature (Shuman et al., 2016), which has been adapted to perform pooling in GNNs (Simonovsky & Komodakis, 2017; Bianchi et al., 2019b). The nodes are partitioned in two sets, according to the signs of the Laplacian eigenvector associated with the largest eigenvalue. One of the two sets is dropped, reducing the number of nodes by approximately half. Kron reduction is then used to generate the coarsened graphs by connecting the remaining nodes.

Model-Based Pooling These approaches learn how to a coarsen graphs through learnable functions, which take as input the nodes features $\mathbf{X}$ and are parametrized by weights optimized on the task and data at hand. While being more flexible, model-based approaches are mostly based on heuristics and, as shown in Sec. 5, tend to fail in several cases, leading to instability in the training process.

DiffPool (Ying et al., 2018) is a pooling method that uses a MP layer to compute a clustering of the input graphs. DiffPool was one of the first attempts at learning a pooling operator end-to-end, and is regularized by minimizing the entropy of the cluster assignment along with a link prediction loss.

The approach dubbed *Top-K* pooling (Hongyang Gao, 2019; Lee et al., 2019), learns a projection vector that is applied to each node feature to obtain a score. The nodes with the K highest scores are retained, while the others are dropped. Since the top-K selection is not differentiable, the scores are also used as a gate (or attention) for the node features, letting the projection vector be trained with backpropagation. Top-K is generally less effective than DiffPool in benchmarks but is significantly more memory efficient as it avoids generating dense cluster assignments.

We conduct an in-depth comparative analysis of MinCutPool, DiffPool, and Top-K in Section 5, showing some significant drawbacks of the two previous learnable approaches w.r.t. to our method, and highlighting how the inductive bias inherited by SC leads to significant improvements in performance in MinCutPool.

5. Experiments

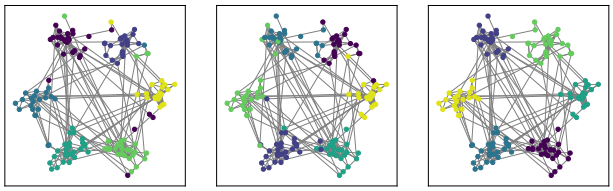
We consider both supervised and unsupervised tasks to compare MinCutPool with traditional SC and with other GNN pooling strategies. The Appendix provides further details on the experiments and a schematic depiction of the architectures used in each task. In addition, the Appendix reports an additional experiment on supervised graph regression. The implementation of MinCutPool is available both in Spektral[1] and Pytorch Geometric [2].

5.1. Clustering the Graph Nodes

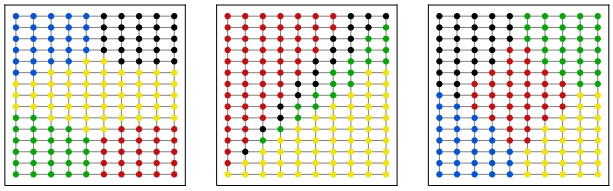
In this experiment, we evaluate the quality of the assignments $\mathbf{S}$ found by our method on two unsupervised tasks. We implement a one-layer GNN followed by a single-layer MLP to compute $\mathbf{S}$, and train the overall architecture by minimizing $\mathcal{L}_u$. For comparison, we configure a similar GNN architecture based on DiffPool where we optimize the auxiliary DiffPool losses (see Sec. 4.2) without any additional supervised loss. We also consider the clusters found by SC, which, unlike our approach, are based on the

[1] https://graphneural.network/layers/pooling/#mincutpool

[2] https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.dense.mincut_pool.dense_mincut_pool



(a) SC (b) DiffPool (c) MinCutPool



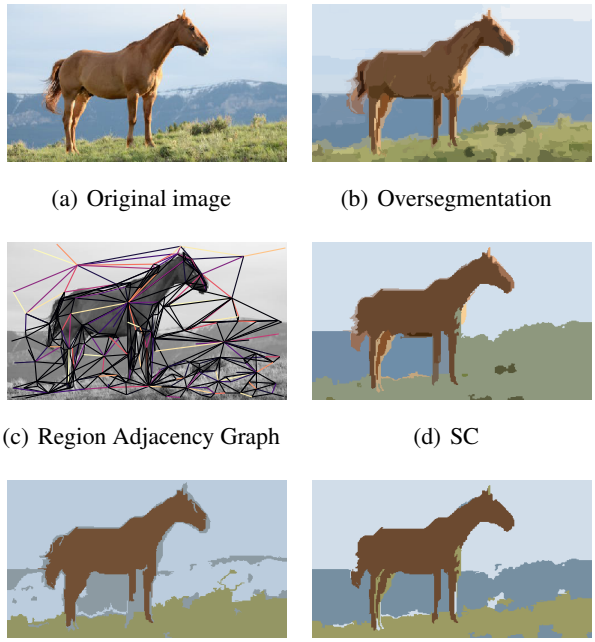
(d) SC (e) DiffPool (f) MinCutPool

Figure 3. Node clustering on a community network (K=6) and on a grid graph (K=5).

spectrum of the graph. In the results, our approach is always indicated as MinCutPool for simplicity, although this experiment only focuses on the clustering results and does not involve the coarsening step. A similar consideration holds for DiffPool.

Clustering on Synthetic Networks We consider two simple graphs: the first is a network with 6 communities and the second is a regular grid. The adjacency matrix $\mathbf{A}$ is binary and the features $\mathbf{X}$ are the 2-D node coordinates. Fig. 3 depicts the node partitions generated by SC (a, d), DiffPool (b, e), and MinCutPool (c, f). MinCutPool generates very accurate and balanced partitions, demonstrating that the cluster assignment matrix $\mathbf{S}$ is well-formed. In particular, we note that the inductive bias carried by the node features in MinCutPool leads to a different clustering than SC but results in an overall better performance (c.f. following discussion and Figure 5). On the other hand, DiffPool assigns some nodes to the wrong community in the first example and produces an unbalanced partition of the grid.

Image Segmentation Given an image, we build a region adjacency graph (Trémeau & Colantoni, 2000) using as nodes the regions generated by an over-segmentation procedure (Felzenszwalb & Huttenlocher, 2004). The SC technique used in this example is the recursive normalized cut (Shi & Malik, 2000), which recursively clusters the nodes until convergence. For MinCutPool and DiffPool, node features consist of the average and total colour in each over-segmented region. We set the number of desired clusters to $K = 4$. The results in Fig. 4 show that MinCutPool yields a precise and intuitive segmentation. On the other hand, SC and DiffPool aggregate wrong regions and, also, SC finds too many segments.

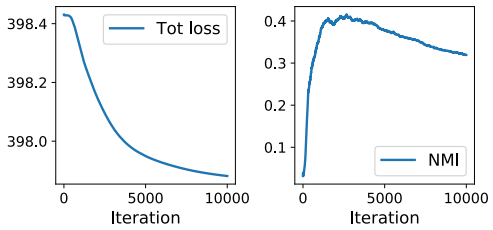


(a) Original image (b) Oversegmentation

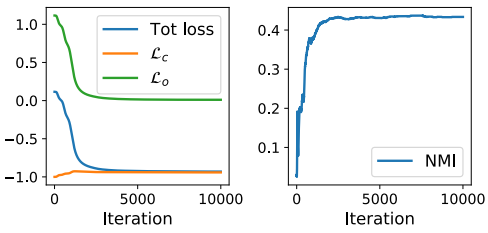
(c) Region Adjacency Graph (d) SC

(e) DiffPool ($K = 4$) (f) MinCutPool ($K = 4$)

Figure 4. Image segmentation by clustering the nodes of the Region Adjacency Graph.



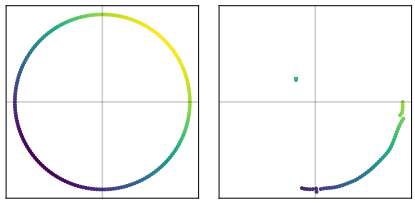
(a) DiffPool



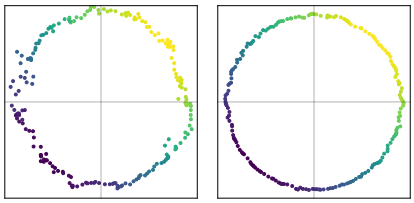
(b) MinCutPool

Figure 5. Unsupervised losses and NMI of DiffPool and MinCutPool on Cora.

Clustering on Citation Networks We cluster the nodes of three citation networks: Cora, Citeseer, and Pubmed. The nodes are documents represented by sparse bag-of-words feature vectors and the binary undirected edges indicate citation links between documents. Each node is labelled

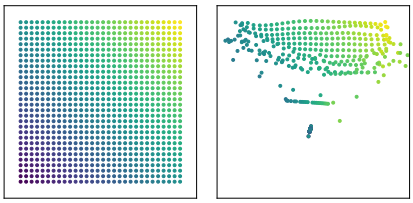


(a) Original (b) Top-K

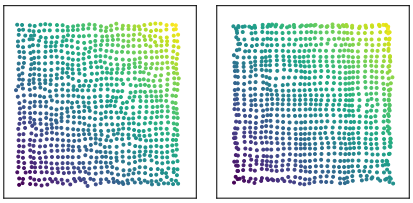


(c) DiffPool (d) MinCutPool

Figure 6. AE reconstruction of a ring graph



(a) Original (b) Top-K



(c) DiffPool (d) MinCutPool

Figure 7. AE reconstruction of a grid graph

with the document class, which we use as ground truth for the clusters. To evaluate the partitions generated by each method, we check the agreement between the cluster assignments and the true labels. Tab. 1 reports the Completeness Score $\text{CS}(\tilde{\mathbf{y}}, \mathbf{y}) = 1 - \frac{H(\tilde{\mathbf{y}}|\mathbf{y})}{H(\tilde{\mathbf{y}})}$ and Normalized Mutual Information $\text{NMI}(\tilde{\mathbf{y}}, \mathbf{y}) = \frac{H(\tilde{\mathbf{y}}) - H(\tilde{\mathbf{y}}|\mathbf{y})}{\sqrt{H(\tilde{\mathbf{y}}) - H(\mathbf{y})}}$, where $H(\cdot)$ is the entropy.

Once again, our GNN architecture achieves a higher NMI score than SC, which does not account explicitly for the node features when generating the clusters. MinCutPool outperforms also DiffPool, since the minimization of the unsupervised loss in DiffPool fails to converge to a good solution. A pathological behaviour in DiffPool is revealed by Fig. 5, which depicts the evolution of the NMI scores as the unsupervised losses in DiffPool and MinCutPool are minimized in training (note how the NMI of DiffPool eventually decreases). From Fig. 5, it is also possible to see the interaction between the `minCUT` loss and the orthogonality loss in our approach. In particular, the `minCUT` loss does not converge to its minimum ($\mathcal{L}_c = -1$), corresponding to one of the degenerate solutions discussed in Sec. 3. Instead, MinCutPool learns the optimal trade-off between $\mathcal{L}_c$ and $\mathcal{L}_o$ and achieves a better and more stable clustering performance than DiffPool.

5.2. GNN Autoencoder

To quantify the amount of information retained by different pooling layers, we train an autoencoder (AE) to reconstruct an input graph signal $\mathbf{X}$ from its pooled version. We compare MinCutPool, DiffPool, and Top-K. The AE is trained by minimizing the mean squared error between the original and the reconstructed graph features, $\|\mathbf{X} - \mathbf{X}^{\text{rec}}\|^2$. All the pooling operations keep 25% of the original nodes.

To upscale the coarsened graph back to its original size, in MinCutPool we transpose the pooling operations:

$$\mathbf{X}^{\text{rec}} = \mathbf{S}\mathbf{X}^{\text{pool}}; \quad \mathbf{A}^{\text{rec}} = \mathbf{S}\mathbf{A}^{\text{pool}}\mathbf{S}^T. \tag{9}$$

A similar operation is performed for DiffPool. For Top-K, we use the un-pooling operation from the original paper (Hongyang Gao, 2019).

Fig. 6 and 7 report the original graph signal $\mathbf{X}$ (the node features are the 2-D coordinates of the nodes) and the reconstruction $\mathbf{X}^{\text{rec}}$ obtained by the different pooling methods, for a ring and a regular grid graph. The reconstruction produced by DiffPool is worse for the ring graph, but is good for the grid graph, while MinCutPool yields almost perfect results in both cases. On the other hand, Top-K fails to generate a coarsened graph that maintains enough information to reconstruct the original graph.

This experiment highlights a major issue in Top-K pooling, which retains the nodes associated with the highest K values of a score vector $\mathbf{s}$, computed by projecting the node features onto a trainable vector $\mathbf{p}$: $\mathbf{s} = \mathbf{X}\mathbf{p}$. Nodes that are connected on the graph usually share similar features, and their similarity further increases after the MP operations, which combine the features of neighbouring nodes. Retaining the nodes associated with the highest K scores in $\mathbf{s}$ corresponds to keeping those nodes that are alike and highly connected, as it can be seen in Fig. 6-7. Therefore, Top-K drops entire portions of the graph, making it impossible to recover the discarded information. This explains why Top-K fails to recover the original graph signal when used as bottleneck for the AE, and motivates the lower performance of Top-K in graph classification (Sec. 5.3).

Table 1. NMI and CS obtained by clustering the nodes on citation networks over 10 different runs. The number of clusters K is equal to the number of node classes.

Dataset	K	Spectral clustering		DiffPool		MinCutPool	
		NMI	CS	NMI	CS	NMI	CS
Cora	7	0.025 ± 0.014	0.126 ± 0.042	0.315 ± 0.005	0.309 ± 0.005	**0.404** ± 0.018	**0.392** ± 0.018
Citeseer	6	0.014 ± 0.003	0.033 ± 0.000	0.139 ± 0.016	0.153 ± 0.020	**0.287** ± 0.047	**0.283** ± 0.046
Pubmed	3	0.182 ± 0.000	**0.261** ± 0.000	0.079 ± 0.001	0.085 ± 0.001	**0.200** ± 0.020	0.197 ± 0.019

Table 2. Graph classification accuracy. Significantly better results ($p < 0.05$) are in bold.

Dataset	WL	Dense	No-pool	Graclus	NDP	DiffPool	Top-K	SAGpool	MinCutPool
Bench-easy	92.6	29.3±0.3	98.5±0.3	97.5±0.5	97.9±0.5	98.6±0.4	82.4±8.9	84.2±2.3	**99.0±0.0**
Bench-hard	60.0	29.4±0.3	67.6±2.8	69.0±1.5	**72.6±0.9**	69.9±1.9	42.7±15.2	37.7±14.5	**73.8±1.9**
Mutagenicity	**81.7**±1.1	68.4±0.3	78.0±1.3	74.4±1.8	77.8±2.3	77.6±2.7	71.9±3.7	72.4±2.4	79.9±2.1
Proteins	71.2±2.6	68.7±3.3	72.6±4.8	68.6±4.6	73.3±3.7	72.7±3.8	69.6±3.5	70.5±2.6	**76.5±2.6**
DD	78.6±2.7	70.6±5.2	76.8±1.5	70.5±4.8	72.0±3.1	**79.3±2.4**	69.4±7.8	71.5±4.5	**80.8±2.3**
COLLAB	74.8±1.3	79.3±1.6	**82.1±1.8**	77.1±2.1	79.1±1.5	81.8±1.4	79.3±1.8	79.2±2.0	**83.4±1.7**
Reddit-Binary	68.2±1.7	48.5±2.6	80.3±2.6	79.2±0.4	84.3±2.4	86.8±2.1	74.7±4.5	73.9±5.1	**91.4±1.5**

5.3. Supervised Graph Classification

In this task, the i^{th} datum is a graph with N_i nodes represented by a pair $\{\mathbf{A}_i, \mathbf{X}_i\}$ and must be associated to the correct label $\mathbf{y}_i$. We test the models on different graph classification datasets. For featureless graphs, we used the node degree information and the clustering coefficient as surrogate node features. We evaluate model performance with a 10-fold train/test split, using 10% of the training set in each fold as validation for early stopping. We adopt a fixed network architecture and only switch the pooling layers to compare Graclus, NDP, Top-K, SAGPool (Lee et al., 2019), DiffPool, and MinCutPool. All pooling methods are configured to drop half of the nodes in a graph at each layer. As additional baselines, we consider the popular Weisfeiler-Lehman (WL) graph kernel (Shervashidze et al., 2011); a network with only MP layers (*No-pool*), to understand whether or not pooling is useful for a particular task; a fully connected network (*Dense*), to quantify how much additional information is brought by the graph structure w.r.t. the node features alone.

Tab. 2 reports the classification results. MinCutPool consistently achieves equal or better results with respect to every other method. On the other hand, some pooling procedures do not always improve the performance compared to the *No-pool* baseline, making them not advisable to use in some cases. Interestingly, in some datasets such as Proteins and COLLAB it is possible to obtain fairly good classification accuracy with the *Dense* architecture, meaning that the graph structure only adds limited information. We also note the good performance of the WL kernel on Mutagenicity.

6. Conclusions

We introduced a deep learning approach to address important limitations of spectral clustering algorithms, and designed a pooling method for graph neural networks that overcomes several drawbacks of existing pooling operators. Our approach combines the desirable properties of graph-theoretical approaches with the adaptive capability of learnable methods. We tested the effectiveness of our method on unsupervised node clustering tasks, as well as supervised graph classification tasks on several popular benchmark datasets. Results show that MinCutPool performs significantly better than existing pooling strategies for GNNs.

Acknowledgments

This research is funded by the Swiss National Science Foundation project 200021 172671: "ALPSFORT: A Learning graPh-baSed framework FOr cybeR-physical sysTems."

References

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Bianchi, F. M., Grattarola, D., Livi, L., and Alippi, C. Graph neural networks with convolutional arma filters. *arXiv preprint arXiv:1901.01343*, 2019a.

Bianchi, F. M., Grattarola, D., Livi, L., and Alippi, C. Hi-

erarchical representation learning in graph neural networks with node decimation pooling. *arXiv preprint arXiv:1910.11436*, 2019b.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Cangea, C., Veličković, P., Jovanović, N., Kipf, T., and Liò, P. Towards sparse hierarchical graph classifiers. In *Advances in Neural Information Processing Systems – Relational Representation Learning Workshop*, 2018.

Collins, M. D., Liu, J., Xu, J., Mukherjee, L., and Singh, V. Spectral clustering with a convex regularizer on millions of images. In *European Conference on Computer Vision*, pp. 282–298. Springer, 2014.

Damle, A., Minden, V., and Ying, L. Robust and efficient multi-way spectral clustering. *arXiv preprint arXiv:1609.08251*, 2016.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

Dhillon, I. S., Guan, Y., and Kulis, B. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 551–556. ACM, 2004.

Dhillon, I. S., Guan, Y., and Kulis, B. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29 (11):1944–1957, 2007.

Felzenszwalb, P. F. and Huttenlocher, D. P. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

Fey, M., Lenssen, J. E., Weichert, F., and Müller, H. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 869–877, 2018.

Gallicchio, C. and Micheli, A. Fast and deep graph neural networks. *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272. JMLR. org, 2017.

Han, Y. and Filippone, M. Mini-batch spectral clustering. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 3888–3895. IEEE, 2017.

Hongyang Gao, S. J. Graph u-nets. In *Proceedings of the 36th International conference on Machine learning (ICML)*, 2019.

Ikebe, Y., Inagaki, T., and Miyamoto, S. The monotonicity theorem, cauchy's interlace theorem, and the courant-fischer theorem. *The American Mathematical Monthly*, 94(4):352–354, 1987.

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

Law, M. T., Urtasun, R., and Zemel, R. S. Deep spectral clustering learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1985–1994. JMLR. org, 2017.

Lee, J., Lee, I., and Kang, J. Self-attention graph pooling. In *Proceedings of the 36th International conference on Machine learning (ICML-19)*, 2019.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. *International Conference of Learning Representations (ICLR)*, 2016.

McPherson, M., Smith-Lovin, L., and Cook, J. M. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, volume 1, pp. 3, 2017.

Newman, M. E. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003.

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

Shaham, U., Stanton, K., Li, H., Nadler, B., Basri, R., and Kluger, Y. Spectralnet: Spectral clustering using deep neural networks. *arXiv preprint arXiv:1801.01587*, 2018.

Shervashidze, N., Schweitzer, P., Leeuwen, E. J. v., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

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

Shuman, D. I., Faraji, M. J., and Vandergheynst, P. A multiscale pyramid transform for graph signals. *IEEE Transactions on Signal Processing*, 64(8):2119–2134, 2016.

Simonovsky, M. and Komodakis, N. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Tian, F., Gao, B., Cui, Q., Chen, E., and Liu, T.-Y. Learning deep representations for graph clustering. In *AAAI*, pp. 1293–1299, 2014.

Trémeau, A. and Colantoni, P. Regions adjacency graph applied to color image segmentation. *IEEE Transactions on image processing*, 9(4):735–744, 2000.

Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Wen, Z. and Yin, W. A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, 142(1-2):397–434, 2013.

Yi, L., Su, H., Guo, X., and Guibas, L. J. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2282–2290, 2017.

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

Yu and Shi. Multiclass spectral clustering. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pp. 313–319 vol.1, Oct 2003.

Supplementary material

A. Additional Experiments

A.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[3]. 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. 3. 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, MinCutPoolsignificantly 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±0.085	0.651±0.026	**0.538**±0.012	0.559±0.007
alpha	0.197±0.087	0.114±0.001	0.078±0.007	**0.065**±0.006
homo	0.698±0.102	0.712±0.015	0.526±0.021	**0.435**±0.013
lumo	0.601±0.050	0.646±0.013	0.540±0.005	**0.515**±0.007
gap	0.630±0.044	0.698±0.004	0.584±0.007	**0.552**±0.008
r2	0.452±0.087	0.440±0.024	0.261±0.006	**0.204**±0.006
zpve	0.402±0.032	0.410±0.004	0.328±0.005	**0.284**±0.005
u0_atom	0.308±0.055	0.245±0.006	0.193±0.002	**0.163**±0.001
cv	0.291±0.118	0.337±0.018	0.148±0.004	**0.127**±0.002

Table 3. 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.

B. Experimental Details

All GNN architectures in this work have been implemented with the Spektral library[4]. 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[5]. The pooling strategy based on Graclus, is taken from the ChebyNets repository[6].

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

B.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. 4.

Table 4. 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

B.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 MinCutPooland 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[7]. *Bench-easy* and *Bench-hard*[8] 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. 5.

Table 5. 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

C. Architectures Schemata

Fig. 8 depicts the GNN architecture used in the clustering and segmentation tasks; Fig. 9 depicts the GNN architecture used in the graph classification task; Fig. 10 depicts the GNN architecture used in the graph regression task; Fig. 11 depicts the graph autoencoder used in the graph signal reconstruction task.

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

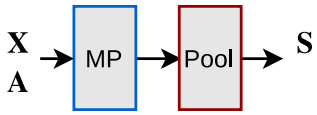


Figure 8. Architecture for clustering/segmentation.

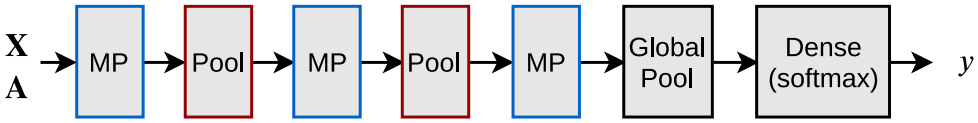


Figure 9. Architecture for graph classification.

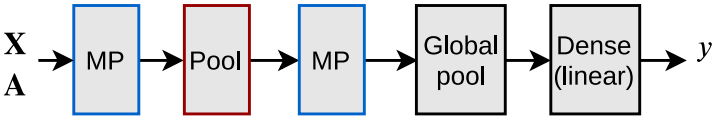


Figure 10. Architecture for graph regression.

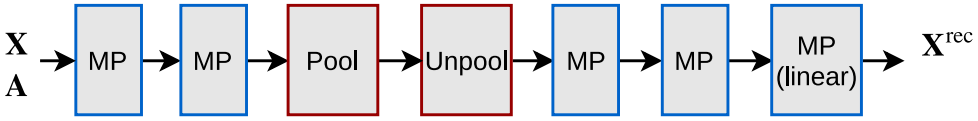


Figure 11. Architecture for the autoencoder.