CS5242 Project - Malware Detection

Group 32
Members: Dai Mingxi (E0227607), Yang Ruizhi (E0227615)
National University of Singapore - School of Computing

1 Introduction

The detection of malware is an important problem in cyber security, especially when more of society becomes dependent on computing systems. Single incidences of malware can cause millions of dollars in damages. According to Anderson et al., the estimated loss cause by malware (consumer and business) is as high as $370 million [1]. On the other hand, auti-virus products are growing increasingly ineffective due to the emergence of new malware. The need for a more effective and agile approach in malware detection has been recognized by the industry, and so many efforts have been made to improve automated malware classification.

In this study, we build a convolutional neural network by using the portable executable (PE) header of exe files without any explicit feature extraction. PE headers are chosen as they are by far the most severe channel for security threats on a PC. In addition to training the best single classifier, we also propose two training schemes which combine several classifiers.The results suggest that combining the predictions from classifiers trained on different train-validation splits of the whole dataset (i.e. ensemble learning) can improve the final prediction to certain extent.

2 Related Work

Most recent work in the application of deep learning to malware detection relies on domain-specific feature extraction. The results of previous work indicate that domain knowledge in the form of information gleaned from PE header is effective at building classifiers that distinguish between benign and malicious executables.

A popular approach to do feature extraction is dynamic malware analysis, where the binary is run in a virtualized environment. The behavior of the malware can be observed and thus information about its execution can be extracted. Attempts have been made to process the sequence of API calls (system calls) a malware file generate using a combination of convolutional and recurrent layers. This version of neural network architecture has achieved an overall accuracy of 89.4% [3].

An other previous work proposes a novel multi-task neural network malware classification architecture called MtNet. This architecture extracted two sets of feature as inputs: a sequence of API calls and a sequence of null-terminated objects recovered from system memory during emulation. This approach achieved a classification error rate of 2.94% [2].

Despite that prior works achieved good performance in malware detection task, the fact that those approaches use manual feature engineering and domain knowledge make them computationally costly and thus hinders their scalability.

Raff et al. (2017) [5] and Marek et al. (2018) [4] have recently proposed two similar simple convolutional deep neural network architectures that do not require manual feature extraction nor pre-processing. Both of the architectures achieved overall accuracy as high as 94.6% and 96.0%, respectively. Because of the nice property that those architecture can perform automatic feature generation and do not require domain knowledge, we constructed a convolutional neural network based on the architecture.

3 Model Architecture and Highlights

The scheme of our network is visualized as below. The reasons for architectural choices follow.

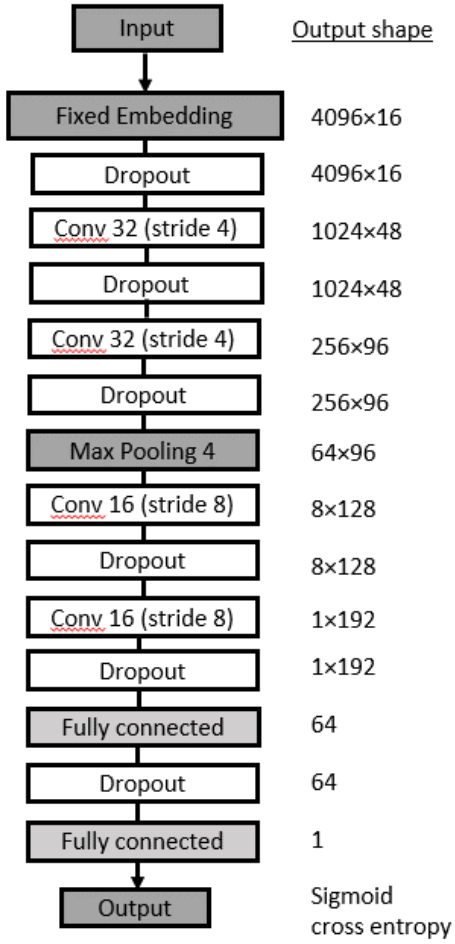

Figure 1: ConvNet Architecture

3.1 Model Architecture

Embedding Layer Each byte of the input sequence is mapped to a 16-dimensional vector at the beginning. This embedding layer is critical because the original byte value representation (numbers in range of [0, 255]) may not correctly reflect the distance between two byte values. By adding this embedding layer, we make the representation of a binary file learnable.

Padding. Zero padding is used for files whose size are less than the maximum.

Convolutional layer. We apply two kernels of size 32 at strike 4 and two kernels of size 16 at strike 8 to mitigate the computational burden. Each layer is activated via `RELU` function.

Fully-connected layer. Two fully-connected layers with output dimensions of 64 and 1 are applied after the convolutional layers, using `SELU` and sigmoid as the activation functions, respectively.

Dropout layer. Dropout layers are applied with a rate of 0.2 after embedding layer and convolutional layers. An additional dropout layer is added after the fully connected layer with dimension 64, with a rate of 0.5. This works as a regularization technique to prevent over-fitting.

Classifier and optimizer. Binary cross entropy

$$-\frac{1}{N}\sum_{i=1}^{N}\{y_i log(h_\theta(x_i)) + (1-y_i) log(1-h_\theta(x_i))\}$$

is used as the loss function, and adam optimizer is used for updating the parameters.

3.2 Model Highlights

There are few highlights of our convolutional network that worth mentioning. It learns from raw sequences of bytes and labels only, that is, no explicit feature construction is needed. Since there is no need to do dynamic malware analysis, computation cost can be reduced to certain extent.

Meanwhile, it obtains a high accuracy rate on the test dataset of 133,223 instances. The high area under the curve (AUC) score also indicates that our classifier has a good diagnostic ability.

4 Experiment Methodology

In the experiment, we propose three training schemes and evaluate each scheme based on AUC, accuracy, and kaggle score (kaggle score is the AUC calculated on more than 130,000 instances).

4.1 Datasets

The dataset consists of more than 110,000 binary files, extracted from PE header of exe files. The size of files are shown in table 1. The majority of file size are 4096, 1024 and 512, and the rest are of various size. It has a complex format with only local sequentially (1-D structure) and the meaning of its byte symbols is vary diverse and context dependent.

	training data		testing data	
file size	count	percentage	count	percentage
4096	19527	17.18%	15769	11.84%
1024	77716	68.40%	86234	64.73%
512	13206	11.62%	28562	21.44%
other	3187	2.80%	2658	2.00%
total	113636	100%	133223	100%

Table 1: Summary of PE header file size

Each training data is given a binary label where 1 indicates that the PE header is malicious and 0 indicates benign. There are 60117 malicious PE headers and 53519 benign ones. Our goal is to correctly classify testing data into the two categories.

All samples in the training dataset are first preprocessed to have the same length of 4096. Samples with length larger than 4096 (0.059% of the total data) are truncated and samples with length smaller than 4096 are padded with trailing 0s. In addition, all samples in the training dataset is further sorted in the descending order of their original length, so that during training all samples in a batch have similar original length.

4.2 Evaluation metrics

The original dataset is randomly split into training set (90%) and validation set (10%). The classification error is evaluated by training accuracy, and AUC score and accuracy on validation dataset.

4.3 Training schemes

In our work we explored three training schemes, in order to increase the prediction power. The first training scheme focuses on tweaking the hyper-parameters to get the best performance, whereas the latter two schemes are designed in a way such that we can leverage on multiple classifiers' predictions to construct a better one.

Single convolutional neural network. This work explores the performance of a convolutional neural network can have by tweaking the hyper-parameters. All the models are learned on the same split between train and validation. We denote the network with the architecture specified in Section 3.1 as ConvNet in the following context.

Ensemble learning. In addition to training a single convolutional neural network which has a high AUC score, we also attempted to combine the results from several models. Each model were trained on different splits of the original dataset. The results are combined by taking the mean value of the predictions from those models. We use ConvNet(E) to denote the ensemble of multiple ConvNets.

Multi-classifier training. Because the proportion of files with size at 512, 1024 or 4096 are quite large in both the training and testing datasets. We also attempted to train a convolutional neural network using only files with a fixed size. That is, we divide the original training and testing dataset into four subsets, the file size in the first three subsets are fixed at 512, 1024 and 4096, respectively. The fourth subsets contains data with various size other than 512, 1024 and 4096.

Subsequently, we train a convolutional neural networks to each of the first three datasets. The model trained using data with a specific file size is used to classify files with that size only. For example, we train a model using files with size at 512 and then this model is only used to classify files that have 512 bytes. For the fourth subset that has various files size, we use the classifier fitted to the original dataset to do prediction.

Lastly, the prediction from each of the four classifiers are combined together as the final prediction. We use M-ConvNet to denote the multiple classifiers trained as above.

5 Results and Discussion

The final performance of each training scheme on Kaggle is shown in table 2.

	ConvNet	ConvNet(E)	M-ConvNet
Kaggle score	0.99169	0.99247	0.99132

Table 2: Performance of models on training dataset.

The highest Kaggle score comes from ConvNet(E). Combining the results from multiple classifiers trained on different subsets is a way to decrease the variance associated with individual prediction from each classifier. Table 3 illustrates the performance of each underlying model of ConvNet(E). The AUC scores are based on validation dataset.

model	Accuracy(train)	Accuracy(valid)	AUC
model_1	0.9868	0.9791	0.99801
model_2	0.9836	0.9792	0.99777
model_3	0.9893	0.9789	0.99745

Table 3: Performance of individual network in ConvNet(E).

As for M-ConvNet, the performance of individual classifier is shown in table 4.

file size	Accuracy (train)	Accuracy (valid)
512	0.9902	0.9833
1024	0.9916	0.9806
4096	0.9677	0.9546

Table 4: Performance individual network in M-ConvNet.

The classifier fitted to data with file size at 512 and 1024 obtain high accuracy on both training and validation datasets. This comparatively better performance trained on data of smaller size and relatively worse performance on the largest size show that our classifier may not be good at predicting files with big size. And so improvement of its predictability on large sized files is needed in future.

Epoch. In order to get the best ConvNet, we monitor accuracy and AUC score after each epoch during training. In terms of implementation, we implement and pass in appropriate callback functions when we train our Keras model.

We find that AUC score on validation set does not necessarily increase as the validation accuracy increase. Therefore, to select a model with good performance, we need to balance between AUC score and prediction accuracy.

To facilitate our model selection procedure, we store the model after each epoch and train for 50 epochs. Then we select the best model from these 50 models based on the AUC score and prediction accuracy that we monitored during training.

6 Conclusion

In our work we build a convolutional neural network to classify PE header of exe files. We proposed three training schemes to improve the performance of our classifier. It turns out that combining the predictions from classifiers trained on different subsets of the original data gives the highest accuracy and AUC score. This implied that ensemble learning techniques could be used to tune the prediction to expected outcome. As for multiple classifiers train on data with different file size, the classifiers' performance vary, and the best performance comes from classifier train on dataset with the most abundant file size. Hence multi-classifier training scheme may work better than single classifier if more data become available for training in future.

In future, this approach could be potentially used for other similar domain like android malware detection, since there is no requirement on domain knowledge.

References

[1] Ross Anderson et al. "Measuring the Cost of Cybercrime". In: (Oct. 2013), pp. 265–300.

[2] Wenyi Huang and Jack W. Stokes. "MtNet: A Multi-Task Neural Network for Dynamic Malware Classification". In: *Detection of Intrusions and Malware, and Vulnerability Assessment* 9721 (2016), pp. 399–418.

[3] Bojan Kolosnjaji et al. "Deep Learning for Classification of Malware System Call Sequences." In: *AI 2016: Advances in Artificial Intelligence* 9992 (2016), pp. 137–149.

[4] Marek Krcal et al. "Deep Convolutional Malware Classifiers Can Learn from Raw Executables and Labels Only". In: *ICLR 2018 Workshop* 9721 (2018), pp. 399–418.

[5] Edward Raff et al. "Malware Detection by Eating a Whole EXE". In: (Oct. 2017).