2020 25th International Conference on Pattern Recognition (ICPR)
Milan, Italy, Jan 10-15, 2021

Malware Detection by Exploiting Deep Learning over Binary Programs

Panpan Qi*, Zhaoqi Zhang*, Wei Wang* and Chang Yao†
*School of Computing, National University of Singapore, Singapore
Email: qipanpan@u.nus.edu, zhaoqi.zhang@u.nus.edu, wangwei@comp.nus.edu.sg
†Institute of Computing Innovation, Zhejiang University, China
Email: yaochang@zjuici.com

***Abstract*—Malware evolves rapidly over time, which makes existing solutions being ineffective in detecting newly released malware. Machine learning models that can learn to capture malicious patterns directly from the data play an increasingly important role in malware analysis. However, traditional machine learning models heavily depend on feature engineering. The extracted static features are vulnerable as hackers could create new malware with different feature values to deceive the machine learning models. In this paper, we propose an end-to-end malware detection framework consisting of convolutional neural network, autoencoder and neural decision trees. It learns the features from multiple domains for malware detection without feature engineering. In addition, since anti-virus products should have a very low false alarm rate to avoid annoying users, we propose a special loss function, which optimizes the recall for a fixed low false positive rate (e.g., less than 0.1%). Experiments show that the proposed framework has achieved a better recall than the baseline models, and the derived loss function also makes a difference.**

I. Introduction

Malicious software (or malware) is a popular and damaging attack vector, leading to $57 billion to $109 billion economic costs in one year [5]. Malware evolves rapidly, with reports showing that 99% disappear after 58 seconds [28] and 3.9 new malware programs are generated per second [3] on average. Therefore, the development of malware detection techniques that are able to generalize to new malware is important.

To build a malware detection model, one key phase is feature engineering. The commercial anti-virus products like Symantec, McAfee, TrendMicro, etc. rely on large databases of malware signatures crafted by security experts [4], [27]. It is challenging to generate signatures quickly enough to handle new malware being generated in a high speed. Machine learning models that can learn to capture the malicious patterns from data partially resolve this issue. However, traditional machine learning methods still require experts to design the features (e.g. printable strings, instructions, control flow graph etc. [19], [17], [32], [31]). These hand-crafted features could be easily exploited to deceive the detection model, e.g. by creating new malware with different feature values.

Recently, deep learning has been successfully applied to research areas such as image recognition and natural language processing. Deep learning models are able to learn abstract representations (i.e. features) from raw input data. This feature learning capability makes deep learning a powerful tool for extracting effective features from new malware. Several malware detection models based on deep learning have been proposed recently. [20] designs a 1D convolution network architecture, called MalConv, which takes a sequence of two million bytes as the input and has achieved a decent detection accuracy. Another convolutional neural network based solution [16] also takes the raw byte sequence as the input. Both methods treat all parts of the file (i.e. program) equally although different parts contain information from different fields. In this paper, we focus on detecting malware in portable executable (PE) file format in Windows systems, which is the most popular malware file format [3]. The header of a PE file includes structured information, and the section parts contain scattered and much more human-unreadable information. To tackle this problem, our proposed method applies two neural networks to process the PE header and PE sections separately. More specifically, the header feature is extracted by a gated convolutional mechanism applied in MalConv [20], while the section feature is learned by an Autoencoder. After that, the concatenated feature of these two parts is fed into Neural Decision Trees [15] for classification.

Anti-virus products are required to keep a low false alarm rate to avoid disturbing the users frequently. Therefore, the models should be trained to achieve a high recall for a fixed low false positive rate (the ratio of incorrectly identifying benign software as malware, usually less than 0.1% [23]). However, the existing models typically focus on malware detection accuracy and pay less attention to the false positive rate. In this paper, we derive a customized loss function to directly optimize the recall given a fixed tiny false positive rate. In particular, we formalize the recall and false positive rate using zero-one loss and then approximate it with cross-entropy loss. As a consequence, the loss function is differentiable and can be optimized with the feature extraction networks end-to-end.

In summary, we make the following contributions:

1) We propose an end-to-end malware detection framework based on deep learning techniques, which achieves the best performance among existing deep learning based methods.
2) We propose an effective loss function for optimizing recall with a fixed tiny false positive rate.

978-1-7281-8808-9/20/$31.00 ©2020 IEEE

2020 25th International Conference on Pattern Recognition (ICPR) | 978-1-7281-8808-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICPR48806.2021.9412227

3) We conduct experiments[1] on a real large dataset to confirm the effectiveness of the proposed feature learning framework and loss function for malware detection.

II. Related Work

A. Commercial Antivirus Solutions

Commercial anti-virus products from McAfee, Symantec, TrendMicro, etc., and online services like VirusTotal have low accuracy and high memory overhead since they rely on large databases of malware signatures [29]. Since a vast number of new malware samples appears every day (1.7 million per day [26]), crafting signatures requires great human expertise. Moreover, the signatures are easily bypassed by encrypting the payload, or by other obfuscation techniques like polymorphism and metamorphisms. In fact, malware creators can avail themselves of many off-the-self packer tools [30] that perform complex obfuscations at next to zero cost.

B. Traditional Machine Learning Solutions

Machine learning models have been applied for malware detection to improve the performance. A crucial phase in using traditional machine learning solutions is feature engineering. Analysis can be performed without executing the malware (static analysis) or by executing it in a real or emulated environment (dynamic analysis).

Schultz et al. [24] are one of the first to introduce the idea of applying machine learning methods for detecting malware based on static analysis. The features include system resource information, strings and byte sequences extracted from the malicious executables. The best result is achieved by the Multi-Naive Bayes algorithm using byte sequences as features. Kolter and Maloof [14] take n-grams of byte codes as features. The experiment results show that boosted decision trees outperform other classifiers. Shafiq et al. [25] present PE-Miner and compare RFR (Redundant Feature Removal), PCA (Principal Component Analysis) and HWT (Haar Wavelet Transform) to reduce the dimensionality of the feature extracted from the PE file. It reports an overall Area Under the ROC Curve (AUC) of 0.991 for classifying malware on their dataset. EMBER (Endgame Malware BEnchmark for Research) [2] is a public and free dataset that integrates many features, such as PE header features, byte-entropy histogram, printable string information, etc. This proposed feature engineering method is detailed, and achieves excellent performance with LightGBM [12] as the classifier.

In general, these machine learning models perform well. However, the human-crafted features that are extracted by feature engineering methods still rely on experts. Crafting the features for new attacks is a time-consuming process. Moreover, the features could be exploited by the attackers due to the limitation of experts' knowledge, resulting in the deficient ability to detect new malware.

[1]The code is available at https://tinyurl.com/y6zkswxw

C. Deep Learning Solutions

As deep learning is successfully applied to research areas including image classification, speech recognition and other applications, researchers have started to explore deep learning architectures for malware classification.

Dahl et al. [7] extract the traditional features and then use random projections to reduce the input dimensionality while retaining the highly discriminative information. The feature vector is fed into a neural network with a single hidden layer. Saxe and Berlin [23] introduce a deep neural network for malware detection and propose a Bayesian model calibration approach for adjusting the classifier scores. It achieves a detection accuracy of 95% and a false positive rate of 0.1% against a dataset of over 400,000 software binaries. However, the performance decays significantly for detecting new malware variants. Later, Meng et al. [18] propose a neural network model named MCSMGS to analyze the malware API call sequences generated by IDA pro. Kolosnjaji et al. [13] also implement a neural network that consists of convolutional and feedforward neural networks and takes the PE metadata, PE import features and Assembly Opcode features as the input.

Although the research works mentioned above use neural networks as their classifiers which improve the performance to some extent, they still rely on the hand-engineered features that require domain knowledge of malware. Several deep learning solutions with minimum usage of domain knowledge have been proposed for extracting effective new features so as to enhance the models' ability to detect unknown malware.

Cylance [6] offers a commercial product that employs Convolutional Neural Networks directly, treating binary samples as images. Raff et al. [21] extract 328 essential features (64 bytes of the MS-DOS header and 264 bytes of the COFF and Optional PE header) from PE headers and train an LSTM with attention mechanism on the samples. As the first application of neural networks on raw byte sequence features for malware detection, it requires minimum domain knowledge. The results show that the feedforward network performs better than all other methods, and the LSTM performs comparably to the domain knowledge approach at times. They further propose a 1D convolutional network architecture, MalConv [20], which reads the raw byte sequences of the entire executable file and achieves a better accuracy. Another work without domain knowledge is [16], in which a deep convolutional neural network is designed. The input is also raw byte sequence. It reports a better performance than that of MalConv on its dataset. Both MalConv [20] and the convolutional neural network [16] process the byte data directly. Gibert et al. [10] propose a deep learning model to learn the feature of the structure entropy stream of the PE files, which is resilient to many obfuscation techniques. However, all these three methods lack an insightful approach to handle the concrete PE header information, which contains more important information than other parts of the data.

III. Background

A. PE File Format

In this work, we focus on the PE files in Windows operating system since they remain the most popular malware file format [3]. PE file is a portable executable file that can run in Windows operating systems. For example, the .exe file, the .dll file. The PE file format defines the structure or the format of these files.

A PE file consists of one PE header and multiple sections. PE header contains the most basic and meaningful information about the file, such as the target machine types, the number of the sections, the operating system version, and so on. The information inside the header is dense and well-aligned. The last part of the PE file is the section part containing code, data, and other content that are valuable for malware detection, especially the executable sections. However, these sections vary in length (usually very long) and the information is scattered, so the features are difficult to extract. Previous works just applied string extraction, n-gram or entropy analysis methods, which are not adequate for extracting the internal particular information. In this work, we propose the methods to extract the features from the executable sections in order to make full use of the entire PE file.

IV. Methodology

The proposed framework can be divided into two parts, the feature learning part and the classification part. For feature learning, in order to make full use of the header and the sections, two structures are designed and applied for processing them separately. The header feature is extracted by a gated convolutional mechanism applied in MalConv [20] which takes the PE header as the input, while the section feature is extracted by an Autoencoder that compresses several executable sections into a short feature representation. Then, the header feature and the extracted section feature are concatenated to form the final feature. The classifier applies Neural Decision Trees and logistic regression over the final feature. We propose a special loss function for optimizing the recall given a fixed false positive rate. Figure 1 is an illustration of the proposed framework.

A. Header Feature Extraction

As stated in section III-A, PE header consists of many discrete fields providing meaningful information of the PE sample. However, many fields in the PE files have no specific meaning for malware recognition. For example, type of the target machine, the minor and the major operating system version. Thus, aiming at filtering the irrelevant information and picking out the most discriminating features, we adopt the gated convolution mechanism in MalConv [20] to extract features from the header. Figure 2(a) is the structure illustration of the header feature extractor.

We truncate and pad the PE header to get a byte sequence of fixed length (e.g., 4096 in this paper). Each byte is converted into a vector representation (e.g., 8 for the length) via the embedding layer, which looks up the embedding vector in an embedding matrix based on the byte index. Then, the embedded features are fed into two convolution layers respectively. Let X_A denote the output of the first convolution layer, and X_B denote the output of the second one; they are combined by $X_A \otimes \sigma(X_B)$, which involves an element-wise multiplication operation. Here, σ is the Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, so $\sigma(X_B)$ is the gate that controls the information from X_A passed to the next layer in the model. Conceptually, the gating mechanism is important because it allows the selection of important and relevant information. Finally, a global max-pooling layer after the convolutional part allows the model to produce the aggregated features.

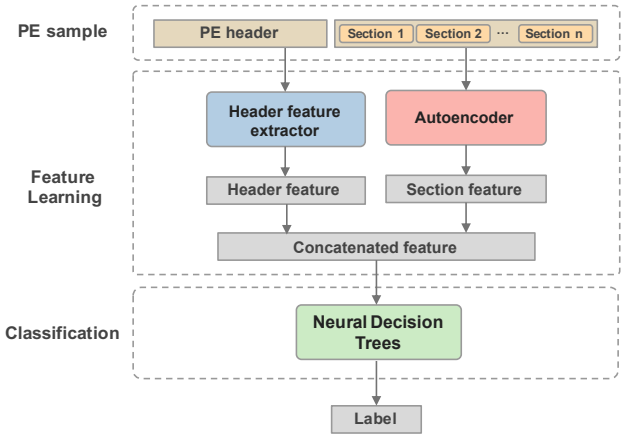


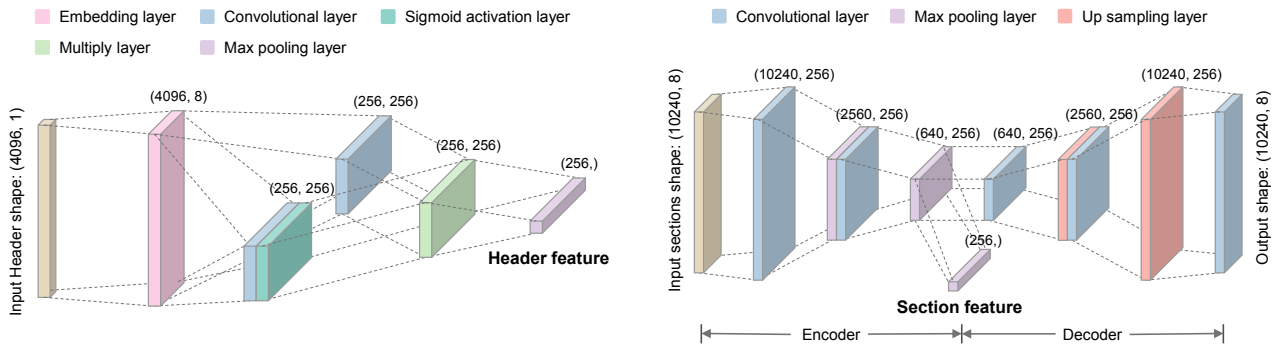
Fig. 1. An illustration of the proposed framework

B. Section Compression

The information inside the sections is significant but scattered. Moreover, the sections are usually very long, leading to the difficulties in efficient processing. An autoencoder model is an unsupervised machine learning algorithm for learning efficient coding and it is widely used for dimensionality reduction.

Figure 2(b) shows the structure of the proposed autoencoder model, which is similar to U-Net [22]. In this model, we attempt to compress several executable sections (e.g., 8 executable sections of length 10,240 in this paper) in a PE file into a short representation. The input shape is (10,240, 8), where the first number is the length of each section and the second is the number of executable sections. We assert that the executable sections are independent, so we apply 1D convolution operation along the byte sequence in each section separately. We utilize multiple convolution layers and max pooling layers to encode the input, and convolution layers and upsampling layer to recover the shape of the output. The loss function for training is the mean squared error between the observed outputs of the final nodes and the inputs.

The obtained latent representation is then concatenated with the header feature to form the final feature F.



(a) Structure of the header feature extractor

(b) Structure of autoencoder

Fig. 2. Detailed Structures of the proposed framework. (m, n) denotes (feature length, # of channels)

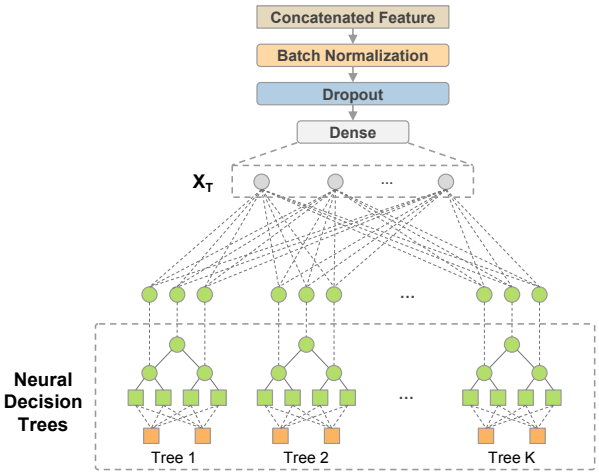


Fig. 3. The structure of Neural Decision Trees

C. Neural Decision Trees

Models based on decision trees (e.g., LightGBM) have achieved good performance for malware detection with hand-crafted features [2]. In order to apply it in our framework and to enable the end-to-end training, we utilize a differentiable version of decision trees, i.e., neural decision tree [15], to process the feature learned by the neural networks.

We first normalize the obtained concatenated feature F by a batch normalization layer so that it resembles a normal distribution. Then, the dropout layer is applied to reduce overfitting. After a dense layer, we get the input X_T of the Neural Decision Trees.

Suppose there are K trees and each tree follows the classical binary tree structure. Each decision node $d \in \mathcal{D}$ holds a decision function $D_d(X_T) \in [0, 1]$, which is the probability that a sample reaches node d and be sent to the left subtree. The decision function is:

$$D_d(X_T) = \sigma(f_d(X_T)), \tag{1}$$

where σ is the Sigmoid function, and $f_d(X_T)$ is the transformation function $f_d(X_T) = W_T X_T + b_T$. Each leaf node $l \in \mathcal{L}$ holds a probability distribution P_l over the labels. Then the probability of a sample predicted as label $Y \in \{0, 1\}$ by tree k is

$$P_{T_k}[Y|X_T] = \sum_{l \in \mathcal{L}} P_{l_Y} \left(\prod_{d \in \mathcal{D}} D_d(X_T)^{I_{left}} \overline{D_d}(X_T)^{I_{right}} \right), \tag{2}$$

where $\overline{D_d}(x) = 1 - D_d(x)$, I_{left} is the indicator function for the sample that will be sent to the left subtree, and P_{l_Y} is the probability for the nodes predicted to be label Y.

There are two methods to make use of the predicted results of all the trees $\mathcal{F} = \{T_1, T_2, ..., T_K\}$, bagging and boosting, which are both ensemble techniques, where a set of weak learners are combined to create a strong learner that obtains better performance than a single one.

For the bagging method, the final prediction for a sample is obtained by averaging the outputs of all the trees (Random Forest):

$$P[Y|X_T] = \frac{1}{K} \sum_{k=1}^{K} P_{T_k}[Y|X_T]. \tag{3}$$

The prediction for the label is

$$\hat{Y} = \underset{Y}{\operatorname{argmax}} P[Y|X_T]. \tag{4}$$

The loss function for the neural random forest is taken as the binary cross entropy function, which is

$$\begin{aligned} \ell_{NDT}(X_T, y) = -\big(&y \log(P[Y = 1|X_T]) \\ &+ (1 - y) log(P[Y = 0|X_T])\big)' \end{aligned} \tag{5}$$

where y is the true label of X_T.

For the boosting method, we adopt the theory of the traditional GBDT (Gradient Boosting Decision Trees) algorithm [9]. In the proposed neural network version, the algorithm is described in Algorithm 1, in which the initial predicted probability g_0 is a trainable parameter that is initialized as 0.5, $g_k(X_T)$ is the prediction of the k_{th} tree, ℓ_{NDT} is the loss function for the Neural Gradient Boosting Decision Tree, and mse is the mean squared error. In each step, we aim to fit the generated decision tree to the residual between the true label

Input : The input feature X_T, the true label y, learning rate ρ, the initial predicted value $g_0(X_T)$

Output: The predicted value $P(X_T)$

$\ell_{NDT} \leftarrow 0.0$;

for $k = 1$ **to** K **do**

  $r_k \leftarrow y - g_{k-1}(X_T)$;

  $g_k(X_T) \leftarrow g_{k-1}(X_T) + \rho \cdot P_{T_k}(X_T)$;

  $\ell_{NDT} \leftarrow \ell_{NDT} + \rho \cdot mse(r_k, P_{T_k}(X_T))$;

end

$P(X_T) = g_K(X_T)$;

Algorithm 1: Neural Gradient Boosting Decision Tree Algorithm

and the prediction of the ensembled previous trees. Thus, the loss for the Neural Gradient Boosting Decision Tree is

$$\ell_{NDT}(X_T, y) = \rho \cdot \sum_{k=1}^{K} mse(r_k, g_k(X_T)). \tag{6}$$

Then the final prediction $P(X_T)$ is the cumulative sum of multiple generated trees.

Following the idea in [11], we apply logistic regression on all the outputs of the decision trees to improve the performance.

Binary cross-entropy loss is taken as the loss function for the logistic regression part, which is denoted as $\ell_{LR}(X_{LR}, y)$. In the next subsection, we specifically optimized this loss function to satisfy the objective of maximizing the recall with a given tiny false positive rate.

Combining the loss functions of the autoencoder, ℓ_{AE}, and the neural decision trees with logistic regression, ℓ_{NDT} and ℓ_{LR}, we get the final loss function:

$$\ell = \ell_{AE}(X_S, Z) + \lambda_1 \ell_{NDT}(X_T, y) + \lambda_2 \ell_{LR}(X_{LR}, y). \tag{7}$$

Here, λ_1 and λ_2 are the parameters to trade off between these three loss functions.

D. Loss Function Optimization

Commercial anti-virus products are required to keep a low false alarm rate to avoid disturbing users frequently. Inspired by [8], we derive a particular loss function for optimizing our specific task, which is to maximize the recall under the constraint that false positive rate is less than or equal to 0.1%. For our proposed model, the optimization process is only applied in the last layer, the logistic regression, which outputs the prediction probabilities of two categories.

We use random variable X to denote the input features, Y to denote the ground truth, i.e. the target label, Y^+ to denote the labels of the positive samples (malware) and Y^- to denote the labels of the negative ones (benign software). We denote a classification function f_b, which consists of a function $f : X \to R$, and a threshold $b \in R$., indicating positive if $f(x) \geq b$. The false positive rate, $FPR(f_b)$ and true positive rate $TPR(f_b)$ of a classification function are defined by

$$FPR(f_b) = \frac{fp(f_b)}{fp(f_b) + tn(f_b)} = \frac{fp(f_b)}{|Y^-|}, \tag{8}$$

$$TPR(f_b) = \frac{tp(f_b)}{tp(f_b) + fn(f_b)} = \frac{tp(f_b)}{|Y^+|}, \tag{9}$$

where tp, fp, tn and fn are the true-positive, false-positive, true-negative and false-negative counts (respectively), and $|Y^+|$ and $|Y^-|$ are the number of the positive and negative samples. The equations for calculating $tp(f_b)$ and $fp(f_b)$ are

$$tp(f_b) = \sum_{i \in Y^+} I(f(x_i) \geq b), \tag{10}$$

$$fp(f_b) = \sum_{i \in Y} I(f(x_i) \geq b). \tag{11}$$

With the notations above, we can formalize our problem as

$$\max_f \frac{tp(f_b)}{|Y^+|} \quad s.t. \frac{fp(f_b)}{|Y^-|} \leq \alpha, \tag{12}$$

which maximizes TPR (Recall) given the upper bound on FPR, i.e., α. By rewriting the $tp(f_b)$ and $fp(f_b)$ using the zero-one loss, we convert Equation 12 into Equation 13.

$$\max_f 1 - \frac{\sum_{i \in Y^+} \ell_{01}(f_b, x_i, y_i)}{|Y^+|} s.t. \frac{\sum_{i \in Y} \ell_{01}(f_b, x_i, y_i)}{|Y^-|} \leq \alpha, \tag{13}$$

where $\ell_{01}(f_b, x_i, y_i) = I(f_b(x_i) \neq y_i)$. Zero-one loss is non-convex and non-differentiable and is thus difficult to optimize. To make the objective differentiable, we use the binary cross entropy loss to lower bound tp and upper bound fp. Equation 13 is then transformed into Equation 14.

$$\max_f 1 - \frac{\sum_{i \in Y^+} \ell(f, x_i, y_i)}{|Y^+|} \quad s.t. \frac{\sum_{i \in Y} \ell(f, x_i, y_i)}{|Y^-|} \leq \alpha, \tag{14}$$

where ℓ is the cross entropy loss. Denote $L^+(f) = \sum_{i \in Y^+} \ell(f, x_i, y_i)$, $L^-(f) = \sum_{i \in Y} \ell(f, x_i, y_i)$. Equation 14 becomes

$$\min_f \frac{L^+(f)}{|Y^+|} \quad s.t. \frac{L^-(f)}{|Y^-|} \leq \alpha. \tag{15}$$

Applying Lagrange multiplier, we get the following transformed objective

$$\min_f \max_{\beta \geq 0} (\frac{L^+(f)}{|Y^+|} + \beta(\frac{L^-(f)}{|Y^-|} - \alpha)). \tag{16}$$

When $\frac{L^-(f)}{|Y^-|} - \alpha > 0$, it reaches the maximum value if β is as large as possible; when $\frac{L^-(f)}{|Y^-|} - \alpha \leq 0$, then maximum value is reached when β is zero. We rewrite the objective and set the β as a fixed big value:

$$\min_f (\frac{L^+(f)}{|Y^+|} + \max(0, \beta(\frac{L^-(f)}{|Y^-|} - \alpha))). \tag{17}$$

TABLE I
THE EXPERIMENTAL RESULTS

Training Dataset	Test Dataset	Model	Without optimized loss function		With optimized loss function	
			AUC (%)	Recall (%)	AUC (%)	Recall (%)
February	March	MalConv	95.45±0.34	33.58±16.21	94.79±0.32	53.17±4.37
		ConvNet	96.21±0.17	45.11±3.88	94.34±0.60	49.92±3.69
		EntropyNet	91.61±0.22	33.88±9.13	88.13±0.73	41.52±4.38
		Proposed Model	**96.47±0.20**	**56.14±3.65**	**96.40±0.19**	**57.52±2.95**
March	April	MalConv	98.50±0.12	50.67±11.75	98.21±0.31	57.41±9.74
		ConvNet	98.82±0.12	63.67±5.50	98.27±0.70	67.39±5.69
		EntropyNet	95.70±0.32	24.53±6.76	93.95±0.48	49.68±8.09
		Proposed Model	**99.16±0.04**	**71.54±3.32**	**99.12±0.07**	**75.25±1.62**
April	May	MalConv	97.95±0.36	52.28±8.12	94.02±1.48	58.55±2.43
		ConvNet	98.33±0.26	55.91±2.68	96.66±0.73	56.96±3.45
		EntropyNet	90.96±0.96	31.33±3.13	81.94±2.33	35.23±3.24
		Proposed Model	**98.60±0.20**	**70.29±1.03**	**98.43±0.35**	**70.69±0.93**

The new loss function is

$$L(f) = \frac{L^+(f)}{|Y^+|} + \max(0, \beta(\frac{L^-(f)}{|Y^-|} - \alpha)), \qquad (18)$$

which implements the last term ℓ_{LR} in Equation 7.

V. EXPERIMENTS

A. Dataset

The dataset spans across 4 months and is provided by a local anti-virus company, SecureAge Technology of Singapore. This company maintains a platform with 12 anti-virus engines to classify the PE files. The classification results are aggregated to get the label of each PE file for model training. If 4 or more engines agree that a particular sample is malicious, then it is labelled as positive. If none of the engines detects it as malicious, it is labelled as negative. Any other samples with 1-3 engines labelled as malicious are deemed inconclusive and therefore discarded (not used for training or testing).

TABLE II
SUMMARY OF THE DATA

Dataset	Positive samples	Negative samples
February	110656	80185
March	100651	92097
April	58394	48595
May	42635	87858

Table II is a summary of the data, where one row represents the statistics of the data in one month. To better measure the model's ability to detect new malware, we divide our dataset into three groups, (February, March), (March, April) and (April, May), where the training and validation data is from the first month and the test data is from the second month. In this way, there is no overlapping between the training and test data. Considering that malware evolves fast, our test data should have included many newly released malware samples.

B. Experimental Setup

We compare our solution with 3 baseline methods that are also based on deep learning models. Each experiment is repeated for 8 times and we report the mean and the standard variation, which enables a robust and reliable estimation of the models' performance.

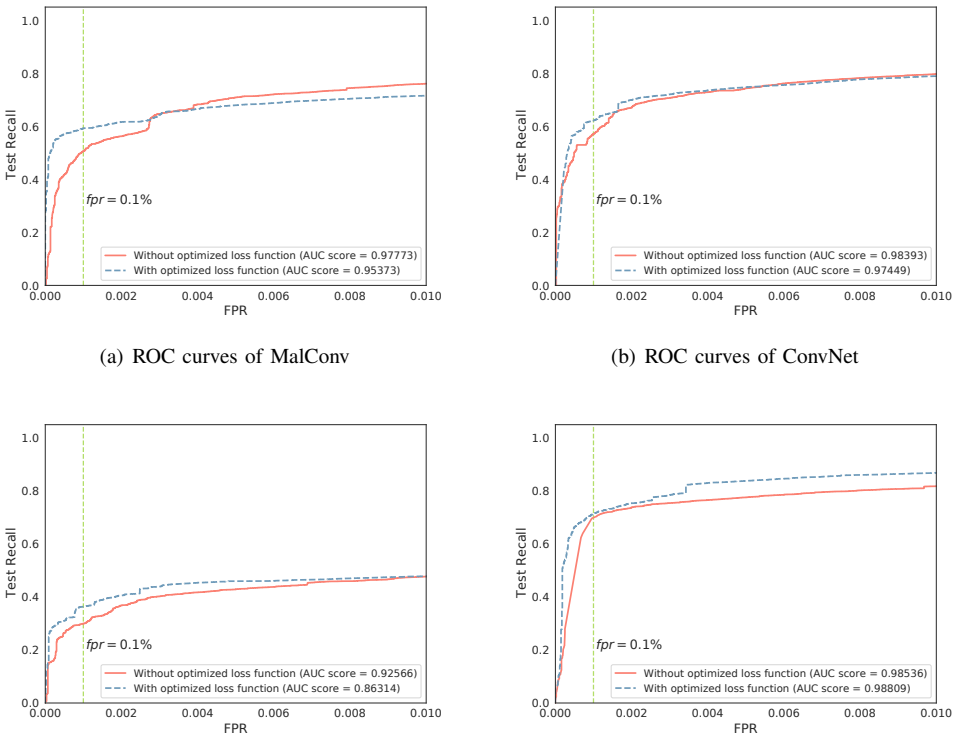
- **MalConv [20]** MalConv's structure is simple but good at processing long 1-dimension sequences and has achieved good performance in malware detection. In this experiment, MalConv takes the first 100,000 bytes in a PE sample as the input.
- **ConvNet [16]** We evaluate the performance of the convolutional neural network proposed by [16] which also takes the first 100,000 bytes in a sample as the input. For ease of explanation, we denote the model as ConvNet in the following text.
- **EntropyNet [10]** [10] is another malware classification method based on deep learning, which learns the feature from the structural entropy stream. In this experiment, we feed 100 non-overlapping chunks (each with 1024 bytes) into the convolutional neural network. We denote the model as EntropyNet in the following text.
- **Proposed Model** In order to account for most of the samples and have a better control of the input size, we limit the maximum header length to 4096 (covering 99.94% PE samples), maximum number of the executable sections to 8 (covering 99.90% executable sections), and the length of each section to 10,240 (covering 29.33% executable sections) in these experiments. The tree depth is set to 5 and the tree number is 200 in both Neural Random Forest and Neural Gradient Boosting Decision Trees. The learning rate in Neural Gradient Boosting Decision Trees is configured as 0.01.

C. Results and Analysis

The experiment results of the baseline models and the proposed model are illustrated in Table I. As malware data in reality is often highly imbalanced, accuracy is virtually useless for the measurement of a model's performance [1]. Thus, two evaluation metrics are considered: ROC AUC score and Recall when fp (false positive) rate is 0.1% or below.

From the experimental results, regardless of the optimization of the loss function, our proposed model achieve the best AUC score and recall among the baseline models for all three dataset groups and it is the most stable and reliable model. Therefore, our effort on extracting the executable sections has made a difference.

Besides, the experimental results in Table I also show that models with the derived optimized loss function generally outperform those without the optimized loss function. Moreover, the optimized loss function improves the stability of the



(a) ROC curves of MalConv

(b) ROC curves of ConvNet

(c) ROC curves of EntropyNet

(d) ROC curves of Proposed Model

Fig. 4. ROC curve comparison

models to some extent. Figure 4 shows the comparison of ROC curves of all these four models. It can be seen that the ROC curve using the optimized loss function is shifted to the left slightly, which indicates that the optimized loss function is able to increase the recall when the fp rate is fixed to a tiny value. Therefore, the derived loss function has made a remarkable effect. We have tried to learn the parameter β by gradient descent method; but in most cases, the performance is neither stable nor ideal. So, in these experiments, we tune the parameter β by cross validation manually. For all the neural network models, the best β is configured as 30.

Lastly, since the training dataset is collected before the test dataset, some new malware variants were generated over time, which means that the distribution of software populations differs between the training and test dataset. For all three dataset groups, the proposed model achieves the best performance, especially when the training dataset is from March, confirming the ability of our model in detecting new and constantly evolving malware.

D. Ablation Study

To evaluate the effect of each component (including the Autoencoder and Neural Decision Trees), some comparison experiments are conducted with different settings shown in Table III. If Autoencoder is No, the decoder part is removed in the model so that the section part is processed by several convolution layers and max pooling layers. If Neural Decision Trees is No, it is replaced by logistic regression. If Logistic Regression is No, we take the result of the Neural Decision Trees Module as the final output instead of collecting the outputs of each single decision trees and applying logistic regression on them.

Table III displays the results when the training dataset is from April and the test dataset is from May. The addition of each component including Autoencoder, the Neural Decision Trees and the Logistic Regression brings an improvement to the performance. Compared to using multiple convolution layers to learn the representation of sections, the autoencoder is a better compression way to process the sections because it has a restoring process and the introduction of the reconstruction loss between the input sections and the decoded sections further enhances the learned section features. Neural Decision Trees is a novel method of transforming the traditional tree structure into a neural network. It enables end-to-end training and is able to reduce overfitting. Experiment results show that the application of Neural Decision Trees improves the performance slightly. The application of logistic regression after the decision trees further improves the performance. Similar observation is presented in [11]. Compared with Neural Decision Trees without logistic regression, the weights of the outputs of different trees are learned instead of just averaging the outputs (in Neural Random Forest) or summing them by a fixed rate (the learning rate in Neural GBDT).

TABLE III
THE RESULTS OF THE PROPOSED MODEL WITH DIFFERENT SETTINGS

Autoencoder	Neural Decision Trees	Logistic Regression	AUC (%)	Recall (%)
No	No	Yes	98.89±0.12	68.99±1.51
Yes	No	Yes	98.86±0.05	69.33±1.75
Yes	Neural Random Forest	No	98.70±0.19	69.89±1.86
Yes	Neural GBDT	No	98.68±0.26	69.75±2.44
Yes	Neural Random Forest	Yes	**98.92±0.11**	70.10±1.53
Yes	Neural GBDT	Yes	98.60±0.20	**70.29±1.03**

VI. CONCLUSION

In this work, we propose an end-to-end hybrid framework for malware detection with convolutional neural network, autoencoder and neural decision trees, and derive an optimized loss function to improve the recall. Experimental results demonstrate that the proposed framework is effective for malware detection. Ablation study over component variations gives us valuable insights on architecture design.

ACKNOWLEDGMENT

This research is supported by the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity RD Programme (Grant No. NRF2016NCR-NCR002-020). We also thank SecureAge Technology of Singapore for sharing the data.

REFERENCES

[1] Akbani, R., Kwek, S., Japkowicz, N.: Applying support vector machines to imbalanced datasets. In: European conference on machine learning. pp. 39–50. Springer (2004)
[2] Anderson, H.S., Roth, P.: Ember: An open dataset for training static pe malware machine learning models. arXiv preprint arXiv:1804.04637 (2018)
[3] AV-TEST: Security Report 2017/18. https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2017-2018.pdf (2017)
[4] Birrer, B., Raines, R.A., Baldwin, R.O., Oxley, M.E., Rogers, S.K.: Using qualia and novel representations in malware detection. In: Intelligent Sensing, Situation Management, Impact Assessment, and Cyber-Sensing. vol. 7352, p. 73520W. International Society for Optics and Photonics (2009)
[5] CEA: The Cost of Malicious Cyber Activity to the U.S. Economy. https://www.whitehouse.gov/wp-content/uploads/2018/03/The-Cost-of-Malicious-Cyber-Activity-to-the-U.S.-Economy.pdf (2018)
[6] Cylance: Cylance: advanced threat prevention (2015), https://www.cylance.com/
[7] Dahl, G.E., Stokes, J.W., Deng, L., Yu, D.: Large-scale malware classification using random projections and neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (2013). https://doi.org/10.1109/ICASSP.2013.6638293
[8] Eban, E.E., Schain, M., Mackey, A., Gordon, A., Saurous, R.A., Elidan, G.: Scalable learning of non-decomposable objectives. arXiv preprint arXiv:1608.04802 (2016)
[9] Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
[10] Gibert, D., Mateu, C., Planes, J., Vicens, R.: Classification of malware by using structural entropy on convolutional neural networks. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
[11] He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers, S., et al.: Practical lessons from predicting clicks on ads at facebook. In: Proceedings of the Eighth International Workshop on Data Mining for Online Advertising. pp. 1–9. ACM (2014)
[12] Ke, G., Meng, Q., Wang, T., Chen, W., Ma, W., Liu, T.Y., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: LightGBM: A highly efficient gradient boosting decision tree. NIPS (2017)
[13] Kolosnjaji, B., Eraisha, G., Webster, G., Zarras, A., Eckert, C.: Empowering convolutional networks for malware classification and analysis. In: Proceedings of the IJCNN (2017). https://doi.org/10.1109/IJCNN.2017.7966340
[14] Kolter, J.Z., Maloof, M.A.: Learning to detect malicious executables in the wild. In: Proceedings of the 2004 ACM SIGKDD (2004). https://doi.org/10.1145/1014052.1014105
[15] Kontschieder, P., Fiterau, M., Criminisi, A., Rota Bulo, S.: Deep neural decision forests. In: Proceedings of ICCV. pp. 1467–1475 (2015)
[16] Krčál, M., Švec, O., Bálek, M., Jašek, O.: Deep convolutional malware classifiers can learn from raw executables and labels only (2018)
[17] Liu, J., Song, J., Miao, Q., Cao, Y.: FENOC: An ensemble one-class learning framework for malware detection. In: Proceedings of CIS 2013 (2013). https://doi.org/10.1109/CIS.2013.116
[18] Meng, X., Shan, Z., Liu, F., Zhao, B., Han, J., Wang, H., Wang, J.: Mcsmgs: Malware classification model based on deep learning. In: CyberC, 2017 International Conference on. pp. 272–275. IEEE (2017)
[19] Moskovitch, R., Feher, C., Tzachar, N., Berger, E., Gitelman, M., Dolev, S., Elovici, Y.: Unknown malcode detection using OPCODE representation. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2008). https://doi.org/10.1007/978-3-540-89900-6_21
[20] Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.: Malware detection by eating a whole exe. arXiv preprint arXiv:1710.09435 (2017)
[21] Raff, E., Sylvester, J., Nicholas, C.: Learning the PE Header, Malware Detection with Minimal Domain Knowledge. In: Proceedings of AISec '17 (2017). https://doi.org/10.1145/3128572.3140442
[22] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234–241. Springer (2015)
[23] Saxe, J., Berlin, K.: Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features. In: CoRR abs/1508.03096 (2015). https://doi.org/10.1109/MALWARE.2015.7413680
[24] Schultz, M., Eskin, E., Zadok, E., Stolfo, S.: Data mining methods for detection of new malicious executables. In: Proceedings of S&P 2001. (2001). https://doi.org/10.1109/SECPRI.2001.924286
[25] Shafiq, M.Z., Tabish, S.M., Mirza, F., Farooq, M.: PE-miner: Mining structural information to detect malicious executables in realtime. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2009). https://doi.org/10.1007/978-3-642-04342-0_7
[26] Symantec Corporation: Internet Security Threat Report. Symantec **2016**(2), 1–3 (2016). https://doi.org/10.1016/S1353-4858(05)00194-7
[27] Venugopal, D., Hu, G.: Efficient signature based malware detection on mobile devices. Mobile Information Systems **4**(1), 33–49 (2008)
[28] Verizon: 2016 Data Breach Investigations Report. Tech. Rep. 1 (2016). https://doi.org/10.1017/CBO9781107415324.004, http://www.verizonenterprise.com/resources/reports/rp_data-breach-investigations-report-2013_en_xg.pdf
[29] Vigna, G.: Antivirus isn't dead, it just can't keep up. Tech. rep., Technical Report. Lastline Labs (2014)
[30] Wei, Y., Zheng, Z., Ansari, N.: Revealing packed malware. IEEE Security and Privacy (2008). https://doi.org/10.1109/MSP.2008.126
[31] Ye, Y., Chen, L., Wang, D., Li, T., Jiang, Q., Zhao, M.: SB-MDS: An interpretable string based malware detection system using SVM ensemble with bagging. Journal in Computer Virology (2009). https://doi.org/10.1007/s11416-008-0108-y
[32] Ye, Y., Li, T., Chen, Y., Jiang, Q.: Automatic malware categorization using cluster ensemble. In: Proceedings of the 16th ACM SIGKDD (2010). https://doi.org/10.1145/1835804.1835820