Discovering Discriminative Nodes for Classification with Deep Graph Convolutional Methods

Liana-Daniela Palcu[1(✉)], Marius Supuran[1(✉)], Camelia Lemnaru[1(✉)], Mihaela Dinsoreanu[1(✉)], Rodica Potolea[1(✉)], and Raul Cristian Muresan[2(✉)]

[1] Computer Science Department, Technical University of Cluj Napoca, Cluj-Napoca, Romania
ldpalcu@gmail.com, marius.supuran@yahoo.com, camelia.lemnaru@cs.utcluj.ro, mihaela.dinsoreanu@cs.utcluj.ro, rodica.potolea@cs.utcluj.ro
[2] Transylvanian Institute of Neuroscience, Cluj-Napoca, Romania
raul.muresan@gmail.com

Abstract. The interpretability of Graph Convolutional Neural Networks is significantly more challenging than for image based convolutional networks, because graphs do not exhibit clear spatial relations between their nodes (like images do). In this paper we propose an approach for estimating the discriminative power of graph nodes from the model learned by a deep graph convolutional method. To do this, we adapt the Grad-CAM algorithm by replacing the part which heavily relies on the 2D spatial relation of pixels in an image, with an estimate of the node importance by its appearance count in the result of the Grad-CAM. Our strategy was initially defined for a real-world problem with relevant domain-specific assumptions; thus, we additionally propose a methodology for systematically generating artificial data, with similar properties as the real-world data, to assess the generality of the learning process and interpretation method. The results obtained on the artificial data suggest that the proposed method is able to identify informative nodes for classification from the deep convolutional models.

1 Context and Motivation

Model interpretability can be important for several reasons: first, it builds trust and confidence in machine learning models when applied to sensible problems (e.g. medical diagnosis and prognosis, terrorism prediction, credit assessment, etc). In such domains, if the model can explain its decisions, it is easier to asses its fairness (does not discriminate against protected groups), privacy-compliance, robustness and the ability to identify causality [1]. Second, it is a potentially powerful tool for generating new domain knowledge in "difficult" domains, such as neuroscience. For example, interpretable models could provide new insights into understanding the effect of alcohol on the brain. In the same line of argument, if the performance of the model beats human performance (e.g. chess, AlphaGO), machine-driven instruction could be used to help humans improve their

M. Ceci et al. (Eds.): NFMCP 2019, LNAI 11948, pp. 67–82, 2020.
https://doi.org/10.1007/978-3-030-48861-1_5

skills. Last, but not least, interpretability can be thought of as a useful tool for understanding and correcting model errors.

In general, we are faced with a trade-off between performance and interpretability. Graph classification is normally a domain which requires the application of complex learning models, such as deep neural networks, which are not interpretable by nature. Several relevant attempts have been made to interpret complex models post-hoc (briefly reviewed in Sect. 2). However, most approaches focus on tabular inputs, or inputs with a known, structured or hierarchical relation between the elements (e.g. the 2D spatial relation between pixels in an image or the 1D temporal relation between words in a sentence). For graph data, we do not have such spatial or temporal semantics to work with, which makes interpreting any model built on such data even more difficult.

The starting point of our research is rooted in neuroscience: trying to identify neurons in the brain which are most affected by alcohol consumption, and which separate between non-alcohol and alcohol affected brain states. Graph/network analysis methods that are applied to this problem need to produce interpretable models, because their aim is to help understand brain behavior. The starting hypothesis is that there is a small subset of neurons whose connection weights are affected by alcohol, and those neurons are responsible for changing the overall behavior and response to alcohol. While initially driven exclusively by this hypothesis, we assess that the methods investigated are applicable to any graph classification problem. We propose a method for graph data classification and model interpretation which generates class-specific relevance heatmaps for the nodes in the graph by applying a modified version of Grad-CAM [2] – an interpretability method initially designed for image CNNs.

The rest of the paper is organized as follows: Sect. 2 overviews the relevant interpretation strategies from literature. Section 3 presents the proposed method, which is evaluated in Sect. 4. The last section contains concluding remarks.

2 Related Work

Some classification models (e.g. decision trees, logistic regression) are inherently interpretable. For the others, which have a black box behavior, interpretability methods can be divided into model-agnostic and model-specific [3]. The first category encompasses methods which can be applied to any classification model, and generally focus either on explaining a model by computing feature relevance scores – globally [4–6], or at instance level [7,8] – or try to build a global or a local interpretable surrogate model, such as LIME [9].

In the context of Convolutional Neural Network (CNN) models, agnostic interpretability methods do not exploit that such models learn new features and concepts in their hidden layers and are computationally inefficient, because they do not use gradient values [3]. For interpreting CNN models, recent works in literature focus either on *perturbing the input (or hidden network layers) and observing the corresponding changes* – generally computationally intensive and can show instability to surprise artifacts (a line of research closely related to

adversarial attacks on CNN architectures) – or *leveraging gradient values to infer feature importance* – computationally efficient, but poses challenges when propagating gradients back through non-linear and re-normalization layers.

[10] proposes the use of deconvolution to identify which part of an image most strongly activates a specific unit in the network: typically, all neurons except one are set to zero in the high level feature map corresponding to the layer of that unit, and we perform a backward pass through the CNN down to the input layer. The resulting reconstructed image shows which part of the input image most strongly activates that unit. Class specific saliency maps [11] are generated by computing the gradient of the class score with respect to the input image. The intuition is to use the gradients to identify input regions that cause the most change in the output. The main difference between the last two techniques is how gradients are passed through non-linear layers such as the Rectified Linear Unit (ReLU): in [11] the gradients of neurons with *negative input* are suppressed, while in [10] the gradients of neurons with incoming *negative gradients* are suppressed. Guided backpropagation [12] combines both strategies, by suppressing the flow of gradients of both negative input and negative gradient neurons.

Class Activation Maps (CAMs) [13] identify the image regions used by a CNN to discriminate between different categories. It can only be applied to a limited set of CNNs and it alters the architecture by adding at the end a Global Average Pooling layer (GAP) and then a fully-connected layer. This is done to preserve the localization ability of any network, which is lost using fully connected layers. However, this change could affect the performance of the model.

A significant shortcoming of the methods presented above is that they do not address re-normalization layers, such as max-pooling. Propagating gradients back through such a layer can be problematic since the functions used are not generally differentiable. Grad-CAM [2] tries to circumvent this problem by relying on the activation maps of the final convolutional layer to build a down-sampled relevance map (heatmap) of the input pixels, which is then upsampled to obtain a coarse relevance heatmap of the input.

3 Interpreting Graph Convolutional Network Models with Grad-CAM

In the up-sampling step, Grad-CAM performs a bi-linear interpolation between neighboring pixels, which is computationally efficient and produces good results for images, but cannot be directly applied to graphs. Consequently, we modify Grad-CAM to address this and allow the generation of class-relevant heatmaps containing estimates of the each node's importance to a specific class. We integrate our solution with the Deep Graph Convolutional Neural Network model (DGCNN) [14]. As the ultimate goal of the strategy is the identification of the relevant nodes in the classification decision, we propose a preprocessing step which consists of removing potentially non-informative edges.

3.1 Graph Sparsification

Sparsification is motivated by the assumption that not all edges are informative, and that small weight edges represent noise. Consequently, sparsification eliminates a certain amount of small weight edges, with the hope of improving classification accuracy and model interpretability. Let $G(V, E)$ be a complete weighted graph, where V represents the set of nodes and E represents the set of edges, each edge being given by $e_i(u, v, w_i)$, with $u, v \in V$ and $w_i \in \mathbb{R}^+$. Let Sum_G be the sum of all the weights from the graph. We sort the edges in descending order by weight, $<e_1, e_2, ..., e_n>$, where $w_1 > w_2 > ... > w_n$, and considering this order we keep only those edges $<e_1, e_2, ..., e_m>$ that have the sum of weights smaller than a certain threshold, computed as a percentage of the total sum of weights:

$$Sum_{G'} = \sum_{i=1}^{m} w_i < p\% * Sum_G = p\% * \sum_{i=1}^{n} w_i,\ p \in [0, 100],\ m <= n = |E|, \quad (1)$$

3.2 Deep Graph Convolutional Neural Networks

End-to-end deep learning architectures, such as the Deep Graph Convolutional Neural Network (DGCNN) [14], take as input graphs of arbitrary structure, (G, y), where y represents the label of the graph, and build a graph classification model by applying end-to-end gradient based training. As opposed to methods which use graph embeddings to transform graphs into tensor data that can be then classified via traditional machine learning algorithms, end-to-end methods solve a single joint optimization problem, which includes both feature learning and classification. This gives them the potential to produce better classification outcomes than the decoupled, embedding-based methods, but increases the complexity of the problem and thus, the computational effort needed to solve it. The DGCNN architecture is composed of three sequential parts. The first part extracts useful features for every node by using Graph Convolutional Networks (GCN). The extracted features characterize the graph topology and based on them, in the middle part, due to the use of the SortPooling layer, an ordering of graph nodes is defined. In the last part, the ordered sequence of nodes are introduced into a 1D convolutional neural network and then into dense layers with the purpose of learning a classification function. For a more in-depth description of the specific principles used by DGCNN, we refer the reader to [14].

3.3 DGCNN Interpretability

The next step after classifying graphs is to find the nodes which best discriminate between classes, in the attempt to interpret the model. We adapted Grad-CAM to graph classification models by starting from the premise that the graph nodes ordering resulting after the SortPooling layer in DGCNN encodes specific structural information (based on the relative structural relevance of the nodes within

the graph), similar – in a way – to how pixels in neighboring regions of an image are correlated. In the following steps we detail our solution.

Let $F_1, F_2, ..., F_n$ be the feature maps in the final convolutional layer and S_c the score of the target class c. The corresponding gradients $(w_1, w_2, ..., w_n)$ are computed by using the formula:

$$w_i = \frac{\partial S_c}{\partial F} | F_i, \forall i = 1, .., n \tag{2}$$

These gradients are global-averaged pooled in order to obtain a weight of the importance of a feature map F_i for a target class c. By multiplying the weights w_i with their corresponding feature maps we obtain the weighted activations:

$$A_i = w_i * F_i, \forall i = 1, .., n. \tag{3}$$

The next step is to sum all the activations of the feature maps and apply the ReLU function, the result being a downsampled feature-importance array:

$$H = ReLU(\sum_{k=1}^{n} A_k) \tag{4}$$

We don't upsample H as it is done for images, we go back through the architecture to find an approximation of a group of nodes that are good predictors for a target class. The part of the architecture were we apply our reverse process is the CNN part, as depicted in Fig. 1. In this example, and even in the architecture which we used, this part is composed of two 1D convolutional layers and a MaxPooling layer. The first 1D convolutional layer combines the resulted features of every node from the SortPooling layer into one feature. The dimensions of the ordered array does not change after this layer. Next, a MaxPooling operation is applied and, depending on the values of the hyperparameters, kernel and step, the dimensions of the previous array changes. A second convolutional layer is applied, changing the dimensions of the array again. We apply Grad-CAM on the result of the previous convolutional layer. Therefore, we can associate an element from H with a group of nodes, FG, by going back trough the architecture. In the illustrated example, $FG(1)$ (Final Group 1) is represented by two previous groups of nodes, where G1 (Group 1) contains the nodes 3 and 6, and G2 (Group2) contains the nodes 2 and 1. In the end, $FG(1)$ points to the nodes 3, 6, 2, 1. H consists of values between 0 (meaning that the group of nodes is not important in classifying the target class) and 1 (meaning that the group of nodes is a very good predictor for the target class). For every node, v_i, we discretize its importance into several bins, by defining an importance array, c_i, where the indices give us decimal intervals from H. For instance, index 0 represents the values between 0 and 0.1, index 1 represents values between 0.1 and 0.2, and so on. C is defined as a frequency matrix where the row c_i is the importance frequency array for the node v_i. This matrix is obtained by applying the *Importance frequency algorithm* (shown below) to every computed H. Based on the C matrix we then generate the interpretability heatmaps (Sect. 4.3) to visualize the discriminative nodes.

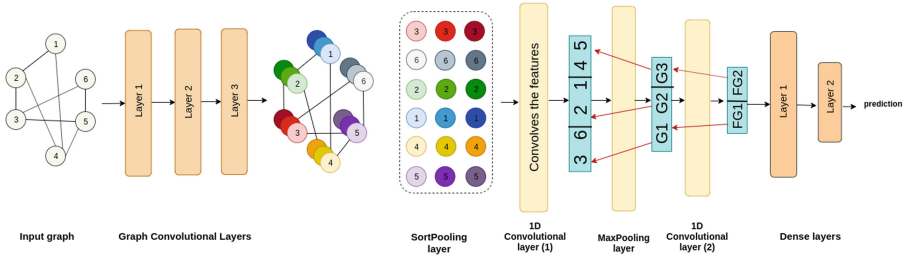


Fig. 1. DGCNN + Grad-CAM

Algorithm 1: Importance frequency algorithm

Input : H - importance array, FG - a list of lists of nodes
Output : C - importance frequency matrix
Initialize: $C(i) \leftarrow 0, \forall\ i = 1, ..., n, where\ n = |V|$
foreach *element* $h \in H$ **do**
  foreach *node* $v \in FG(h)$ **do**
    $idx \leftarrow \lfloor h * 10 \rfloor$;
    $C(v)(idx) \leftarrow C(v)(idx) + 1$
  end
end

4 Experimental Evaluation and Results

The domain specific problem we started off from consisted of graphs representing brain functional networks in two different physiological states. Though the classification accuracy obtained on that data was good, and the heatmaps obtained allowed for reaching a certain understanding of the generated models, we chose to validate the interpretability method more reliably, by generating artificial datasets in which the relation between the nodes in the graph is known in advance.

4.1 Data Generation

Validating interpretability methods for graph classification models is not straightforward, since if we employ real data it might not even be clear what the model should be learning. Because the interpretability model we propose tries to highlight class-relevant nodes, failing to do so may be caused by flaws in the interpretability model itself, but also by the fact that the classification model does not actually learn what it should. To remove the second factor from the analysis (since it is not relevant for the validation of the interpretability method), we turn to synthetic data to analyze the strengths and weaknesses of the proposed interpretation strategy. In generating the data, we followed three main objectives/hypotheses (further detailed in four data generation scenarios):

1. Classification performance on a random class distribution problem should be close to the 50% baseline – addressed by scenario $S1$ below; analyze what the interpretability heatmaps indicate in this situation.
2. Evaluate the robustness of the method to mild graph topologies and distributions which try to mimic the original, real-world problem we started from. This is addressed by generation scenarios $S2$ and $S3$ below.
3. Evaluate the robustness of the method to various complexities inherent in data, which normally affect performance, such as: (i) imbalance, (ii) overlap, (iii) noise and also combinations of these complexities (as most traditional machine learning techniques fail to handle well this aspect). Scenario $S4$ below considers two of these complexities.

The objectives above are materialized in the following 4 generation scenarios:

1. **Random classification problem (S1-Random)**, in which the graphs for both classes are very similar. We expect that the resulting model has very poor performance in this case (close to 50%), and the interpretability heatmaps to show no emerging pattern.
2. **Well separable and interpretable classification problem (S2-Easy)**, in which we select a subset of nodes to drive an almost perfect separation between classes. For this scenario, we expect classification accuracy to be close to 100% and the model to be able to learn which are the important nodes - which should be visible in the resulting interpretability heatmaps.
3. **Well separable, partially interpretable classification problem (S3-Moderate)**, in which we try to give more importance in separation to a subset of nodes, but this importance is not as straightforward as in the previous scenario. In this case, we also expect a very good classification accuracy, and the interpretability heatmaps should be able to indicate (at least partially) the important nodes.
4. **Partially separable, partially interpretable classification problem (S4-Hard)**, in which we attempt to make the data more difficult to separate, by introducing two challenges: imbalance and overlap.

The rest of this section describes the data generation processes for each of the above scenarios. All datasets generated contain 500 synthetic complete weighted, undirected graphs, each having 85 nodes. The graphs belong to 2 different classes - State1 and State2 - the class labels being uniformly distributed (except for S4-Hard, where we introduce imbalanced class distributions). Each node has the same labelling in all the graphs. The weights of the edges are numbers in the $[0, 1]$ range.

For **S1-Random**, the edge weights are drawn randomly from the same distribution for both classes, $\mathcal{N}_1(\mu_1, \sigma_1^2)$. As mentioned above, this should yield around 50% classification accuracy and the resulting heatmaps should not indicate any relevant nodes. For scenario **S2-Easy**, the graphs belonging to the first class are generated as for **S1-Random**; for the graphs belonging to the second class, we select a subset of k nodes for which we use a different edge weight distribution, $\mathcal{N}_2(\mu_2, \sigma_2^2)$. For the rest of the edge weights, we use the initial

distribution, $\mathcal{N}_1(\mu_1, \sigma_1^2)$ - this should yield a (very) well separable classification problem. For this scenario we experimented with two different settings: one in which the weights of the k nodes in the separate community in *State2* were weaker than for the rest of the graph ($S2.1$), and one in which they were stronger ($S2.2$). The reason for this is to observe whether sparsification can affect class separability and model interpretability, since sparsification removes the smaller weight edges (thus it might remove relevant edges in $S2.1$). With **S3-Moderate** we tried to generate graphs that were separable by a well known network metric - the *betweeness centrality* - and see whether the model is able to learn those characteristics. More specifically, for the graphs belonging to the first class, we again generate complete, weighted graphs, drawing the weights randomly from $\mathcal{N}_1(\mu_1, \sigma_1^2)$. For the graphs belonging to the second class, we select a subset of k nodes and generate the weights of the edges connecting these nodes by drawing randomly from $\mathcal{N}_1(\mu_1, \sigma_1^2)$. Then, the rest of the nodes are "split" uniformly at random among these hub nodes. We thus create separated communities, within each community the edge weights being drawn randomly from $\mathcal{N}_2(\mu_2, \sigma_2^2)$. In a last generation step, we connect the nodes belonging to different communities (except for the hub nodes) by very weak connections, drawing their weights from $\mathcal{N}_3(\mu_3, \sigma_3^2)$. In **S4-Hard** we generate three different datasets. We keep the generation strategy from *S3-Moderate*, and try to make the classification problems harder by introducing first class imbalance, then class overlap. Dataset $S4.1$ was generated with an imbalance ratio of approximately 10, the second class being the minority class. For dataset $S4.2$, we employ a balanced class distribution but change the means of the three distributions used to generate the edge weights such as to make them overlap more. Finally, $S4.3$ was generated by applying jointly the strategies from $S4.1$ and $S4.2$.

The specific parameters for the distributions used are presented in Table 1. For k we experimented with three different values: 8, 42 and 77 for *S2-Easy*, and $k = 8$ for *S3-Moderate* and *S4-Hard*.

Table 1. Distributions used for data generation

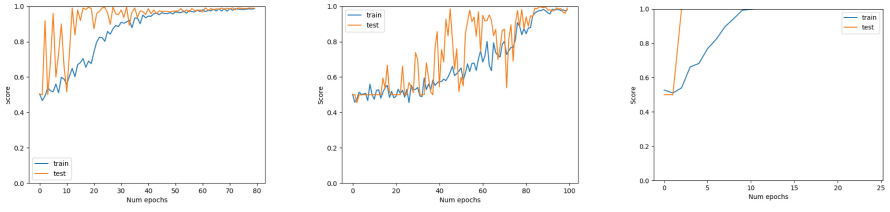
*	$\mathcal{N}_1$		$\mathcal{N}_2$		$\mathcal{N}_3$	
	μ_1	σ_1	μ_2	σ_2	μ_3	σ_3
S1-Random	0.5	0.25	n.a.	n.a.	n.a.	n.a.
S2.1-Easy (weaker)	0.7	0.1	0.5	0.1	n.a.	n.a.
S2.2-Easy (stronger)	0.5	0.1	0.7	0.1	n.a.	n.a.
S3-Moderate	0.7	0.1	0.5	0.1	0.2	0.1
S4.1-Hard (imbalance)	0.7	0.1	0.5	0.1	0.2	0.1
S4.2-Hard (overlap)	0.6	0.1	0.5	0.1	0.4	0.1
S4.3-Hard (imb. + overlap)	0.6	0.1	0.5	0.1	0.4	0.1

4.2 Classification Performance Evaluation

The classification task was performed using the network structure for DGCNN as presented in [14], applied to input graphs sparsified to maintain a certain amount of edges, as specified by $p\%$. We repeated each experiment 10 times, using in each evaluation 80% of the data for training and the remaining 20% as validation (test) set. For setting $p\%$, we experimented with several options, from maintaining all edges (i.e. $p = 100\%$) down to keeping the strongest edges that make up for 50% of the total weights.

As expected, for *S1-Random*, the trained models learn to predict one of the classes, reaching an accuracy of around 50% (e.g. the average accuracy of the final model over the 10 runs for $p = 70\%$ sparsification threshold was 50.1%). For all the other scenarios, all models, in all runs, eventually converge to a 100% accuracy on the validation set. What differs is the speed of convergence and the variability of the accuracy on the validation set.

For example, if we compare the training behavior of the models in *S2-Easy* and *S3-Moderate* - see Fig. 2) - we observe that the latter converge faster, and with less variability, which might indicate that the models find these datasets easier to learn, contrary to our initial assumptions. A potential motivation for this can be found in the effect of sparsification. For *S3-Moderate* data, for the graphs belonging to *State2* we expect sparsification to remove the weak inter-community edges (i.e. the ones generated with $\mathcal{N}_3(\mu_3, \sigma_3^2)$). In contrast, for the graphs belonging to *State2* in *S2-Easy*, sparsification might remove edges from both outside and inside the community formed of the k nodes (with higher probability for the edges generated with the distribution having the lower mean); what is important to note here is that by removing from both outside and inside the community, the problem might become more difficult to learn. Within the same scenario, we observed that keeping more edges in the initial graphs (by increasing $p\%$) makes the models converge more slowly (i.e. in later epochs), which was expected.

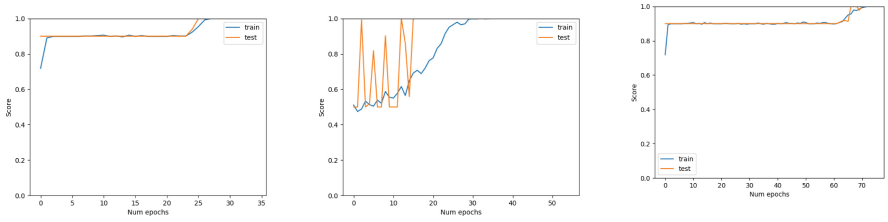


(a) S2.1 (b) S2.2 (c) S3

Fig. 2. Comparison of learning curves for *S2-Easy* and *S3-Moderate*, $k = 8$, $p = 70\%$ (Score = Average Accuracy)

In *S4-Hard* we find that the models have different learning patterns according to the complexities added to the data (imbalance and/or overlap). As illustrated

in Fig. 3a, the model only starts to learn something meaningful around epoch 25, when training accuracy starts to increase from 90% towards 100% (before that epoch, the model always predicted class *State1*, also reflected in the value of the validation set accuracy). We observe a similar behavior in Fig. 3c, only this increase appears later in training, due to the overlap also being a data complexity that the model has to overcome. Comparing with the behavior in *S3-Moderate* (see Fig. 2c), we find that both imbalance and overlap make the models learn more slowly, and overlap induces more variability in the learning process (which can be seen in the validation/test set accuracy).



(a) S4.1 (imbalance) (b) S4.2 (overlap) (c) S4.3 (imbalance + overlap)

Fig. 3. Comparison of learning curves for *S4-Hard*, $k = 8$, $p = 70\%$ (Score = Average Accuracy)

4.3 Interpretability Heatmaps

In order to visualize the discriminative nodes we created a heatmap where the horizontal axis (Oy) represents the nodes of the graph, G, and the vertical axis (Ox) represents the interval [0, 1], the importance of a node in classification (the values from H). The interval [0, 1] is split into 10 bins with a step of 0.1: the values between 0 and 0.1 are removed for better visualization, while we allocate an extra bin for values exactly equal to 1. The color represents the difference between the importance frequency matrix of $State1$, C_1, respectively $State2$, C_2 ($C1 - C2$), the difference being computed per decimal interval. C contains values which indicate how many times a node takes a value from H within a decimal interval. The top of the heatmap is associated with high values from H (for example, 1), while the bottom of the heatmap with low values from H. Therefore, for each node we can visualize its importance in the classification as follows: if the red color appears on the top of the image, and the blue color on the bottom of the image, it means that the node is a good predictor for $State1$; if we have the blue color on the top of the image and the red color on the bottom, it means that the node is very important in classifying $State2$; the green colour shows that the node does not have discriminative power in the model; if red or blue colors appear emphasized only in the middle of the heatmap, it might

indicate that our problem is difficult to learn, the difference between classes being less noticeable. We created an average heatmap across the 10 folds in order to capture the strongest common features of the models resulting from different evaluation folds.

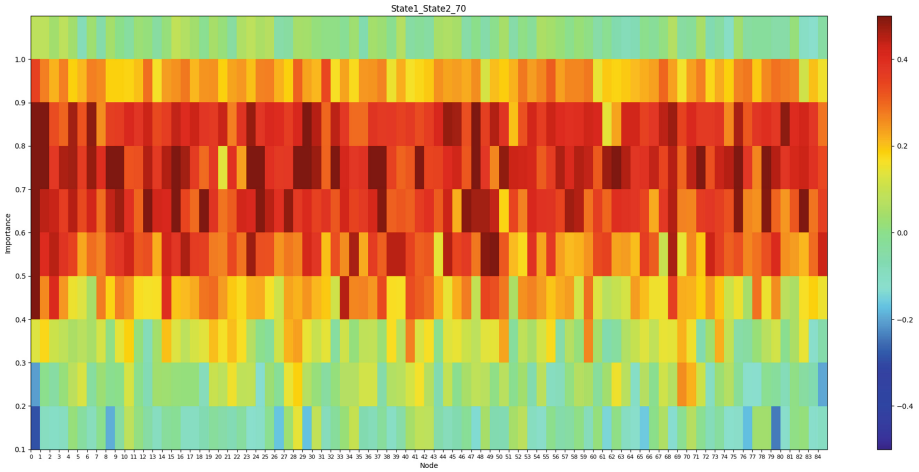
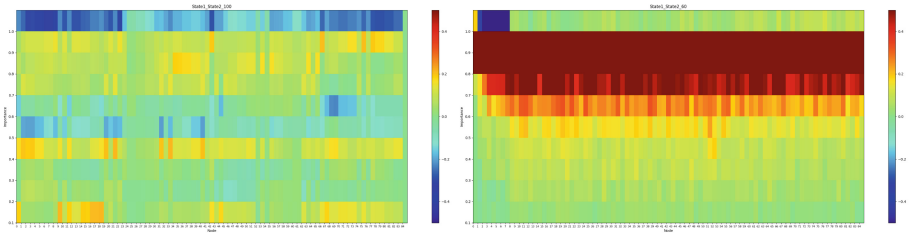


Fig. 4. The heatmap for *S1-Random* data generation strategy where $p = 70\%$.

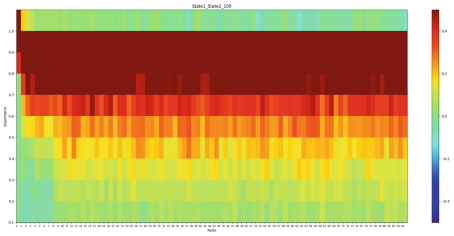
Figure 4 illustrates the average heatmap for the models learned for *S1-Random*. The model predicts any graph in the test/validation set as belonging to the *State1* class; the heatmap indicates that all nodes in the graph are relevant for predicting that class, which is to be expected.

For *S2.1-Easy (weaker)* data generation strategy we performed experiments and computed heatmaps for the following sparsification percentages: $p = 100\%$, $p = 70\%$ and $p = 50\%$. The purpose of this experiment was to highlight the k nodes whose edge weights were generated using a different distribution in *State*2. We always choose the k nodes to be the first in the graph (i.e. the leftmost 8 columns of the heatmap in Fig. 5). In Fig. 5a, we notice that if we keep all the edges no clear patterns emerge, because the information is actually distributed across the nodes. But if we sparsify the graphs using a percentage $p = 70\%$, Fig. 5b shows how our classifier distinguishes between classes by highlighting the 8 nodes that are good predictors for *State*2 (and the model performance is almost the same). If we sparsify more, $p = 50\%$, Fig. 5c does not indicate clear patterns because through sparsification the nodes are losing their importance (for example, the discriminative edges are eliminated).

In a next experiment, we modified k, the number of nodes for which we employed a different distribution for generating the edge weights (for the graphs belonging to the second class). The results for $p = 70\%$ can be visualized in Fig. 6. Figure 6a illustrates a clear difference between the discriminative nodes

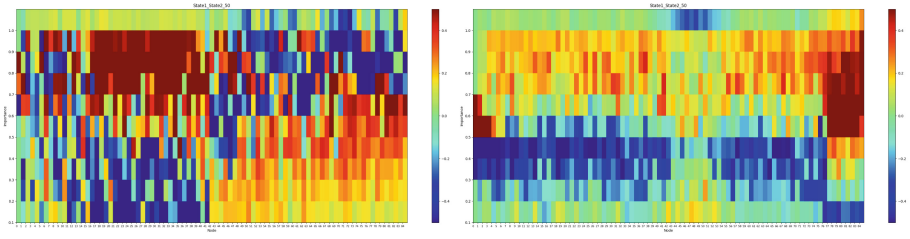


(a) The heatmap for $p = 100\%$ (b) The heatmap for $p = 70\%$



(c) The heatmap for $p = 50\%$

Fig. 5. The heatmaps for *S2.1-Easy (weaker)* data generation strategy for each sparsification percentage considered, where $k = 8$.



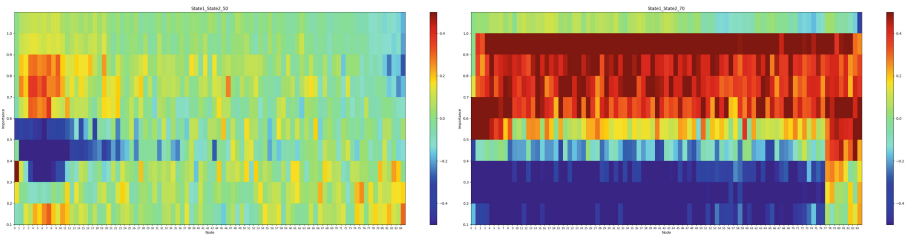
(a) The heatmap for $k = 42$ (b) The heatmap for $k = 77$

Fig. 6. The heatmaps for *S2.1-Easy (weaker)* data generation strategy, at $p = 70\%$.

for *State*1 (left part), and the good predictors for *State*2 (right part). Theoretically, by increasing k, we should have more discriminative nodes for one state. Figure 6b shows the opposite: actually the nodes (left part) whose edges weights have not been generated from another distribution are the most important ones for *State*1. Also, we can notice that there are fewer good predictors (the nodes from the middle of the heatmap) for *State*2 than in the previous case when $k = 42$. In the case of *S2.2-Easy (stronger)*, the first 8 modified nodes are more important in classifying *State*1 rather than *State*2 as it is shown in Fig. 7a, while in Fig. 7b all nodes are relevant in classifying *State*1.

In *S3-Moderate* the first 8 nodes were selected to be the hubs in *State*2. As Fig. 8 shows, only a part of them are highlighted as being important in classifying *State*2.

Even though *S4.1-Hard* represents a class imbalance problem, the same patterns as in *S3-Moderate* emerge in Fig. 9a, only the colors are less intense, which might indicate that the model is less certain in how the two classes separate. A similar phenomenon can be observed when the two classes overlap more, in *S4.2-Hard* (Fig. 9b), where the strong shades of blue and red appear more towards the middle bins (as opposed to the top or bottom of the heatmap - as for models which converged faster and are - intuitively - more confident in their separation). An interesting phenomenon can be observed for *S4.3-Hard* (Fig. 9c), where, as expected, the emphasized patterns appear in the middle of the heatmap, but the heatmap is flipped (blue appears more on the top of the heatmap, while red more on the bottom part).



(a) The heatmap for $k = 8$ (b) The heatmap for $k = 77$

Fig. 7. The heatmaps for *S2.2-Easy (stronger)* data generation strategy for $k = 8$, respectively $k = 77$, where $p = 70\%$

5 Discussion

The interpretability method proposed in this paper attempts to extract information about the importance of graph nodes in achieving class separation for deep graph convolutional models. The evaluation attempted to assess the validity of the method on several classification tasks for which - intuitively - we know what to expect from the models. A first important observation is that sparsification affects the outcome of the interpretability method, and this is because it affects how the underlying classification model learns to separate between the classes. When the information is dense (i.e. we keep all graph edges), individual nodes matter less in learning how to separate between the classes - which is to be expected. Naturally, the "right" amount of sparsification is highly dependent on the problem, and - even if not observed in the current evaluations - sparsification affects not only interpretability, but also the classification performance. Consequently, a future step is to study these interactions more systematically.

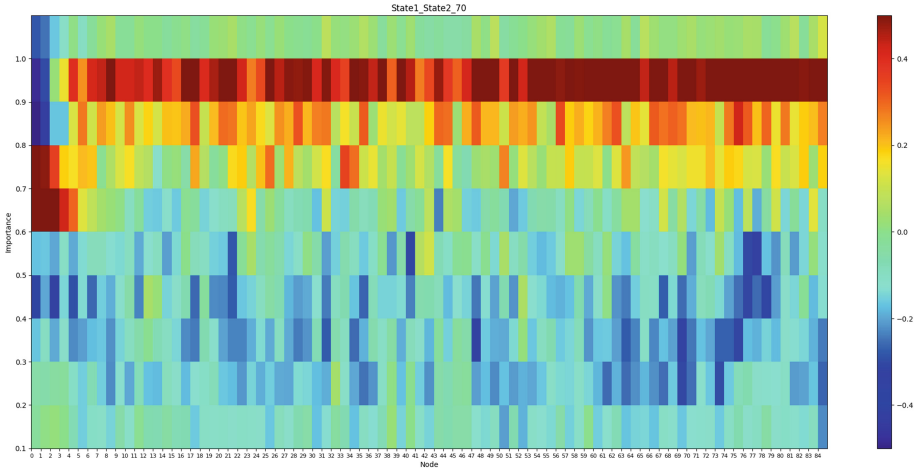
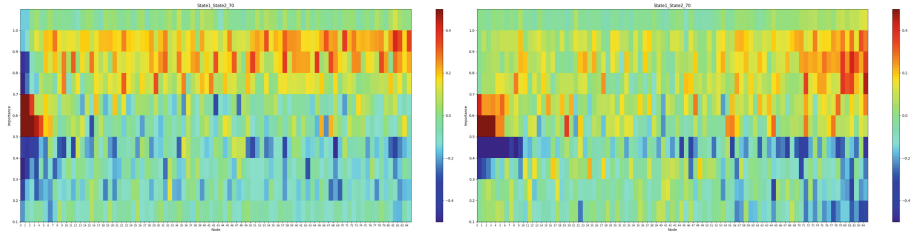
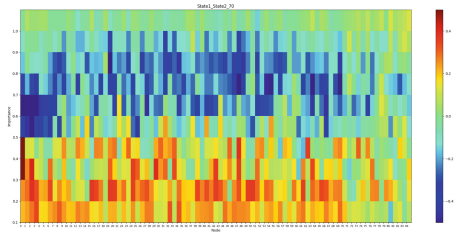


Fig. 8. The heatmap for *S3-Moderate* data generation strategy where $p = 70\%$.



(a) The heatmap for *S4.1-Hard (imbalance)* (b) The heatmap for *S4.2-Hard (overlap)*



(c) The heatmap for *S4.3-Hard (imb + overlap)*

Fig. 9. The heatmaps for *S.4-Hard* data generation strategy where $p = 70\%$.

By comparing the heatmaps for *S3-Moderate* and *S4-Hard*, and considering also how the corresponding models converge, we believe that the heatmaps may capture also the confidence of the model's predictions. However, this phenomenon needs to be studied further, especially for classification problems which are not perfectly separable.

The proposed modification to Grad-CAM performs a very rough approximation to compute graph node relevance. We are currently exploring more accurate alternatives for doing this (such as adapting the deconvolution method initially proposed for the interpretation of image convolutional models).

6 Conclusion

Interpretability is – in many application domains – crucial towards gaining acceptance for machine learning models. Graph convolutional models add an extra layer of difficulty for interpretability methods, because graphs do not exhibit clear spatial relations between their nodes (like images do).

In this paper we propose a method for graph classification and model interpretation, which combines DGCNN with a modified Grad-CAM algorithm, to obtain heatmaps representing each node's relevance to the classification of a specific graph. We alter the Grad-CAM algorithm to apply only operations which do not assume a specific locality for nodes. We evaluate our method on synthetic datasets which were generated to emulate a real dataset representing brain functional networks in different physiological states. These functional networks are represented by complete, weighted graphs that need to be sparsified. The resulting heatmaps are generally able to identify the nodes which we intended to be relevant for the identification of a specific class. Interrestingly, we believe they manage to also capture some degree of "uncertainty" associated to the predictions of the model, but this aspect needs further investigation, together with the effect of sparsification on the resulting models and heatmaps.

Acknowledgments. This work was supported by a grant from the Romanian National Authority for Scientific Research and Innovation, CNCS-UEFISCDI (project number COFUND-NEURON-NMDAR-PSY), a grant by the European Union's Horizon 2020 research and innovation program – grant agreement no. 668863-SyBil-AA, and a National Science Foundation grant NSF-IOS-1656830 funded by the US Government.

References

1. Doshi-Velez, F., Kim, B.: Towards a rigorous science of interpretable machine learning. arXiv e-prints, February 2017
2. Selvaraju, R.R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., Batra, D.: Grad-CAM: why did you say that? Visual explanations from deep networks via gradient-based localization. CoRR abs/1610.02391 (2016)
3. Molnar, C.: Interpretable machine learning (2019). https://christophm.github.io/interpretable-ml-book/
4. Greenwell, B.M., Boehmke, B.C., McCarthy, A.J.: A simple and effective model-based variable importance measure (2018)
5. Zhao, Q., Hastie, T.: Causal interpretations of black-box models (2019)
6. Fisher, A., Rudin, C., Dominici, F.: All models are wrong but many are useful: variable importance for black-box, proprietary, or misspecified prediction models, using model class reliance. arXiv e-prints, January 2018. arXiv:1801.01489

7. Goldstein, A., Kapelner, A., Bleich, J., Pitkin, E.: Peeking inside the black box: visualizing statistical learning with plots of individual conditional expectation. J. Comput. Graph. Stat. **24**(1), 44–65 (2015)
8. Štrumbelj, E., Kononenko, I.: Explaining prediction models and individual predictions with feature contributions. Knowl. Inf. Syst. **41**(3), 647–665 (2014)
9. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why should I trust you?": explaining the predictions of any classifier. CoRR abs/1602.04938 (2016)
10. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. CoRR abs/1311.2901 (2013)
11. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: visualising image classification models and saliency maps. CoRR abs/1312.6034 (2013)
12. Springenberg, J., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: the all convolutional net. In: ICLR (Workshop Track) (2015)
13. Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., Torralba, A.: Learning deep features for discriminative localization. In: CVPR (2016)
14. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: AAAI, pp. 4438–4445 (2018)