An Evaluation of Backpropagation Interpretability for Graph Classification with Deep Learning

Kenneth Teo Tian Shun
NTU Singapore

Eko Edita Limanta
NTU Singapore

Arijit Khan
NTU Singapore

***Abstract*—The end-to-end learning in convolutional neural networks (CNNs) and their ability to extract localized deep features, have turned them into powerful tools for learning from a large corpus of data including graphs. Deep neural networks such as CNNs are "black-box", therefore various interpretability methods have been developed to understand which aspects of the input data drive the decisions of the network. However, interpretability for graph convolutional neural networks (GCNNs) is an open area of research. To this end, we investigate three backpropagation-based interpretability methods: saliency map with contrastive gradients (CG), gradient-weighted class activation mapping (Grad-CAM), and deep learning important features (DeepLIFT) in conjunction with three state-of-the-art GCNNs: GCNN+GAP, DGCNN, and DIFFPOOL, as well as with their variants, for the graph classification problem. We discuss novel challenges and our solutions towards integrating these deep learning frameworks, measuring their efficiency and performance both qualitatively and quantitatively. With our extensive empirical analysis over five real-world graph datasets from different categories, we report their quantitative, visualization, and active subgraph based performance, compare them with the classic significant subgraph mining results, summarize their trade-offs and surprising findings. We conclude by discussing our recommendations on the road ahead.**

I. Introduction

Graph data are ubiquitous in many domains, such as social networks, knowledge graphs, biological networks, geo-spatial road networks, and internet-of-things. There are plenty of interest and applications in developing high-quality machine learning techniques for classifying graph objects, including cheminformatics (e.g., compounds that are active or inactive for some target) [24], [7] and bioinformatics (e.g., classifying proteins into different families) [18], malware detection and classification with call graphs [8]. There are two research directions for classification tasks on graphs: **(1)** Given a single large graph, infer labels of individual nodes (*the node classification problem*) [3], [11]; and **(2)** given a set of graphs with different structure and sizes, predict the class labels of unseen graphs (*the graph classification problem*) [7], [28], [29]. We focus on the second problem — graph classification, which is more difficult because graph datasets contain graphs with varying numbers of nodes and edges.

Moreover, nodes from different graphs may have different correspondence, and a problem-specific node ordering is often unavailable. Unlike images, general graphs do not have regular grid-like structures, and the neighborhood size of each node varies to a great extent. In practise, the lack of ordered vector representation limits the applicability of machine learning on graphs, and consequently, makes it difficult to build a classifier directly over the graph space [29], [28]. In the past, graph kernels (e.g., random walks, shortest paths, cyclic patterns, subtrees, graphlets, and subgraph kernels) [23] and feature based classification (e.g., frequent and significant subgraphs) methods [25], [16] were developed. Due to the challenges of feature engineering, coupled with the hardness of subgraph mining and isomorphism testing, deep learning methods are increasingly becoming popular. In particular, the end-to-end nature of learning in convolutional neural networks (CNNs) and their ability to extract localized spatial features, have turned them into powerful tools for learning from a large corpus of data including graphs. We focus on three state-of-the-art graph convolutional neural networks (GCNNs): GCNN with global average pooling (GCNN+GAP)[11], [13], DGCNN [29], and DIFFPOOL [28], as well as their variants.

However, deep learning models and neural networks are "black-box" — *it is inherently difficult to understand which aspects of the input data drive the decisions of the network*. Interpretability can improve the model's transparency related to fairness, privacy, and other safety challenges, thus enhancing the trust in decision-critical applications [6], [14]. Network data are complex, noisy, and content-rich. End users (e.g., data scientists, business analysts) often find them difficult to query and explore [10]. Hence, it is even more critical to provide human-understandable interpretation over classification results on such datasets. While interpretability for GCNNs is an open area of research, inspired by recent interpretability tools for CNNs, in this work we investigate three backpropagation based post-hoc interpretability methods: **(1)** saliency map with contrastive gradients (CG) [20], **(2)** gradient-weighted class activation mapping (Grad-CAM) [17], and **(3)** deep learning important features (DeepLIFT) [19]. We conduct a thorough experimental evaluation of these interpretability methods in conjunction with three state-of-the-art graph convolutional neural networks: GCNN+GAP [11], [13], DGCNN [29], and DIFFPOOL [28], and with their variants.

Challenges and our contributions. Ours is the first work that evaluates backpropagation-based interpretability methods over recent GCNN frameworks with sophisticated graph convolutional and pooling layers. When integrating these two orthogonal deep learning tools — GCNNs in one side and interpretability approaches in the other side, we encounter several technical, decision-critical, and optimization challenges.

978-1-7281-6251-5/20/$31.00 ©2020 IEEE

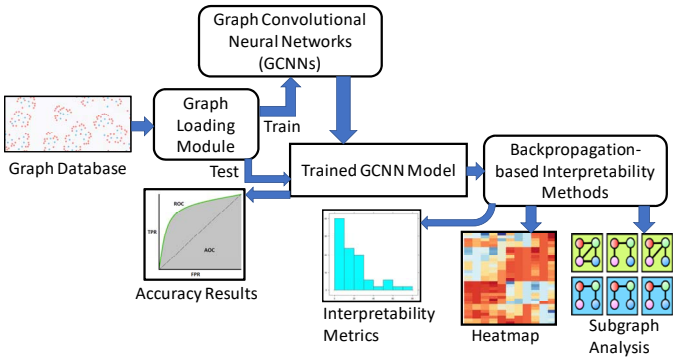


Fig. 1. Architecture of our framework

(1) We find it *non-trivial to integrate Grad-CAM with DIFFPOOL*. Grad-CAM computes a feature importance map, by associating the feature maps in the final convolutional layer with particular classes, based on the gradients of each class with respect to each feature map. However, in case of DIFFPOOL the final embedding vector from the last layer is used as input to a softmax classifier, and this embedding vector does not have direct notions of feature maps. Moreover, in DIFFPOOL an input node's assignment to any intermediate cluster is probabilistic, making it difficult to upscale weighted activations of the feature maps into the input node-space. We discuss these challenges and our solutions in § IV-B.

(2) *Visualizing higher-layer active features* of a GCNN can be more meaningful — which subgraphs (as opposed to individual nodes) drive the classification decision of the GCNN. Finding a one-to-one mapping between an intermediate active neuron and an important subgraph is difficult. We resolve this problem in two ways: (a) by computing frequent subgraphs among most active nodes identified via interpretability methods (§ V-D), and (b) by visualizing higher-level node clusterings with DIFFPOOL (§ V-C).

(3) To compare quality of the interpretability methods, we employ three quantitative metrics: fidelity, contrastivity, and sparsity. Fidelity is computed as the difference in accuracy obtained by "occluding" nodes with importance scores greater than a threshold. In graph space, occlusion of a node can be achieved in many ways: In one extreme, we can delete this node and all its incident edges. However, this could drastically change the graph's structure. Instead, we consider a *milder occlusion*: We remove associated features of the node, while keeping the graph structure unchanged. Among various possible node occlusion techniques, we find that the aforementioned approach consistently produces the best fidelity scores, thus highlighting the effectiveness of the interpretability methods.

The main contributions of this paper are as follows.

— We develop and open source an end-to-end framework in which one can plug and play with different graph datasets, GCNNs, and backpropagation-based interpretability tools, obtain results with various accuracy and interpretability metrics, generate heatmap visualization of a test graph indicating which nodes (and subgraphs) contribute positively to the output classification, and those that have a negative effect on it. The architecture of our system is given in Fig. 1. Our codebase [1] will facilitate reproducibility and future research.

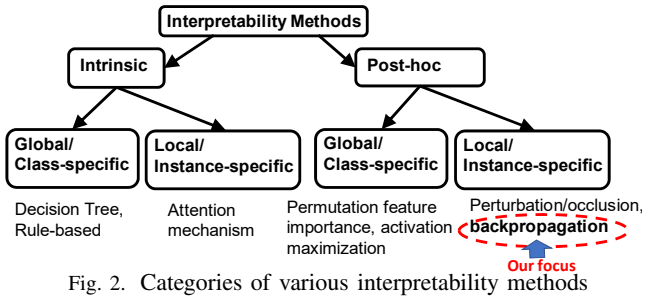


Fig. 2. Categories of various interpretability methods

— We investigate three interpretability methods: CG, Grad-CAM, and DeepLIFT in conjunction with three state-of-the-art GCNNs: GCNN+GAP, DGCNN, and DIFFPOOL, and with their variants. We design novel strategies for integrating them, measuring their efficiency and performance both qualitatively and quantitatively.

— We implement GCNNs and interpretability tools in a common environment, using same hyperparameters selection criteria, and present extensive empirical comparisons over five real-world graph datasets from different categories. We report their quantitative, visualization, and active subgraph based performance, compare them with the classic significant subgraph mining results [16], summarize their trade-offs, and provide guidelines for researchers and practitioners.

II. Related Work

Interpretability methods for deep learning. Recent developments on interpretability can be grouped into two broad categories: **(1)** *intrinsic*, i.e., by constructing self-explanatory models which incorporate interpretability directly to their structures, and **(2)** *post-hoc*, i.e., by creating a second model to provide explanations for an existing model. The post-hoc ones could be limited in prediction performance, while keeping the underlying model accuracy intact. Interpretability methods can also be classified as: **(1)** *global*, where users can understand how the model works globally by inspecting the structures and parameters of a complex model, thereby explaining the "essence" of a class; and **(2)** *local*, that locally examines an individual prediction of a model, trying to figure out why the model makes the decision that it makes. Example interpretability methods for each category are given in Fig. 2. For surveys and recommendations, we refer to [6], [14], [4].

Perturbation/ occlusion-based models determine the contribution of a feature by measuring how prediction score changes when the feature is altered. While being model-agnostic, such methods can be computationally inefficient as each perturbation requires a separate forward propagation through the network. Backpropagation methods, in contrast, are model-specific, and calculate the gradient (or its variants) of a particular output with respect to the input using backpropagation to derive the contribution of features. The intuition is that the magnitude of larger gradient may represent more substantial relevance of a feature to the prediction. Backpropagation methods are more efficient, since they usually need only one forward and another backward pass.

Interpretability in deep learning based graph classification. There is limited work on interpretability of neural network-

based graph classification models. We discuss two recent ones: GNNExplainer [27] formalizes the notion of importance using mutual information, thus identifies a subgraph of the input graph and a subset of node features that are most influential for the model's prediction. This is similar to perturbation/ occlusion-based interpretability methods. The second method, GroupINN [26] performs node grouping across multiple training graphs, and incorporates such node groupings into the framework of neural network, thereby identifying the most predictive subgraphs within several task-specific contexts. This is different from instance-specific and backpropagation-based interpretability methods that we investigate in this paper.

Some GCNN models also provide interpretability via attention mechanisms [22]. However, attention mechanisms are intrinsic (i.e., incorporating interpretability directly into the models), and often cannot explain predictions by jointly considering both graph structure and node features [27].

Recently, [15] empirically compared a few backpropagation based interpretability methods over a simpler GCNN model. In contrast, we are the first to conduct a comprehensive interpretability study with state-of-the-art GCNNs: DGCNN and DIFFPOOL, and with their variants. They are more complex due to their sophisticated graph convolutional and pooling layers. We also consider DeepLIFT, a recent and more advanced interpretability tool compared to the ones used in [15], e.g., considering both positive and negative contributions, DeepLIFT reveals dependencies missed by other backpropagation approaches. Due to more complex GCNNs and interpretability tools that we consider in this work, their integration is more challenging. Unlike [15], we open source an end-to-end framework in which one can plug and play with different graph datasets, GCNNs, and backpropagation-based interpretability tools, thereby facilitating reproducibility and future research. Moreover, we additionally report running times, compare them with the classic significant subgraph mining, summarize their trade-offs, and provide guidelines.

Graph convolutional neural networks (GCNNs). Convolutional neural networks (CNNs) consist of alternatively stacked convolutional layers and spatial pooling layers. Researches that generalize convolution to graphs are categorized as spectral [3] and spatial [7] methods. In spectral approaches, filters are applied on a graph's frequency modes computed via graph Fourier transform. One limitation of spectral formulations is that they rely on the fixed spectrum of the graph Laplacian, and thus are suitable only for graphs with a single structure (and varying features on nodes). Spatial approaches, in contrast, are not restricted to a fixed graph structure, as they extract local features by propagating features between neighboring nodes. Later, Kipf and Welling [11] develop a first-order approximation of the spectral convolution, which also results in propagation between neighboring nodes.

However, these previous works have relatively poor performance on graph classification tasks. One reason for this is that after extracting localized node features, these features are directly summed up as a graph-level feature used for graph classification. Therefore, to better preserve the relationship between nodes and to get the entire graph representation, recent studies develop more sophisticated pooling operations for graphs.We shall discuss three of them, namely, GCNN+GAP[11], [13], DGCNN [29], and DIFFPOOL [28], and their two variants in § III.

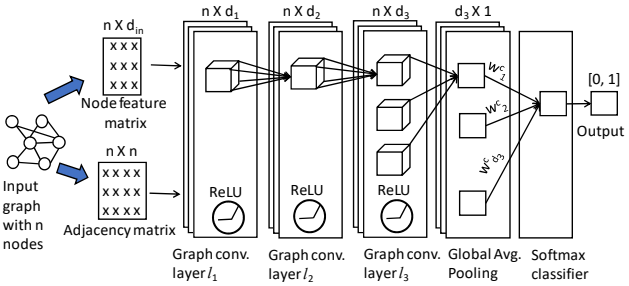


Fig. 3. GCNN with global average pooling (GCNN+GAP)

Significant subgraphs mining. Given a set of training graphs (from both positive and negative classes), [25], [16] mine significant subgraphs — subgraphs that are "over-represented" in each class. These works can be categorized under the classic subgraph mining literature published a decade ago. Once significant subgraph features are identified, each input graph is represented as a vector, and machine learning models are employed. Thus, significant subgraph mining based interpretability methods are intrinsic and class-specific.

III. Graph Classification with Deep Learning

We study three state-of-the-art graph convolutional neural networks (GCNNs): GCNN with global average pooling (GCNN+GAP)[11], [15], DGCNN [29], and DIFFPOOL [28], as well as their variations. They all support end-to-end gradient-based training with the original graphs, without the need to first transform graphs into vectors.

A. GCNN + global average pool (GCNN+GAP)

This architecture in Fig. 3 consists of graph convolutional layers, followed by a global average pooling (GAP) layer, and a softmax classifier. Convolutional layers extract localized spatial features from the input graph, and the GAP layer enforces correspondences between feature maps and classes. Moreover, there is no parameter to optimize in the global average pooling, thus it acts as a structural regularizer [13].

We denote a graph G having n nodes with the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and the node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d_{in}}$. The degree matrix for this graph is: $\mathbf{D}_{ii} = \sum_{j=1}^{n} \mathbf{A}_{ij}$. Let $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_n$ be the adjacency matrix with added self loops, where $\mathbf{I}_n \in \mathbb{R}^{n \times n}$ is the identity matrix, and $\widetilde{\mathbf{D}}_{ii} = \sum_{j=1}^{n} \widetilde{\mathbf{A}}_{ij}$. Following [11], we formally define the graph convolutional layer as follows:

$$\mathbf{F}^{l}(\mathbf{X}, \mathbf{A}) = \sigma(\widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2}\mathbf{F}^{l-1}(\mathbf{X}, \mathbf{A})\mathbf{W}^{l}) \qquad (1)$$

Here, $\mathbf{W} \in \mathbb{R}^{d_l \times d_{l+1}}$ is a matrix of trainable graph convolution parameters for layer l, converting d_l feature maps to d_{l+1} maps in the next layer. We denote by $\sigma(\cdot)$ element-wise nonlinear activation function, e.g., ReLU, defined as $y = \max(0, x)$. For initialization, we set $\mathbf{F}^{0}(\mathbf{X}, \mathbf{A}) = \mathbf{X}$.

Let the k^{th} graph convolutional feature map at layer l be defined as: $\mathbf{F}^l_k(\mathbf{X}, \mathbf{A})$, which is a vector of length n, and is computed as given in Eq. 2. We denote by $\mathbf{W}^l_k$ the k^{th} column of matrix $\mathbf{W}^l$.

$$\mathbf{F}^l_k(\mathbf{X}, \mathbf{A}) = \sigma(\widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2}\mathbf{F}^{l-1}(\mathbf{X}, \mathbf{A})\mathbf{W}^l_k) \tag{2}$$

For a node $i \in [1, n]$, the k^{th} feature at the l^{th} layer is denoted by $F^l_{k,i}(\mathbf{X}, \mathbf{A})$. Then, the GAP feature after the final convolutional layer, L, is:

$$e_k = \frac{1}{n}\sum_{i=1}^{n} F^L_{k,i}(\mathbf{X}, \mathbf{A}) \tag{3}$$

The class score is calculated as: $y^c = \sum_{k=1}^{d_L} w^k_c e_k$. The weight w^k_c encodes the importance of feature k for the predicting class c, and is a learnable parameter. Finally, the softmax layer reports the class probability as: $\frac{\mathbf{e}^{y^c}}{\sum_c \mathbf{e}^{y^c}}$.

B. GCNN with sort pool (DGCNN)

DGCNN consists of graph convolutional layers, followed by a *SortPooling* layer, traditional convolutional and dense layers, and finally a softmax classifier. The convolutional layers extract localized spatial features from the input graph, as well as define a sorting order among the nodes. The SortPooling layer sorts the nodes based on the previously defined order and selects the top-k among them. The traditional convolutional and dense layers read the sorted graph representations and predict the class label with the help of the softmax classifier. We show the DGCNN framework in Fig. 4.

In DGCNN, the graph convolutional layer is defined as:

$$\mathbf{F}^l(\mathbf{X}, \mathbf{A}) = \sigma(\widetilde{\mathbf{D}}^{-1}\widetilde{\mathbf{A}}\mathbf{F}^{l-1}(\mathbf{X}, \mathbf{A})\mathbf{W}^l) \tag{4}$$

We initialize $\mathbf{F}^0(\mathbf{X}, \mathbf{A}) = \mathbf{X}$. For graph convolutional layers, $\sigma(\cdot) = tanh(\cdot)$ activation function is used.

Next, we concatenate the outputs $\mathbf{F}^{1:L} = [\mathbf{F}^1, \mathbf{F}^2, \ldots, \mathbf{F}^L]$, where L is the number of graph convolutional layers, and $\mathbf{F}^{1:L} \in \mathbb{R}^{n \times \sum_{l=1}^{L} d_l}$. In the concatenated output $\mathbf{F}^{1:L}$, each row can be regarded as a "feature descriptor" of a node, encoding its multi-scale local substructure information. We input $\mathbf{F}^{1:L}$ to the SortPooling layer, and its output is a matrix $\mathbf{F}^{sp} \in \mathbb{R}^{k \times \sum_{l=1}^{L} d_l}$. The node order is based on $\mathbf{F}^L$, and within $\mathbf{F}^L$, by first sorting nodes using the last channel of $\mathbf{F}^L$ in a descending order. If two nodes have the same value in the last channel, the tie is broken by comparing their values in the second last channel, and so on. Then, SortPooling reports the top-k sorted rows. The intuition is that the SortPooling layer sorts the localized node features instead of summing them up, which can keep much more information, and permits us learning from the global graph topology. In addition, it unifies graph sizes: Graphs with different numbers of nodes are unified on their sizes by retaining the top-k nodes.

We next reshape $\mathbf{F}^{sp}$ into a $k(\sum_{l=1}^{L} d_l) \times 1$ vector row-wise. Several 1-D convolutional layers and MaxPooling layers are added. Finally, we add a fully-connected layer, followed by a softmax classifier that predicts the class label of the input graph. ReLU activation function is used in 1-D convolutional and fully-connected layers.

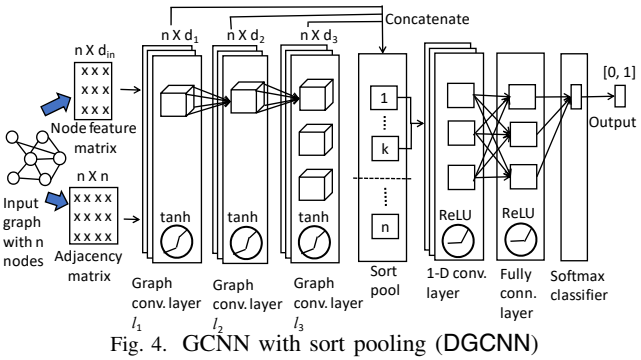


Fig. 4. GCNN with sort pooling (DGCNN)

C. GCNN with differentiable pool (DIFFPOOL)

DIFFPOOL generates hierarchical representations of graphs, by learning a differentiable soft cluster assignment for nodes at each layer of a graph neural network (GNN), mapping nodes to a set of clusters, which then form the coarsened input for the next GNN layer (Fig. 5).

DIFFPOOL introduces a framework that jointly learns the pooling and node embeddings. Formally, we denote the learned cluster assignment matrix at layer l as $\mathbf{S}^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$. Each row of $\mathbf{S}^{(l)}$ corresponds to one of the n_l nodes (or clusters) at layer l, and each column of $\mathbf{S}^{(l)}$ corresponds to one of the n_{l+1} clusters at the next layer $l+1$, with $n_{l+1} < n_l$. Intuitively, $\mathbf{S}^{(l)}$ provides a soft assignment of each node at layer l to a cluster in the next coarsened layer $l+1$.

At layer l, we denote by $\mathbf{A}^{(l)} \in \mathbb{R}^{n_l \times n_l}$, $\mathbf{X}^{(l)} \in \mathbb{R}^{n_l \times d_{in}}$, and $\mathbf{Z}^{(l)} \in \mathbb{R}^{n_l \times d_{in}}$ the adjacency matrix, node (or cluster) feature matrix, and node (or cluster) embedding matrix, respectively. We initialize $\mathbf{A}^{(l)} = \mathbf{A}$ and $\mathbf{X}^{(l)} = \mathbf{X}$. They are computed as follows.

$$\mathbf{Z}^{(l)} = \text{GNN}_{l,\text{embed}}(\mathbf{A}^{(l)}, \mathbf{X}^{(l)}) \tag{5}$$

$$\mathbf{S}^{(l)} = \text{softmax}(\text{GNN}_{l,\text{pool}}(\mathbf{A}^{(l)}, \mathbf{X}^{(l)})) \tag{6}$$

$$\mathbf{X}^{(l+1)} = {S^{(l)}}^T Z^{(l)} \tag{7}$$

$$\mathbf{A}^{(l+1)} = {S^{(l)}}^T A^{(l)} S^{(l)} \tag{8}$$

In the first two equations above, the two GNNs consume the same input (i.e., feature matrix $\mathbf{X}^{(l)}$ and adjacency matrix $\mathbf{A}^{(l)}$); however, they have distinct parameterizations, produce outputs of different dimensionality, and thus play separate roles. The embedding GNN generates new embeddings for the input clusters at this layer l, whereas the pooling GNN computes a probabilistic assignment of the input clusters to coarsened n_{l+1} clusters. The softmax function is applied in a row-wise manner. The original DIFFPOOL paper [28] advocates using GraphSAGE [9] together with mean aggregator as the underlying GNN module. In brief, GraphSAGE learns a function that generates embeddings by sampling and aggregating features from a node's local neighborhood, and it also generalizes GCNN to use trainable aggregation functions.

The last two equations above coarsen the current graph, generating a new coarsened adjacency matrix $\mathbf{A}^{(l+1)}$, and a new matrix $\mathbf{X}^{(l+1)}$ of features for each of the clusters in the coarsened graph. Embedding vectors from the last DIFFPOOL layer are combined to form one single embedding vector of dimensionality d_{in} by taking maximum across each

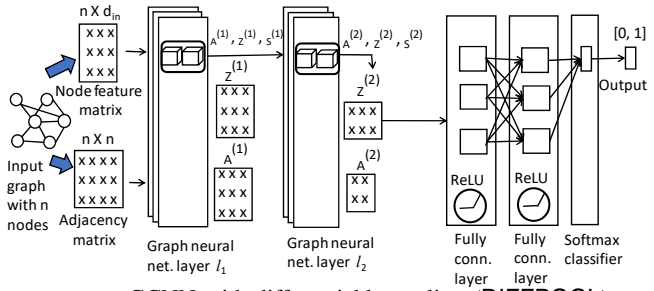


Fig. 5. GCNN with differentiable pooling (DIFFPOOL)

embedding dimension. This final vector is used as input to fully-connected layers, followed by a softmax classifier to predict input graph's class label.

D. Horizontal comparison among three GCNNs

All three GCNNs support graph classification in an end-to-end fashion, without the need to first transform graphs into vectors. Their pooling layers are invariant under node permutations. Any permutation in the adjacency matrix produces an isomorphic graph, thus outputs the same prediction. This is important for GCNNs.

Notice that three GCNNs have different types of graph convolutional and pooling layers, thus the final accuracy comparison for GCNN+GAP, DGCNN, and DIFFPOOL is unknown. Therefore, our experimental analysis is critical. On the other hand, DIFFPOOL could identify meaningful clusters in the input graphs (e.g., groups of atoms and bonds representing functional units in a molecule), thus it can be more interpretable compared to GCNN+GAP and DGCNN.

For both DGCNN and DIFFPOOL, the pooling layers are more complex than that in GCNN+GAP. However, GCNN+GAP's convolution operation is more expensive. Thus, their final efficiency comparison is also unknown, and our empirical analysis is required.

Other variants. We observe that the graph convolutional layers in GCNN+GAP and DGCNN are slightly different (Eq. 1 and 4). In case of GCNN+GAP, the graph convolution form is same as the spectral filter proposed in [11]. For DGCNN [29], it is shown to be a closer approximation to a soft version of the Weisfeiler-Lehman kernel, as well as the propagation kernel. Thus, we consider a variation of GCNN+GAP in our experiments, referred to as GCNN_D, where we use the graph convolutional layer of DGCNN.

For DIFFPOOL, we also consider a variant where we employ DGCNN's convolution (instead of GraphSAGE that is advocated in the original paper [28]) as the underlying GNN module. We refer to this version as DIFFPOOL_D.

IV. Interpretability Methods for Deep Learning

We study three backpropagation-based post-hoc interpretability methods: saliency map with contrastive gradients (CG) [20], gradient-weighted class activation mapping (Grad-CAM) [17], and deep learning important features (DeepLIFT) [19]. They assign an attribution value, called "relevance" or "contribution", to each input node of a test graph with respect to an output class label. Backpropagation propagates an importance signal from an output neuron backwards through the layers to the input in one pass, while the weights are fixed to those found during the training. This makes backpropagation based interpretability methods highly efficient.

A. Saliency map with contrastive gradients (CG)

This is a straight-forward and well-established method that uses the gradient of the output with respect to nodes of an input graph to compute a saliency map (or attribution map), which is a heatmap showing attributions of all input nodes, in the context of classification tasks. In this saliency map, the norm of the gradient over input nodes indicates their relative importance. The resulting gradient in the input space points in the direction corresponding to the maximum positive rate of change in the model output. Therefore, the negative values in the gradient are discarded to only retain the parts of input that positively contribute to the solution. Formally,

$$r^c_{GC}(i) = ||ReLU(\frac{\delta y^c}{\delta i})|| \tag{9}$$

where y^c is the score for class c before the softmax layer, and i is the input node. Due to the zero-ing out of negative gradients, CG can fail to highlight inputs that contribute negatively to the output. Additionally, it may also suffer from gradient saturation problem, and discontinuities in the gradients can cause undesirable results.

B. Gradient-weighted class activation mapping

The earlier CG method highlights fine-grained details in the input, but is not class-discriminative. In contrast, gradient-weighted class activation mapping (Grad-CAM) is highly class-discriminative, since it computes a coarse grained feature importance map by associating the feature maps in the final convolutional layer with particular classes, based on the gradients of each class with respect to each feature map, and then using the weighted activations of the feature maps as an indication of which inputs are most important.

Formally, let $\mathbf{F}^L_k \in \mathbb{R}^{n\times 1}$ be the k^{th} feature map of the last convolutional layer, L (in case of GCNN+GAP and DGCNN). For a node $i \in [1, n]$, the k^{th} feature at the L^{th} layer is denoted by $F^L_{k,i}(\mathbf{X}, \mathbf{A})$. Next, Grad-CAM computes feature map weights $\alpha^c_{L,k}$, related to class score y_c (before the softmax layer) for the predicting class c, as follows.

$$\alpha^c_{L,k} = \frac{1}{n}\sum_{i=1}^{n}\frac{\delta y^c}{\delta F^L_{k,i}} \tag{10}$$

Clearly, the weight $\alpha^c_{L,k}$ encodes the importance of the k^{th} feature map of the last convolutional layer on the predicting class c. By upscaling each feature map to the size of the input graph (to undo the effect of pooling layers), the class-specific heatmap in the node-space becomes:

$$r^c_{Grad-CAM}(i) = ||ReLU(\sum_{k=1}^{d_L}\alpha^c_{L,k}F^L_{k,i})|| \tag{11}$$

Grad-CAM with DIFFPOOL. The same method cannot be directly applied with DIFFPOOL. First, the embedding vector from the last layer is used as input to a softmax classifier, but it does not have notions of feature maps. We thus employ each embedding dimension as a feature map, and there are total d_{in}

embedding dimensions. Second, an input node's assignment to any intermediate cluster is soft, i.e., probabilistic. Hence, we determine the node cluster membership by taking the argmax of its cluster assignment probabilities. Last, input nodes in the last DIFFPOOL layer L are assigned to fewer clusters. Since all nodes in the same cluster would have the same attribution scores, considering the final embedding vectors from the last layer is not meaningful for generating a diverse set of attribution scores. Instead, we consider node assignments in the first layer l, where there are more clusters. At layer l, $\mathbf{Z}^{(l)} \in \mathbb{R}^{n_l \times d_{in}}$ denotes the corresponding cluster embedding matrix, and $Z^{(l)}_{k,i}$ the value at embedding dimension k for the i^{th} cluster in that layer. We compute the weight $\alpha^c_{l,k}$, which encodes the importance of the k^{th} embedding dimension at layer l on the predicting class c, as follows.

$$\alpha^c_{l,k} = \frac{1}{n_l} \sum_{i=1}^{n_l} \frac{\delta y^c}{\delta Z^{(l)}_{k,i}} \tag{12}$$

Assuming that node j is assigned to cluster i at layer l, the class-specific heatmap in the node-space is computed as:

$$r^c_{Grad-CAM}(j) = ||ReLU(\sum_{k=1}^{d_{in}} \alpha^c_{l,k} Z^{(l)}_{k,i})|| \tag{13}$$

The Grad-CAM strategy inherits some of the limitations of CG caused by zero-ing out negative gradients during back-propagation. It also suffers from gradient saturation problem and discontinuities in gradients. On other hand, it requires partial backward pass per input sample (e.g., from output neuron backwards only to last convolutional layer for DGCNN).

C. Deep Learning Important Features (DeepLift)

DeepLIFT starts at output layer L (before softmax), and proceeds in backward fashion. Each neuron a is assigned an attribution that denotes relative effect of the neuron activated at the original input graph G, compared to the activation at some reference input G^0 (Eq. 14). The reference graph G^0 is chosen to be isomorphic to the input graph G, while the features of all nodes are removed (i.e., node feature matrix $\mathbf{X} = \mathbf{0}$). Reference values for all neurons are determined running a forward pass through the network, using the baseline G^0 as input, and recording the activation of each neuron.

$$r^{c,L}(a) = \begin{cases} y^c(G) - y^c(G^0), & \text{if } c \text{ is the predicting class} \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

$$r^{c,l}(a) = \sum_b \frac{z_{ba} - z^0_{ba}}{\sum_{a'} z_{ba'} - \sum_{a'} z^0_{ba'}} r^{c,l+1}(b) \tag{15}$$

where $y^c(G)$ is the score for class c at the last layer L (i.e., before the softmax layer) considering the original input G, and $y^c(G^0)$ is the score for class c at the last layer L with the reference input G^0.

Following the "rescale rule" in the original paper [19], the backward relevance propagation is described in Eq. 15. Here, a, a' are neurons at layer l, and b is a neuron at the next layer $l+1$. The first summation on the right-hand side is taken over all neurons b at layer $l+1$. Analogously, the summations at the denominator are taken over all neurons a' at layer l. We define $z_{ba} = w^{l+1,l}_{b,a} G(a)$ to be the weighted activation of a neuron a onto neuron b in the next layer, where $w^{l+1,l}_{b,a}$ is weight between neurons a and b learnt during the training stage, and $G(a)$ is the activation of neuron a determined by running a forward pass through the network using the original input G. Similarly, we define $z^0_{ba} = w^{l+1,l}_{b,a} G^0(a)$ to be the weighted activation of a neuron a onto neuron b when the reference input G^0 is fed into the network. Finally, the class-specific heatmap for a node i is computed as:

$$r^c_{DEEPLIFT}(i) = r^{c,l=0}(i) \tag{16}$$

We notice that by considering both positive and negative contributions, DeepLIFT can reveal dependencies missed by other backpropagation based approaches. Moreover, by using a difference from reference, DeepLIFT can propagate an importance signal even in situations where the gradient is zero and avoids problems caused by discontinuities in the gradient. Ancona et al. [2] have discussed theoretical equivalence of DeepLIFT with other existing backpropagation-based interpretability tools, e.g., layer-wise relevance propagation, integrated gradients, and gradient $\times$ input.

D. Horizontal comparison among three interpretability tools

The three interpretability methods that we study are backpropagation-based and post-hoc. Among them, Grad-CAM is highly class-discriminative. DeepLIFT requires a reference input; by considering both positive and negative contributions it reveals dependencies missed by other back-propagation based approaches. Also, DeepLIFT can avoid problems due to saturation and discontinuities in the gradients.

All three interpretability approaches are highly efficient. In particular, GC requires one forward and one backward pass, Grad-CAM requires one forward and one partial backward pass, while DeepLIFT does two forward passes (due to input graph and reference graph) and one backward pass.

V. Experimental Results

We conduct experiments to compare three backpropagation-based interpretability methods over five GCNNs (three state-of-the-art and their two variants), and report their quantitative, visualization, and active subgraph feature-based results, using five real-world networks. The architecture of our system is given in Fig. 1. We employ PyTorch v1.5 deep learning library to build GCNN models. The NetworkX library is used to load and transform graphs as Python lists. We modify Captum interpretability framework [12], originally developed for standard neural networks, to fit with GCNNs, and integrate it with PyPlot API for visualisation. We perform experiments on a single machine with 16GB, 3GHz Intel i5-7400 processor. Our GPU platform is GeForce GTX 1070 (8GB VRAM) with CUDA 10.1. Our codebase and datasets are at [1].

Datasets. We use five real-world graph datasets from two different categories (Table I). The first four belong to bioinformatics benchmarks [1]: *MUTAG* [5] is a dataset of mutagenic

[1] https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets
[2] https://pubchem.ncbi.nlm.nih.gov/
[3] https://github.com/quarkslab/dataset-call-graph-blogpost-material

TABLE I
PROPERTIES OF DATASETS

dataset	#classes & distribution	#node-features	(avg.) #nodes / graph	(avg.) #edges/ graph	(avg.) #node-features/ graph	(avg.) #node-features/ node
MUTAG	2 [inactive:63, active:125]	7	18	20	3	1
NCI-H23	2 [inactive:2000, active:500]	18	27	29	3	1
Tox21_AR	2 [no-toxic:8969, toxic:380]	42	18	19	3	1
PTC_FR	2 [no-toxic:230, toxic:121]	19	15	15	3	1
Callgraph	2 [sane:546, malware:815]	27	782	1852	10	4

aromatic and heteroaromatic nitro compounds classified according to their mutagenic effects on a bacterium. *Tox21_AR* [24] contains qualitative toxicity measurements of compounds. *PTC_FR* [21] is a dataset of chemical compounds classified according to their carcinogenicity on female rats. *NCI-H23* [2] contains anti-cancer screens for cell lung cancer over small molecules [25]. Our last dataset is a *callgraph* dataset [3], consisting of 1361 portable executable, 546 of them are sane files and 815 are malicious ones. Call graphs are extracted in a static way from these executables with the help of radare2. Each graph is much larger in this dataset. For every node, which represents a function, we assign the most frequent opcodes (and their counts) as its features.

Node features are binary except for *callgraph*. In *callgraph* a node (i.e., a function) can have multiple (≈ 4) features (i.e., frequent opcodes), and we store in the feature matrix the count of these features (opcodes) within a node (function).

GCNN architectures and hyperparameters selection. We perform 5-fold cross-validation. The learning rates and the number of epochs are selected from {0.01, 0.001, 0.0001, 0.00001} and {50, 100, 150, 200}, respectively, via the ADAM optimizer. We follow the same GCNN architectures (e.g., number of different layers and their sizes) as in the original papers, since they consistently perform well in our experiments. We omit details due to lack of space, and they are well documented in our webpage for the codebase [1].

Finally, we design a significant subgraph mining based graph classification model: Significant subgraph features are identified via GraphSig [16]. Next, each input graph is represented as a binary vector, based on the presence/ absence of these significant subgraph features. The vectors are passed through a deep neural network (DNN) consisting of dense layers, and followed by a softmax layer for classification. We refer to this architecture as GraphSig+DNN. We notice that GraphSig identifies significant subgraphs when each node has a single binary feature; it is non-trivial to extend GraphSig for nodes having multiple non-binary features. Thus, we cannot run GraphSig+DNN over *callgraph*.

A. Classification performance

AUC-ROC and AUC-PR [15], [24], [25], [16] are used to analyze the performance of six classification models. They are special-purpose evaluation metrics suitable for evaluating classifiers over imbalanced datasets (such as ours in Table I)[4]. Based on the AUC-ROC and AUC-PRC results in Table II, *there is no common winner*. We notice the following trends. **(1)** *Generally, DIFFPOOL$_D$ performs the best, closely followed by DIFFPOOL and DGCNN.* These results illustrate the benefit of using our variant of DIFFPOOL, namely DIFFPOOL$_D$. Among the five GCNNs, GCNN+GAP and GCNN$_D$, with much simpler model, has worse performance. **(2)** *GraphSig+DNN exhibits quite fluctuating performance over our datasets. We suspect that this is due to varying qualities of the "hand-crafted" patterns mined from training graphs*. Moreover, GraphSig+DNN method cannot work over *callgraph* with nodes having multiple non-binary features. This shows the advantage of using end-to-end GCNNs over classic significant subgraph mining based approaches.

Different classification models require different number of epochs for optimal performance over various datasets. For a fairer comparison of efficiency of these models, we show the running times of one forward and one backward pass in Table III. We find that **(1)** among the GCNNs, *GCNN$_D$ is the fastest in both forward and backward passes*. This demonstrates the efficiency benefit of using our variant of GCNN+GAP, namely GCNN$_D$. In fact, the original model GCNN+GAP, due to its more expensive convolutional operation, is time-consuming over larger *callgraph* dataset. **(2)** *The forward and backward pass times for GCNN+GAP, DGCNN, DIFFPOOL, and DIFFPOOL$_D$ are comparable*. We find that both DIFFPOOL and DIFFPOOL$_D$ require longer time over *callgraph* due to the usage of two DIFFPOOL layers (suggested in [28]), and also due to larger graphs in this data. **(3)** *GraphSig+DNN can be 3∼6 orders of magnitude slower than GCNNs*. We recall that significant subgraph mining based approaches are not end-to-end for graph classification. We, therefore, add subgraph mining time and graph-to-vector conversion time (both are quite expensive) in forward pass time, while only graph-to-vector conversion time in backward pass time. These results show the efficiency benefit of GCNNs over classic significant subgraph mining methods.

B. Interpretability: quantitative performance

The node attribution values assigned by interpretability methods are normalized between 0 to 1 for both CG and Grad-CAM, while they are between -1 to 1 for DeepLIFT. Since our interpretability methods are instance-specific, we normalize for each input instance (i.e., graph) by dividing the attribution values of its nodes with the largest attribution value (in magnitude) within that instance. Next, to compare these interpretability methods, we employ three quantitative

[4] AUC-ROC measures the area under the curve that visualizes the tradeoff between true positive rate (TPR) and false positive rate (FPR). In ROC plot, classifiers with random performance show a diagonal line, thus baseline AUC-ROC score is 0.5. Higher AUC-ROC score means that the curve is more top-left, implying a higher TPR and lower FPR for each threshold, this indicates that the classifier is better. Similarly, AUC-PR reports the area under the curve that combines precision and recall in a single visualization. The baseline of PR is determined by the ratio of positives (P) and negatives (N) as: P/(P + N), e.g., for *Tox21_AR* the baseline AUC-PR is 380/(380+8969) = 0.04, indicating only 4% correct predictions, on an average, among the positive predictions by a random classifier. A higher AUC-PR score than this baseline implies that the classifier is better.

TABLE II
COMPARISON OF CLASSIFICATION ACCURACY

	AUC-ROC						AUC-PRC					
	GCNN+GAP	GCNN$_D$	DGCNN	DIFFPOOL	DIFFPOOL$_D$	GraphSig+DNN	GCNN+GAP	GCNN$_D$	DGCNN	DIFFPOOL	DIFFPOOL$_D$	GraphSig+DNN
MUTAG	0.78	0.74	**0.90**	0.83	**0.90**	0.71	0.88	0.86	**0.95**	0.90	**0.95**	0.86
NCI-H23	0.62	0.58	0.68	**0.70**	**0.70**	**0.70**	0.29	0.25	0.35	**0.43**	0.40	0.34
Tox21_AR	0.69	0.77	0.72	**0.79**	**0.79**	0.54	0.08	0.27	0.29	**0.42**	0.38	0.37
PTC_FR	**0.65**	0.60	0.63	0.61	0.58	0.63	**0.56**	0.49	0.53	0.51	0.46	0.27
Callgraph	0.86	0.84	**0.89**	0.87	0.87	CAN'T RUN	0.89	0.90	**0.92**	0.90	0.90	CAN'T RUN

TABLE III
COMPARISON OF TRAINING TIMES FOR CLASSIFICATION

	forward pass						backward pass					
	GCNN+GAP	GCNN$_D$	DGCNN	DIFFPOOL	DIFFPOOL$_D$	GraphSig+DNN	GCNN+GAP	GCNN$_D$	DGCNN	DIFFPOOL	DIFFPOOL$_D$	GraphSig+DNN
MUTAG	1.42 ms	**0.55 ms**	1.53 ms	2.15 ms	1.91 ms	4.88 sec	0.76 ms	**0.68 ms**	2.55 ms	2.02 ms	1.49 ms	0.91 sec
NCI-H23	1.61 ms	**0.81 ms**	1.39 ms	2.36 ms	2.05 ms	299.68 sec	0.87 ms	**0.79 ms**	2.21 ms	2.08 ms	1.40 ms	0.46 sec
Tox21_AR	1.73 ms	**0.73 ms**	1.54 ms	2.43 ms	2.08 ms	4222.22 sec	0.91 ms	**0.74 ms**	2.47 ms	2.16 ms	1.44 ms	3.55 sec
PTC_FR	1.39 ms	**0.73 ms**	1.53 ms	2.21 ms	1.98 ms	8.54 sec	0.74 ms	**0.73 ms**	2.56 ms	2.13 ms	1.51 ms	0.08 sec
Callgraph	29.41 ms	3.93 ms	**2.91 ms**	44.37 ms	201.97 ms	CAN'T RUN	5.00 ms	**4.49 ms**	4.62 ms	52.58 ms	152.73 ms	CAN'T RUN

TABLE IV
COMPARISON OF FIDELITY

		MUTAG	*NCI-H23*	*Tox21_AR*	*PTC_FR*	*Callgraph*
GCNN+GAP	CG	0.551	0.030	0.164	0.126	0.020
	Grad-CAM	**0.782**	0.460	0.502	0.493	**0.123**
	DeepLIFT	0.764	**0.549**	**0.549**	**0.507**	0.085
GCNN$_D$	CG	0.061	0.011	-0.005	0.024	0.011
	Grad-CAM	**0.692**	**0.578**	-0.164	0.525	**0.822**
	DeepLIFT	0.568	0.548	**0.612**	**0.534**	0.056
DGCNN	CG	0.021	0.023	0.125	0.030	0.016
	Grad-CAM	**0.233**	**0.587**	**0.236**	**0.441**	**0.209**
	DeepLIFT	0.049	0.246	0.233	0.229	0.041
DIFFPOOL	CG	0.179	0.061	0.065	0.116	-0.001
	Grad-CAM	**0.194**	-0.033	0.043	0.198	**0.213**
	DeepLIFT	0.229	**0.245**	**0.128**	**0.240**	0.047
DIFFPOOL$_D$	CG	0.141	0.090	0.137	0.028	0.047
	Grad-CAM	**0.378**	**0.693**	**0.194**	**0.345**	**0.227**
	DeepLIFT	0.296	0.468	0.184	0.332	0.081

TABLE V
COMPARISON OF CONTRASTIVITY

		MUTAG	*NCI-H23*	*Tox21_AR*	*PTC_FR*	*Callgraph*
GCNN+GAP	CG	0.069	0.030	0.043	0.036	0.009
	Grad-CAM	**0.640**	**0.868**	**0.781**	**0.548**	**0.068**
	DeepLIFT	0.052	0.053	0.018	0.030	0.001
GCNN$_D$	CG	0.000	0.026	0.047	0.001	0.010
	Grad-CAM	**0.779**	**0.939**	**0.962**	**0.714**	**0.851**
	DeepLIFT	0.001	0.060	0.132	0.059	0.005
DGCNN	CG	0.120	0.051	0.041	0.116	0.006
	Grad-CAM	**0.858**	**0.715**	**0.543**	**0.719**	**0.268**
	DeepLIFT	0.125	0.036	0.033	0.112	0.002
DIFFPOOL	CG	0.003	0.000	0.001	0.004	0.000
	Grad-CAM	**0.987**	**1.000**	**1.000**	**0.983**	**1.000**
	DeepLIFT	0.003	0.000	0.002	0.005	0.000
DIFFPOOL$_D$	CG	0.030	0.006	0.078	0.044	0.007
	Grad-CAM	**0.854**	**1.000**	**0.792**	**0.813**	**0.803**
	DeepLIFT	0.077	0.031	0.137	0.137	0.003

metrics: fidelity, contrastivity, and sparsity [5] [15], [4]. Higher accuracy and fidelity could help improving the user's trust in the classification model, while higher contrastivity and sparsity indicate that the interpretation is more useful.

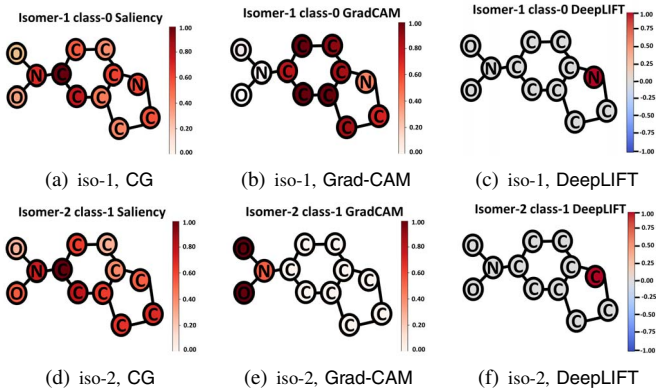
Following quantitative results in Tables IV-VI, once again there is no common winner. General trends are as follows. **(1)** *Grad-CAM produces the best fidelity scores, closely followed by DeepLIFT. CG's fidelity scores are much worse, indicating that both Grad-CAM and DeepLIFT identify more important nodes in regards to classification*. **(2)** *Grad-CAM has the highest contrastivity*, which is expected as it is class-discriminative, and higher contrastivity indicates that class-specific features highlighted by an interpretability method would differ significantly between classes. Both DeepLIFT and CG's contrastivity scores are much lower, and they are generally comparable with each other. **(3)** *DeepLIFT has the highest sparsity, closely followed by Grad-CAM. CG's sparsity scores are much worse*. This implies that CG produces many salient nodes (but of lower quality), whereas both DeepLIFT and Grad-CAM result in a fewer, yet highly important, salient nodes. **(4)** *Considering saliency generation time (Table VII), Grad-CAM is the most efficient*. This is because Grad-CAM requires one forward and one partial backward pass. *Efficiency-wise, Grad-CAM is closely followed by CG*, since CG requires one forward and one backward pass. *DeepLIFT is the most expensive among these three*, as DeepLIFT does two forward passes (due to input graph and reference graph) and one backward pass. Nevertheless, *all three interpretability methods generate saliency maps in less than 900 ms over our datasets*.

We exclude GraphSIG from Tables IV-VII, because the explainability features (significant subgraphs) were mined once during training, and they remain unchanged while predicting over test instances (graphs). In contrast, for our three interpretability methods, the salient nodes are generated for each

[5] **Fidelity** is computed as the decrease in accuracy (AUC-ROC) by occluding all nodes with attribution values greater than a threshold (0.5). Such nodes are referred to as the salient nodes. To compute fidelity, features of salient nodes are removed, while keeping the graph structure unchanged. Intuitively, occlusion of salient nodes identified via interpretability methods would decrease the classification performance significantly. Fidelity could be negative if accuracy improves after occluding salient nodes returned by an interpretability method. This happens more often if the classification result is poor, coupled with a lower quality interpretability method (e.g., CG). **Contrastivity** is the ratio of the Hamming distance d_H between binarized (i.e., based on identified salient nodes) saliency maps $\hat{m}_0$,$\hat{m}_1$ for positive and negative classes, normalized by the total number of salient nodes identified by either method, $\hat{m}_0 \cup \hat{m}_1$, i.e., $\frac{d_H(\hat{m}_0,\hat{m}_1)}{\hat{m}_0 \cup \hat{m}_1}$. Higher contrastivity indicates that class-specific features highlighted by an interpretability method should differ significantly between classes. **Sparsity** is one minus the number of identified salient nodes in either class ($\hat{m}_0 \cup \hat{m}_1$), divided by the total number of nodes n in the input graph, i.e., $1 - \frac{\hat{m}_0 \cup \hat{m}_1}{n}$. Sparsity measures localization of an interpretation, that is useful for larger graphs. Notice that fidelity can be increased by reporting many nodes as salient nodes, however the sparsity would be lower. Hence, there should be an appropriate trade-off between fidelity and sparsity for a better interpretability method.

TABLE VI
COMPARISON OF SPARSITY

		MUTAG	*NCI-H23*	*Tox21_AR*	*PTC_FR*	*Callgraph*
GCNN+GAP	CG	0.320	0.262	0.320	0.618	0.975
	Grad-CAM	**0.680**	**0.566**	0.610	**0.726**	0.966
	DeepLIFT	0.362	0.641	**0.746**	0.695	**0.995**
GCNN$_D$	CG	0.033	0.199	0.430	0.147	0.963
	Grad-CAM	0.610	0.531	0.519	**0.640**	0.574
	DeepLIFT	**0.886**	**0.574**	**0.567**	0.623	**0.994**
DGCNN	CG	0.720	0.623	0.618	0.615	0.984
	Grad-CAM	0.541	0.642	0.728	0.625	0.866
	DeepLIFT	**0.732**	**0.827**	**0.798**	**0.729**	**0.993**
DIFFPOOL	CG	0.273	0.368	0.538	0.325	0.933
	Grad-CAM	0.506	0.500	0.500	0.500	0.500
	DeepLIFT	**0.850**	**0.798**	**0.761**	**0.672**	**0.993**
DIFFPOOL$_D$	CG	0.049	0.132	0.398	0.256	0.972
	Grad-CAM	**0.571**	0.471	0.471	0.516	0.561
	DeepLIFT	0.350	**0.488**	**0.664**	**0.669**	**0.994**



(a) iso-1, CG (b) iso-1, Grad-CAM (c) iso-1, DeepLIFT

(d) iso-2, CG (e) iso-2, Grad-CAM (f) iso-2, DeepLIFT

Fig. 6. Visualization of two isomers' saliency maps (*MUTAG*, DGCNN). Best viewed in color.

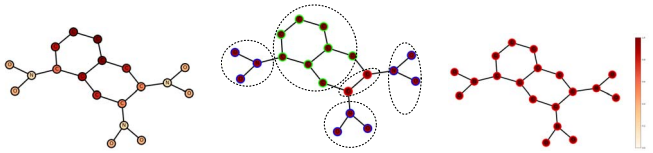
test instance based on its predicted class scores. This is a key difference of CG, Grad-CAM, and DeepLIFT, with the classic significant subgraph mining based approaches, since *the salient nodes generated by these interpretability methods are dependent on both training and test data*.

C. Interpretability: visualization performance

We next focus on visualizing saliency maps produced by interpretability methods, where red color indicates nodes that contribute positively to the activation of the target output, and blue color indicates nodes that have a negative effect. We present our results in Fig. 6 with two isomers (pair of graphs having same structure, but different node features) from *MUTAG*, which belong to two different classes. DGCNN is used as the underlying classifier. We observe that with CG, many salient nodes are reported for both isomers (Fig. a,d), but they are neither class-discriminative, nor they can highlight the difference between the two isomers. For example, the same "C" node has the highest attribution score in both classes according to CG (Fig. a,d), which is not very useful. For Grad-CAM, the salient nodes between two isomers are highly class-discriminative (Fig. b,e), indicating higher contrastivity of Grad-CAM. With DeepLIFT (Fig. c,f), we use the other isomer as the reference graph, thereby highlighting the difference between two isomers in their saliency maps. *Our visualization result illustrates that the use-cases of Grad-CAM and DeepLIFT could be different. Grad-CAM is suitable for class-discriminative visualization, whereas DeepLIFT is useful in identifying differences between two individual instances*.

TABLE VII
COMPARISON OF SALIENCY MAP GENERATION TIME

		MUTAG	*NCI-H23*	*Tox21_AR*	*PTC_FR*	*Callgraph*
GCNN+GAP	CG	**2.18 ms**	2.41 ms	**2.27 ms**	2.11 ms	34.08 ms
	Grad-CAM	2.47 ms	1.78 ms	2.30 ms	**2.08 ms**	**33.73 ms**
	DeepLIFT	3.90 ms	4.32 ms	3.85 ms	3.59 ms	62.08 ms
GCNN$_D$	CG	1.47 ms	1.85 ms	1.56 ms	1.66 ms	11.36 ms
	Grad-CAM	**1.44 ms**	**1.37 ms**	**1.34 ms**	**1.32 ms**	**7.53 ms**
	DeepLIFT	2.43 ms	3.08 ms	2.61 ms	2.47 ms	17.67 ms
DGCNN	CG	3.69 ms	4.20 ms	3.88 ms	3.94 ms	9.57 ms
	Grad-CAM	**3.67 ms**	**3.55 ms**	**3.23 ms**	**3.39 ms**	**8.21 ms**
	DeepLIFT	6.44 ms	7.70 ms	10.19 ms	6.53 ms	15.14 ms
DIFFPOOL	CG	4.16 ms	4.32 ms	4.39 ms	3.86 ms	88.72 ms
	Grad-CAM	**2.95 ms**	**3.22 ms**	**3.27 ms**	**3.13 ms**	**44.31 ms**
	DeepLIFT	7.02 ms	8.99 ms	11.77 ms	7.38 ms	134.16 ms
DIFFPOOL$_D$	CG	2.80 ms	3.04 ms	2.97 ms	2.89 ms	777.42 ms
	Grad-CAM	**2.47 ms**	**2.56 ms**	**2.66 ms**	**2.47 ms**	**342.40 ms**
	DeepLIFT	5.60 ms	6.74 ms	9.51 ms	5.67 ms	831.60 ms



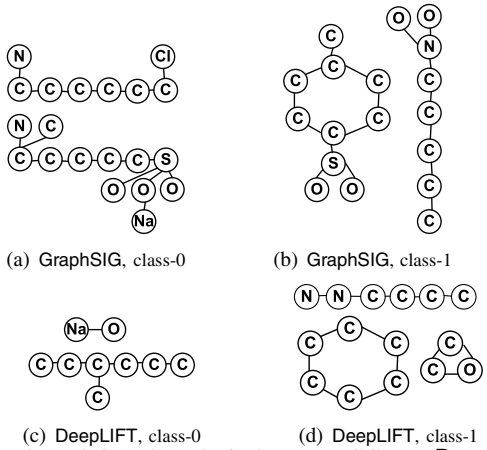
(a) input graph (b) DIFFPOOL's layer-1 (c) DIFFPOOL's layer-2

Fig. 7. DIFFPOOL's higher-layer clustering (*MUTAG*, Grad-CAM). Best viewed in color.

Saliency map with DIFFPOOL's higher-layer clustering. DIFFPOOL (and its variant DIFFPOOL$_D$) can identify node clustering at various granularity, where node cluster membership at a DIFFPOOL layer is determined by taking the argmax of its cluster assignment probabilities. Such hierarchical clustering, coupled with the interpretability methods, has the potential to correctly identify active subgraphs in the context of graphs classification. In Fig. 7, we present hierarchical node clusterings obtained at three different DIFFPOOL layers, and their corresponding attribution scores via Grad-CAM. We observe that *the active subgraphs (marked in dark red) from layer-1 are of modest size, and hence they could be useful in discovering active subgraphs that contribute the most in predicting the graph's class label*.

D. Interpretability: subgraph-based performance

To directly compare the results of deep learning based interpretability methods with that of significant subgraphs obtained via the classic GraphSig, *we compute frequent subgraphs in each class induced by the most active nodes (attribution score $>$ 0.3) identified via the interpretability methods*. We reckon that visualizing active subgraphs (as opposed to active individual nodes), which drive the classification decision of the GCNN, could be more useful. Among the subgraphs identified via GraphSig and DeepLIFT (as mentioned above), we present in Fig. 8 the largest ones for each class over *PTC_FR*. *The subgraphs via DeepLIFT are generally smaller than those by GraphSig, however they often contain important sub-structures of GraphSig's subgraphs*. For example, (a) "Na-O" and (b) chain-like sub-structures with "C" are preserved



(a) GraphSIG, class-0 (b) GraphSIG, class-1

(c) DeepLIFT, class-0 (d) DeepLIFT, class-1

Fig. 8. subgraph-based analysis between GCNN (DeepLIFT) and classic significant subgraph mining (GraphSIG), *PTC_FR*

in class-0 (Fig. a,c), while (c) chain-like sub-structures with "C", followed by "N" and (d) hexagon structures with "C" are maintained in class-1 (Fig. b,d). Which categories of subgraphs are more useful (larger ones vs. smaller ones preserving important sub-structures) is application specific, and it would be interesting to verify them with domain experts.

E. Discussion and recommendation

Table VIII summarizes recommendation levels of each interpretability method according to different performance metrics. The scale is from 1 to 4 stars, larger number of stars stands for higher recommendation. Clearly, there is no single winner.

(1) Considering various trade-offs, *our recommended interpretability tools for graph classification are DeepLIFT, Grad-CAM, in combination with recent (and more accurate) GCNNs: DGCNN, DIFFPOOL, and its variant* $\mathsf{DIFFPOOL}_D$. **(2)** Grad-CAM is useful for class-discriminative visualization, whereas DeepLIFT can identify differences between two individual instances. **(3)** Finally, DIFFPOOL and $\mathsf{DIFFPOOL}_D$ can assist visualizing node clustering at various granularity, and this has the potential to identify active subgraphs, e.g., active functional groups for biological networks classification.

VI. Conclusions

We investigated three backpropagation interpretability methods: CG, Grad-CAM, and DeepLIFT. We conducted a thorough experimental evaluation of them with three state-of-the-art GCNNs: GCNN+GAP, DGCNN, DIFFPOOL, and their variants: GCNN_D, $\mathsf{DIFFPOOL}_D$. We also compared their performance with the classic significant subgraph mining (e.g., GraphSig) and demonstrated their benefits both qualitatively and quantitatively. We discussed our recommendations such as: DeepLIFT and Grad-CAM, coupled with recent (and more accurate) GCNNs: DGCNN, DIFFPOOL, and its variant $\mathsf{DIFFPOOL}_D$, are the best-performing interpretability tools considering various trade-offs. We open sourced an end-to-end framework in which one can plug and play with different graph datasets, GCNNs, and backpropagation-based interpretability tools. In future, it would be interesting to test and model how high accuracy and fidelity could impact the user's trust in the underlying graph classification tool, as well as to conduct usability analysis due to higher contrastivity, sparsity, explanation subgraph size, and efficiency.

TABLE VIII
Summary and recommendation

method	fidelity	contrastivity	sparsity	running time	subgraph size
CG	★	★★	★	★★★	★★★
Grad-CAM	★★★★	★★★★	★★★	★★★★	★★★
DeepLIFT	★★★	★★	★★★★	★★	★★
GraphSig	—	—	—	★	★★★★

Acknowledgement. Arijit Khan is supported by MOE Tier1 and Tier2 grants RG117/19, MOE2019-T2-2-042.

References

[1] Our codebase: https://github.com/tsKenneth/interpretable-graph-classification.
[2] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, "Towards Better Understanding of Gradient-based Attribution Methods for Deep Neural Networks," ICLR, 2018.
[3] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral Networks and Locally Connected Networks on Graphs," ICLR, 2014.
[4] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso, "Machine Learning Interpretability: A Survey on Methods and Metrics," Electronics, vol. 8, no. 8, 2019.
[5] A. K. Debnath, d. C. R. Lopez, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with Molecular Orbital Energies and Hydrophobicity," J. Medicinal Chemistry, vol. 34, 1991.
[6] M. Du, N. Liu, and X. Hu, "Techniques for Interpretable Machine Learning", Commun. ACM, vol. 63, no. 1, 2020.
[7] D. Duvenaud, D. Maclaurin, J. A.-Iparraguirre, R. G.-Bombarelli, T. Hirzel, A. A.-Guzik, and R. P. Adams, "Convolutional Networks on Graphs for Learning Molecular Fingerprints," NeurIPS, 2015.
[8] M. Fan, X. Luo, J. Liu, M. Wang, C. Nong, Q. Zheng, and T. Liu, "Graph Embedding based Familial Analysis of Android Malware using Unsupervised Learning," ICSE, 2019.
[9] W. L. Hamilton, R. Ying, J. Leskovec, "Representation Learning on Graphs: Methods and Applications", IEEE Data Eng. Bull., vol. 40, no. 3, 2017.
[10] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri, "Querying Knowledge Graphs by Example Entity Tuples", IEEE Trans. Knowl. Data Eng., vol. 27, no. 10, 2015.
[11] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks", ICLR, 2017.
[12] N. Kokhlikyan, V. Miglani, M. Martin, E. Wang, J. Reynolds, A. Melnikov, N. Lunova, and O. Reblitz-Richardson, "PyTorch Captum,", https://github.com/pytorch/captum, 2019.
[13] M. Lin, Q. Chen, and S. Yan, "Network In Network", ICLR, 2014.
[14] Z. C. Lipton, "The Mythos of Model Interpretability", Commun. ACM, vol. 61, no. 10, 2018.
[15] P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann, "Explainability Methods for Graph Convolutional Neural Networks", CVPR, 2018.
[16] S. Ranu and A. K. Singh, "GraphSig: A Scalable Approach to Mining Significant Subgraphs in Large Graph Databases", ICDE, 2009.
[17] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization", ICCV, 2017.
[18] R. Sharan, S. Suthram, R. M. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. M. Karp, and T. Ideker, "Conserved Patterns of Protein Interaction in Multiple Species", PNAS, vol. 102, no. 6, 2005.
[19] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning Important Features Through Propagating Activation Differences", ICML, 2017.
[20] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR, 2013.
[21] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, "Statistical Evaluation of the Predictive Toxicology Challenge 2000-2001", Bioinform., vol. 19, no. 10, 2003.
[22] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks", ICLR, 2018.
[23] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph Kernels", J. Mach. Learn. Res., vol. 11, 2010.
[24] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. S. Pande, "MoleculeNet: A Benchmark for Molecular Machine Learning", CoRR, vol. abs/1703.00564, 2017.
[25] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining Significant Graph Patterns by Leap Search", SIGMOD, 2008.
[26] Y. Yan, J. Zhu, M. Duda, E. Solarz, C. S. Sripada, and D. Koutra, "GroupINN: Grouping-based Interpretable Neural Network for Classification of Limited, Noisy Brain Data", KDD, 2019.
[27] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "GNNExplainer: Generating Explanations for Graph Neural Networks", NeurIPS, 2019.
[28] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical Graph Representation Learning with Differentiable Pooling", NeurIPS, 2018.
[29] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An End-to-End Deep Learning Architecture for Graph Classification", AAAI, 2018.