General and Interpretable TCP attack classification based on Modified Line Graph and Graph Neural Network

Jeongsol Kim
Department of Bio and Brain Engineering
KAIST
Daejeon, South Korea
wjdthf3927@kaist.ac.kr

ABSTRACT

Transmission Control Protocol (TCP) allows interactions between computers based on internet. Due to the its universality and importance, TCP has been used as a target to attack other connected computers or devices. Hence, it is required to predict and to prevent the attack through TCP system. To do this, a proper classification technique for the TCP attack based on the connection history is necessary. As one line of the classification study, we propose the graph neural network based model that reflects the dynamic connection. First, we define edge features that contain the information of each connection including connection timing and used port number. Then, we transform the graph into modified line graph so that we could define the problem as a form of graph classification. We apply the GraphSAGE to train the classification task for the given TCP connection history. Furthermore, we apply the gradient-weight CAM (Grad-CAM) to get the interpretability of the model about each TCP attacks pattern which is disappeared when we transform the problem from the edge classification into the graph classification. We analysed our model using F1 score, auc score and GRAD-CAM and we showed that our model capture the important patterns of TCP attacks efficiently so that the model outperforms the baseline models.

1. INTRODUCTION

Nowdays, most internet communication are based on packet switching. In packet switching, data is transmitted across digital network as a form of small packets for more efficient transferring [1]. Transmission control protocol (TCP) is a protocol for the packet switching that allows to rearrange small packets into the original data and to check the packet loss. Thus, TCP connection is highly universal over most of computers using internet. Because of its fundamental importance and necessary usage, TCP connection has been used a way to attack other connected computers by internet [2]. Definitely, preventing and predicting the attack trough TCP connection is important for computer security. One possible resource for counteracting these attacks is TCP connection history because we can identify each TCP attack using the patterns of the connection along time. If we could identify the markers for each attack, it will be easier to recognize and response to the attack.

One can try to recognize the pattern of each attack by reading as many as possible histories. However, it is not a reasonable idea owing to the tremendous many computers and real-time connections between them. Indeed, we need to build a proper model to recognize the patterns (or types) of TCP attack. So far, many researches in computer science field have done based on the statistical model [3, 4, 5]. They showed efficient TCP attack detection model and further proposed the way to defense the attack. One lack of these techniques is the generalized architecture for detecting the various TCP attacks. Most of works focus on Denial of Service (Dos) attack including Distributed Denial of Service (DDos) or the SYN flood.

In this project, we propose a generalized model for TCP attack recognition by making full use of the given TCP connection data. Especially, we utilize the graph neural network which is recently in the spotlight of a wide field of research. Due to the ubiquitous structure of the graph, the neural network on the graph already has been applied to physics field [6], [7] chemistry field [8], [9] and biology field [10], [11]. Big advantage of the graph neural network is that it allows to reflect both the value (or attributes) of data points and the connectivity between data points even when the data does not exist in a grid structure. We want to use this advantage of the graph neural network to our task; our purpose of using the graph neural network is to reflect not only the TCP connection pattern (pattern of timing that the transition of the data occurs) but also network connectivity (which IP is connected to which IP). To avoid the confusing from the notation 'connection pattern' and 'network connectivity', we will replace 'connection pattern' as 'pattern' from now on.

Now, we will introduce the flow of what we have done for this project. We have 30 minutes of TCP connection data that include source and destination IP addresses, port numbers, timestamps, and for training data, we know the types of connection (benign or attack type). For the first, we converted this data into graph structure so that we could define a problem as a familiar form to apply the graph neural network. The natural idea is constructing directed graphs by assigning each IP to each node and each connections to directed edges between nodes. However, the obstacle was that the given histories are dynamic. In other words, the connectivity changes along time. To simplify the dynamic

.

graph with small loss of information, we take an approach to convert each dynamic connection into static edge feature vector size of 1800; one entry corresponds to one second. In other words, our approach is the off-line method. Consequently, we get the static directed graph G whose nodes are corresponds to each IP of the TCP connection history and edges are corresponds to each connection. We handle the details of this approach in section 2.2.

As a result of previous graph construction, each node does not have features, but each edge has features that contains the dynamic pattern information. The next procedure is converting each edge of G into nodes so that two new static and undirected graphs H_1 and H_2 are generated from G. The node attribute is the corresponding edge features of G defined in the previous part. For H_1, we connect two nodes if they have the common source IP. For H_2, we connect two nodes if they have the common destination IP. Thus, two generated graphs have the same vertice set and node features with different connectivity. This is actually the modified version of line graph. The reasons for transforming to modified line graph are: 1) to improve the performance of model by compacting up the diversity of local structure 2) with maintaining the dynamic pattern and important connectivity information as much as possible and 3) to allow the usage of Grad-CAM for interpretability of the proposed model. We will deal with the details in section 2.3.

Based on the constructed graph H_1 and H_2, we applied a graphSAGE model [13] to do the inductive graph classification task. We compared the performance of our proposed mode, which uses both the pattern and connectivity, with simple fully-connected neural network model which do not use the connectivity between IPs. From this experiment, we want to show the advantages of the proposed graph-based model. Also, to see the improvement from the proposed modified line graph, we compared our model with the graphSAGE model which is trained on the line graph made from the G. The detailed information about applying the graphSAGE model will be given in section 2.4 and the comparison result will be handled in section 3.

Remark that we defined our problem as the graph classification. Thus, the prediction of our model is types of TCP attack which are contained in a connection history, not the certain IP connections that are used for attack. Definitely, this is not enough to our purpose of detecting TCP attacks from the history. Hence, we applied Grad-CAM method [15] which is widely used to get the interpretability of neural network model on image domain. Using this technique, we can predict not only the types of TCP attack but also connection candidates for each predicted attack. There exist several methods to make the interpretable model [16] but, we choose the Grad-CAM method which gives nice information with low computational cost. We will give the details about this part in section 2.5.

The entire framework of the proposed model is depicted in figure 1. Our contributions through this project are following:

- We propose a novel graph construction scheme for TCP attack classification such that maximize the usage of information given by TCP connection history.
- We propose the graph neural network based model for general TCP attack classification that can reflect both the TCP patterns and connectivity.
- Our model is also interpretable such that it can improves the one's level of understanding TCP attack pattern.

2. PROPOSED METHOD

2.1 Prerequisites

In this report, we will use normal small letter to denote the scalar value and bold small letter to denote the vectors. For the matrices, we will denote it by bold capital letter. $V(G)$ denotes the vertice set of a graph G. In the same line, $E(G)$ denotes the edge set of the graph G. For the node $u \in V(G)$, the corresponding node feature is denoted by ${}^{G}\mathbf{x_u} \in \mathbb{R}^d$. In the same line, ${}^{G}\mathbf{y_{uv}} \in \mathbb{R}^k$ denotes the feature of edge e_{uv} when there exists a connection from node u to v. In the matrix form, the node features of graph G are denoted by ${}^{G}\mathbf{X} \in \mathbb{R}^{d\times|V(G)|}$ and the edge features are denoted by ${}^{G}\mathbf{Y} \in \mathbb{R}^{k\times|E(G)|}$ where $|V(G)|$ denotes the cardinality of the set $V(G)$.

2.2 Simplifying the dynamics of graph

In this part, we will explain about the detailed procedure to converting the given TCP history into graphs. We first conceptually assigned each IP to each node and assigned the each connection to each edge between corresponding nodes. However, there are additional dimensions, time and port number. The edge dynamics along time axis is important because it contains pattern information. Also, as far as we understand, the port number can be used as a marker for some types of TCP attack [12]. Thus we need to consider both of dimensions, but the later one is easier to reflect than the earlier one. We decide to maintain the port number as the magnitude of the feature. In the case of time axis, one can decompose a history into multiple graphs depending on the time stamp. But, we should further consider the relationship between each graph which is still complicate. In addition, the connectivity in one graph may be too small to extract meaningful information (i.e. over-simplified). Thus, we decide to vectorize the time dimension and assign it as an edge feature size of 1800 defined by

$$ {}^{G}\mathbf{y_{uv}}[t] = \begin{cases} \frac{\gamma}{65535}\sum_{p_i\in P_{uv,t}} p_i \text{ if } P_{uv,t} \text{ is not empty set} \\ 0 \qquad\qquad\qquad \text{othrewise} \end{cases} \tag{1} $$

where $t = 1, 2, ..., 1800$ denotes time points in order of second, $\frac{\gamma}{65535}$ denotes normalization constant for port number, $P_{uv,t}$ is a collection of the port numbers used for connections between u and v at time point t. Note that the collection $P_{uv,t}$ allows the duplication of the port number. The role of the normalization constant γ is to scaling the port number from $[0, 65535]$ to $[0, \gamma]$. This normalization is required for numerical stability for subsequent calculation of graph neural network.

2.3 Transform into the modified line graphs

We can define the task as an edge classification on the constructed graph G in the previous section. However, we further convert the graph into the other form of graphs. The proposed transformation is based on the constructing the line graph. But, we modified some parts as following.

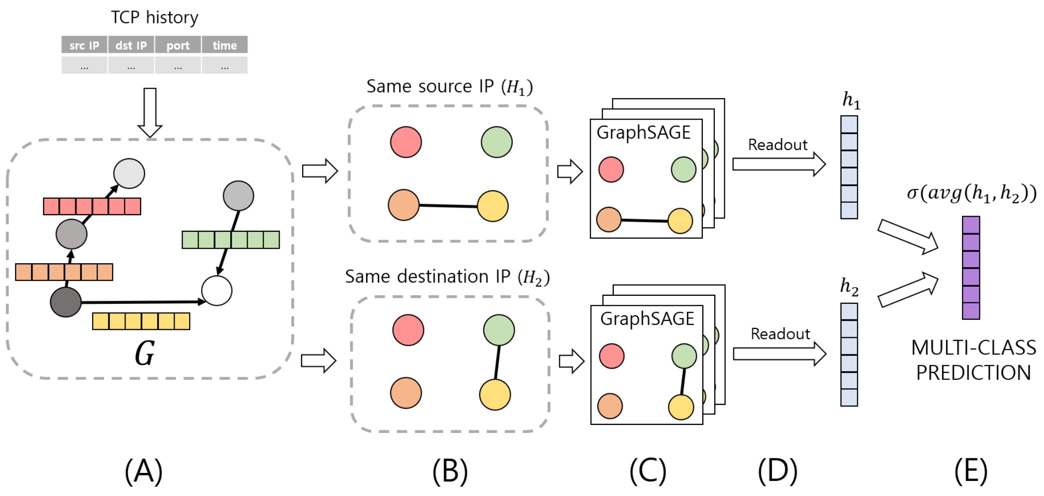

Figure 1: This figure is the entire scheme of our proposed method. (A) We first construct graphs using each TCP history by assigning each IP to nodes (gray color circles) and each connection between IPs to edges (black directed arrows). Also, we convert the information of each connection into edge feature vectors (color vector on arrows). (B) We further transform the constructed graph G into modified line graphs, H_1 and H_2. By the definition of line graph, each edge features in G is assigned to nodes in H_1 and H_2. Depending on the type of incident node for each edge in G, we decide the connectivity of H_1 and H_2 as a form of undirected edge (back lines between nodes). (C) After converting, we apply the GraphSAGE model to H_1 and H_2 independently. (D) We define our problem as graph classification. Thus, our model have additional readout layer that makes a vector from embedded node feature vectors. (E) Because we apply the GraphSAGE independently to H_1 and H_2, we have two embedded graph vectors. We average them and use as the multi-class prediction.

First we set the node features of new two graphs H_1 and H_2 as the edge features of the graph G.

$$^{H_1}\mathbf{X} = {}^{H_2}\mathbf{X} = {}^{G}\mathbf{Y} \tag{2}$$

Here, we designed the connectivity of H_1 and H_2 in different way to the line graph. For H_1, we connect node i and j when the corresponding i-th edge and j-th edge in G share the common source node (i.e. common source IP) whereas we connect nodes in H_2 when the corresponding edge in G share the common destination node (i.e. common destination IP).

The original line graph defines the connectivity by connecting the nodes when the corresponding edges have the same incident node. Note that the line graph is not isomorphic to the original graph. (i.e. two different graphs can be converted into the same line graph). This can cause undesired result in the subsequent tasks. One simple example is depicted in figure 2A. The given graphs in figure 2A are definitely different. In terms of TCP connection, they may be classified to different types of connection. However, their line graph is the same so that the following graph neural network may not classify them as different structure.

In contrast, our modified line graphs can address the issue. The key point is the designed connectivity depending on the common source and destination nodes. In figure 2B, we can see that our modified line graphs can convert the different structure differently. Of course, there still exists some cases that two different graphs are convert into the same graph like figure 3. But interestingly, this characteristic rather may improve the model by compacting up the diversity of local structures. Although two original graphs in figure 3 are not isomorphic, they have the equivalent connectivity in terms of TCP connection; connection from one IP to two IPs and connection from two IPs to one IP. TCP attack types is independent to the exact number of IP. Thus, it is highly likely that the given two original graphs in figure 3 have the same structure. In this situation, our method converts two essentially the same local structures (in our task) into the same structure.Therefore, this may reduce the diversity of local structures in the complicated graph that the subsequent graph neural network should learn, and improve the stability and performance of models.

We assumed that connections that do not share source IPs or destination IPs with all other connections are less likely to TCP attack. Thus, a single-sided cycle, such as right graph of figure 2B, has no connection in the modified line graph. These isolated nodes will be mapped linearly without message passing from neighborhood nodes (for graph neural network without non-lineality). As a result, they will have different characteristics to other connections, so they are expected to be classified differently to TCP attacks.

2.4 Multi-label classification problem on graph

The next step is to applying the graph neural network model to modified line graphs, H_1 and H_2. We designed our problem as a graph classification problem. Thus, the graph neural network should gives us one graph feature vector from one graph at the end.

We choose the graphSAGE model as our graph neural network because it can perform the inductive learning on various graphs. H_1 and H_2 are constructed with different concepts, so we applied two independent graphSAGE models to them with the same number of layers and the same

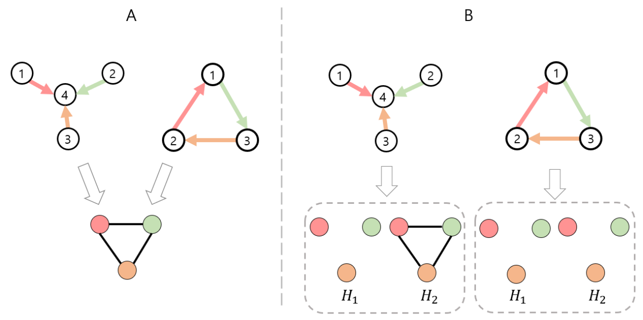

Figure 2: Upper two graphs are the original non-isomorphic graphs and lower graphs are corresponding original/modified line graph. The original line graph construction A is not isomorphism, but the proposed construction B can address this partially.

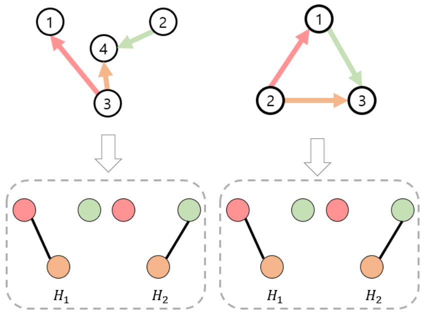

Figure 3: Upper two graphs are the original non-isomorphic graphs and low graphs are modified line graphs. Because the original graphs share the type of incident nodes (either source or destination), the modified line graphs are the same.

operations. Hence, we will give two node feature matrices from H_1 and H_2 (figure 1c). Like the commonly used readout layer for the graph classification problem, we average the node features across the all nodes and linearly embedding them once more (figure 1d). Then we get two graph representation vectors. We average them and apply the sigmoid function as the activation function (figure 1e). Note that our problem is the multi-label classification which means that the prediction of the model is binary vector, not one-hot vector like the multi-class classification problem. Thus, we apply the sigmoid rather than the softmax.

Formally, we can express the graphSAGE that we used as followings. Note that we only arrange the equations for the one modified line graph H_1, but the same equations are applied to the other one, H_2.

$$^{H_1}\mathbf{h}_v^0 = {}^{H_1}\mathbf{x}_v \tag{3}$$

$$^{H_1}\mathbf{h}_{N(v)}^k = AGGREGATE({}^{H_1}\mathbf{h}_u^{k-1},\ \forall u \in N(v)) \tag{4}$$

$$^{H_1}\mathbf{h}_v^k = Relu(COMBINE(\mathbf{W}_k \cdot {}^{H_1}\mathbf{h}_{N(v)}^k,\ \mathbf{B}_k\, {}^{H_1}\mathbf{h}_v^{k-1})) \tag{5}$$

$$^{H_1}\mathbf{h} = READOUT(\{{}^{H_1}\mathbf{h}_v^L,\ \ v \in V(H_1)\}) \tag{6}$$

where $k = 1, ..., L$ with the total number of layers L. $^{H_1}\mathbf{h}_v^k$ denotes the feature vector of node v in graph H_1 at k-th graphSAGE layer. The dimension of feature vector at k-th layer is denoted by d_k which is hyper-parameter that we can choose. $\mathbf{W}_k,\ \mathbf{B}_k \in \mathbb{R}^{d_k \times d_{k-1}}$ are embedding matrices at k-th layer for center nodes and neighbour nodes respectively. For $AGGREGATE(\cdot)$ we tried various candidate operations such as mean, pool, and we used mean for $COMBINE(\cdot)$. All these setting are equivalently applied to another graphSAGE for H_2.

For the readout layer,

$$READOUT({}^{H_1}\mathbf{h}_v^L) = \frac{1}{|V(H_1)|} \sum_{v \in V(H_1)} {}^{H_1}\mathbf{h}_v^L \cdot \mathbf{Q_1} \tag{7}$$

where $\mathbf{Q_1} \in \mathbb{R}^{d_L \times C}$ is a linear embedding matrix with the number of classes C. We just average the embedded node features across the all nodes and multiply the embedding matrix linearly.

Now, we get two graph feature vectors denoted as $^{H_1}\mathbf{h}$ and $^{H_2}\mathbf{h}$. Here, we have two variations for the last prediction of the proposed model. One is averaging the two representations and the other one is applying the Relu to $^{H_1}\mathbf{h}$ before averaging them like (8). Empirically, the second variation gives a significant improvement of the performance. This final output is called model's prediction or class score vector.

Finally, the loss equation for our model training is

$$BinaryCrossEntropy(\hat{\mathbf{h}},\ \ \sigma\left(\frac{Relu({}^{H_1}\mathbf{h}) + {}^{H_2}\mathbf{h}}{2}\right)) \tag{8}$$

where $\hat{\mathbf{h}}$ is corresponding label, σ denotes the sigmoid function. The binary cross entropy loss is defined by

$$BCE(x, y) = \frac{1}{N} \sum_{n=1}^{N} (y_n log x_n + (1 - y_n) log(1 - x_n)) \tag{9}$$

where $N = |V(H_1)| = |V(H_2)|$. For the inference, we deals with the class score higher than a threshold as positive and the class score lower than the threshold as negative.

2.5 Applying the Grad-CAM to the proposed model

One important lack of our design is that the model only predicts the types of the attack without the corresponding connections. To address this limitation, we tried to establish an interpretable model that gives the reasons (i.e connections) the model's prediction. There exist several algorithms for interpretable neural network. Attention is one of the great concepts that improves both the performance and the interpretability. However, we choose gradient-weight CAM (Grad-CAM) method [15] due to it's lower computational cost. To get the heat-map from Grad-CAM, gradients is all what need to calculate and because of auto-grad system of current python libraries such as pytorch and tensorflow, we can easily do it. Originally, Grad-CAM is method for convolutional neural network (CNN) with task on image domain. Grad-CAM calculate the gradient of the class-prediction with respect to the last hidden feature map. So, the gradient-weighted heat map from Grad-CAM tells us how much each region contribute to the class-prediction.

In similar way, [17] adapt the Graph-CAM method to Graph-convolutional Network (GCN [18]). They calculated the gradient of the predicted class score (for node classification) with respect to hidden node feature matrices. So they could get the gradient-weighted heat map for node feature matrix at each GCN layer. The concept of GCN is not much different to graphSAGE; it also aggregates the neighbour node features, combine them using average and embedding to lower dimensional space. In other words, the formula of the graphSAGE is generalized version of GCN. Thus, we thought that the same Grad-CAM method can work for graphSAGE too.

For our application, the Grad-CAM's weight for class c at layer l and for feature k (i.e. for k-th entry of the feature vector) is calculated by

$$\alpha_k^{l,c} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial y^c}{\partial F_{n,k}^l} \tag{10}$$

where n denoted indices for nodes, $N = |V(H_1)| = |V(H_2)|$ and $F_{n,k}^l$ is (n,k) element of the output features at the l-th layer. Using this weights, the heat map is calculated as

$$L_{Grad-CAM}^c[l,n] = ReLU(\sum_k \alpha_k^{l,c} F_{n,k}^l(\mathbf{X}, \mathbf{A_1}, \mathbf{A_2})) \tag{11}$$

where $\mathbf{X}$ is the input node feature matrix and $\mathbf{A_1}, \mathbf{A_1}$ are the adjacency matrices of H_1 and H_2, respectively. Especially, we are interested in input connections that are highly responsible to the predicted class score. So, we calculated the gradient of the predicted class score with respect to the input features (i.e. $l = 0$). The heat map is a matrix size of 25 by $|V(H_1)|$. If a value of the heat map at (l,n) is higher than other entries, it means that n-th connection is more likely to be responsible to the prediction of the l-th attack type.

3. EXPERIMENTS

3.1 Experiment details

We trained our model using the given TCP history files that are recorded for 30 minutes. The training data are composed by source and destination IPs, port numbers, time points in order of seconds and type of the attack of each IP connection. Because we do not have enough large number of data for training, the over-fitting problem occurs easily. Thus, we did not use the entire training set for each epoch. Instead, we uniformly sampled 70% of the training data per epoch as a kind of data augmentation.

We want to compare the classification performance of the simple feed forward network, the GraphSAGE model on the line graph, and the proposed model. From the comparison with the simple feed forward network, we want to examine the improvement which comes from the graph neural network using connectivities. Alos, from the comparison with the GraphSAGE one the line graph, we want to examine the improvement which comes from the proposed modified line graph. Thus, we unified all hyper-parameters and the structures. For every experiments, we set number of epochs as 10, learning rate as 1e-3 and $\gamma = 1.0$. For the all models, we unified the number of layers of as 2, hidden dimension as 256.

For the metric of the performance, we used two scores; weighted average of F1-score and area under the curve (auc) score from the receiver operating characteristic (ROC) curve. Because the given data contains multi types of attack and are imbalanced between benign connection and attack connection, the weighted average of F1-score is more proper than the accuracy. The weighted average of F1-score for the TCP connection history g is defined as

$$F_1^w = \frac{1}{2} \left[\frac{1}{|B|} \sum_{g \in B} F_1(g) + \frac{1}{|A|} \sum_{g \in A} F_1(g) \right] \tag{12}$$

$$F_1(g) = 2 \times \frac{prec(g) \cdot recall(g)}{prec(g) + recall(g)} \tag{13}$$

$$prec(g) = \frac{TruePositives(g)}{TruePositives(g) + FalsePositives(g)} \tag{14}$$

$$recall(g) = \frac{TruePositives(g)}{TruePositives(g) + FalseNegatives(g)} \tag{15}$$

where B represents the set of networks without any attacks and A indicates the set of networks containing TCP attacks. The highest F1-score is 1, indicating that the model shows perfect precision and recall. In other words, the model successfully classifies the data into multiple classes.

To get more information, we also used auc score which is defined as the area under the ROC curve. Similar to the F1-score, the highest value of auc score is 1, indicating that the model distinguishes between the positive and negative. In contrast to the F1-score, auc score is calculated for each class, so we can check the performance of the model for each class independently.

To check whether the result is consistent, we did 10 times experiment and reports the averaged values.

3.2 Results

3.2.1 Weighted Average of the F1-score

Our experimental results comparing the weighted average of the F1-score of the proposed method and two baseline methods are shown in table 1. The graph neural network based models outperform the simple feed forward network which does not use network connectivities. Especially, there is significant improvement in F1-score for the TCP histories containing attacks ($F1^A$) which indicates that the usage of IP connection information makes better classification of the TCP attack types. Furthermore, our proposed method achieves the best performance among the three methods which indicates that the proposed modified line graph makes improvement. Interestingly, when we add an additional Relu layer to the graph representation vector from the H_1, the F1-score becomes higher.

3.2.2 AUC score

In addition to the F1-score, we want to see the classification performance for each type of attacks independently. Thus, we calculated the auc score for each class and the results are shown in table 2.

First of all, note that the auc score for the benign is about 0.5 for every tested models. When we checked the ROC curve for the benign class, it was close to $y = x$ graph which implies that the model cannot distinguish the benign type into the positive or negative meaningfully. By combining this result with the high F1-score for the set of networks without any attacks ($F1^B$), we can conclude that our model identifies the benign history based on the existence of TCP attacks, not based on the feature of benign connection. This is reasonable because the benign TCP connections may not have certain features like attacks so that the neural network models could not extract consistent feature that implies the benign connection. For the other attack types, our proposed method shows the higher auc scores than baselines for more than half classes. Significant improvements occurs in neptune, nmap, pod, portsweep, rootkit, smurf, snmpgetattack and warezclinet. For the attack named saint, satan and warez, the baseline methods are better than our proposed model. However, note that the auc score of the baseline models for the first two attacks are quite low which implies that actually the baseline models are also not good at distinguish them as the positive or negative. Hence, we conclude that our methods consistently better that the baseline methods for distinguishable TCP attacks.

Table 1: F1-scores on the validation set: FFN represents the simple feed forward network and SAGE represents the GraphSAGE model on the line graph. (m), (p) indicates mean and pool aggregation respectively. (+r) indicates Relu in (8). Values are mean ± std.

Method	$F1^A$	$F1^B$	$F1^W$
FFN	0.636±0.009	0.977±0.011	0.807±0.006
SAGE (m)	0.657±0.015	0.962±0.017	0.809±0.008
SAGE (p)	0.651±0.008	0.974±0.007	0.812±0.005
Ours (m)	0.675±0.015	0.940±0.022	0.808±0.006
Ours (p)	**0.683 ± 0.014**	0.956±0.018	0.819±0.008
Ours (m+r)	0.661±0.012	**0.985 ± 0.008**	**0.823 ± 0.006**
Ours (p+r)	0.665±0.015	0.981±0.012	**0.823 ± 0.005**

3.2.3 *GRAD-CAM analysis*

To test whether the GRAD-CAM applied to the proposed method provides a correct interpretation of its prediction, we first checked the heat map for the training data set, since only the training data set shows matches between the IP connection and the type of attack. The results are shown in figure 4. Although the GRAD-CAM is quite simple method, it makes a meaningful heat-map that provides the interpretation about the prediction of our model. Figure 4B is the heat map from GRAD-CAM based on the prediction of our model about a training file that contains only the snmpgetattack. Our model predicts the type of attack as snmpgetattack with score 0.9385. According to the heat map, connection 0, 1, 9 and 13 are considered as connection containing snmpgetattack. From these candidates, we select the connection with the highest heat map value in order to check whether the selected connection really contains the predicted attack in the training file. The selected connection was connection 1 and the initial attributes for the selected connection is shown in figure 4A. Note that the selected connection occurs very frequently. When we manually checked the training file, we found that the selected connection is classified as snmpgetattack. Furthermore, we saw the heat map from the another training file that does not contains any attacks. Our model classified the file as benign and the heat map based on our model is shown in figure 4C. The heat map definitely has low values, which indicates that no attacks are detected. The first row of the heat map is black which implies that our model does not detect the benign connection directly and identifies the benign by eliminating the possibility of other types of attack. This result is consistent with the quantitative analysis in the previous sections. Definitely, the GRAD-CAM method makes our model interpretable.

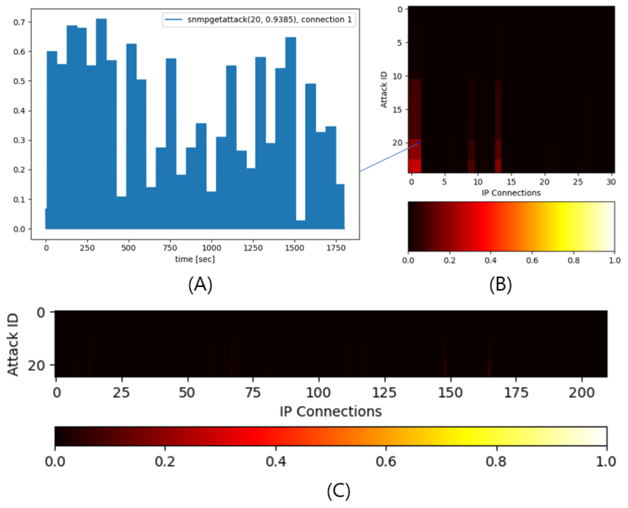

Figure 4: GRAD-CAM result for the training data. (A) The initial attribute of the node 1 (or connection 1). This connection has the highest heat map value for the model prediction 'snmpgetattack'. In other wards, this connection is likely to be responsible to the prediction. (B) The entire heat map from GRAD-CAM for the train_732.txt file. The label attack type is 'snmpgetattak'. (C) The entire heat map from GRAD-CAM for the train_100.txt file. The label attack type is 'benign'. Unlike the (B), there is not significant heat map because our model predicts it as benign.

We also applies the GRAD-CAM methods to validation data. One example is shown in figure 5. We checked the validation file as same as the previous analysis and founded that the GRAD-CAM methods correctly provides the responsible connection to the prediction of our model.

4. DISCUSSION

For the F1 score, our model was better than the baseline methods. However, among variations of my model, two types of aggregation are comparable. When we use mean operation as aggregation, the F1 score for benign networks is lower than the pool operation while the F1 score for attacked networks is higher than the pool operation. If we add the Relu layer, the result becomes opposite. The F1 score for benign networks is higher when the aggregation is mean while the F1 score for attacked networks is higher when the aggregation is pool. From the point of view of the overall

Table 2: auc-scores on the validation set: Number at each name of attacks represents the class ID that we used to identify them. FFN represents the simple feed forward network and SAGE represents the GraphSAGE model on the line graph. (m), (p) indicates mean and pool aggregation respectively. (+r) indicates Relu in (8). Values are mean $\pm$ std and Top-2 values are indicated by the bold letter (yellow: the highest, green: the 2nd). Because there were no apache2 and smurfttl attacks in the validation data, the values for them are reported as nan.

Method	benign (0)	apache2 (1)	back (2)	dict (3)	guest (4)	httptunnel-e (5)	ignore (6)	ipsweep (7)	mailbomb (8)
FFN	**0.528 ± 0.010**	nan	**0.951 ± 0.016**	**0.560 ± 0.026**	0.558 ± 0.020	0.323 ± 0.033	0.742 ± 0.045	0.648 ± 0.016	**0.987 ± 0.029**
SAGE (m)	**0.524 ± 0.013**	nan	0.870 ± 0.027	0.490 ± 0.044	**0.764 ± 0.051**	**0.544 ± 0.074**	0.834 ± 0.047	0.543 ± 0.014	0.917 ± 0.125
SAGE (p)	0.511 ± 0.014	nan	0.817 ± 0.129	0.557 ± 0.058	0.657 ± 0.122	**0.710 ± 0.116**	**0.857 ± 0.036**	0.585 ± 0.047	0.920 ± 0.075
Ours (m)	0.506 ± 0.014	nan	0.854 ± 0.037	0.559 ± 0.046	0.644 ± 0.048	0.432 ± 0.051	**0.838 ± 0.064**	**0.695 ± 0.023**	0.922 ± 0.125
Ours (p)	0.472 ± 0.032	nan	0.853 ± 0.067	0.511 ± 0.079	0.597 ± 0.100	0.283 ± 0.045	0.596 ± 0.112	**0.719 ± 0.044**	**0.939 ± 0.096**
Ours (m+r)	0.497 ± 0.017	nan	**0.877 ± 0.024**	**0.588 ± 0.047**	**0.661 ± 0.018**	0.321 ± 0.038	0.827 ± 0.037	0.627 ± 0.031	0.845 ± 0.222
Ours (p+r)	0.497 ± 0.036	nan	0.825 ± 0.076	0.515 ± 0.049	0.620 ± 0.052	0.317 ± 0.071	0.701 ± 0.112	0.683 ± 0.044	0.920 ± 0.050

Method	mscan (9)	neptune (10)	nmap (11)	pod (12)	portsweep (13)	processtable (14)	rootkit (15)	saint (16)	satan (17)
FFN	0.609 ± 0.047	0.516 ± 0.069	0.562 ± 0.027	0.458 ± 0.018	0.773 ± 0.019	0.034 ± 0.055	0.781 ± 0.017	0.060 ± 0.018	**0.243 ± 0.026**
SAGE (m)	**0.925 ± 0.025**	0.483 ± 0.047	0.579 ± 0.028	0.309 ± 0.030	0.758 ± 0.021	**0.133 ± 0.290**	0.321 ± 0.048	**0.063 ± 0.018**	0.239 ± 0.056
SAGE (p)	**0.975 ± 0.007**	0.474 ± 0.094	**0.586 ± 0.018**	0.253 ± 0.064	0.742 ± 0.016	0.068 ± 0.070	0.381 ± 0.178	**0.124 ± 0.051**	**0.339 ± 0.188**
Ours (m)	0.702 ± 0.064	0.537 ± 0.040	0.578 ± 0.046	0.462 ± 0.025	**0.800 ± 0.033**	0.023 ± 0.015	**0.845 ± 0.034**	0.017 ± 0.006	0.156 ± 0.070
Ours (p)	0.786 ± 0.118	**0.547 ± 0.049**	**0.588 ± 0.049**	**0.515 ± 0.076**	0.776 ± 0.023	0.055 ± 0.048	0.763 ± 0.070	0.021 ± 0.024	0.104 ± 0.064
Ours (m+r)	0.914 ± 0.030	**0.561 ± 0.079**	0.572 ± 0.068	0.361 ± 0.045	**0.805 ± 0.037**	**0.111 ± 0.296**	**0.860 ± 0.041**	0.010 ± 0.004	0.109 ± 0.022
Ours (p+r)	0.898 ± 0.083	0.507 ± 0.142	0.524 ± 0.027	**0.471 ± 0.048**	0.755 ± 0.024	0.044 ± 0.026	0.790 ± 0.109	0.004 ± 0.002	0.089 ± 0.035

Method	smurf (18)	smurfttl (19)	snmpgetattack (20)	snmpguess (21)	teardrop (22)	warez (23)	warezclient (24)
FFN	0.870 ± 0.033	nan	0.765 ± 0.016	0.538 ± 0.031	0.290 ± 0.037	0.560 ± 0.035	0.883 ± 0.011
SAGE (m)	0.981 ± 0.005	nan	0.772 ± 0.018	0.545 ± 0.037	**0.538 ± 0.058**	**0.695 ± 0.037**	0.881 ± 0.014
SAGE (p)	0.973 ± 0.012	nan	0.789 ± 0.020	**0.612 ± 0.127**	0.446 ± 0.073	**0.714 ± 0.039**	0.870 ± 0.021
Ours (m)	**0.988 ± 0.007**	nan	0.756 ± 0.024	**0.599 ± 0.036**	**0.479 ± 0.045**	0.557 ± 0.027	**0.885 ± 0.007**
Ours (p)	0.952 ± 0.073	nan	**0.803 ± 0.026**	0.574 ± 0.055	0.432 ± 0.120	0.590 ± 0.042	0.872 ± 0.013
Ours (m+r)	**0.989 ± 0.012**	nan	0.796 ± 0.030	0.521 ± 0.053	0.358 ± 0.066	0.634 ± 0.061	**0.895 ± 0.009**
Ours (p+r)	0.851 ± 0.055	nan	**0.806 ± 0.033**	0.526 ± 0.040	0.377 ± 0.094	0.635 ± 0.031	0.858 ± 0.024

weighted average of the F1 score, the mean operation with Relu layer shows a highest score. We have to take into account, however, that this higher score is not the best metric, which implies a better multi-label classification for the TCP attacks. Nevertheless, our methods with Relu layer outperform the baseline methods regardless of the aggregation type. The auc score also provides similar discussion point that higher F1 score does not mean the better performance always. Although the proposed method has higher F1 score than the baseline methods, the baseline methods are better to distinguish the positive or negative of few types of attack.

An underlying assumption of the neural network model is that the model extracts the important features from the given data and uses them for downstream tasks. The above two metrics can provide quantitative evidence of the performance of our model, but they cannot provide evidence that the model is actually extracting features according to the assumption. This is the reason that we want to give interpretability to our model by applying the GRAD-CAM. Figure 6 implies that our model detects meaningful features of TCP attack. The used validation history contains neptune attack but our model said that it contains nmap, snmpgetattack and warez attacks with high score. Definitely our model predict incorrectly. However, note that the GRAD-CAM heat map has only one line at the connection 181. Our model concluded that there are three types of attack based on this one connection. Although the prediction was wrong, the heat map implies two positive things. First, this implies that our model can detect uncommon patterns of connection. When we checked other connections in the same validation file, most of connections were very sparse (about < 5 connections during 30mins). In contrast, the detected connection in figure 6 contains continuous pattern with large port numbers. Remark that the magnitude of the initial attribute is proportional to the port number according to (1). Second, we empirically saw that the predicted attack types are related to the repeated connection pattern like figure 4A. The detected pattern also contains similar repeated pattern so that our model confused. Of course, this means that our model should be better than now.

Through our analysis, we conclude that it is necessary to check additional metrics such as the auc score and additional interpretation from GRAD-CAM instead of just using the F1 score.

5. CONCLUSIONS

In this project, we propose an inductive TCP attack classification model based on the graph neural network. To simplify dynamic TCP connection data, we propose a way of creating static feature vectors that contain the connection dynamics and the port numbers. In particular, we propose a novel graph construction concept called modified line graphs to capture the structural information that plays an important role in classifying the attack type. The proposed method outperforms the baseline methods; one is the model that does not use network connectivity and the other is the model that works on the line graph. In addition to the quantitative analysis, we apply the GRAD-CAM method to our model and show that the proposed model successfully captures unusual patterns.

Some questions remain. First, we did not analyse the reason of improvement from the additional Relu layer at the

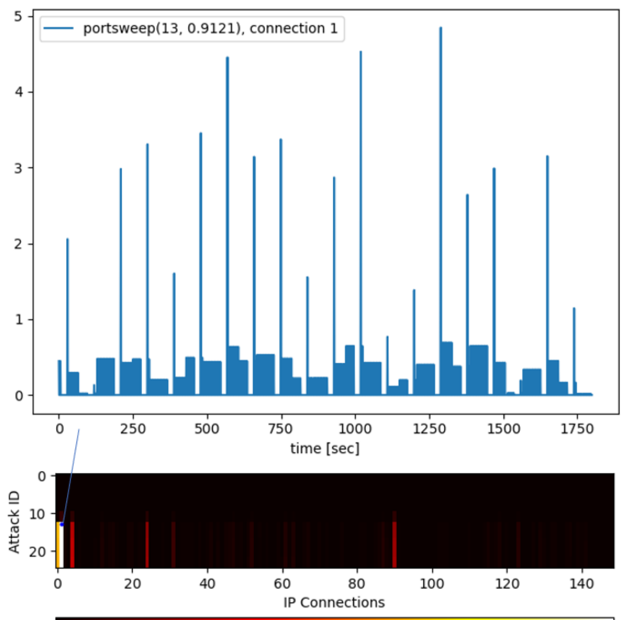

Figure 5: GRAD-CAM result for a validation data (valid_query_128.txt). The heat map and the initial node attribute corresponding to the highest value are depicted.

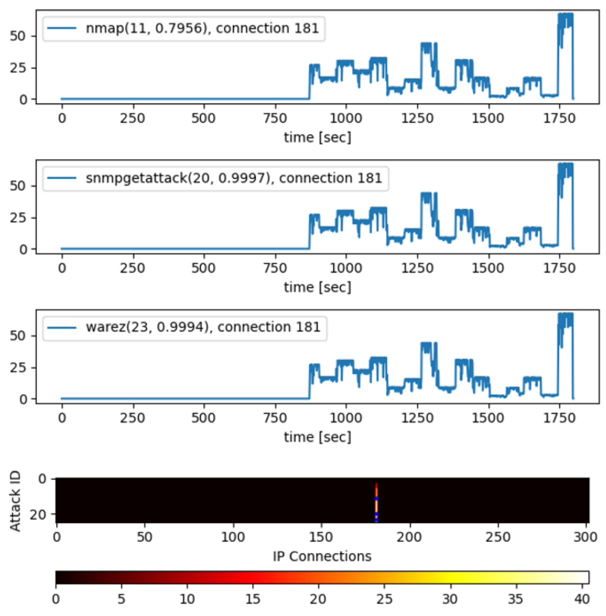

Figure 6: GRAD-CAM result for a validation data (valid_query_078.txt). The heat map and the initial node attribute corresponding to the highest value are depicted. The predicted types were nmap, snmpgetattack and warez but the label is neptune.

graph representation of H_1. We need to reveal the effect of imbalanced application of non-linear layer to the modified line graphs. Second, our model is off-line model. For real application, on-line model is more useful because it can immediately detect the TCP attack when it occurs. Extension of our framework to the on-line model can be another interesting topic. Third, the GRAD-CAM is kind of the simplest method among the techniques to give interpretability to the neural network model. Nevertheless, it works well with our method. However, one can increase the interpretability by applying other improved version of methods.

6. REFERENCES

[1] packet switch definition: https://avinetworks.com/glossary/packet-switching/.

[2] computer network security: https://www.utc.edu/center-academic-excellence-cyber-defense/pdfs/course-paper-5620-attacktcpip.pdf.

[3] JBD Caberera, B Ravichandran, and Raman K Mehra. Statistical traffic modeling for network intrusion detection. In *Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No. PR00728)*, pages 466–473. IEEE, 2000.

[4] Jelena Mirkovic, Gregory Prier, and Peter Reiher. Attacking ddos at the source. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 312–321. IEEE, 2002.

[5] Laura Feinstein, Dan Schnackenberg, Ravindra Balupari, and Darrell Kindred. Statistical approaches to ddos attack detection and response. In *Proceedings DARPA information survivability conference and exposition*, volume 1, pages 303–314. IEEE, 2003.

[6] Xiangyang Ju, Steven Farrell, Paolo Calafiura, Daniel Murnane, Lindsey Gray, Thomas Klijnsma, Kevin Pedro, Giuseppe Cerati, Jim Kowalkowski, Gabriel Perdue, et al. Graph neural networks for particle reconstruction in high energy physics detectors. *arXiv preprint arXiv:2003.11603*, 2020.

[7] Junyoung Park and Jinkyoo Park. Physics-induced graph neural network: An application to wind-farm power estimation. *Energy*, 187:115883, 2019.

[8] Connor W Coley, Wengong Jin, Luke Rogers, Timothy F Jamison, Tommi S Jaakkola, William H Green, Regina Barzilay, and Klavs F Jensen. A graph-convolutional neural network model for the prediction of chemical reactivity. *Chemical science*, 10(2):370–377, 2019.

[9] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[10] Mona Alshahrani, Mohammad Asif Khan, Omar Maddouri, Akira R Kinjo, Núria Queralt-Rosinach, and Robert Hoehndorf. Neuro-symbolic representation learning on biological knowledge graphs. *Bioinformatics*, 33(17):2723–2730, 2017.

[11] Xiaoxiao Li, Nicha C Dvornek, Yuan Zhou, Juntang Zhuang, Pamela Ventola, and James S Duncan. Graph neural network for interpreting task-fmri biomarkers. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 485–493. Springer, 2019.

[12] Port number and tcp attack: http://xeushack.com/practicing-for-ddos.

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[14] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[15] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[16] Christoph Molnar. *Interpretable Machine Learning*. Lulu. com, 2020.

[17] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10772–10781, 2019.

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

APPENDIX

A. APPENDIX

A.1 Labor Division

In this project, I performed the following tasks

- Data preprocessing
- Design the problem as graph classification
- Construct the scheme to address the problem
- Experiments on the real data
- Write the final report
- Prepare presentation

A.2 The reason for one-man project

Although I take this lecture alone, I tried to make a team. However due to the online lecture, it was hard to contact to others. I found the post that is written by professor in Classum and I once contact to one student. But, he already formed his team and I judged that it is better to construct my own research hypothesis due to the time limit at that time. Sorry about not to make a team.