Weight-Based Deep Model Compression of DGA Detectors for Small Devices

by

Nussara Tieanklin

Supervised by Dr. Juhua Hu

A senior thesis submitted in partial fulfillment of the departmental honors requirements for the degree of

Bachelor of Science
Computer Science & Systems
University of Washington Tacoma

June 2021

Presentation of work given on June 2, 2021.

The student has satisfactorily completed the Senior Thesis, presentation and senior elective course requirements for CSS Departmental Honors.

Faculty advisor: [illegible] Date 06/14/2021

CSS Program Chair: [illegible] Date 06/21/2021

Weight-Based Deep Model Compression of DGA Detectors for Small Devices

***Abstract* -** The sophisticated model such as deep neural networks (DNNs) based architecture is capable of handling Domain Generating Algorithms (DGAs) malware threats particularly well. Regardless of such remarkable success, DNN models have high computational and storage overhead. It is very difficult to deploy the model into the resource-constrained devices that everyday users have access to, such as mobile phones, home routers, embedded gadgets, or IoT devices. In this thesis, we explore an effective way to compress the DNN model while preserving the effectiveness and enhancing the efficiency. We introduce filter by filter pruning together with additional pre-processing techniques, such as normalization to avoid any biases from dominant values, and a clustering technique to identify redundancy to remove. Adapting the proposed weight-based deep model compression methodology to the pre-trained DGA classifiers, we are able to facilitate the DNN model size reduction by removing 40% of the filters, while preserving the effectiveness of the model. Therefore, reducing energy consumption and memory overhead, our proposed methodology facilitates accessibility of many deep model applications, promotes higher security measures, and enables energy saving for day-to-day users.

Index Terms - Deep Learning, Model Compression, Pruning, Clustering

INTRODUCTION

Many malware attacks have caused significant losses to governments, private sectors, industries, and various main information infrastructures. One of the well-known malware is the Domain Generating Algorithms (DGAs). DGA is a popular technique unscrupulous players use to install malwares by dynamically generating ten of thousands of domain names daily, and the majority are unregistered. They masquerade the registered domains, allowing the infected botnets to slip through and evade automatic detections from cybersecurity measures. DGAs employ various malwares families that challenge the command-and-control (C&C) countermeasure.

A variety of strategies have been proposed to detect the DGAs malwares. In recent years, many efforts have been devoted to finding effective malware defenses to protect our cyberspace on computers and networks. One of the previous work is using a Hidden Markov Model (HMM) framework to generate distributions of DGA malware families and benign domains [7]. With the increasing severity of the damage from cybersecurity threats, it requires a highly-developed approach to target the entire DGA malware families. Deep neural networks (DNN) model, a part of the Machine Learning family, has proven to be one of the most effective approaches [9, 10, 11]. Particularly, the DNN model is trained on a pre-registered or blacklist of all the domains, which are static and in a sequence of characters or words, the model can easily recognize infected domain names to help prevent invasion on C&C. The convolutional neural network (CNNs) based architecture of the DNN model outperforms other models by effectively identifying malicious and difficult-to-detect DGA families. Despite its sophisticated performance, the challenge presented by neural network models is that they can only operate in larger systems due to their size. In response to rapidly growing numbers of Internet of Things (IoT) devices, we seek to compress the model to improve its usability and allow them to operate in smaller and affordable IoT or edge devices.

As the continuation of the prior work of the CNN model, which was pre-trained by the Data Scientist team at Infobox, we explore various technical discoveries to downsize the model while preserving the effectiveness and improving the efficiency. In this thesis, inspired by the pruning technique proposed in [1], we focus on using a basic but rather powerful property, such as weights in the convolutional layer to locate redundant information involved in the pre-trained model and thus remove them for model compression. Instead of doing neuron by neuron pruning that is time consuming and computationally expensive, we propose to do filter by filter pruning, where each filter contains a certain number of neurons depending on the size designed by the neural network, where each neuron is associated with a weight. Another benefit of filter by filter pruning is that the corresponding following layers can be pruned accordingly.

Each filter is represented by a matrix. To find redundant filters, it is important to define the similarities between filters. Then, similar filters can be removed for compression. We propose to do clustering over filters or group similar filters together. As a pre-processing step for similarity calculations of filters, we normalize their weights into a specific range to avoid any potential bias that can be introduced by dominating large weights and flatten each filter from a matrix format to a vector format to be fed to a clustering algorithm. Considering that we do not have any domain knowledge to provide a predefined value of how many clusters all the filters can group, we adopt HDBSCAN [6] as our clustering method. HDBSCAN groups redundant filters in the same cluster, without requiring an input on the number of clusters needed. Additionally, it works particularly well with data that have varying density.

After applying this methodology to the pre-trained DGA classifier, some filters are grouped together into one cluster. Then, we can remove certain redundant filters within a cluster to compress the deep model. We propose to use the sum of absolute weights to indicate one filter's importance. Intuitively, higher the weight sum, more important the filter. Thereafter, we can remove filters within a cluster of less importance, so that it will not degenerate the model's prediction performance. By removing filters in one cluster according to their importance, we are able to remove around 40 percent of filters without sacrificing too much prediction performance. This demonstrates the effectiveness of the proposed weight-based methodology for deep model compression to promote accessibility and security in small and affordable devices for everyday users.

Related Work

Our work has drawn a variety of related works on model compression, particularly on pruning, and clustering (i.e., HDBSCAN). All details are listed below.

I. Model Compression

Working with multiple layers of neural networks within the deep learning model can often be so computationally intensive that it is unsuited for lightweight IoT devices. The majority of existing deep model compression methods apply a pruning method to decrease the model size utilizing the weight information in the convolutional layer. Though our work has incorporated more fine-tuned pruning, there are a number of scholars working on refining the state-of-art pruning that this work draws from.

Network pruning is one of the basic and popular approaches that aim to reduce redundancy and compress the heavyweight model. The majority of pruning network techniques are derived from an algorithm proposed by Han et al. [1]. The algorithm is first trained to find meaningful or dominant criteria. Then each of the parameters in the network is issued a score. The network will then use these scores as a reference to prune the model, anything below a threshold will be removed. Since pruning on the first iteration reduces the accuracy, pruning is then followed by greedy iterative prunings to retrain such iteration (known as *fine-tuning*) to improve the accuracy. This will help establish the best connection or dominant criteria of each parameter. During the retraining phase, the redundant information or parameters with zero connection can be automatically removed safely using gradient descent and regularization reducing the network size while preserving the model accuracy. The author mentioned that probabilistically pruning parameters based on their absolute value provides worse results. The literature review [2] found that many papers have proposed a variation of the algorithm such as a technique of pruning that starts at initialization [3], periodically prunes while training [4], or even include additional parameters to use as a score to prune [5].

Various techniques of pruning can accomplish many different goals that favor different design choices, requiring different metrics of evaluation. This includes reducing storage requirements, computational cost, energy consumption, and etc. Regardless of efficiency, the non-structure sparse model from Han et al. [1] can not be supported by off-the-shelf libraries, thus it requires specialized hardware and software, which is not practical and expensive. More importantly, the issued importance scores from many pruning strategies often disregard less relevant parameters entirely and only take more dominant neurons into account of evaluation. Different from these existing pruning techniques that require a certain degree of training or retraining involved, this thesis aims to develop a pruning method applicable for pre-trained GDA classifiers without any additional retraining.

II. Clustering

Clustering establishes natural grouping or clusters within multidimensional data based on some similar measures [13]. Therefore, this is a popular technique that can be used to group similar items, while we also aim to find similar filters in this work. Existing clustering techniques can be categorized into two main directions: point assignment based approaches and hierarchical clustering approaches [14]. Point-assignment based algorithms like k-means [15] often require an additional input as the number of clusters needed, which is very hard to decide in advance in our problem. Therefore, we adopt the hierarchical clustering method in this work to find redundant filters.

Specifically, Hierarchical Density-Based Clustering (HDBSCAN) is used. HDBSCAN is implemented based on Density-based-spatial clustering of applications with noise (DBSCAN) [6]. DBSCAN separates clusters with high density from low density. It works particularly well in sorting data into clusters of varying sizes and shapes. It will also detect and ignore the outliers as noises. However, for clusters that have varying density, DBSCAN fails to recognize that some of the noises are a part of the cluster. So it separates the very low-density noises into their own individual mini clusters. HDBSCAN, on the other hand, performs clustering better in the datasets when it comes to such varying density as the algorithm will focus on clustering high density. Besides, HDBSCAN requires less predefined parameters than DBSCAN to perform clustering, which is better suited in our problem. Finally, at the 200,000 record points, DBSCAN algorithm takes twice as much time as HDBSCAN would [6], and thus HDBSCAN's efficiency in finding groups of filters is also desired.

Methodology

Given a pre-trained neural network, in this section, we aim to compress the model by looking at the convolutional layers. Specifically, in the convolutional layer, a deep neural network for DGA classification often contains N filters and each filter is a matrix of size $n \times m$ as shown in the left plot of Figure 1, where n is the specific elements of the input to perform the filter pruning and m is the embedding size. Instead of looking at each weight in each matrix one by one to remove less important elements, which is time consuming and hard to compress the following layers accordingly, we aim to find redundant filters by looking at each matrix (i.e., each filter) as a whole. Consequently, if a filter is removed, the corresponding following layers of this filter can be removed accordingly to easily compress the whole model not just within the convolutional layer.

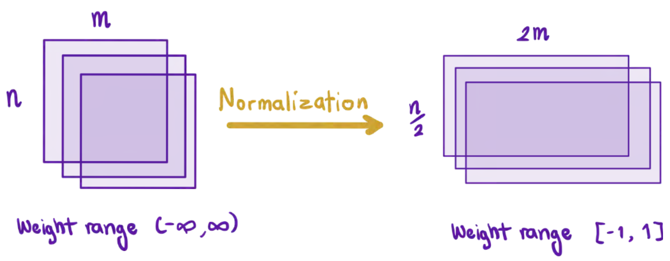


Figure 1: Weight normalization

As a first step to remove redundant filters, it is intuitive to check if there is any filter whose weights are too low to be useful. It should be noted that, in a pre-trained model, it is often hard to find a filter whose weights are all zeros. Therefore, we aim to group similar filters, so as to find redundant filters within a cluster and remove them to compress the model. Because the values in a matrix of a filter can be any real value, matrices with large values in matrices may dominate the similarity calculation. Therefore, we first normalize the weights for all filters into range $[-1, 1]$ and flatten the matrix to a vector for the ease of similarity calculation. Specifically, as shown in Figure 1, we often use bigrams to represent a domain name in GDA classification and thus $n = 2$. Therefore, the new vector representation of each filter will be of size $(n/2) \times (2m) = 1 \times (2m)$.

Now given the vector representation of each filter whose element values are in range $[-1, 1]$, we are aiming to find the redundancy between these filters. It is straightforward to apply a clustering algorithm to group similar filters together and thus the redundant filters can be discovered within each formed cluster. Therefore, we feed the vectors to HDBSCAN to cluster the filters as shown in Figure 2.

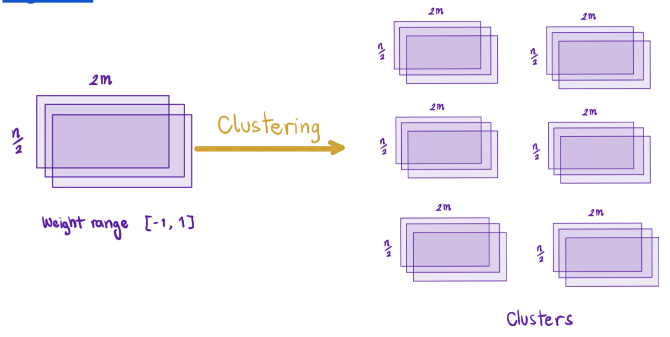


Figure 2: Clustering

With the filters grouped in clusters, it is intuitive to consider that all filters in one cluster are similar and thus are redundant. Therefore, a simple way to compress the model is to keep only one representative filter in each cluster. However, due to the definition of different similarity functions, some important local weights can be hidden. Blindly keeping only one filter in each cluster may degenerate the model's prediction performance. Therefore, we proposed to calculate the importance score of each filter within a cluster using the absolute sum of weight values. Assembling the normalization, clustering, and sorting algorithms, we can describe the full pipeline of the proposed methodology in Figure 3. Then, the filters in each cluster can be gradually removed from the cluster according to their importance score (i.e., removing from the lowest important filter until the 2nd highest important filter). Thereafter, a specific number of filters within each cluster can be removed according to the performance impact based on some sufficient test data.

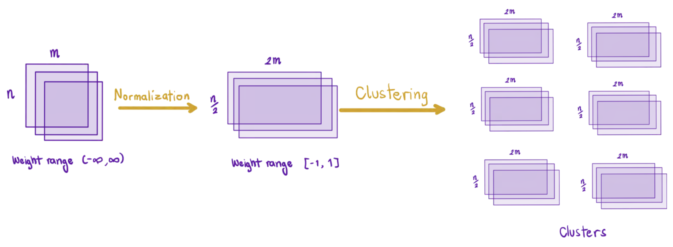


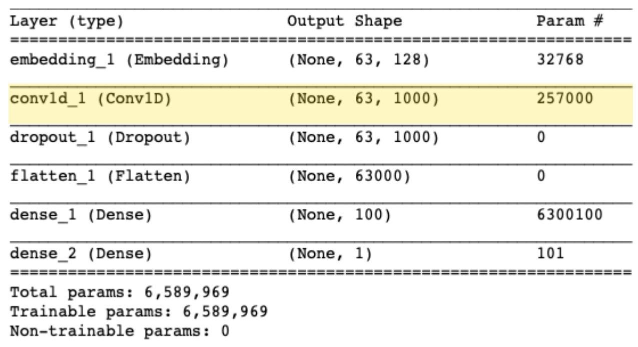
Figure 3: The overall procedure of the proposed methodology prior gradual removal in each cluster.

Experiment

In this section, we aim to evaluate the proposed deep model compression method using a pre-trained DGA classifier (i.e., a CNN model) and sufficiently large test data.

I. *Model Information*

The dataset and the initial uncompressed model [10] are provided by the Data Science team from Infobox company. The detailed model architecture is illustrated in Figure 4. It has (1) an embedding layer, (2) a convolutional layer, (3) a dropout layer, and (4) two dense layers. It can be observed that this model accepts a domain name in the length of 63 characters, where a domain name's Second Level Domain (SLD) name is used as the input. It should be noted that not every domain name's SLD is of length 63. Therefore, zeros are padded in front to make the same length of input.



```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 63, 128)           32768
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 63, 1000)          257000
_________________________________________________________________
dropout_1 (Dropout)          (None, 63, 1000)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 63000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               6300100
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 6,589,969
Trainable params: 6,589,969
Non-trainable params: 0
```

Figure 4: Model Architecture

Then, each character is embedded to a new space, where the dimensionality is 128. More importantly, this pre-trained model contains 1,000 filters in the convolutional layer, which contributes a huge amount of parameters in the first dense layer after the flatten layer. Now, considering an example of compression by removing 500 filters, we can reduce half the number of parameters for both the convolutional and the 1st dense layer, and thus reduce about 50% of the model size.

The provided Qname8 dataset contains 27 millions of possible malicious and 27 millions possible benign domain names. The domain names are represented using a total of 75 columns, but we are only interested in the SLD name string representing only the first 63 characters. Any names longer than 63 characters are truncated, any shorter ones are padded with zeros. We take two subsets of domain names from this dataset as our test data, which are the 1st 100k and 2nd 100k domain names in Qname8. The basic statistics of these two subsets are summarized in Table I. It can be observed that each test data contains a certain number of malicious and benign domain names. The prediction accuracy of the pre-trained model on these two subsets are varying around 95%, which indicates two varied test data for fair evaluation.

TABLE I
BASIC INFORMATION OF THE TWO DIFFERENT SUBSETS OF TEST DATA

Subsets of test data	Performance (without any removal)	#malicious (1) and #benign (0) samples
1st 100k of Qname8	97.544 %	54,432 (1) and 40,504 (0)
2nd 100k of Qname8	94.726%	35,725 (1) and 57,256 (0)

II. *Weight Normalization and Clustering*

As mentioned, the convolutional layer (see the yellow-highlight in Figure 4) has a total of 1,000 filters, each of size 2 × 128. There is no filter whose weights are all zeros, which empirically confirms our statement in the methodology. The weight distribution looking at all filters is summarized in Figure 5. In summary, the weights of filters are in the range of [-1.49, 1.40] with an average weight of -0.01.

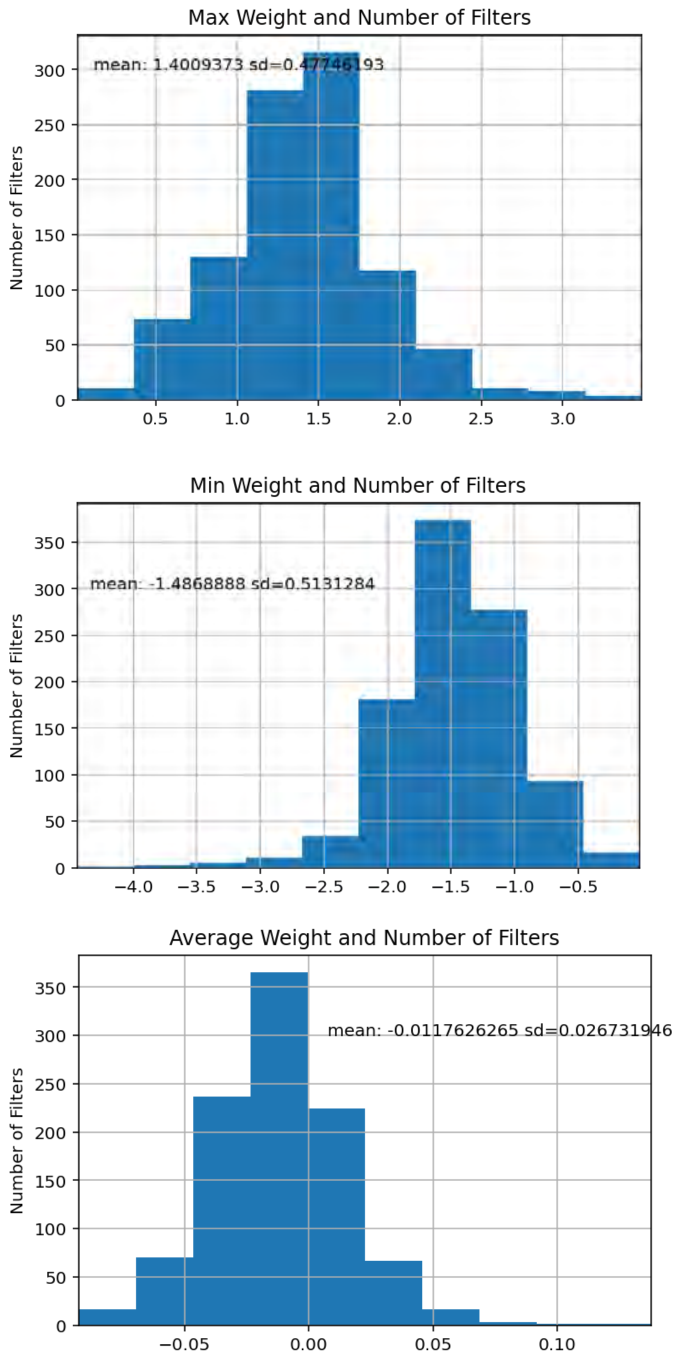


Figure 5: The weight distributions

To avoid any potential bias due to varying density and dominant values of weights among the filters, the weights of the convolutional layer are normalized to be in range [-1, 1]. Afterward, each filter is flattened to be in a vector form size 2 × 128 to be 1 × 256 feeding into the clustering algorithm. The normalized weights are then clustered using HDBSCAN algorithm to identify similarities among the filters.

We perform HDBSCAN over the 1,000 filters with default parameters, where Euclidean is used as a metric with at least 2 filters in each cluster. The clustering algorithm results in a total of 6 clusters. It starts with a cluster group of -1 to be a cluster of outliers. Cluster 3 comes out to be the *SuperCluster* as it contains the most number of filters in the cluster. The rest of the clusters have only 2 filters in each cluster. Table II summarizes the number of filters in each cluster.

TABLE II
CLUSTERS STRUCTURE SUMMARY

Cluster Number	Number of Filters
-1	228
0	2
1	2
2	2
3	764
4	2
Total	1,000

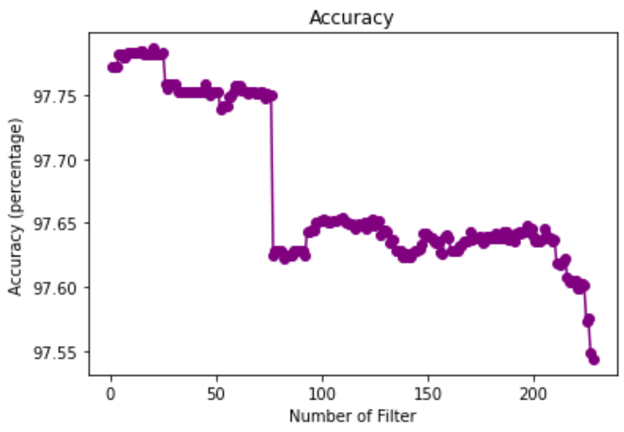
III. *Cluster Investigation*

After the clusters are identified, as mentioned in the proposed methodology, we sort the filters in each cluster according to the absolute sum of the weights as the importance score. The higher of the sum indicates the greater importance of the filter. In this subsection, we investigate for each cluster on how removing each filter within a cluster is going to affect the prediction performance on the test data.

With similar filters grouped in each cluster, each cluster allows us to remove certain redundant and less important filters to favor model size reduction, while still being able to preserve the model's effectiveness in malware detection. In the following subsections from A through C, we perform a series of fine-grained experiments to evaluate the model's performance change trends when the number of filters kept in each cluster is changed according to their importance. Specifically, we evaluate the model's prediction accuracy on each test data when the number of filters kept in each cluster is increased one by another. That is, the most important one is kept at first and then the next important one is added until all filters are kept.

A. Cluster -1 (Outliers)

As we know, cluster -1 should be a cluster of outliers and no correlation is expected. Figure 6 shows the accuracy change on both test data when the number of filters kept in cluster -1 is increased. It can be observed that the 1st 100k (top) and 2nd 100k (bottom) of the Qname8 data are showing opposite trends in accuracy although the same sequence of filters are kept. Therefore, it is not desired to remove any filters in cluster -1 for compression.



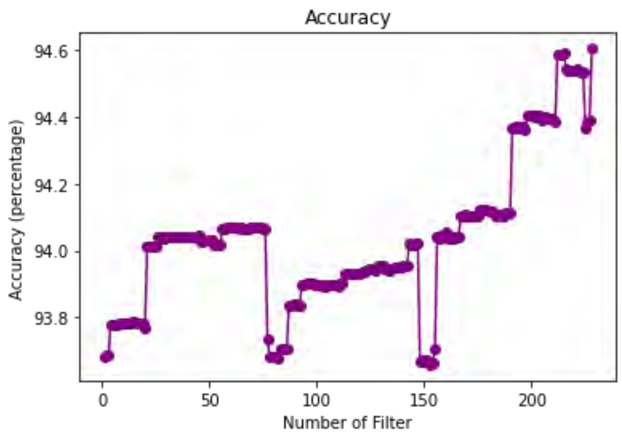


Figure 6: Cluster -1's model prediction shows opposite trends among the two test data when keeping filters with regards to their importance score (i.e., the x-axis shows the number of filters kept).

B. Clusters 0, 1, 2, and 4 (small clusters with 2 filters)

According to the cluster structures in Table II, cluster 0, 1, 2, and 4 have only 2 similar filters being grouped in each cluster. We find that removing one less important filter in any of these clusters results in very slight change in model's prediction accuracy on both test data. For example, removing the less important filter in cluster 1, 2, or 4 will not change the model's performance as shown in Figure 7 for cluster 1, and that of cluster 0 will only decrease the accuracy by about only 0.004%. Therefore, it demonstrates that groups of filters discovered by clustering can be used to find redundant filters, where less important ones can be removed for model compression.

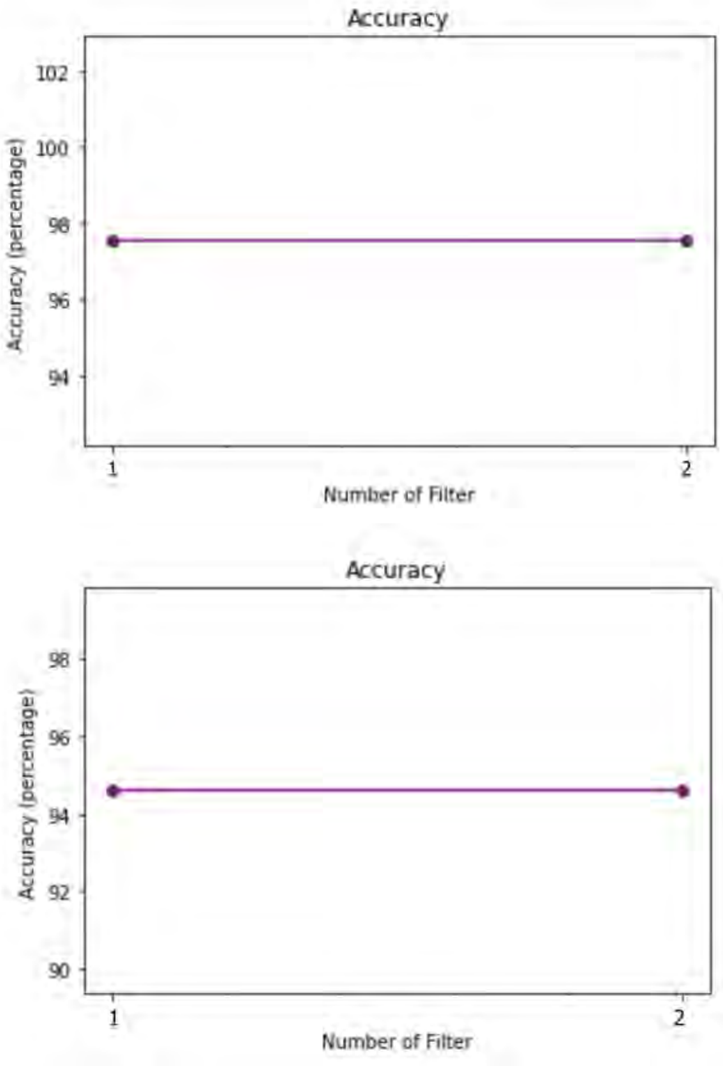


Figure 7: Cluster 1's model prediction changes trend when keeping filters with regards to their importance score (i.e., the x-axis shows the number of filters kept).

C. Cluster 3 (SuperCluster)

As cluster 3 is one of the largest clusters, it contains a total of 764 filters as shown in Table II, we refer to this cluster as *SuperCluster*. With the similar fashion as the rest of the cluster, SuperCluster's filters are kept in the same sequence with regards to their importance score.

Figure 8 illustrates the model's prediction performance changes when the number of kept filters is increasing for both test data. It can be observed that the accuracy is generally increasing for the 1st 100k, while that of the 2nd 100k has some fluctuation. The general increasing trend indicates that our importance score using the sum of absolute weights is not ideal but useful.

Interestingly, we can observe that by keeping about 450 filters for the SuperCluster (i.e., removing about 314 filters), we can have the same prediction accuracy for the 1st 100k as the original model without reduction, while surprisingly increasing the prediction accuracy for the 2nd 100k. Furthermore, we will not sacrifice the model's performance too much (at most 1% drop) if keeping only half of the filters in the SuperCluster (i.e., removing 382 filters). Therefore, we propose to remove 382 filters from the SuperCluster to reduce the model size.

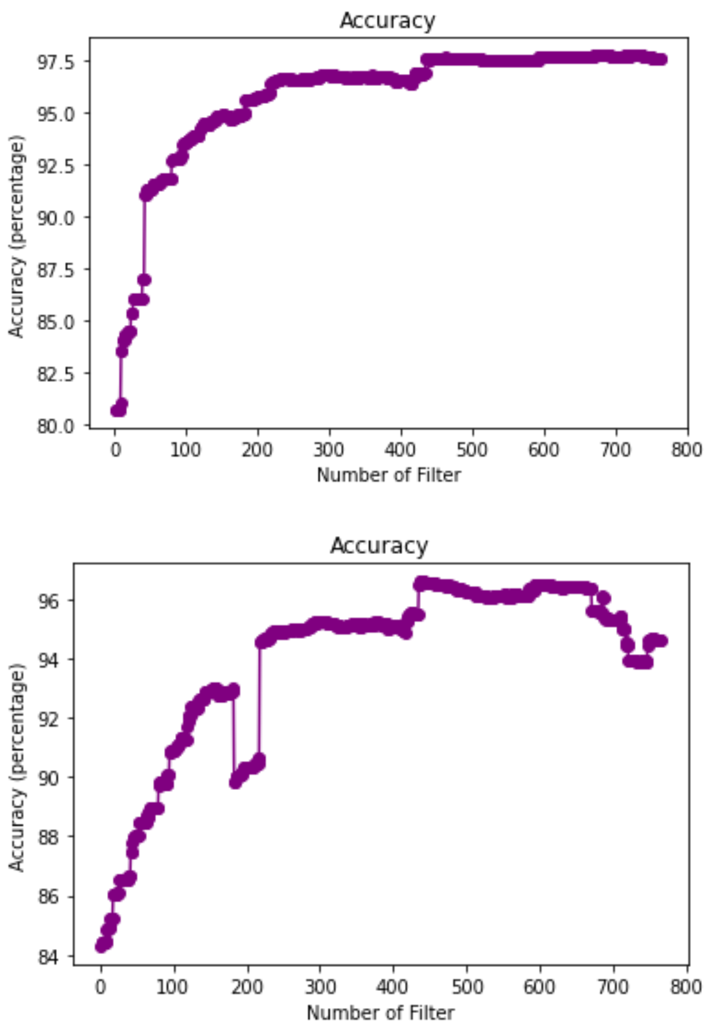


Figure 8: SuperCluster's model prediction illustrates an increasing trend when keeping more filters with regards to their importance score (i.e., the x-axis shows the number of filters kept).

IV. Model Size Reduction

Based on the investigation in the above subsection, we propose to reduce the model size by removing the less important filters in each small cluster (i.e., cluster 0, 1, 2 and 4) and removing 382 less important filters in the SuperCluster (i.e., cluster 3). The following Table III shows the performance summary of the original model and its reductions on each test data. It can be observed that the original model has prediction accuracy of 97.54% for the 1st 100k and 94.73% for the 2nd 100k. After we remove the SuperCluster in half with the 382 filters being disregarded, the accuracy for the 1st 100k drops less than 1% while that for the 2nd 100k in fact increases slightly. By further removing the 4 less important filters from each small cluster, we can keep almost the same level of prediction performance, in which we can remove 386 filters in total. In summary, this is about 40% of model size reduction considering removing the corresponding channels in the dense layer.

TABLE III
PERFORMANCE SUMMARY

Model	**Performance 0 - 100K**	**Performance 100K - 200K**	**# Filter Removed**
Original	97.54%	94.73%	0
Removed Half SuperCluster	96.67%	95.18%	382
Keep the highest weight in 0,1,2,4 and half SuperCluster	96.65%	95.18%	386

To further demonstrate our proposed method for model compression, we show how the choices of other clusters (i.e., except the SuperCluster) will affect the model's performance when gradually changing the SuperCluster in Figure 9. The four choices included for discussion are 1) keep all other clusters, that is, no reduction in other clusters; 2) keep cluster -1 (i.e., outliers) and the highest important filters in small clusters (i.e., cluster 0, 1 2, and 4), that is, our final reduction; 3) remove the whole cluster -1 while keeping the highest important filters in small clusters; and 4) remove all other clusters. Figure 9 demonstrates our final reduction illustrated by the green line. More importantly, it can be observed that we can further remove all other clusters if we can keep a few more filters in the SuperCluster without sacrificing the performance too much. This further demonstrates the effectiveness of the proposed weight-based model compression method.

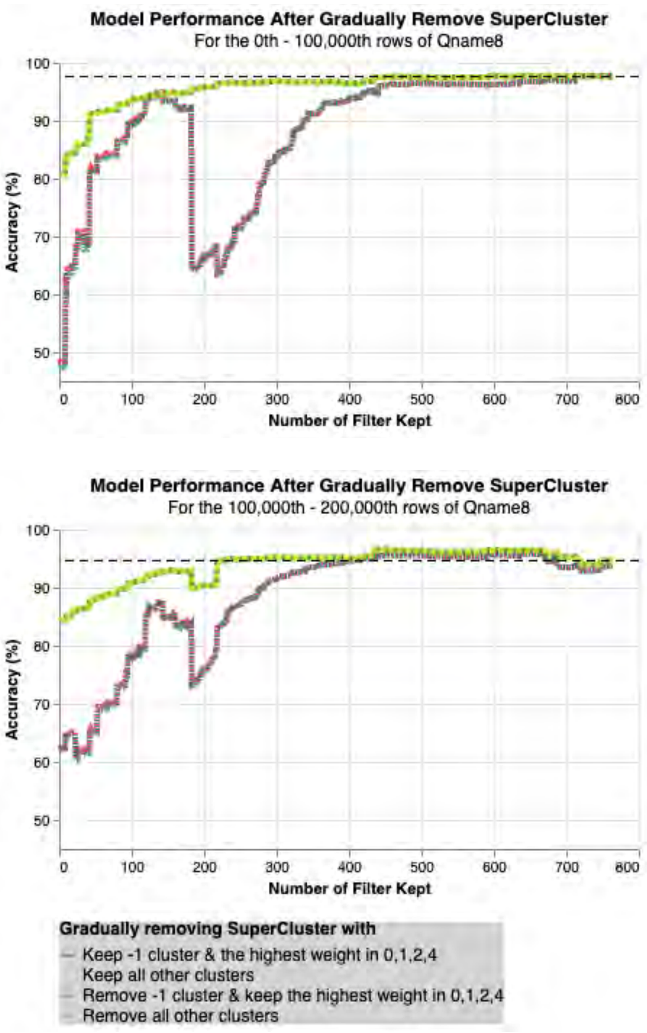


Figure 9: Performance summary of gradually changing SuperCluster on different choices of other clusters (the black dash line indicates the original model's accuracy).

CONCLUSION AND FUTURE WORK

The widespread use of many malware families generated by DGAs to establish command and control (C&C) connection by the threat actors has made it difficult for defenders to prevent the attacks. DGAs is developed specially so that malwares can generate a list of domains for the attackers to use during malware attacks. In the past, software security can simply block and take down the malicious domains that are often listed in the blacklist. However, the algorithm allows the attackers to switch to a different unknown domain to attack, making it more difficult for security measures to detect these malicious domains. Not to mention that the defenders have to go through the process working with ISP and decode the algorithm to take down the malicious domain one by one. Thankfully, the convolutional neural network (CNNs) have enhanced the capability for predicting incoming domain features from the machine learning framework and the ground truth datasets. Yet, such models require more storage space and power than the majority of affordable IoT devices can handle, which everyday users have access to.

Introducing our proposed methodology, instead of pruning neuron by neuron which is time consuming, we introduce filter by filter pruning incorporated with normalization of the weights and HDBSCAN clustering to identify similarities among filters for compression. The results prove for the given data that we can retain the same prediction model performance compared to the uncompressed model, while being able to remove about 40% of the model size considering that the dense layer will also be further reduced.—However, we did find that the importance score based on the sum of absolute weights is not an ideal importance indicator, where the monotonicity cannot be observed. Moreover, a special property of the domain names has been ignored in this work, that is, we have a finite number of characters that are used in domain names. Therefore, we have a finite set of bigrams. This information can be used to do activation-based model compression, which will be an interesting future direction to do deep model compression in malware detection.

References

[1] Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both Weights and Connections for Efficient Neural Network. ArXiv, abs/1506.02626.

[2] Blalock, D.W., Ortiz, J.G., Frankle, J., & Guttag, J. (2020). What is the State of Neural Network Pruning? ArXiv, abs/2003.03033.

[3] Lee, N., Ajanthan, T., & Torr, P.H. (2019). SNIP: Single-shot Network Pruning based on Connection Sensitivity. ArXiv, abs/1810.02340.

[4] Gale, T., Elsen, E., & Hooker, S. (2019). The State of Sparsity in Deep Neural Networks. ArXiv, abs/1902.09574.

[5] Molchanov, D., Ashukha, A., & Vetrov, D. (2017). Variational Dropout Sparsifies Deep Neural Networks. ArXiv, abs/1701.05369.

[6] McInnes et al, (2017), hdbscan: Hierarchical density based clustering, Journal of Open Source Software, 2(11), 205, doi:10.21105/joss.00205.

[7] Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., & Dagon, D. (2012). From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. USENIX Security Symposium.

[8] The hdbscan Clustering Library. The hdbscan Clustering Library - hdbscan 0.8.1 documentation. (n.d.). https://hdbscan.readthedocs.io/.

[9] Sivaguru, Raaghavi & Choudhary, Chhaya & Yu, Bin & Tymchenko, Vadym & Nascimento, Anderson & De Cock, Martine. (2018). An Evaluation of DGA Classifiers. 5058-5067. 10.1109/BigData.2018.8621875.

[10] Yu, Bin & Pan, Jie & Gray, Daniel & Hu, Jiaming & Choudhary, Chhaya & Nascimento, Anderson & De Cock, Martine. (2019). Weakly Supervised Deep Learning for the Detection of Domain Generation Algorithms. IEEE Access. 7. 51542-51556. 10.1109/ACCESS.2019.2911522.

[11] Li, Yi & Xiong, Kaiqi & Chin, Tommy & hu, Chengbin. (2019). A Machine Learning Framework for Domain Generation Algorithm (DGA)-Based Malware Detection. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2891588.

[12] Luo, Jian-Hao & Wu, Jianxin & Lin, Weiyao. (2017). ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression.

[13] Berkhin P. (2006) A Survey of Clustering Data Mining Techniques. In: Kogan J., Nicholas C., Teboulle M. (eds) Grouping Multidimensional Data. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-28349-8_2

[14] Xu, Rui & Wunsch, Donald. (2005). Survey of Clustering Algorithms. Neural Networks, IEEE Transactions on. 16. 645 - 678. 10.1109/TNN.2005.845141.

[15] Jain, A.K., Murty, M.N., & Flynn, P. (1999). Data clustering: a review. ACM Comput. Surv., 31, 264-323.