A Simple Yet Effective Method Improving Graph Fingerprints for Graph-Level Prediction

Jiaxin Ying* University of Michigan
Jiaqi Ma* University of Michigan
Qiaozhu Mei University of Michigan

ABSTRACT

Graph fingerprints form an important group of graph representation methods and have been shown effective in graph machine learning tasks. However, such methods usually first compute feature descriptors of nodes in a graph, and then average or sum over them to obtain a graph-level feature descriptor. The simple pooling methods tend to cause significant information loss in the graph-level representation. Further, the computation of graph fingerprints is mostly not parameterized and cannot be tailored for a supervised learning task. In this paper, we test a simple fuzzy histogram approach that is applicable to a wide range of graph fingerprints. Through extensive benchmark evaluation, we demonstrate that this simple method significantly improves the supervised learning performance with graph fingerprints, and achieves similar performance with popular graph neural networks for graph classification, while remaining computationally efficient. We suggest graph fingerprints enhanced with the histogram approach should be considered as strong baselines in the context of graph-level prediction tasks.

ACM Reference Format:
Jiaxin Ying, Jiaqi Ma, and Qiaozhu Mei. 2021. A Simple Yet Effective Method Improving Graph Fingerprints for Graph-Level Prediction. In *Proceedings of GLB '21: The Workshop on Graph Learning Benchmarks at the Web Conference 2021. (GLB '21).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Graphs are a family of general representations of data and graph-structured data are commonly observed in the real world. In the area of machine learning (ML), the topological structures of a graph, *locally* and *globally*, have been shown as useful features for many prediction tasks. Taking examples in social network analysis, how likely a user is willing to adopt a new product is related to the structural diversity of their local neighborhood [25]; the popularity of a Tweet is related to its global retweeting tree structure at an earlier stage [11]. To better utilize the graph information in the prediction tasks, there have been rapidly growing developments of graph representation learning methods in recent years.

Graph fingerprints [3, 5, 18, 23, 24, 27] form an important group of graph representation methods. Such representations usually first calculate a series of feature descriptors for each node, which smoothly capture the graph properties at different granularities, from local to global. The node features are then pooled together (e.g., by summation or average) to obtain the graph-level fingerprints. While the graph fingerprints have been shown to provide decent discriminative power for ML on graphs, there are two shortcomings most of them suffer from. First, the simple pooling methods cause significant information loss. Second, most graph fingerprints are calculated by a fixed algorithm independent of the downstream ML tasks, which prevent them from fully leveraging the label information in a graph-level supervised learning.

Recently, Li et al. [10] propose to first calculate a histogram over the node-level Heat Kernel Signature (HKS) [23][1], and then apply a convolution neural network (CNN) on top of the histogram to obtain a parameterized graph fingerprint. This work examines whether this histogram approach can be effectively applied to other types of graph fingerprints. We also extend this approach by using two variants of fuzzy histograms and try to further parameterize the fuzzy histograms. The histogram approaches are simple, easy-to-implement, and applicable to a wide range of graph fingerprints. We implement both the (original) discrete histogram approach and the extended fuzzy histogram approach on three types of graph fingerprints, and evaluate them on popular graph classification datasets. We demonstrate clear improvements of the histogram approaches over the vanilla graph fingerprints. We also show that the improved graph fingerprints achieve similar performance as graph neural networks, with a better trade-off between model performance and computational cost, which suggests graph fingerprints enhanced by the histogram approaches should be considered as strong baselines in the context of graph classification.

2 RELATED WORK

2.1 Graph Fingerprints

There has been a large body of literature for graph fingerprints. The Heat Kernel Signature (HKS) [23] is obtained by restricting the heat kernel to the temporal domain while inherits rich information of the heat kernel. NetLSD [24] computes both heat kernel and wave kernel from the eigen-decomposition of the graph Laplacian. For the purpose of graph comparison, NetLSD calculates graph embeddings from the trace of heat kernel or wave kernel. The trace of heat kernel is equivalent to summing over the node-level fingerprints of HKS. Verma and Zhang [27] discover family of graph spectral distances (FGSD) and calculate f-spectral distance of two nodes based on it. They further use 1-D histogram to process the distance matrix to get the graph spectrum. de Lara and Pineau [3] propose to use the spectral decomposition of graph Laplacian as a simple and fast algorithm for graph-level classification tasks. Specifically,

*Both authors contributed equally. Emails: jiaxinyi@umich.edu, jiaqima@umich.edu. Code available at https://github.com/jiaxinying/HistogramGraphFingerprints.

[1]This is a type of graph fingerprint.

GLB '21, April 12–16, 2021, Online
© 2021 Association for Computing Machinery.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

they set a hyperparameter k and use the k smallest positive eigenvalues for graph Laplacian and denote this embedding as spectral features (SF). They also show that each eigenvalue of the Laplacian has a natural physical interpretation, which corresponds to the energy level of the nodes in the embedding space. Gao et al. [5] introduce geometric wavelet scattering transform which generalize (Euclidean) wavelet scattering transform [12] by applying a graph wavelet transform on top of it. The proposed model sums wavelet values at different scales and different orders, and concatenates them to get graph-level fingerprint. Rozemberczki and Sarkar [18] model the node-level feature descriptors, named FEATHER, with a characteristic function defined by aggregating node features in local neighborhoods. This fingerprinting method is able to effectively leverage node attributes. To get graph-level representations, they apply a mean or max pooling over the node-level fingerprints. Most the aforementioned graph fingerprints except for a variant of FEATHER are calculated independent of the downstream ML tasks and the parametric variant of FEATHER was only used in node-level prediction tasks rather than graph-level prediction tasks. And all of the above methods use relatively simple pooling methods transforming node-level feature descriptors to graph-level fingerprints, which suffer from significant information loss.

Li et al. [10] build on top of the node-level HKS with a discrete histogram followed by a CNN. This work extends the histogram approach by Li et al. [10] and apply it to more types of graph fingerprints.

2.2 Graph Neural Networks

One of the most popular and fast growing family of methods are graph neural networks (GNNs) [30]. The majority of existing GNNs fall into the message-passing (MP) paradigm (e.g., GCN [7], GAT [26], or GIN [28]). These MP-GNNs model the representation of a node or a sub-structure of the graph by aggregating the features of its neighbors through neural network layers. When applied to complex graph-level prediction tasks (e.g., molecular property prediction [13]), the design of MP-GNNs is often faced with the trade-off between expressive power (as measured by Weisfeiler-Leman (WL) test) and computational efficiency [13, 16, 28]. The graph fingerprints, instead, are not limited by the WL expressiveness and serve as great alternatives for graph-level learning tasks.

3 THE HISTOGRAM APPROACHES

Suppose there is a graph with N nodes. A graph fingerprint method usually first calculates node-level feature descriptors, $\Phi \in \mathbb{R}^{N\times M}$, where each row corresponds to a node. The M columns usually correspond to graph properties at different granularities, and there is continuity between consecutive columns. To form a graph-level representation that can be used in a downstream ML task, we need to eliminate the dependence of N. In most previous work [18, 24], this is done by applying a simple pooling operation (sum, max, or average) over the N nodes, which inevitably causes significant information loss. Li et al. [10] apply histograms with B bins to summarize the distribution of each column of Φ over the N nodes, resulting a histogram matrix $H \in \mathbb{R}^{B\times M}$. A CNN is then applied to H as the rows and columns of H respectively present inherent continuity.

While Li et al. [10] only consider the HKS graph fingerprint, we further examine other types of graph fingerprints. We also generalize this histogram approach to fuzzy histograms. Formally, suppose we are given a kernel function $f : \mathbb{R} \to \mathbb{R}^+$ satisfying

$$\int f(x)dx = 1, \int xf(x)dx = 0, \text{ and } \int x^2 f(x)dx > 0.$$

Given B pairs of location and scale parameters $\{(\mu_b, \sigma_b)\}_{b=1}^{B}$, we can calculate a fuzzy histogram $H \in \mathbb{R}^{B\times M}$ from the Φ, such that, for any $b = 1, \cdots, B$, and $m = 1, \cdots, M$,

$$H_{bm} = \sum_{i=1}^{N} f(\frac{\Phi_{im} - \mu_b}{\sigma_b}).$$

In particular, the discrete histogram used by Li et al. [10] is a special case when we choose a rectangular kernel $f(x) = \frac{1}{2}\mathbb{1}\,[|x| < 1]$ with proper location and scale parameters. In practice, we further normalize H by $H_{bm} \leftarrow \frac{H_{bm}}{\sum_{d=1}^{B} H_{dm}}$. We also test various design choices of fuzzy histograms as described below.

Kernel functions. We implement three types of kernel functions: rectangular, Gaussian $\left(f(x) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{x^2}{2})\right)$, and Silverman $\left(f(x) = \frac{1}{2}\exp(-\frac{|x|}{\sqrt{2}})\sin(\frac{|u|}{\sqrt{2}} + \frac{\pi}{4})\right)$.

Parameters. The location and scale parameters can be either treated as hyper-parameters or treated as learnable parameters (except for the rectangular kernel function).

Initialization of parameters. By default, we set the hyper-parameters (or initializations) of the location parameters uniformly for a given range and set those of the scale parameters as constant. We also try to set them by fitting a Gaussian mixture model from the node-level feature descriptors.

More details of the approach can be found in Appendix A.

4 EXPERIMENTS

In this section, we present the benchmark experiments. Due to page limit, some experiment details are left to Appendix B.

4.1 Methods for Comparison

Graph fingerprints. We implement the different variants of histogram approaches on three base graph fingerprints, **NetLSD** [24], **GeoScatteirng** [5], and **FEATHER** [18]. We append **-R**, **-G**, and **-S** to the base method to indicate the histogram approaches with rectangular, Gaussian, and Silverman kernel functions respectively. We further append an **-L** if the location and scale parameters are learnable, and append an **-I** if they are initialized from a fitted Gaussian mixture. When both -I and -L should be appended, we simplify it as **-IL**. For example, "NetLSD-R" indicates applying the histogram with rectangular kernel on NetLSD, while "FEATHER-G-L" indicates applying the histogram with Gaussian kernel and learnable location/scale parameters on FEATHER.

Graph Kernel Methods. We include three types of graph kernels as our baseline models, shortest-path kernel (ShortestPath) [2], Weisfeiler-Lehman subtree kernel (WL) [20] and Weisfeiler-Lehman Optimal Assignment kernel (WL-OA) [9].

Table 1: Performance on the real-world benchmark datasets. 10-fold cross-validation accuracy is reported on TU datasets, and average AUC of 5 random splits is reported on Karateclub datasets. Bold entries indicate the best among all methods and underlined entries indicate the best within the subgroup of methods. Some results are missing due to out of memory.

	Method	TU Dataset					Karateclub Dataset			
		MUTAG	NCI1	PROTEINS	IMDB-BIN	IMDB-MULTI	Deezer Egos	Github StarGazers	Twitch Egos	Reddit Threads
Kernel	ShortestPath	0.806±0.094	0.661±0.025	0.761±0.028	0.564±0.039	0.394±0.032	0.497±0.016	0.687±0.005	/	/
	WL	0.772±0.094	0.814±0.018	**0.762±0.023**	0.722±0.041	0.507±0.033	0.514±0.008	0.668±0.010	/	/
	WL-OA	0.856±0.087	**0.854±0.019**	0.754±0.039	0.729±0.040	0.497±0.028	0.506±0.003	/	/	/
MP-GNN	1-GNN	0.672±0.127	0.677±0.023	0.725±0.040	0.728±0.048	0.486±0.036	0.510±0.021	0.693±0.011	0.720±0.001	0.837±0.003
	GIN	0.783±0.085	0.719±0.026	0.726±0.044	0.733±0.029	0.491±0.051	0.526±0.005	0.748±0.008	0.722±0.002	0.843±0.002
	1-2-3-GNN	0.828±0.081	0.756±0.025	0.755±0.035	0.716±0.031	0.497±0.033	0.522±0.006	0.777(0.004)	**0.725±0.001**	0.842±0.002
NetLSD	NetLSD	0.761±0.086	0.610±0.016	0.729±0.051	0.61±0.035	0.462±0.049	0.527±0.005	0.630±0.010	0.630±0.003	0.818±0.001
	NetLSD-R	0.839±0.085	0.725±0.033	0.740±0.063	0.723±0.034	0.493±0.053	0.529±0.007	0.713±0.005	0.721±0.001	0.838±0.002
	NetLSD-G	0.839±0.071	0.711±0.023	0.746±0.030	0.692±0.030	0.475±0.034	0.522±0.004	0.712±0.003	0.720±0.001	0.837±0.001
	NetLSD-S	0.872±0.053	0.705±0.023	0.743±0.035	0.709±0.051	0.466±0.052	0.517±0.006	0.709±0.005	0.719±0.001	0.836±0.001
	NetLSD-G-L	0.806±0.088	0.708±0.023	0.746±0.039	0.716±0.046	0.479±0.052	0.518±0.005	0.715±0.003	0.719±0.0005	0.838±0.001
	NetLSD-S-L	0.878±0.073	0.706±0.016	0.750±0.042	0.708±0.051	0.463±0.027	0.504±0.020	0.710±0.005	0.721±0.001	0.838±0.001
	NetLSD-G-IL	0.844±0.073	0.710±0.028	0.751±0.051	0.715±0.036	0.501±0.068	0.526±0.006	0.723±0.007	0.719±0.002	0.838±0.001
	NetLSD-S-IL	0.856±0.105	0.719±0.021	0.744±0.041	0.726±0.041	**0.516±0.023**	0.522±0.006	0.723±0.006	0.721±0.001	0.839±0.001
GeoScattering	GeoScattering	0.817±0.075	0.509±0.038	0.600±0.040	0.715±0.050	0.446±0.058	0.535±0.007	0.719±0.002	0.723±0.001	0.817±0.001
	GeoScattering-R	0.867±0.065	0.718±0.020	0.740±0.039	0.705±0.046	0.470±0.037	0.528±0.008	0.766±0.007	0.723±0.001	0.843±0.002
	GeoScattering-G	0.883±0.067	0.721±0.017	0.737±0.035	0.711±0.061	0.497±0.055	**0.539±0.009**	0.767±0.004	0.724±0.001	0.843±0.002
	GeoScattering-S	0.867±0.079	0.709±0.020	0.749±0.032	0.710±0.029	0.493±0.052	0.535±0.011	0.771±0.005	0.724±0.001	0.843±0.002
	GeoScattering-G-L	0.878±0.035±	0.721±0.024±	0.745±0.049	0.707±0.077	0.489±0.055	**0.539±0.012**	0.768±0.005	0.724±0.001	0.843±0.002
	GeoScattering-S-L	0.867±0.084	0.720±0.020	0.749±0.041	0.737±0.032	0.488±0.042	**0.539±0.008**	0.773±0.006	0.724±0.001	0.843±0.002
	GeoScattering-G-IL	0.894±0.061	0.751±0.020	0.750±0.050	0.712±0.031	0.489±0.037	0.533±0.012	0.779±0.007	0.724±0.001	0.843±0.002
	GeoScattering-S-IL	0.867±0.115	0.745±0.023	0.737±0.037	**0.740±0.032**	0.503±0.044	0.534±0.006	0.784±0.006	**0.725±0.001**	**0.844±0.002**
FEATHER	FEATHER	0.833±0.086	0.590±0.020	0.699±0.048	0.72±0.053	0.466±0.047	0.533±0.006	0.751±0.005	0.721±0.001	0.831±0.002
	FEATHER-R	0.861±0.075	0.740±0.020	0.738±0.038	0.716±0.056	0.493±0.022	0.523±0.003	0.783±0.005	0.721±0.002	0.841±0.002
	FEATHER-G	**0.906±0.046**	0.739±0.028	0.742±0.042	0.729±0.062	0.485±0.039	0.521±0.004	0.781±0.005	0.722±0.001	0.841±0.002
	FEATHER-S	0.900±0.104	0.735±0.024	0.748±0.037	0.717±0.040	0.488±0.039	0.521±0.007	**0.791±0.006**	0.723±0.001	0.843±0.002
	FEATHER-G-L	**0.906±0.053**	0.745±0.012	0.741±0.047	0.730±0.038	0.485±0.048	0.520±0.003	0.779±0.006	0.721±0.001	0.841±0.002
	FEATHER-S-L	0.889±0.074	0.736±0.014	0.745±0.045	0.713±0.029	0.498±0.056	0.522±0.011	0.789±0.007	0.722±0.002	0.842±0.002
	FEATHER-G-IL	0.861±0.095	0.751±0.028	0.734±0.052	0.724±0.040	0.477±0.048	0.518±0.017	0.762±0.010	0.722±0.001	0.841±0.002
	FEATHER-S-IL	0.878±0.057	0.756±0.316	0.737±0.052	0.720±0.052	0.487±0.050	0.516±0.014	0.779±0.006	0.721±0.002	0.843±0.002

Graph neural networks. We primarily compare with three types of GNNs, **GIN** [28], **1-GNN**, and **1-2-3-GNN** [16]. Where the first two are 1-order GNNs and 1-2-3-GNN is a high-order GNN. In the experiments on synthetic data (Section 4.3), we further include **P-GNN** [13], which is a more powerful high-order GNN but has very high computation cost.

4.2 Experiments on Real-World Datasets

We first evaluate different methods on real-world datasets. Our primary goal here is to demonstrate that the histogram approaches clearly improve various types of vanilla graph fingerprints. We also provide the performance of popular GNNs for graph-level predictions as a reference.

Datasets. We adopt two groups of graph classification datasets. The first group (TU datasets [15]) consists of five popular graph kernel benchmark datasets [29], including 3 bioinformatics datasets MUTAG, NCI1, PROTEINS and 2 social network datasets IMDB-BIN, and IMDB-MULTI. The second group (Karateclub datasets [17]) consists of four social network datasets, GitHub StarGazers, Twitch Egos, Reddit Threads, and Deezer Egos, which have been used to benchmark various graph fingerprints.

Experiment setup. On the TU datasets, we train and evaluate all the models except for the vanilla graph fingerprints following exactly the same experiment setup as in Morris et al. [16] (10-fold cross validation, average test accuracy is reported). For NetLSD, Geoscattering and FEATHER, a logistic regression is applied on top of the unsupervised graph fingerprints, so we do not need a validation set. Therefore for these vanilla graph fingerprints, we merge the validation set into the training set when fitting the logistic regression.

On the Karateclub datasets, we randomly split the dataset into training, validation, and test sets by the ratio of 65:15:20. For NetLSD, Geoscattering and FEATHER, again we merge the validation set

Table 2: Average test MAE on the sythetic regular graph dataset. Bold entries indicate the best performance.

Model	3-clique	4-clique
1-GNN	0.7932±0.0239	0.5928±0.1988
GIN	0.7957±0.0212	0.5892±0.1991
1-2-3-GNN	0.4834±0.2749	0.5760±0.2024
P-GNN	0.0972±0.1122	**0.1953 ±0.1415**
GeoScattering-G	**0.0940 ± 0.2175**	0.3160±0.1529

into the training set when fitting the logistic regression. We repeat 5 independent splits, and report the average AUC of the 5 trials.

Results on real-world datasets. The results on the two groups of real-world datasets are shown in Table 1. We first observe that, compared to the vanilla graph fingerprints, applying the histogram approaches almost always improve the performances, regardless the variant of histogram approaches. Next, using a fuzzy histogram (i.e., histograms with Gaussian or Silverman kernels) often outperforms the variant using a discrete histogram (i.e., histograms with a rectangular kernel). However, there is no consistent evidence to show that making the location/scale parameters learnable or using a more sophisticated initialization method leads to a better performance. Finally, we see that the graph fingerprints improved by the histogram approaches achieve better or similar performance compared to popular GNNs and kernel methods for graph classification, which makes graph fingerprint methods strong baselines to be considered in the context of graph classification.

4.3 Experiments on Synthetic Data

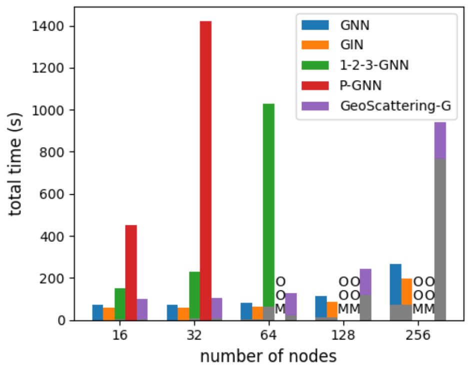
We further conduct experiments on synthetic graphs to better evaluate the expressive power and the computational costs of different methods.

To evaluate the expressive power, we generate a benchmark synthetic dataset of random regular graphs. A k-regular graph is defined as a graph where each node has exactly the same degree k. Distinguishing regular graphs is a relatively hard task. In particular, 1-order MP-GNNs are proved to fail in distinguishing regular graphs [19], and (strongly) regular graphs are commonly used as benchmark for the graph isomorphism test [14].

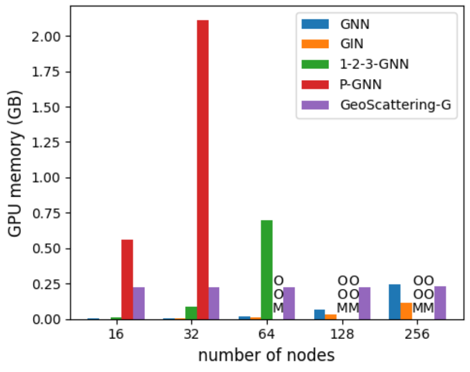
To cleanly measure the computational costs of different methods in terms of the graph size, we generate a series of synthetic datasets of Poisson random graphs with different graph sizes. In this way, we are able to robustly evaluate the average computational costs of different methods at different graph sizes.

The detailed experiment setups for synthetic data are described in Appendix B.3.

Prediction performance on regular graphs. The experiment results on the synthetic regular graph dataset are shown in Table 2. First we observe that the 1-order MP-GNNs, i.e., 1-GNN and GIN, are among the worst on both tasks, which verifies the theory that they are not able to distinguish the regular graphs. P-GNN achieves the best performance due to its superior expressive power on 4-clique task. However, Geoscattering-G achieves the best performance on



(a) Total time and preprocessing time. The total time consists of both preprocessing time (grey area) and training time.



(b) Peak GPU memory.

Figure 1: Computational costs of the GeoScattering-G and various MP-GNNs on Poisson random graphs with varying graph sizes. OOM means out of memory.

3-clique task and is much better than all other MP-GNNs except for P-GNN. The reason why 1-2-3-GNN does not perform well on the two tasks might be due to the local approximation in its official implementation [16].

Computational cost on Poisson random graphs. As can be seen in Figure 1, higher-order MP-GNNs have significantly larger computational costs compared to 1-order MP-GNNs and the proposed GeoScatteirng-G. Both the time and memory costs grow quickly for P-GNN and 1-2-3-GNN in terms of the graph size. And P-GNN can barely scale to graphs with a couple hundreds of nodes. GeoScattering-G has larger preprocessing time compared to 1-order MP-GNNs but has similar training time. In Appendix B.3, we also show GeoScattering-G has similar training time per epoch as 1-order MP-GNNs.

5 CONCLUSION

In this paper, we generalize the discrete histogram approach on HKS by Li et al. [10] as a fuzzy histogram approach, and apply it on various types of graph fingerprints. Through extensive benchmark study, we demonstrate that this simple method can significantly improve all the tested graph fingerprint methods, achieving on-par performance with popular GNNs for graph classification. The improved graph fingerprint methods are not limited by 1-WL in terms of expressive power, and have significantly lower computation costs compared to high-order GNNs. We suggest graph fingerprints enhanced by the histogram approach should be considered as strong baselines in the context of graph classification tasks.

Acknowledgement. The authors would like to thank Jing Zhu for her contributions to this project.

REFERENCES

[1] Vladimir Batagelj and Ulrik Brandes. 2005. Efficient generation of large random networks. *Physical Review E* 71, 3 (2005), 036113.

[2] Karsten M Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE, 8–pp.

[3] Nathan de Lara and Edouard Pineau. 2018. A simple baseline algorithm for graph classification. *arXiv preprint arXiv:1810.09155* (2018).

[4] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[5] Feng Gao, Guy Wolf, and Matthew Hirn. 2019. Geometric scattering for graph data analysis. In *International Conference on Machine Learning*. PMLR, 2122–2131.

[6] Jeong Han Kim and Van H Vu. 2003. Generating random regular graphs. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. 213–222.

[7] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[8] Boris Knyazev, Graham W Taylor, and Mohamed Amer. 2019. Understanding attention and generalization in graph neural networks. In *Advances in Neural Information Processing Systems*. 4202–4212.

[9] Nils M Kriege, Pierre-Louis Giscard, and Richard C Wilson. 2016. On valid optimal assignment kernels and applications to graph classification. *arXiv preprint arXiv:1606.01141* (2016).

[10] Cheng Li, Xiaoxiao Guo, and Qiaozhu Mei. 2016. Deepgraph: Graph structure predicts network growth. *arXiv preprint arXiv:1610.06251* (2016).

[11] Cheng Li, Jiaqi Ma, Xiaoxiao Guo, and Qiaozhu Mei. 2017. Deepcas: An end-to-end predictor of information cascades. In *Proceedings of the 26th international conference on World Wide Web*. 577–586.

[12] Stéphane Mallat. 2012. Group invariant scattering. *Communications on Pure and Applied Mathematics* 65, 10 (2012), 1331–1398.

[13] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2019. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*. 2156–2167.

[14] Rudolf Mathon. 1978. Sample graphs for isomorphism testing. *Congressus Numerantium* 21 (1978), 499–517.

[15] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. 2020. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*. arXiv:2007.08663 www.graphlearning.io

[16] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4602–4609.

[17] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. *arXiv* (2020), arXiv–2003.

[18] Benedek Rozemberczki and Rik Sarkar. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. *arXiv preprint arXiv:2005.07959* (2020).

[19] Ryoma Sato. 2020. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078* (2020).

[20] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, 9 (2011).

[21] Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. 2020. GraKeL: A Graph Kernel Library in Python. arXiv:1806.02193 [stat.ML]

[22] Angelika Steger and Nicholas C Wormald. 1999. Generating random regular graphs quickly. *Combinatorics, Probability and Computing* 8, 04 (1999), 377–396.

[23] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. 2009. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*, Vol. 28. Wiley Online Library, 1383–1392.

[24] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. 2018. Netlsd: hearing the shape of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2347–2356.

[25] Johan Ugander, Lars Backstrom, Cameron Marlow, and Jon Kleinberg. 2012. Structural diversity in social contagion. *Proceedings of the National Academy of Sciences* 109, 16 (2012), 5962–5966.

[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[27] Saurabh Verma and Zhi-Li Zhang. 2017. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*. 88–98.

[28] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[29] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1365–1374.

[30] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

A METHOD DETAILS

We introduce more details of the histogram approach here. In particular, we use the process of applying the histogram approach on FEATHER [18] as an example to illustrate the idea.

Notations. Before we dive into the details, we first introduce a few general notations that will be used across this section. We denote a graph as

$$G = (V, E, A, X),$$

where $V = \{1, 2, \cdots, N\}$ is the set of N nodes, $E \in V \times V$ is the set of edges, $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix, and $X \in \mathbb{R}^{N \times K}$ is the node feature matrix. We write the degree matrix as $D \in \mathbb{R}^{N \times N}$, which is a diagonal matrix with $D_{ii} = \sum_{j \in V} A_{ij}$ for any $i \in V$. We also denote the random walk transition matrix as

$$P = D^{-1} A.$$

A.1 A Brief Introduction on FEATHER

The FEATHER fingerprint is motivated by modeling the distribution of node features through a characteristic function. Specifically, for each feature dimension $k = 1, \cdots, K$ and each node $j \in V$, Rozemberczki and Sarkar [18] define a series of R characteristic functions $\phi_{jk}^r : \mathbb{R} \to \mathbb{C}$, for $r = 1, \cdots, R$, such that

$$\phi_{jk}^r(t) = \sum_{l \in V} \left[P^r\right]_{jl} e^{itX_{lk}},$$

where i is the imaginary unit and recall that P^r is the r-step random walk transition matrix. The underlying assumption can be thought of that the distribution of the feature of a node is modeled by the features of its neighbors weighted by the "closeness" between them. Given M pre-specified evaluation points $t_1, t_2, \cdots, t_M$, the node-level FEATHER fingerprints are then defined as a 4-dimensional complex tensor $\Phi \in \mathbb{C}^{K \times R \times N \times M}$, where, for any $k = 1, \cdots, K$, $j \in V$, $r = 1, \cdots, R$, and $m = 1, \cdots, M$,

$$\Phi_{kjrm} = \phi_{jk}^r(t_m).$$

To simplify the notations in the following sections, we further transform Φ into a 3-dimensional real tensor $\Psi \in \mathbb{R}^{2KR \times N \times M}$ by concatenating the real part and imaginary part of Φ and merging the first two dimensions.

A.2 Multi-Channel Fuzzy Histograms and Convolution Layers

From node-level to graph-level representation. Note that the FEATHER Ψ are node-level fingerprints and have a dimension of size N. To form a graph-level representation that can be used in a graph-level prediction task, we need to eliminate the dependence of N. In previous work [18, 24], this is usually done by simply applying a sum or average pooling operation over the N nodes, which inevitably causes significant information loss. To alleviate such information loss, Li et al. [10] propose to use a histogram with B bins to summarize the distribution of the node-level fingerprints over the N nodes, and apply it to HKS [23]. To generalize this histogram idea to other fingerprints and get a smoother histogram representation, we come up with a multi-channel fuzzy histogram approach. We note that we omit the description of "channel" in Section 3 for simplicity.

Multi-channel fuzzy histograms. Suppose we concatenate all the node-level fingerprints into a 3-dimensional real tensor $\Psi \in \mathbb{R}^{L \times N \times M}$, where in the case of FEATHER, $L = 2KR$. We refer the first dimension as L channels.

For each channel $l = 1, \cdots, L$, we can define a kernel function $f^l : \mathbb{R} \to \mathbb{R}^+$ satisfying

$$\int f^l(x)dx = 1, \int x f^l(x)dx = 0, \text{ and } \int x^2 f^l(x)dx > 0. \quad (1)$$

And each channel l is also associated with B pairs of location and scale parameters $\{(\mu_b^l, \sigma_b^l)\}_{b=1}^B$. We can calculate a fuzzy histogram $H \in \mathbb{R}^{L \times B \times M}$ from the node-level fingerprints Ψ, such that, for any $l = 1, \cdots, L$, $b = 1, \cdots, B$, and $m = 1, \cdots, M$,

$$H_{lbm} = \sum_{i=1}^N f^l \left(\frac{\Psi_{lim} - \mu_b^l}{\sigma_b^l} \right).$$

We note that the application of the kernel functions to the node-level fingerprints Ψ can be calculated in fully parallel. So the transformation from Ψ to H is very efficient on GPU. In practice, we further normalize H by $H_{lbm} \leftarrow \frac{H_{lbm}}{\sum_{d=1}^B H_{ldm}}$ for any $l = 1, \cdots, L$ and $m = 1, \cdots, M$.

The choice of the kernel functions can be very flexible as long as it satisfies the conditions (1). And the choices for different channels can differ, though we make them to be the same in our implementation. The choice of kernel functions and the configurations of the location/scale parameters have already been discussed in Section 3.

Convolution layers. A nice property of the fuzzy histogram H (with 3 dimensions $\times B \times M$) is that the entries of H come with continuity in the last two dimensions respectively. The second dimension represents the (soft) bins of the histogram and the third dimension represents the evaluation points of the graph fingerprints. It is therefore compatible with 2-D convolution layers. We apply a convolutional neural network on top of the fuzzy histogram to learn the higher-level representations of the graph and make the final predictions. The location/scale parameters of the fuzzy histogram kernels and the parameters of the convolutional neural network can be jointly learned through the supervision of the graph-level labels.

B EXPERIMENT DETAILS

B.1 Baseline Methods

Graph Kernel models

- **ShortestPath** [2]: ShortestPath kernel method is based on shortest paths calculation.
- **WL** [20]: WL subtree kernel has a tight connection with 1-WL algorithm, generating WL node embedding through an iterative relabeling.
- **WL-OA** [9] Based on WL node embedding scheme, WL-OA kernel apply an optimal assignment between two sets of node embeddings in order to maximize the sum of similarities between assigned points, unlike WL subtree kernel which consider all pairwise similarity.

Graph kernels can be used to measure the similarity of two graphs. We use GraKeL library [21] to obtain kernel values for all pairs of graphs, and then we use SVM on top of it to perform classification.

MP-GNN models.

- **GIN** [28]: Graph Isomorphism Network (GIN) is a type of 1-order MP-GNN with the maximum expressive power (in the sense of WL test) among the class of 1-order MP-GNNs.
- **1-GNN** and **1-2-3-GNN** [16]: 1-GNN is another type of 1-order MP-GNN; and 1-2-3-GNN is a hierarchical high-order MP-GNN with up to 3-WL expressive power.
- **P-GNN** [13]: Powerful GNN (P-GNN) is another type of high-order MP-GNN with provably 3-WL expressive power.

In particular, P-GNN and 1-2-3-GNN are two of the state-of-the-art MP-GNN models for many graph-level prediction tasks. For 1-GNN, 1-2-3-GNN, and P-GNN, we use the official implementations in our experiments. For GIN, we use the implementation of the PyTorch-Geometric library [4]. While theoretically 1-2-3-GNN is able to achieve set 3-WL expressive power, the practical implementation uses a local approximation for better computational efficiency, which may compromise its expressive power.

Graph fingerprint methods.

- **NetLSD** [24] The NetLSD model gets the graph fingerprints based on calculating heat kernel traces of the graph at multiple time scales.
- **GeoScatteirng** [5] Geoscattering model utilize geometric wavelet scattering transform to get graph fingerprints by calculating different order geometric scattering moments.
- **FEATHER** [18]: This baseline method applies a logistic regression on top of the unsupervised FEATHER fingerprints (see Appendix A.1).

Among various unsupervised graph fingerprint methods [24, 27], FEATHER has been shown to significantly outperform others on many datasets [18]. For both the baseline NetLSD, GeoScattering, and FEATHER, we adopt the implementation from the KarateClub library [17] to obtain graph embeddings, and use logistic regression on top of it to perform graph classification.

B.2 Experiments on Real-World Datasets

Statistics of real-wolrd datasets Statistics about total number of graphs, average number of nodes and average number of edges are included in Table3.

Hyper-parameters. For all models, we grid-search hyper-parameters on the Github Stargazers dataset with the validation performance, and apply the same hyper-parameters to all other datasets. We use Adam to train all neural network models and search the initial learning rate from $\{0.0001, 0.001, 0.01\}$. For GNNs, we search the number of hidden units from $\{16, 32, 64, 128\}$. The learning rate and the number of hidden units of each GNN are individually selected. For histogram-histogram based models, there are many different variants of models but they share almost the same set of hyper-parameters. To avoid the risk of overfitting in hyper-parameter search, we search these hyper-parameters only using the FEATHER-G-L model, and apply the same value to all other histogram-based models on all datasets. We further list the major

Table 3: Summary statistics of the real-world datasets.

Dataset	Statistics		
	#Graphs	Avg. #Nodes	Avg. #Edges
MUTAG	188	17.93	19.79
NCI1	4110	29.87	32.30
PROTEINS	1113	39.06	72.82
IMDB-BIN	1000	19.77	96.53
IMDB-MULTI	1500	13.00	65.94
Deezer Egos	9629	23.49	65.25
GitHub StarGazers	12725	113.79	234.64
Twitch Egos	127094	29.67	86.59
Reddit Threads	203088	23.93	24.99

hyper-parameters we use for the histogram-based models as follows. We set the learning rate to be 0.0001 and set the the number of bins B to be 16. We use a convolution neural network with 3 convolution layers and 2 fully connected layers, with the output channels of the convolution layers being 32, 64, 64, and the number of hidden units of the fully connected layers as 1024. We apply BatchNorm2d and RELU activation function on each convolution layers, and apply MaxPool2d on second and third convolution layer. For both histogram-based models and the vanilla graph fingerprint methods, we set the number and values of evaluation points (as well as other hyper-parameters related to each fingerprint method) as default in the KarateClub library.

B.3 Experiments on Synthetic Data

Experiment setup on the regular graphs. We generate 10 different groups of regular graphs using the random_regular_graph function in NetworkX [6, 22]. Each group contains 500 graphs with the same number of nodes N and degree k. For each graph, we calculate two labels for it: (1) the number of 3-cliques (closed triangles), and (2) the number of 4-cliques. We also normalize the labels within each group. The task of counting triangles in a graph is also adopted in a synthetic data in previous work [8]. We perform two regression tasks on the random regular dataset with targets respectively as the normalized 3-clique numbers and 4-clique numbers. For each group, we use 5 cross validation to get the average test error. For each model, we apply early stopping on the validation set and report the average of the 5 test mean average error (MAE).

Experiment setup on the Poisson random graphs. For the computational cost experiments, we generate graphs through the fast_gnp_random_graph function in NetworkX [1], setting $n \in \{16, 32, 64, 128, 256\}$ and $p = 0.2$. We generate 5 datasets, each dataset has the same number of nodes and there are 1000 graphs in each dataset. We use the normalized number of 3-cliques as the graph label. For each dataset, we run 500 epochs for each model. Finally we report the peak GPU memory, preprecossing time, total training time, and training time per epoch.

Training time per epoch. Figure 1b shows the training time per epoch of different methods.