Explaining AI for Malware Detection: Analysis of Mechanisms of MalConv

Shamik Bose
Department of Computer Science
Florida State University
Tallahassee, Florida
bose@cs.fsu.edu

Timothy Barao
Department of Computer Science
Florida State University
Tallahassee, Florida
barao@cs.fsu.edu

Xiuwen Liu
Department of Computer Science
Florida State University
Tallahassee, Florida
liux@cs.fsu.edu

***Abstract*—In recent years, machine learning has been used in a very wide variety of applications and malware detection is no exception. Because of its fast and widespread adaptation to various diverse fields, machine learning can, and often is, treated as a black box. The disadvantage of doing so is that the decisions can often be difficult to interpret which can be especially challenging in the field of malware detection. Training deep neural networks also requires a vast amount of data from all classes which can be quite challenging in the field of proprietary software, specially for smaller research labs. In this paper, we introduce a framework which interpolates between samples of different classes at different layers to see how a deep network architecture generalizes to samples that are not in the training set, explaining the results of deep networks in real-world testing. Using this framework, we attempt to demystify the mechanisms behind the MalConv architecture [1] by analyzing the weights and gradients of multiple layers in its architecture and decipher what the architecture learns by analyzing raw bytes from the binary. For this architecture, our analysis shows that the network assigns much higher weights to specific portions of the executable Indicating that these portions contribute significantly more to the classification than other portions of the executable. Through the proposed framework, we can explain the mechanisms behind machine learning algorithms and explain their decisions better. In addition, the analyses will allow us to look inside existing networks without training them from scratch.**

***Index Terms*—Cybersecurity, Malware Detection, Explainable AI, Machine Learning, Neural Network**

I. INTRODUCTION

Neural networks, and machine learning in general, have become the default choice as a solution to tasks that were previously considered to be generally unsolvable either in reasonable amounts of time or with high enough accuracy on real-world test situations. These include object detection [2], image classification [3], [4] and natural language processing [5], [6]. Many of the solutions provided work very well in real-world situations, but the decisions taken by these systems are not always easily interpretable. This has limited the use of machine learning in fields like malware analysis where the reasoning for a decision is just as important as the decision itself. In addition, the data requirements for training these networks are immense and while large datasets are becoming more available, they are not available for proprietary software, specially to smaller research labs. We introduce a framework to analyze the different steps of a trained neural network and provide insight into the reasoning for its decisions.

Many of the tasks that are solved by neural networks are presented as classification problems and one of the most important classification tasks in the field of security is being able to differentiate between malware and goodware. Malware attacks were estimated to have cost more than 2.7bn in damages in 2018 [7]. To combat malware, there have been numerous approaches over the years, including but not limited to antivirus programs, virtual environments, dynamic analysis and symbolic analysis [8], [9]. Each one of these has its share of advantages and shortcomings. Antivirus programs, which are most widely used, use signatures which can be very specific if they want to reduce their false negatives or extremely generalized to reduce false positives. To get around this, pattern recognition and machine learning have been used for the malware detection task since 2001 [10]. All these methods had to deal with creating features from the executable, be it external library functions or DLLs and strings [10], n-grams of byte-codes [11], [12] or a combination of all or some of them [13], [14]. Machine learning methods can automatically learn discriminative features from raw inputs, but all these methods require manual feature extraction and a significant amount of domain knowledge which is complicated, time-consuming, non-trivial and in some cases, they might not capture the best features required for classification. The MalConv [1] architecture aims to remove these shortcoming by providing the raw bytes from a file as input to a convolutional neural network which then classifies the given binary as either malicious or benign. The core idea is that the network will be able to successfully extract discriminative features like they have for other tasks like image classification [15], speech and signal processing [16] and text understanding [17].

Additionally, the global interest in machine learning and its applications has caused a huge number of models being trained for the same tasks. This has incurred a substantial energy cost. For instance, training a big Transformer model with Neural Architecture Search has the same carbon footprint as five family cars over their entire lifetime [18]. Similarly, training the base BERT model [5] can cost up to $12,500 [18]. If we

This research was funded by a SEED Grant from Cyber Florida

978-1-7281-6026-2/20/$31.00 ©2020 IEEE

have access to trained networks and we can look inside them to understand their workings, then we can tweak them to our requirements without needing to retrain them, thus avoiding the cost and carbon impact.

In this paper, we analyze the MalConv architecture using our framework in an attempt to understand how the system learns to discriminate between malicious and benign executables using raw bytes. Our contributions include gradient analysis at various stages of the trained network to see how the system assigns weights to different portions of the executable, analysis of the filter weights and their activations for different files. Finally, we also show how generalization in neural networks can be achieved though linear interpolation between samples.

The paper is organized as follows. Section II introduces some work that has been carried out in this field followed by their results and Section III describes the architecture of the MalConv [1] model, the open-source model (*emberMalConv*) [19] we use for our analyses, our data and our testing results compared to MalConv [1] and emberMalConv [19]. The next section describes the various analyses carried out by our framework, the findings and our interpretations of it. In the last section, we conclude with a summary of our findings and describe future plans for our framework and how it can aid the machine learning community move towards more explainable AI.

II. RELATED WORK

Limited work has been carried out in the area of analysis of deep networks used for malware classification.To the best of our knowledge, there are two notable studies in this field [20], [21]. The first of these, explained in Section II-A looks at the byte activation [20] of the network although it uses a different network [22] than MalConv [1]. The other notable work [21], detailed in Section II-B uses the same network that we used, but their analyses are different.

A. Byte Activation Analysis

Byte-activation analysis [20] looks at the response of a network to a given input and maps the activations of the various bytes. The network used in this case is the CNN proposed by FireEye [22] and is shown in Figure 1. In this architecture, three networks are trained with a combination of different parameters (dropout on/off) and training set sizes (7M files vs 15M files). For the rest of the paper, we will use the following names for the corresponding models:

1) **Baseline**: This model was trained on 7M samples with an equal split between malicious and benign files
2) **Large**: This model was trained on 15M samples with an 80/20 split between benign and malicious files
3) **Dropout**: This model was also trained on the same dataset as "Large", but the network included dropout layers

The analyses were carried out on three different levels and their results are as follows:

- **Byte relationships**: The analysis carried out at this level was a hierarchical clustering using HDBSCAN [23]. The two clusters created are of bytes that can be easily interchanged and ones that would cost considerably more to swap (outliers). Numbers of outliers grows from the Baseline (13), to the Large (31), to the Dropout (158) model. The outlier bytes for the Baseline and Large models are typically registers and the `call` instruction.
- **Low-level features**: The analysis at this level shows that the majority of activations happen on a single filter, import name and common instruction features are instrumental to the classification across all three models. For both malware and benign files, the largest number of activations are around the beginning of the file in the Baseline and Large models.
- **High-level features**: The analysis of the end-to-end features shows that a number of features are extracted at this level which are routinely used by analysts. These include detection of the Rich header present exclusively in Microsoft binaries and checksum validation (0 in Baseline and Large models, valid value in Dropout)

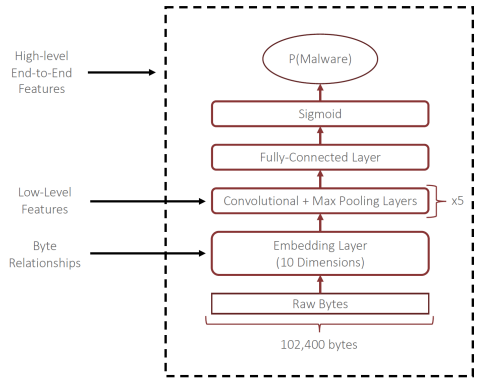

Fig. 1. FireEye CNN for malware classification [20]

B. Adversarial Vulnerability

Adversarial examples are defined as inputs having small, imperceptible perturbations [24], [25] which cause a neural network to misclassify it. Neural networks designed to detect malware automatically are not immune to attacks like these and that is the focus of the second approach [21] to analysing the MalConv architecture. This approach analyzes the network to create adversarial examples which are misclassified by the network. A similar approach has also been tried before [26], [27], but the analysis carried out in this work is in direct contrast to the results from [20]. While that makes it interesting, it also allows our research to reconcile these two seemingly disparate findings. For their analyses, the authors used a technique called *feature attribution* [28] where the most significant features to each contribution are identified by calculating their integrated gradients [28] against a baseline signal. For their experiments, Demetrio et al. [21] chose an empty file as a baseline. The main findings from [21] are as follows:

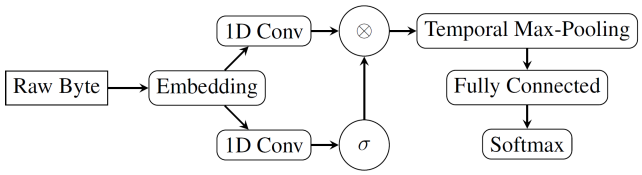

Fig. 2. Architecture for the MalConv model [1]

- Bytes from the DOS header are used for classification, which is strange since modern operating systems don't read the DOS header except for the magic number 'MZ'.
- The highest gradient values are observed for the COFF and other headers, with other sections like *.text* and *.rsrc* with minimal gradients.

As with both previous works [20], [21], our framework allows us to plot the gradients of the network layers as a response to an input signal in order to explain the decision boundary of the network. However, our framework also allows us to interpolate between samples of different classes in order to explain the underlying reason for neural networks to be able to generalize from training samples to unseen data. In addition, we are also able to run architecture-specific analyses as shown in Section IV-C.

III. THE NETWORK MODEL

In this section, we describe the original MalConv model, the emberMalConv model we used for our analyses and the accuracy and F1-scores achieved on our dataset.

A. MalConv Architecture

The model we analyse in this paper is called MalConv [1]. It was introduced in 2018 as an end-to-end convolutional neural network which can discriminate between malicious and benign files reliably on the basis of raw bytes alone without any feature engineering. The architecture of the network is shown in Figure 2. The embedding layer is learned during training and used as a lookup thereafter. It produces an 8-dimensional embedding for each byte, which are then fed into two convolutional layers. Each of the convolutional layers have 128 filters of size 500 and stride 500, meaning there is no overlap between two convolutions. The outputs of the two convolutional layers are then multiplied, the result of which is forwarded to a max-pooling layer followed by a fully-connected layer. The final softmax layer produces a prediction value (p) between 0 and 1. If $p \geq 0.5$, then the file is classified as malicious, otherwise it is classified as benign.

B. The emberMalConv Model

According to the authors of MalConv [1], their model cannot be trained on a single GPU due to the massive memory requirements. Without access to the required hardware, we were unable to train this model from scratch and the authors do not have a trained model for researchers to use. Instead, we used an open-source implementation of the model called emberMalConv [19]. There are some minor differences between the two models, which are as follows:

- It uses a special padding byte. The authors of MalConv did not specify if they used a special padding byte.
- It uses a smaller batch size (100 vs 256). The smaller batch size is supposed to improve training stability and generalization performance [29], but the improved generalization might affect accuracy results between the two models. Additionally, generalization might not be a desirable quality in this task as we explain in Section IV
- While MalConv allowed files up to 2MB, this architecture only allows files up to 1MB due to memory limitations on their GPU. Smaller file sizes could lead to different features being learned across the file
- Maybe most significantly, the model was trained on the EMBER dataset [19], which is a fair bit smaller than the dataset used to train MalConv. The different dataset used for training will lead to different features being learned by the different layers

TABLE I
OUR ACCURACY AND F1-SCORES AGAINST MALCONV [1] AND EMBERMALCONV [19]

Dataset	*Accuracy*	*F1-Score*
Reduced	**87.1**	**87.3**
MalConv [1]	94.0	98.2
EMBER MalConv [19]	92.2 (97.3 with 1% FP)	-

C. Data and Testing Results

For our testing, we used a dataset of around 700 files, all of which are under 1MB as required by the model. It is a nearly even split of files between the two classes (357 benign, 332 malicious). The benign files are taken from Windows System32 folders and the malicious files are a subset of the MALICIA dataset [30]. One of the biggest challenges we faced while carrying out our analysis was the lack of a dataset for benign executables. While we did have a large dataset of malicious executables, we didn't want to skew our results towards a specific class, so the sizes of both datasets were kept very similar. In Table I, we present the results we achieved on our dataset ("Reduced"). Our scores are highlighted in bold. As we can see, the results on the small dataset are a few points short of the reported findings from MalConv [1] and emberMaclConv [19], but this can be attributed to the size of our dataset. In the next section, we present our analyses and findings using our framework.

IV. ANALYSES

To analyze a neural network, common approaches involve gradient analysis of the output with respect to different input signals and/or intermediate representations to understand the decision boundaries between different classes. This section describes the various analyses we ran using our framework

on the emberMalConv [19] model to explain the mechanisms behind its discriminative decisions. Before we do that, it should be noted that this framework can easily be extended to any neural network. We chose the task of malware detection because it is a significant strain on cybersecurity personnel to hand-engineer features for anti-virus signatures, sometimes requiring up to 10 hours of work [31] per family of malware. This domain knowledge requirement allows malware authors to simply obfuscate their code using a custom packer to avoid detection from traditional detection methods. A machine learning system should be able to avoid that pitfall and learn beyond byte sequences to understand mechanisms based on their success in other fields using unsupervised systems [3], [15]. MalConv [1] was one of the most recent systems to apply machine learning to this task. Additionally, it requires no additional parsing like PE Imports or strings and this is the reason we chose to analyse this specific network.

A. Gradient analysis

Using our framework, the first thing we wanted to analyse was the gradients of each block with respect to the output. While there are definite spikes in the first block which contains the header, it is not significantly higher than the gradient of other blocks in the file. This is in contrast to the results from [21] where the gradients of the headers are found to be much higher than the other portions of the file. We present the results of our gradient analysis in Figures 3 and 4 below. In the figures, the sections of the graph in red indicate the headers and the blue indicate the other sections of the file. As we can see in the first graph in Figure 4, the packed files can sometimes cause the disassembler to misread the length of the header. We only show a small subset of our dataset spanning all file sizes, but this trend was noticed across the entire dataset, showing that all portions of the file contribute to the classification even though the gradients for the headers are consistently high. Additionally, we find the gradients of the output with respect to the outputs of the max-pooling layer. These are shown in Figures 5 and 6. We noticed a constant peak in all the files at filter #45. Since this is a framework to analyse all kinds of neural networks, we didn't analyze what that specific filter was checking as that would make it specific to the malware detection domain, but that can be achieved quite easily.

B. Interpolation between samples

The next analysis from our framework involves interpolating between correctly classified samples from the two classes to find a decision boundary between the two classes in an effort to explain the discriminative abilities of the network. To pick the files, we ranked the files in order of their predictions i.e. closest to 1 was the most malicious and closest to 0 was the most benign. We then interpolated between these files with a step size of 0.05, leading to a total of 20 samples between each samples, counting the original samples. The results of the interpolation can be seen in Fig. 7. None of the 225 possible combinations go back and forth across the

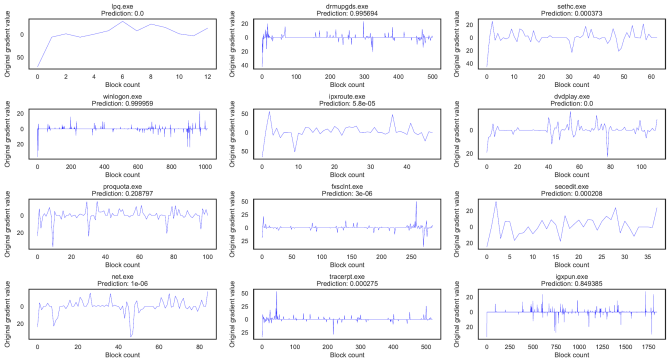

Fig. 3. Gradients of *embeddings* for a subset of the *benign* files

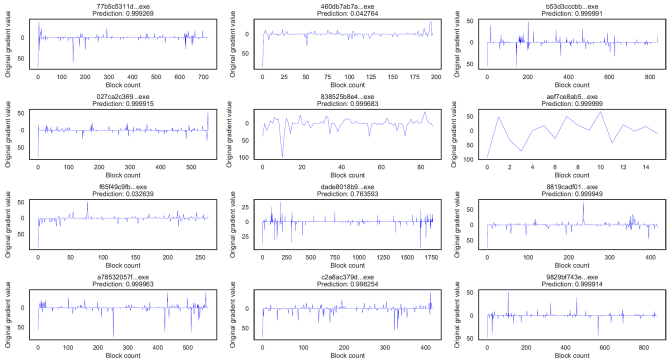

Fig. 4. Gradients of *embeddings* for a subset of the *malicious* files

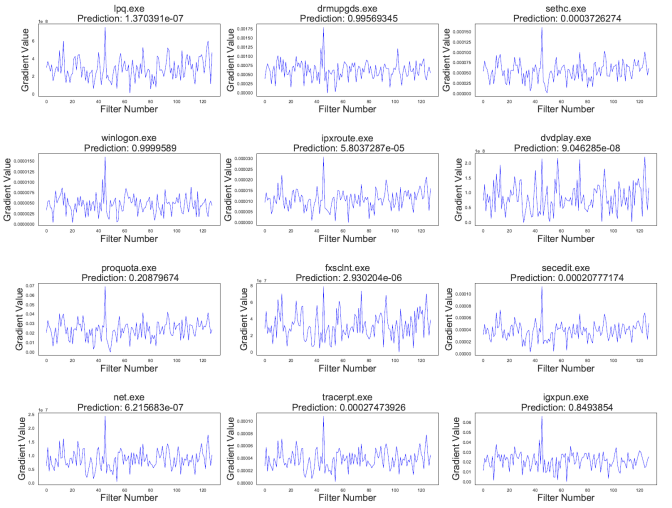

Fig. 5. Gradients of *max-pooling outputs* for a subset of the *benign* files

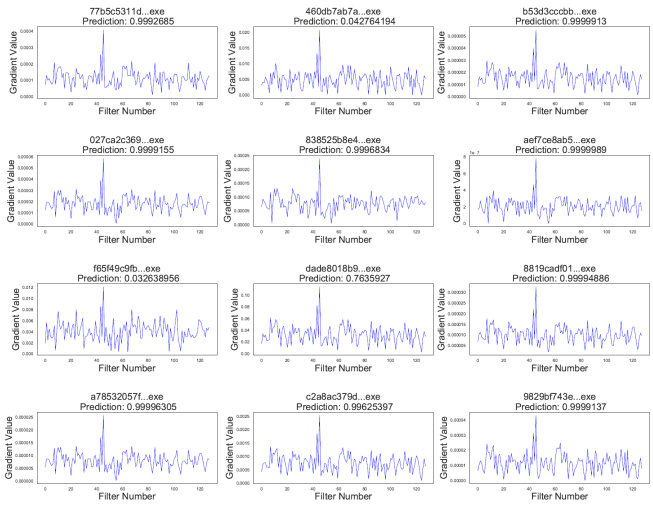

Fig. 6. Gradients of *max-pooling outputs* for a subset of the *malicious* files

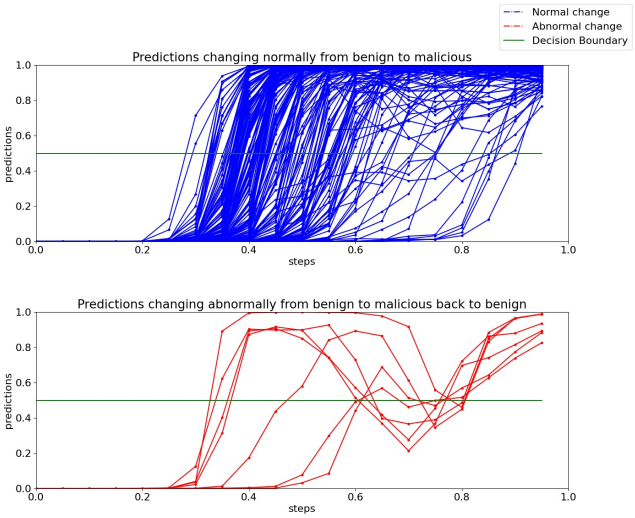

Fig. 8. Trend for decision boundary between random interpolated samples from each class

decision boundary, showing that there's an increasing trend between benign files and malicious files when it comes to predictions about their maliciousness. It also strongly hints that linearly interpolating between samples could be how machine learning algorithms generalize to unseen inputs. In addition

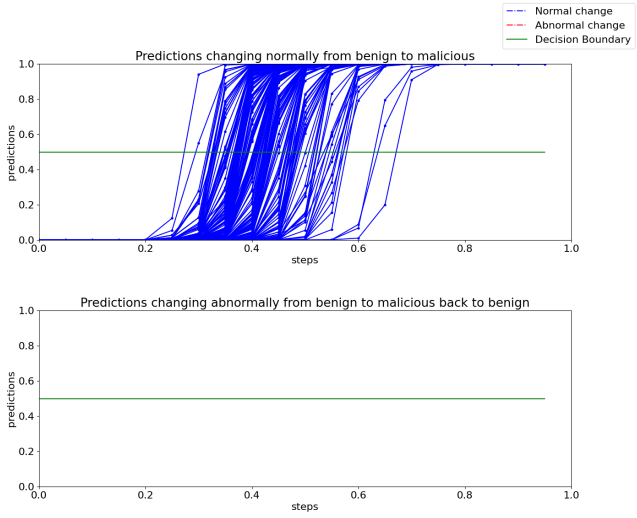

Fig. 7. Trend for decision boundary between interpolated samples from most benign to most malicious

to interpolating between the strongest representatives of each class, we also interpolate between random samples from each of the classes shown in Fig. 8. As we can see, only a small percentage (approximately 3%) of the interpolated samples go back and forth across the decision boundary before settling on a class before reaching the malicious file. To further expand on this idea, we interpolate between two files and run the gradients analysis from Section IV-A on these samples. To present our results in a readable manner, we use a step size of 0.125, resulting in 8 interpolated samples. We ran the analysis on multiple pairs of files and present the results from one of the pairs here in Fig 9. For similar sized files, the trend is that the predictions change very sharply instead of smoothly, as we had expected. We should note that there is a max-pooling

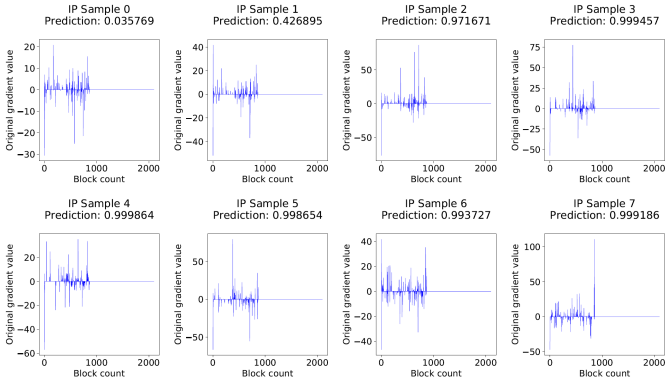

Fig. 9. Gradients of *blocks* in interpolated samples

layer after the convolutions which is not a linear layer. Using the framework, we extract the results of the max-pooling layer and interpolate between those outputs for pairs of files. These results are shown in Figure 10. As we can see, the change in predictions is much more linear at this point. We present the results for only one set of files, but this holds true for all the pairs we tested. The peak value of the filters is seen in filter 45, which is also true for all the samples we tested. It also aligns with other results [20] where the vast majority of activations also happened on a single filter.

C. Filter Correlation

Following our analysis showing the highly prevalent activations for a single filter on all the samples (true and

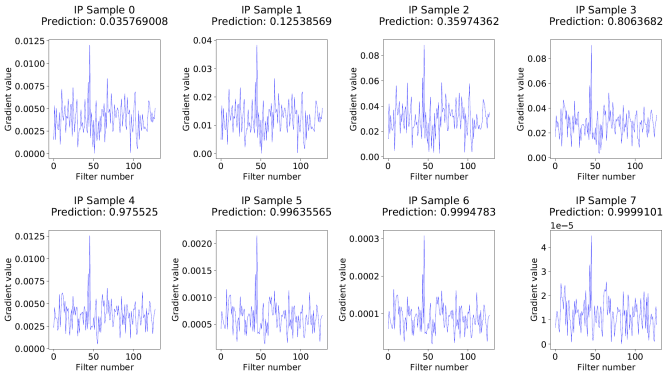

Fig. 10. Gradients of *max-pooling outputs* in interpolated samples

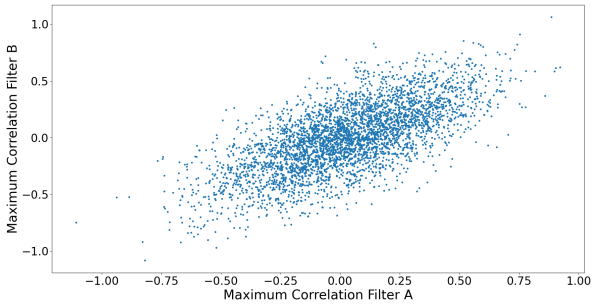

Fig. 11. Maximum correlation between Filters A and B

interpolated), we wanted to see the correlation between and within the two sets of filters A and B in order to see if they were learning similar or disparate things. These results are shown in Figures 11 and 12. As we can see from the figures, the filters A and B are strongly correlated, leading us to believe that the filters are learning mostly similar features. We wanted to see if the filters were correlated within themselves, so we generated correlation heatmaps for the filters themselves. These are shown in Figures 13 and 14. As we can see, the filters themselves are not correlated, leading us to believe that each filter learns a distinct feature, only one of which is strongly activated. It should be noted that this analysis is architecture-specific and while it is part of our framework, it cannot be applied to all networks.

D. Additional Experiments

Keeping in mind the correlative properties of the filters, we wanted to run an experiment where the filters are replaced, one at a time, by the other one. Since the filters are identical except for their activation functions, this involved copying a model from the original and then changing the weights of one filter with that of the other. We got some surprising results which are shown in Table II. The accuracy on our test set went up by about 4 points when we replaced B with A and dropped to about 50% when we replaced A with B. We denote these models with **A**→**B** (B replaced by A) and **B**→**A** (A replaced by B)

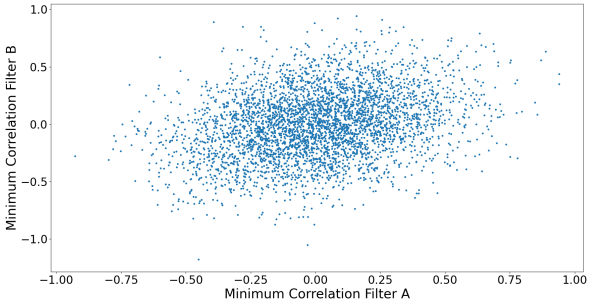

Fig. 12. Minimum correlation between Filters A and B

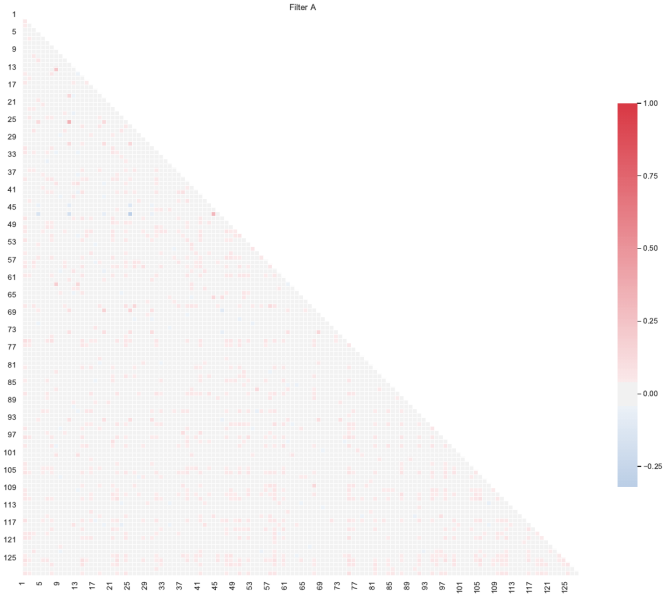

Fig. 13. Filter A Correlation heatmap

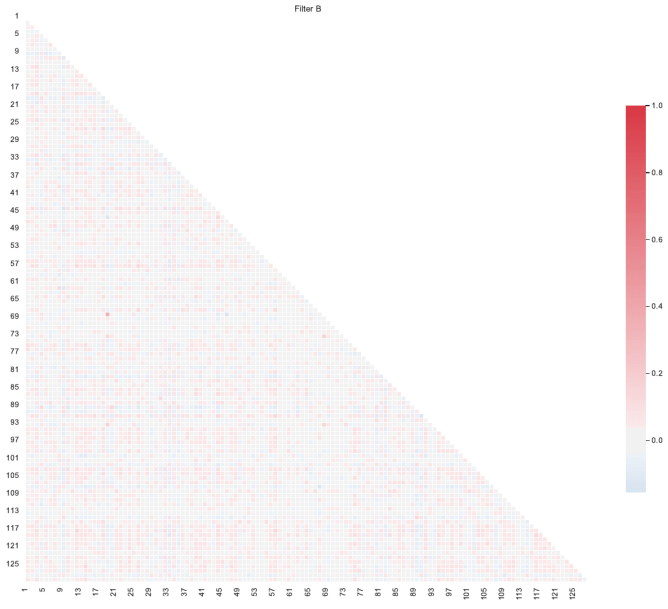

Fig. 14. Filter B Correlation heatmap

As we can see from Table II, B→A misclassifies almost every goodware sample, but correctly classifies nearly every malware sample. On the other hand, A→B improves on the false negatives, but has a higher false positive. Intuitively, it seems like Filter A is responsible for detecting the parts of the file which are goodware, but it is also a much more generalized filter whereas Filter B seems to be a very focused filter that seeks out the specific portions of a file that make it malicious. This seems to align with the finding from analysts that most parts of an executable are not malicious, but specific parts of it make it, or at least make it look, malicious. Another hypothesis for the improvement in the replaced model could be due to ineffective training. This could mean that the training reaches a local minimum region in the parameter space and the optimization algorithm can not escape or the training was not run for a long enough period of time. It also shows that the proposed technique can not only provide better insights how the classifier classifies different inputs, but also explore other solution regions with potential better performance.

TABLE II
ACCURACY AND F1-SCORES BY REPLACING FILTERS

Model	Accuracy	F1-Score	Misclassified Malware Count	Misclassified Goodware Count
Reduced	87.1	87.3	25	64
B→A	48.2	64.9	**2**	355
A→B	**91.2**	**90.7**	34	**27**

V. CONCLUSION AND FUTURE WORK

The findings from our analyses are able to reconcile the disparate findings from [20] and [21]. While there are definitely high gradient values at the beginning of the files indicating that the headers contribute strongly to the classification, there are also gradient peaks elsewhere (sometimes exceeding the header peaks) in the file, indicating that other portions of the file are also responsible for the classification results. Additionally, the two filters seem to learn two sets of features, one that allows it to generalize (Filter A) and another that focuses on the malicious aspects of the file (Filter B). This is in line with the findings from [20] which uses a different model [22] with overlapping filters. This brings up the question of how much information from a filter is retained even without any overlap between them. It is also possible that a single filter in the emberMalConv architecture is large enough to capture all the features that required multiple filters in the FireEye architecture [22].

The interpolation between samples shows that as we move from one class to another, the neural network classifies samples close to one class as that class even though the sample itself might not be a valid one. This could be the reason that neural networks are able to generalize well to samples that it has not yet encountered, but is close to some interpolated values between two classified samples.

We would like to emphasize that while we used our framework to analyze an end-to-end malware detection network, this framework can be used to analyze any other neural network which performs classification. By assigning and examining values of the outputs of various layers within a neural networks, we should be able to better interpret the results of a neural network and hopefully expand the areas of its application without retraining it from scratch and avoid incurring its associated energy and carbon costs.

REFERENCES

[1] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[2] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *Advances in neural information processing systems*, 2013, pp. 2553–2561.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[6] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[7] F. B. of Investigation Internet Crime Complaint Center, "2018 internet crime report," https://pdf.ic3.gov/2018_IC3Report.pdf.

[8] J. C. King, "Symbolic execution and program testing," *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[9] C. Cadar and K. Sen, "Symbolic execution for software testing: three decades later," *Communications of the ACM*, vol. 56, no. 2, pp. 82–90, 2013.

[10] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*. IEEE, 2001, pp. 38–49.

[11] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, vol. 7, no. Dec, pp. 2721–2744, 2006.

[12] I. Santos, Y. K. Penya, J. Devesa, and P. G. Bringas, "N-grams-based file signatures for malware detection." 2009.

[13] D. Gavriluţ, M. Cimpoeşu, D. Anton, and L. Ciortuz, "Malware detection using machine learning," in *2009 International Multiconference on Computer Science and Information Technology*, Oct 2009, pp. 735–741.

[14] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2015, pp. 11–20.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[16] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[17] X. Zhang and Y. LeCun, "Text understanding from scratch," *arXiv preprint arXiv:1502.01710*, 2015.

[18] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *arXiv preprint arXiv:1906.02243*, 2019.

[19] H. S. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," *ArXiv e-prints*, Apr. 2018.

[20] S. E. Coull and C. Gardner, "Activation analysis of a byte-based deep neural network for malware classification," 2019.

[21] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Explaining vulnerabilities of deep learning to adversarial malware binaries," *arXiv preprint arXiv:1901.03583*, 2019.
[22] J. Johns, "Representation learning for malware classification," https://www.fireeye.com/content/dam/fireeye-www/blog/pdfs/malware-classification-slides.pdf.
[23] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2013, pp. 160–172.
[24] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.
[25] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014.
[26] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 533–537.
[27] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," *arXiv preprint arXiv:1802.04528*, 2018.
[28] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3319–3328.
[29] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *arXiv preprint arXiv:1804.07612*, 2018.
[30] A. Nappa, M. Z. Rafique, and J. Caballero, "The MALICIA Dataset: Identification and Analysis of Drive-by Download Operations," *International Journal of Information Security*, vol. 14, no. 1, pp. 15–33, February 2015.
[31] A. Mohaisen, O. Alrawi, J. Park, J. Kim, D. Nyang, and M. Mohaisen, "Network-based analysis and classification of malware using behavioral artifacts ordering," *arXiv preprint arXiv:1901.01185*, 2019.