Pooling Methods in Graph Neural Networks - Comparison and Applications

Christian Bohn, Lili Georgieva, Henry Martin
Department of Computer Science, ETH Zurich, Switzerland
(Dated: January 18, 2020)

***Abstract*—Graph neural networks are deep learning based methods that have been gaining increasing popularity in recent years. Inspired by convolutional neural networks, graph pooling methods are still an open research topic. The aim of this project is to compare three recently proposed pooling techniques — Self-Attention Graph Pooling, Differential Pooling and G-Pooling — in the context of node-level graph classification problems. We propose a novel approach of combining each method with unpooling layers in a Graph U-Net architecture and generalize how performance depends on graph characteristics. Our results show that the usage of pooling methods is especially valuable on large and noisy graphs or on graphs with low label percentages.**

I. INTRODUCTION

Graph neural networks (GNNs) are Deep Learning-based methods inspired by convolutional neural networks (CNNs) that operate on the graph domain and can be used to process structured, non-Euclidean data inputs with complex relationships and interdependencies between objects. GNNs are designed for various tasks, such as explicitly extracting high-level representations, network embeddings, node representations (e.g. via Recurrent GNNs) or learning graph generative distributions (e.g. graph autoencoders). Our research interest in GNNs is motivated by their potential to extend CNNs and RNNs to the graph domain by their ability to consider a graph's structure while learning features.

Relevance of pooling and different types of pooling. Pooling layers are an important part of every CNN architecture for computer vision tasks. Here, pooling is used to create a more high-level view of the input data by stripping away unimportant details using different down-sampling techniques. This decreases the size of the inputs and at the same time also increases the effective receptive field of the convolution operation [1].

Analogous to CNNs, pooling can be defined on graph data for GNNs. For graphs, there are global and non-global pooling techniques. While global pooling generates a vector-valued summary for an entire graph, non-global pooling methods are more challenging as the structure of the problem (e.g., the graph topology) is not constant. Therefore, non-global pooling operations have to create a new graph with fewer nodes that is a higher-level representation of the original graph.

Recently, several new non-global pooling methods have been proposed, amongst which are: Self-Attention Graph Pooling (SAGPool) [2], Differential Pooling (diffPool) [3] and Graph Pooling (gPool, alternatively, also known as TopK Pooling). [4].
However, graph pooling methods are still an open research topic and little is known about how the choice of a pooling method impacts the problem-specific performance. Also, most of the listed pooling methods do not yet provide unpooling methods, which are required for node-level graph problems as described in [4].
Research questions. Based on these gaps in the scientific literature, we propose to investigate the following research questions with a focus on the pooling methods mentioned above:

1: What are suitable unpooling methods for the investigated pooling methods?

2: How do the investigated graph pooling methods compare in terms of performance, their memory consumption and computational complexity?

3: Which of the proposed graph pooling methods is most suitable to solve a node classification problem based on the properties of the graph problem?

The remainder of this report is structured as follows: Section II starts with a review of graph pooling methods with a focus on Graph U-Nets that will be used as a common architecture for all our pooling methods. In Section III we first present an overview of the experiments designed to answer the identified research questions. We then introduce the datasets used and describe their properties. The main part of this section then is designated to reporting results of the experiments. In Section IV the experimental results are discussed with respect to the identified research questions, and finally, we summarize our contributions in Section V.

II. MODELS AND METHODS

In the following, we present a review of the Graph U-Net architecture that was used with all pooling methods and the three recent pooling methods that we focus on in this project.

A. Graph U-Nets

In their paper *Graph U-Nets* [4], Gao & Ji extend the *U-Net* architecture first introduced in [5] to node classification in graph data.

These U-Nets consist of encoding and decoding blocks, where each encoding block is made up of a pooling layer followed by a graph-convolutional layer, and each decoding block consists of an unpooling layer and a subsequent graph-convolutional layer. Every U-Net is made up of some number of encoding blocks, followed by the same number of decoding blocks. In our experiments, this number of encoding/decoding blocks was fixed to two. In the graph convolutional layers, the feature vectors X_{l+1} in layer $l+1$ are computed as follows:

$$X_{l+1} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}X_l W_l) \tag{1}$$

where $\hat{A} = A + I$ adds self-loops to the original adjacency matrix A, and $\hat{D}$ denotes the diagonal node degree matrix which is necessary to normalize $\hat{A}$. The matrix W_l contains the trainable weights used in the linear transformation of the feature vector X_l. The activation function is represented by σ, it can be chosen freely. Unless otherwise specified, in the subsequent experiments, we always used ReLU activations here.

Moreover, every U-Net architecture includes skip connections from the first encoding block to the last decoding block, the second encoding block the the second-to-last deconding block, and so on. The feature vectors transmitted over these skip connections from encoding to decoding blocks are either added or concatenated to the feature matrix. In all experiments in the subsequent sections, these skip connections are concatenated, as that is the less restrictive option.

B. Graph pooling layers (gPool)

The main idea of this pooling operation is to have a trainable vector $\mathbf{p}$ onto which we will project the the feature vectors $\mathbf{x_i}$ contained in the rows of X^l.
Given a node i with feature vector $\mathbf{x_i}$, the projection onto $\mathbf{p}$ is $y_i = \mathbf{x_i}\mathbf{p}/||\mathbf{p}||$. The intuitive reasoning for having this vector $\mathbf{p}$ trainable, is for it to learn to extract the most relevant features and ignore unimportant ones. In order to retain as much information as possible in the pooling operation, we select the k nodes with the largest projection values on $\mathbf{p}$ to form the new graph. This new graph then consists of the selected nodes, connected by the edges for which both end-nodes are retained.

The formal, layer-wise propagation rule of the gPool layer l is:

$$\mathbf{y} = X^l\mathbf{p}^l/||\mathbf{p}^l|| \tag{2}$$
$$idx = rank(\mathbf{y}, k) \tag{3}$$
$$\tilde{\mathbf{y}} = \sigma(\mathbf{y}(idx) \tag{4}$$
$$\tilde{X}^l = X^l(idx, :) \tag{5}$$
$$A^{l+1} = A^l(idx, idx) \tag{6}$$
$$X^{l+1} = \tilde{X}^l \odot (\tilde{\mathbf{y}}\mathbf{1}_C^T) \tag{7}$$

Here, *rank* returns the indices of the k largest values in $\mathbf{y}$, $\mathbf{1}_C$ is the vector of ones of feature length C, and $\odot$ represents element-wise matrix multiplication.

The unpooling layer l is defined as follows:

$$X^{l+1} = distribute(\mathbf{0}_{N\times C}, X^l, idx) \tag{8}$$

where *distribute* just returns the rows in X^l back to the positions in the all-zero matrix $\mathbf{0}_{N\times C}$ where they were before the corresponding pooling layer. One should note that the featue vectors of the new nodes are set to all-zero. A^{l+1} is set to the adjacency matrix before the corresponding pooling operation.

C. Differential Pooling

DiffPool was presented in [3]. It's main contribution is the proposition of a fully differentiable pooling method that allows to *learn* a lower level graph representations. In [3] the authors define a pooling layer using a two-step approach. At first, they define the adjacency matrix of the next layer as $A^{(l+1)}$ and then they define a soft assignment matrix S that matches all nodes in the adjacency matrix of the layer $A^{(l)}$ with size n_l to the adjacency matrix of the next layer $A^{(l+1)}$, where $n_{l+1} << n_l$. Equations 9 and 10 show the update steps of the diffpool pooling layer.

$$X^{(l+1)} = {S^{(l)}}^T Z^{(l)} \in \mathbb{R}^{n_{l+1}\times d}, \tag{9}$$
$$A^{(l+1)} = {S^{(l)}}^T A^{(l)} S^{(l)} \in \mathbb{R}^{n_{l+1}} \times \mathbb{R}^{n_{l+1}} \tag{10}$$

Here $Z^{(l)}$ is the original node feature matrix and $X^{(l+1)}$ is the learned node embedding. Additionally, the authors of [3] propose an entropy-based auxiliary loss to regularize the adjacency matrix and to reward distinct cluster. In this work, we follow the authors instructions and include the loss in our experiments.

[3] was originally designed for graph classification tasks and therefore the authors do not specifically provide an unpooling method. However, the matching between the nodes between the original and the pooled graph layer is explicitly encoded in the $S^{(l)}$ matrix, so we can use it to inverse the matching operation.

We use DiffPool pooling method to create two different Graph U-Nets. DiffPool1 is created for comparability to the other pooling method with our standard architecture (cf. Figure 3 in the appendix). The second Graph U-Net was created to circumvent performance problems that were observed with the DiffPool1. Inspired by the original U-Net [5] it has double convolutions instead of the single convolutions in the horizontal direction.

D. Self-Attention Graph Pooling

Self-Attention Graph Pooling (SAGPool) [2] is a hierarchical graph pooling method based on self-attention

and graph convolution, which allows consideration of both node features and graph topology for the calculation of attention scores. Attention mechanisms are exploited to distinguish between more and less important features, allowing input features to be the criteria for the attention itself.

SAGPool uses a GNN to provide self-attention scores. Using projection scores for selecting top ranked nodes is a method also used in gPool; however, there graph topology is not considered. To improve on that, SAGPool proposes a self-attention mask that allows input features to be the criteria for attention, while obtaining self-attention scores via graph convolution allows the result of the pooling to be based on both graph features and topology. Moreover, this is achieved with a reasonable time and space complexity.

When applying the graph convolution formula of Kipf & Welling ([6]), the self-attention score $Z \in \mathbb{R}^{N\times 1}$ is calculated as:

$$Z = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}X\Theta_{att}), \tag{11}$$

where σ is the activation function ($tanh$), $\hat{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix with self-connections, $\hat{D} \in \mathbb{R}^{N\times N}$ is the degree matrix of $\hat{A}$, $X \in \mathbb{R}^{N\times F}$ are the F-dimensional features of a graph with N nodes, and $\Theta_{att} \in \mathbb{R}^{F\times 1}$ is the only parameter of the SAGPool layer. Then, SAGPool adopts the node selection method of Gao & Ji [4]; Cangea et al. [7], which retains a portion of nodes of the input graph even when graphs of varying sizes and structures are inputted. The top $\lceil kN \rceil$ nodes are selected based on the value of $\mathcal{Z}$. An illustrative figure of the SAG method is presented in Appendix B, Figure 4.

III. EXPERIMENTS AND RESULTS

In order to answer the proposed research questions, we executed the following steps:

1: To compare the different graph pooling methods for a node classification task, we at first designed a generic U-net architecture suitable for graph problems based on [4], that is compatible with all investigated pooling methods. This architecture is further described here II **2:** We designed unpooling methods for SAGPool and Diffpool, as no unpooling method was provided in the original publications. The unpooling methods are described together with their respective pooling methods in Section II. **3:** We implemented a Graph U-Net architecture with the following 3 pooling methods: SAG, Diffpool and gPool and their respective unpooling methods. **4:** Finally, we ran the following experiments to enable us to answer research question 2 and 3:

- Train all 3 Graph U-Nets to solve a semi-supervised classification task on the following datasets: Cora, PubMed and Cora-Full
- Evaluation of classification performance on Cora when training with a varying number of training nodes (labeled nodes).
- Train all 3 Graph U-Nets on Cora while monitoring GPU memory consumption, training time and training progress per epoch.

Dataset Description. To demonstrate the effectiveness of our proposed approach, we conducted experiments on three datasets, widely used in the graph domain for the node-level classification task [8]. The datasets represent citation networks of scientific publications classified into one of a number of classes, where nodes represent documents and edges citation links.

Dataset	# classes	# nodes	# edges	# features
Cora	7	2,708	5,278	1,433
PubMed	3	19,717	44,324	500
Cora-full	70	19,793	63,421	8,710

TABLE I. Statistics of datasets

Baseline. As a baseline, we use the simple 2-layer GCN proposed in [6] which is often referred to as KipfNet. KipfNet has become famous because it significantly simplified previous GCN approaches such as ChebNet [9] but could maintain state-of-art level performance. It consists of only 2 layers with 16 filters which compute new node embeddings by calculating a weighted sum of a node and all of its neighbours. For the exact architecture we refer the reader to the original paper [6].

Training procedures. We evaluated the pooling methods over 10 random seeds using cross validation via data masks, as available in the *torch_geometric* library. A total of 200 testing results were used to obtain the final accuracy of each method on each dataset. We used Adam optimizer (Kingma & Ba, 2014), patience, and fixed hyperparameter selection across all methods, as summarized in Table II on page 3. We initially used a pooling ratio of 0.5 (the fraction of nodes to retain in each pooling step), but due to memory constraints for diffPool we fixed the parameter at 0.35 for all methods.

Parameter	**Value**
Horizontal convolution	Single GCN
Dropout	0.92 neuron dropout, 0.2 edge dropout in input graph
Pooling ratio	0.35
Graph U-Net Depth	2
Optimizer, Learning rate	ADAM, 0.01
Skip connection	1x1 Convolution after concatenation
Hidden channels	Fixed to 32

TABLE II. Hyperparameters for U-Nets used across all models for consistency

A. Performance on different datasets

We performed experiments to evaluate the three pooling methods in the U-net architecture, with newly implemented unpooling layers, on the node classification task (summary in Table III). Results on three benchmark datasets with differently sized graphs: Cora, PubMed and Cora-Full. On average, SAGPool showed superior classification performance on the mid- to large-sized datasets, while using a reasonable number of parameters. However, gPool slightly outperformed SAGPool on the small Cora dataset, demonstrating its superiority when the number of classes and features is relatively small.

Pooling Method	Cora	PubMed	Cora-full
KipfNet	79.5 ± 0.9	76.79 ± 0.5	46.6 ± 4.0
diffPool1	53.4 ± 12.8	-	-
diffPool2	77.6 ± 3.1	-	-
SAGPool	80.2 ± 1.2	**76.84 ± 1.6**	54.2 ± 1.2
gPool	**81.1 ± 1.0**	76.5 ± 1.0	**54.5 ± 0.7**

TABLE III. Average accuracy and standard deviation of 10 random seeds. The abbreviations denote: gPool (Graph g-pooling), diffPool1 and diffPool2 (Differential pooling with single and double convolutions, respectively), SAGPool (Self-Attention Graph pooling)

B. Performance with a varying percentage of training nodes

In order to compare the different pooling methods (and the baseline) in terms of efficiency w.r.t. the number of training samples, we tested each method with different fractions of the graph nodes used as training data. In all cases, the remaining graph nodes were split evenly between the validation and test sets, and we report the best accuracies on the test data after training for 200 epochs. All models shown use the same hyperparameters, in order to be comparable. The results of these experiments can be seen in Figure 1. Each experiment was repeated 10 times, and the plot shows the mean results along with the respective standard deviations as error bars.
For these results, note that the fractions of the nodes used for training below 10% are of particular interest, any larger fractions certainly lead to overfitting to the training data, and are only included for the sake of completeness, but add little insight.
For the fractions below 10%, one can clearly see that the SAG-Pool and TopK/GPool methods consistently outperform both the baseline, as well as both DiffPool models by a significant margin.

C. Comparison of computational complexity, training efficiency and memory usage for different methods

Finally, we compare the different pooling methods in terms of their time complexity, their required training time and their GPU-Memory requirements. This experiment follows the same procedure as in the other experiments (fixed hyperparamters, 10-times repetition of result, report test error based on best validation error).

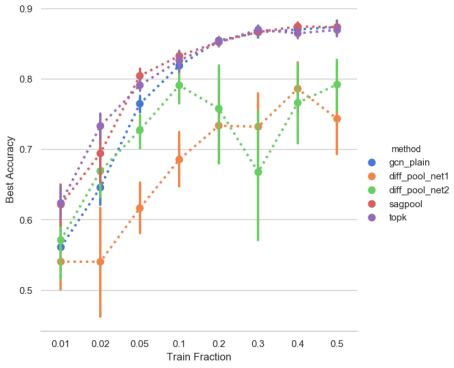

FIG. 1. Test accuracies and their standard deviations over 10 repetitions on the Cora dataset with different fractions of graph nodes used for training.

Figure 6 shows the required time to train for a given number of epochs. As expected, the training time grows linearly with the number of epochs for all methods. However, the plot reveals great differences between the individual methods. It shows that both DiffPool-based architectures are significantly slower than the other methods. The other two pooling methods require approximately the same training time and the baseline model is the fastest by a large margin. We note that our findings are in line with theoretical values. SAGPool is able to learn hierarchical representations using relatively few parameters, gPool requires theoretical storage complexity of $O(|V| + |E|)$, whereas DiffPool requires $O(k|V|^2)$ (V, E, and k denote vertices, edges, and pooling ratio, respectively).

Figure 2 shows an analysis of the training efficiency. All networks except for the diffpool1 architecture reach approximately the same final accuracy, however there are great differences in the training efficiency. The baseline architecture shows the fastest convergence, closely followed by the TopK/gPool and the SAGPool-based architectures. The DiffPool2 architecture is considerably slower and less efficient than the other three methods. Additionally, the error bars reveal that the diffpool1 architecture is unstable and its final result is highly dependent on weight-initialization.

Lastly, Figure 5 in Appendix B shows the memory consumption of the different methods. This is measured as the peak memory allocation on the GPU for each model during one training run. The figure shows that SAGpool has remarkably low memory requirements, close to those of the baseline, which does not feature any pooling method. This plot reveals the problems that are resulting from the fully connected adjacency matrix that is defined in every diffpool layer. The size of the error bars shown in this plot indicate that the peak memory allocation is

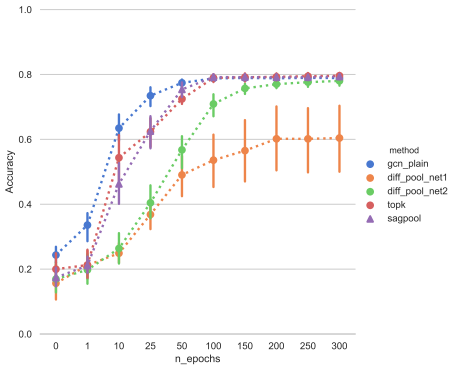

FIG. 2. Test accuracy of the different graph U-Nets during the training process. x-Axis not scaled linearly.

deterministic for all models.

IV. DISCUSSION

Discussion of results with respect to the research questions Opposing to our prior belief, it was relative straightforward to construct suitable unpooling methods for DiffPool and SAGPool. DiffPool provides an explicit representation of the matching between the different pooling layers, which can be equally used in an unpooling layer. For SAGPool it turned out that the unpooling layer used in the original Graph U-Net [4] is well suited. An in-depth comparison of several different pooling layers is left for future work as the focus of this work was on the remaining research questions.

Table III show that the results vary greatly between the datasets. It is very clear that DiffPool is not a suitable method for any of the experiments either due to the bad performance or the high memory requirements that prevented the usage of DiffPool in two out of three experiments. The results show that the performance does not vary significantly between gPool and SAGPool (considering the standard deviation of the results). This is also mostly true for the comparison of gPool and SAGPool with the baseline, except for the Cora-full experiment where both perform sigificantly better than KipfNet.

The differences between the methods become more clear when comparing the computational complexity in terms of training time as it is done in 6. Here, and in the GPU memory consumption shown in 5, SAGPool outperforms gPool significantly. Here again the difference between SAGPool as a sparse pooling method and DiffPool as a dense pooling method is evident.

All pooling methods except DiffPool have a good performans on all datasets, the only datasets where they are able to significantly outperform the baseline is on Cora-full. The Cora dataset was created based on the Cora-full dataset can be seen as a clean subset of the Cora-ful dataset. This is evidence that the usage of gPool and SAGPool has a strong added value on large and noisy graphs.

Finally, the investigation of the percentage of available labels on the graph, which can be seen in figure 1, showed that the addition of pooling methods is especially valuable in domains with a low ($\leq 5\%$ on Cora) percentage of available labels. Here gPool and SAGpool could outperform the KipfNet baseline significantly.

Contribution and impact. The main contributions of this work can be summarized as follows: DiffPool as a dense pooling method is not suitable for larger graphs and is often unable to outperform our baseline without pooling. gPool and SAGPool often perform very similar to each other and similar to the baseline. SAGPool however, does outperform gPool in terms of computational complexity and GPU memory requirements and both methods can outperform the baseline on large and noisy graphs or on graphs with low label percentages. These results are of particular importance for practitioners, as they offer insights that are helpful for choosing the correct pooling methods when designing an architecture for a given graph problem.

Limitations and Future Work. We limited the scope of this paper to a single Graph U-Net architecture (with an exception for the DiffPool method) with shared hyper parameters over all methods and all datasets. While we note note that the performance might be affected by the fixed hyper parameters such as the pooling ratio of nodes, tuning all hyper parameters for all datasets is a time-consuming task that might not offer additional insights as it limits the comparability between the different methods. Further experiments could include the in-depth exploration of the best performing methods of this paper. For example SAGPool offers attention scores that are calculated using multi-hop connected nodes as described in [2] or the repetition of this analysis with omitted pooling methods such as relational pooling [10] or newly proposed pooling methods. Another direction is the creation of new architectures by combination of different pooling and unpooling layers in a Graph U-Net architecture.

V. SUMMARY

In this work we proposed a Graph U-Net framework to compare three of the most common graph pooling methods: SAG, DiffPool and gPool using several standard dataset for node-level classification tasks: Cora, PubMed and Cora-Full. The pooling methods were compared in terms of their performance, impact on generalization, required GPU memory requirements and computational complexity for training, and respective suitability for graph problems with different properties such as size of the graph, number of node-features and percentage of labeled nodes. Our findings show that DiffPool is unsuitable for larger graphs and that the addition of pooling methods is especially valuable on large and noisy graphs or on graphs with low label percentages.

[1] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard S. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," *CoRR*, vol. abs/1701.04128, 2017.

[2] Junhyun Lee, Inyeop Lee, and Jaewoo Kang, "Self-attention graph pooling," in *Proceedings of the 36th International Conference on Machine Learning*, 09–15 Jun 2019.

[3] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec, "Hierarchical Graph Representation Learning with Differentiable Pooling," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., pp. 4800–4810. Curran Associates, Inc., 2018.

[4] Hongyang Gao and Shuiwang Ji, "Graph U-Nets," *arXiv:1905.05178 [cs, stat]*, May 2019, arXiv: 1905.05178.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," *ArXiv*, vol. abs/1505.04597, 2015.

[6] Thomas N Kipf and Max Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[7] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò, "Towards sparse hierarchical graph classifiers," 2018.

[8] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," 2016.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 3844–3852.

[10] Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro, "Relational Pooling for Graph Representations," *arXiv:1903.02541 [cs, stat]*, May 2019, arXiv: 1903.02541.

Appendix A: Graph U-Net Architecture

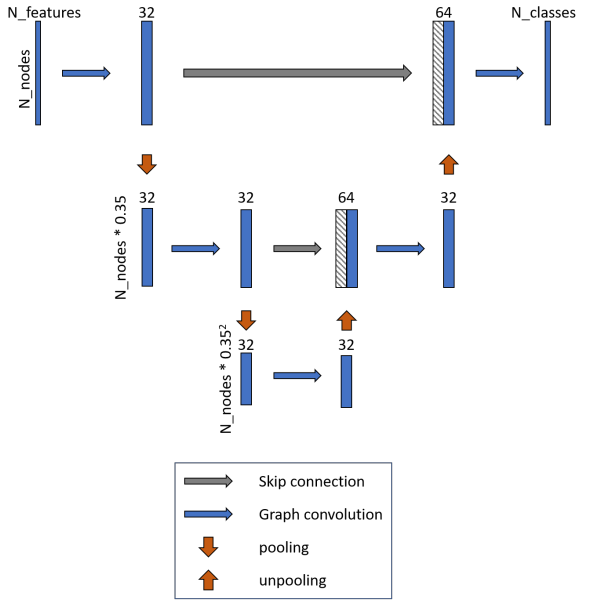

FIG. 3. An overview of the U-Net Architecture used for all pooling methods

Appendix B: Additional Graphs and Charts

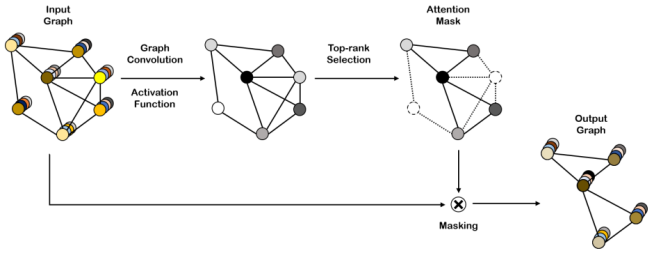

FIG. 4. An illustration of the SAGPool layer

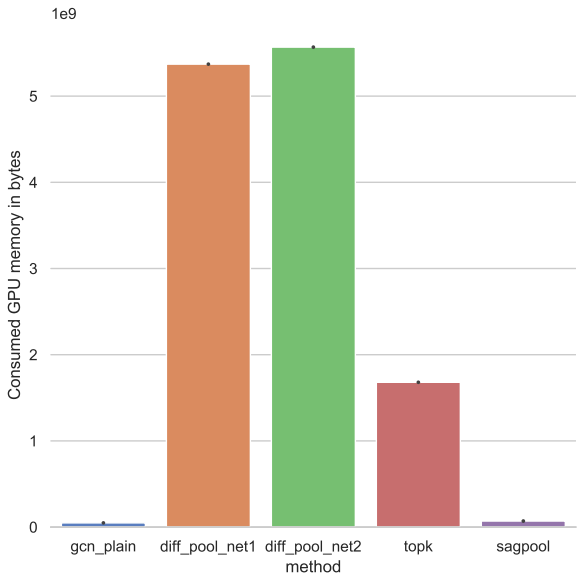

FIG. 5. GPU Memory consumption of the different graph U-Nets during training.

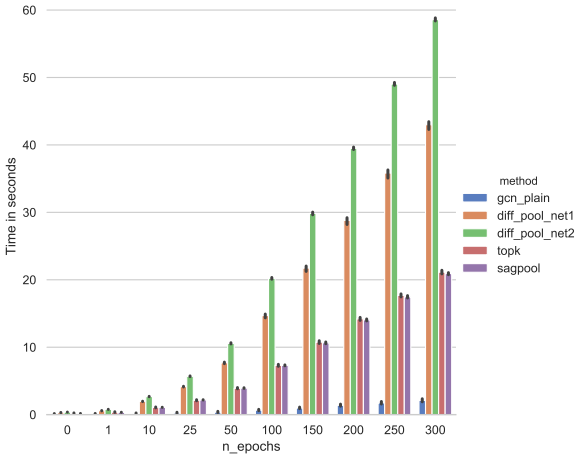

FIG. 6. Training time of the different graph U-Net architectures. x-Axis not scaled linearly.