An Effective Ensemble Deep Learning Framework for Malware Detection

Dinh Viet Sang
Hanoi University of Science and Technology
Hanoi, Vietnam
sangdv@soict.hust.edu.vn

Dang Manh Cuong
Hanoi University of Science and Technology
Hanoi, Vietnam
dangmc95@gmail.com

Le Tran Bao Cuong
Hanoi University of Science and Technology
Hanoi, Vietnam
ltbclqd2805@gmail.com

ABSTRACT

Malware (or malicious software) is any program or file that brings harm to a computer system. Malware includes computer viruses, worms, trojan horses, rootkit, adware, ransomware and spyware. Due to the explosive growth in number and variety of malware, the demand of improving automatic malware detection has increased. Machine learning approaches are a natural choice to deal with this problem since they can automatically discover hidden patterns in large-scale datasets to distinguish malware from benign. In this paper, we propose different deep neural network architectures from simple to advanced ones. We then fuse hand-crafted and deep features, and combine all models together to make an overall effective ensemble framework for malware detection. The experiment results demonstrate the efficiency of our proposed method, which is capable to detect malware with accuracy of 96.24% on our large real-life dataset.

CCS CONCEPTS

• **Security and privacy** → **Malware and its mitigation**; • **Computer systems organization** → **Neural networks**;

KEYWORDS

Malware Detection, Residual Convolutional Neural Network, Ensemble Method.

ACM Reference Format:
Dinh Viet Sang, Dang Manh Cuong, and Le Tran Bao Cuong. 2018. An Effective Ensemble Deep Learning Framework, for Malware Detection. In *SoICT '18: Ninth International Symposium on Information and Communication Technology, December 6–7, 2018, Da Nang City, Viet Nam*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/xxx.xxx

SoICT '18, December 6–7, 2018, Da Nang City, Viet Nam

ACM ISBN 978-1-4503-6539-0/18/12…$15.00
https://doi.org/10.1145/xxx.xxx

1 INTRODUCTION

Malware is a huge concern of the world. These malicious programs can perform a variety of functions, including stealing, encrypting or deleting sensitive data, altering core computing functions and monitoring users' computer activity without their permission. In 2017, a cyberattack has taken place on an enormous scale. The attack has affected 200.000 computers in 150 countries. A type of malware called WannaCry tried to access a personal computer and encrypted all data on it. The user is required to send a ransom to receive data, otherwise all data will be deleted. WannaCry is considered a nightmare for computer users in recent times. From those dangers, the problem of how to detect malware is attracting the attention of many researchers and businesses.

The malware detection problem can be defined as a classification task: given a training dataset containing Portable Executable (PE) files and their corresponding labels: malware or benign. The computer needs to learn from the training set to automatically classify each new PE file into one of two categories: malware or benign.

Recently, machine learning has helped us to deal with various difficult tasks including malware detection. Traditional machine learning approaches for malware detection are usually based on analyzing binary files and divided into two categories: static or dynamic analysis [6].

In dynamic analysis, binary files are executed and their actions are recorded. In theory, dynamic analysis provides a way to directly observe the behavior of malware [13]. However, in practice, executing binary files is not simple. Malware is increasingly sophisticated, it can detect whether it is being run in the sandbox to prevent malicious behavior to it [7].

In static analysis, binary files are decompiled into assembly representation. Static analysis is an appropriate approach to use machine learning for malware detection.

Kephart et al. were the first to use machine learning for malware detection [12]. Machine learning for malware detection usually consists of two phases: The first is extracting features from binary files, the second is the learning phase that trains a classification model using features from the first phase. A lot of static features were proposed to extract from binary files such as entropy histogram [15], n-grams [3], opcodes, informational entropy [17], image representation [14].

Nowadays, with the development of deep learning techniques, CNN models such as ResNet [8], VGG [16], DenseNet [10] are widely used in classification problems. There are

some studies that use simple feed forward neural networks to deal with malware detection [4] [5] [12]. Nevertheless, there is still not much research on applying more advanced deep learning techniques for malware detection.

In this paper, we use different methods to extract features from PE files and propose different effective deep neural networks to detect malware with high accuracy rate. Finally, we combine all models together to form an ensemble framework to achieve higher accuracy of malware detection. Our system has high scalability to deliver deployable detectors.

Our main contributions are: (1) to propose different effective deep neural networks and combine them together to make an effective deep learning framework for malware detection; (2) to collect a large set of benign software programs which is potentially made available for research community.

The rest of the paper is organized as follows. In section 2, we briefly introduce Residual Network (ResNet) and our proposed method. Finally, in section 3, we introduce our real-life dataset and show the experimental results. The conclusion is in section 4 with a discussion for the future work.

2 OUR PROPOSED FRAMEWORK

Our ensemble framework is divided into three main steps. The first step is to extract four different types of features from the static benign and malicious binaries. The next step is to train different proposed models with features extracted from the previous step. Here we use two kinds of models: feed-forward neural networks and residual deep neural networks. The final step is to ensemble all models together to build an overall effective framework for malware detection. In the remainder of this section, we describe each component of our framework in detail.

2.1 Feature Engineering

2.1.1 n-gram. A n-gram is a contiguous sequence of n items from a given sequence. A PE sample is as a sequence of hex values (Figure. 2). Each element in a byte sequence get integer value from 0 to 255. Because of that, a PE sample can be effectively represented by n-gram. 1-gram (1G) features are an example of n-gram, which represent just one byte frequency, and thus they are described by a 256-dimensional vector. 2-gram (2G) features represent two combinational bytes frequency, and hence they are described by a 256^2-dimensional vector. Due to the lack of computational complexity and the limitation of memory space, in our experiments, we just use 1-gram features.

2.1.2 Entropy Histogram. This feature was proposed by Saxe et al. in [15]. To extract entropy histogram, they slide a 1024 byte window over an input binary. For each window, the 2-base entropy of the window is computed. Entropy value e_i of the i^{th} sliding window is defined as follows:

$$e_i = -\sum_{j=1}^{m} p_j log_2(p_j), \quad (1)$$

where p_j is the number of occurrences of byte j, and m is the number of different bytes in the i^{th} window.

Each individual byte occurrence in the window and the computed entropy are stored in a list of 1024 pairs. In the next step, they compute two dimensional histogram base on this list, where the entropy has sixteen evenly sized bins over the range $[0, 8]$, the byte has sixteen evenly sized bins over the range $[0, 255]$. Finally, a 256 dimensional vector is created.

Algorithm 1 describes the pseudo-code of entropy histogram feature extraction algorithm.

Algorithm 1 Entropy histogram algorithm

```
Input: A set S of continous byte in PE file.
Output: A 256-dimensional vector.
procedure EntropyHistogramFeature
    window_size = 1024;
    S = A set of continous byte with size equal to window_size in PE file;
    for s ∈ S do
        e = entropy(s);
        for i = 0; i ¡ window_size; i++ do
            x = s[i] % 16;
            v[x][e] += 1;
        end for
        result = []
        cnt = 0
        for i = 0; i ¡ 16; i++ do
            for j = 0; j ¡ 16; j++ do
                cnt += 1;
                result[cnt] = v[i][j];
            end for
        end for
    end for
    return result;
end procedure
```

2.1.3 Image Representation. This feature is highly motivated by the work in [14]. A binary file can be read as a vector of 8 bit unsigned integers and organized into a 2D array. Then, this array can be visualized as a gray image with pixel value in the range [0,255]. The width of each image depends on the size of the corresponding binary file. Table 1 describes the dependency of image width with respect to file size. In order to train convolutional neural network, all images have to be resized to 64×64 pixels. Fig. 1 illustrates some images after resizing. Algorithm 2 describes how to get the image representation from a PE file.

2.1.4 Application Programming Interfaces. APIs can be used to determine the behavior of software programs. However, the total number of APIs is extremely large, considering them all would bring little or no meaningful information for malware detection. Therefor, we just used 794 APIs that were analyzed on 500,000 malware samples [2].

(a) Malware **(b) Benign**

Figure 1: The images of malware and benign samples after resizing.

```
0000   FF D8 FF E1   1D FE 45 78   69 66 00 00   49 49 2A 00
0010   08 00 00 00   09 00 0F 01   02 00 06 00   00 00 7A 00
0020   00 00 10 01   02 00 14 00   00 00 80 00   00 00 12 01
0030   03 00 01 00   00 00 01 00   00 00 1A 01   05 00 01 00
0040   00 00 A0 00   00 00 1B 01   05 00 01 00   00 00 A8 00
0050   00 00 28 01   03 00 01 00   00 00 02 00   00 00 32 01
0060   02 00 14 00   00 00 B0 00   00 00 13 02   03 00 01 00
0070   00 00 01 00   00 00 69 87   04 00 01 00   00 00 C4 00
0080   00 00 3A 06   00 00 43 61   6E 6F 6E 00   43 61 6E 6F
0090   6E 20 50 6F   77 65 72 53   68 6F 74 20   41 36 30 00
00A0   00 00 00 00   00 00 00 00   00 00 00 00   B4 00 00 00
00B0   01 00 00 00   B4 00 00 00   01 00 00 00   32 30 30 34
00C0   3A 30 36 3A   32 35 20 31   32 3A 33 30   3A 32 35 00
00D0   1F 00 9A 82   05 00 01 00   00 00 86 03   00 00 9D 82
00E0   05 00 01 00   00 00 8E 03   00 00 00 90   07 00 04 00
```

Figure 2: An example of PE file.

Algorithm 2 Image representation algorithm

Input: A list S of continous byte in PE file.
Output: A gray image represents corresponding PE file.
procedure ImageRepresentation
    image = []
    for i = 0; i ¡ S.size; i++ **do**
        image[i] = S[i]
    end for
    /* Choose the size of image based on the size of PE file */
    w, h = chooseSize(image);
    /* Reshape the image */
    image = reshape(image, w, h)
    return *image*;
end procedure

Table 1: Image size corresponds to file size

File size (KB)	Image width (pixel)
< 10	32
10 - 30	64
30 - 60	128
60 - 100	256
100 - 200	384
200 - 500	512
500 - 1000	784
> 1000	1024

2.2 Modelling

2.2.1 Simple feed-forward neural network architectures. In this section, we try to use a simple feed-forward neural network architecture to detect malware from three types of feature: n-gram feature, entropy histogram feature and application programming interface feature. This network contains 2 hidden layers. Number of neurons in input layer and hidden layer is showed in Table 2.

In each hidden layer, we use ReLU activation function combined with Batch Normalization layer [11]. The output layer

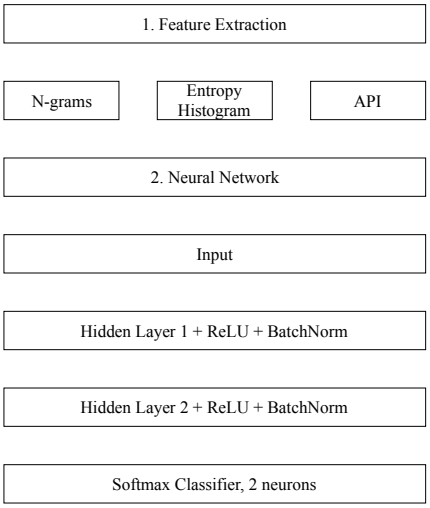


Figure 3: Our proposed simple feed-forward neural networks for hand-crafted features.

Table 2: Number of neurons in each layer with corresponding feature

Feature	Number of neurons in input layer	Number of neurons in hidden layer
n-grams	256	128
Entropy histogram (EH)	256	128
API	794	256

is a softmax classifier with 2 neurons. We called this architecture as **NN-Ngram, NN-EH, NN-API**, respectively. Fig. 3 shows the overview of our architecture.

2.2.2 Deep residual convolutional neural network. ResNet [8] is known as one of the most efficient CNN architectures so far. In order to enhance the information flow between layers, ResNet uses shortcut connections between layers. The original variant of ResNet is proposed by He et al. in [8] with different numbers of hidden layers: ResNet-18, ResNet-34 or ResNet-50, ResNet-101 and ResNet-152. He et al. then introduce an improved variant of ResNet (called ResNet_v2) in [9] which shows that the pre-activation order "conv - batch normalization - ReLU" is consistently better then post-activation order "batch normalization - ReLU - conv".

Based on the aforementioned ResNet_v2 architecture, we propose a ResNet_v2-like deep neural network to detect malware. The input of our network is a 64×64 gray image created by the method mentioned in above section. Our ResNet_v2-like architecture has 3 residual blocks. Each block contains some *sub-sampling blocks* and *identity blocks*. The architectures of identity blocks and sub-sampling blocks are shown in Fig. 5a and Fig. 5b. For both these two kinds of blocks,

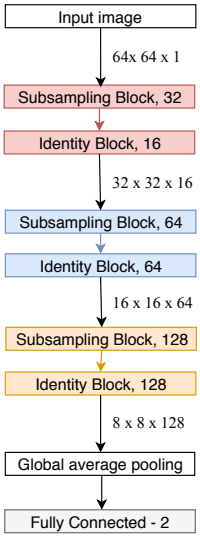


Figure 4: Our proposed residual network architecture - RNSIMG.

we use the bottleneck architecture with *base depth* m that consists of three conv layers: a 1×1 conv layer with m filters followed by a 3×3 conv layer with m filters and a 1×1 conv layers with $4m$ filters. The identity blocks and sub-sampling blocks are distinguished by the stride value in the second conv layer and the shortcut connection. In sub-sampling blocks, we use a conv layer with stride 2 instead of stride 1 as in identity blocks. We called our proposed residual network as RNSIMG (see Fig. 4).

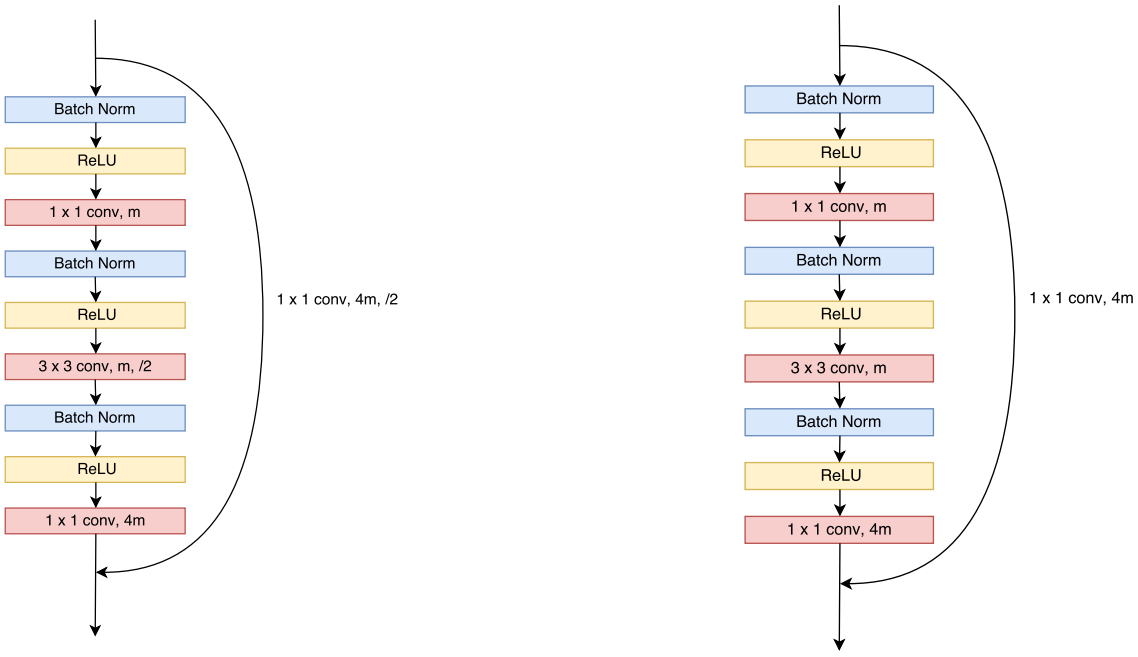
2.2.3 Feature fusion and ensemble framework. We also propose to improve RNSIMG model by fuse deep feature extracted from RNSIMG with n-gram feature, entropy histogram feature and API feature. Given an input image, after the global average pooling layer of RNSIMG, we obtain a 128-dimensional deep feature vector. We then concatenate this vector with corresponding n-gram feature, entropy histogram feature and API feature of the input image to form a new 1434-dimensional vector. Next, we use two fully connected layers with 256 neurons and a final softmax classifier with 2 neurons. We called this architecture as RNSALL (see Fig. 6).

Finally, we propose to combine all model including NN-Ngram, NN-EH, NN-API, RNSIMG, RNSALL to make an ensemble framework that is shortly called as ENSEMBLE. The way that we combine different models is to simply average the probabilistic prediction results of single models.

3 EXPERIMENTS AND EVALUATION

3.1 Dataset

In this paper, we use a set of 28.000 PE files. Among these PE files, 14.000 samples of malware were collected from the Virusshare [1], 14.000 samples of benign were collected from



(a) Sub-sampling block

(b) Identity block

Figure 5: The architecture of identity blocks and sub-sampling blocks.

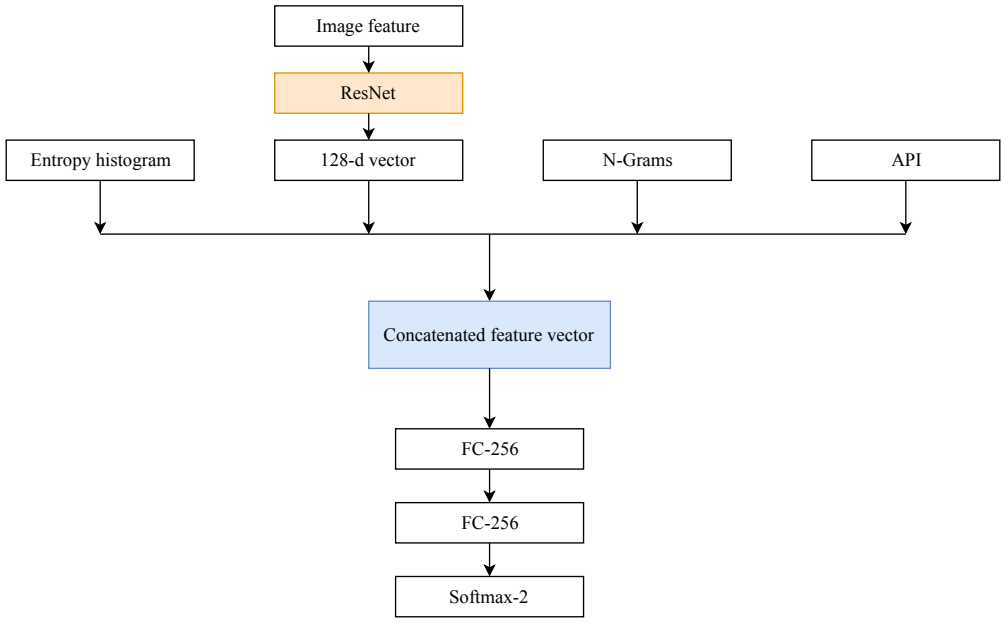


Figure 6: Our proposed RNSALL architecture.

VIETTEL Corp. This dataset is divided in training set and test set as follows:

- Training set: 10.500 samples of malware and 10.500 samples of benign.
- test set: 3.500 samples of malware and 3.500 samples of benign.

3.2 Implementation details

We conduct the experiments using Python programming-language and TensorFlow framework on a computer with the following specifications: Ubuntu Operating System 16.04 64 bit, 16GB RAM, GPU NVIDIA GeForce GTX 1080.

Training phase: Our models are trained by using Stochastic Gradient Descent alogrithm with momentum 0.9. We set the batch size equal to 64. We initialize all weights using a Gaussian distribution with zero mean and standard deviation 0.01. The L2 weight decay is $\lambda = 0.005$. We trained our networks with 40.000 steps. The initial value of learning rate is 0.001 and we decrease learning rate two times every 10.000 steps.

Our models are evaluated based on k-fold cross-validation. This method splits the training data into k parts with the same size. The model evaluation is performed through loops, each loop selects $k-1$ parts of data for training the models and the rest is used as validation set. In this paper, we use $k = 10$. The average cross-validation accuracy of 10 folds is then used for comparison between different models.

Testing phase: For each model among NN-Ngram, NN-EH, NN-API, RNSIMG and RNSALL and for each fold during 10-fold cross-validation, we choose the best checkpoint with the highest validation accuracy. Hence, for each model we have 10 best checkpoints corresponding to 10 folds. These checkpoints are then ranked based on the their validation accuracy. Finally, five top checkpoints of each model are selected for ensemble inference. It means we combine 25 different checkpoints of 5 models for evaluating the test set. The combining way is to simply average the probabilistic results of all 25 checkpoints.

3.3 Experimental results

We report our accuracy with different models in Table 3. As one can see, using ensemble framework gives higher accuracy than using each single model in malware detection.

With simple neural network architecture, the best accuracy we achieve is **94.86%** with entropy histogram feature. When we used all of 4 features with RNSALL model, we get a better result with **95.14%** accuracy. In particular, with the ensemble model, we achieve the highest performance **96.24%**. We also show our True/False Positive/Negative value on our dataset with difference models in Table 4.

Besides the overall accuracy, we also use other performance measurements to compare our models. Particularly, we use four measurements including true positive rate (TPR), false positive rate (FPR), true negative rate (TNR) and false negative rate (FRN), which are defined as follows:

$$TPR = \frac{\#\,true\,positives}{\#\,true\,positives + \#\,false\,negatives} \quad (2)$$

$$FNR = \frac{\#\,false\,negatives}{\#\,true\,positives + \#\,false\,negatives} \quad (3)$$

$$TNR = \frac{\#\,true\,negatives}{\#\,true\,negatives + \#\,false\,positives} \quad (4)$$

Table 3: Accuracy on our dataset with different models

Model	10-folds cross-validation Accuracy	Test accuracy
NN-Ngram	94.40%	92.74%
NN-EH	95.05%	94.86%
NN-API	94.45%	93.56%
RNSIMG	95.48%	93.80%
RNSALL	95.80%	95.14%
ENSEMBLE	-	**96.24%**

Table 4: True/False Positive/Negative results on our dataset with different models

Model	TP	FP	FN	TN
NN-Ngram	3232	240	268	3260
NN-EH	3300	160	200	3340
NN-API	3363	314	137	3186
RNSIMG	3222	156	278	3344
RNSALL	3316	156	184	3344
ENSEMBLE	3386	149	114	3351

$$FPR = \frac{\#\,false\,positives}{\#\,true\,negatives + \#\,false\,positives} \quad (5)$$

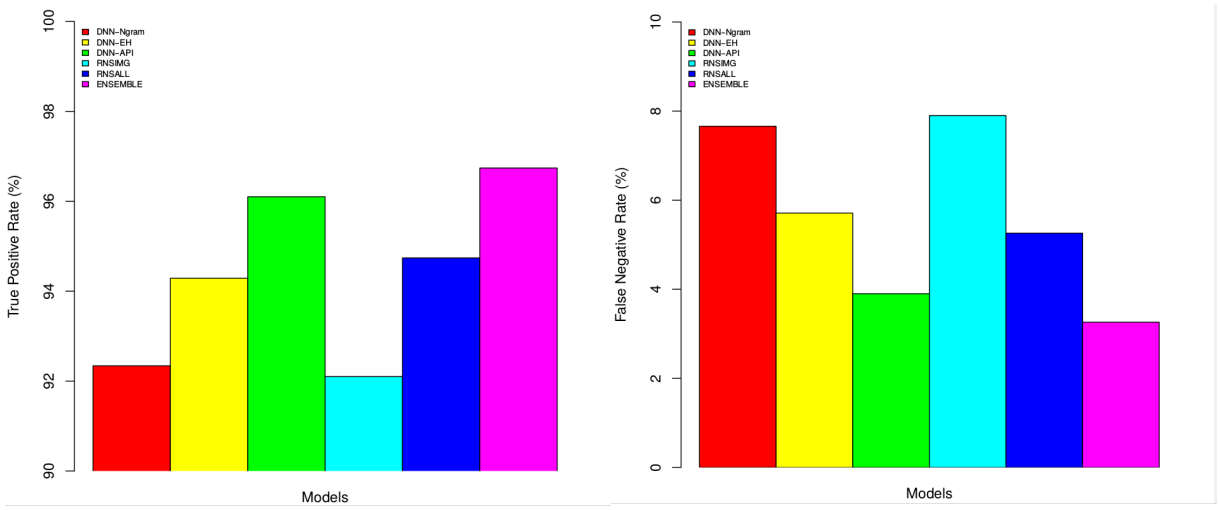
Fig. 7a and Fig. 7b show TPR (which is also known as recall) and FNR of our different models. The higher TPR value is, the better model is. Conversely, the lower FNR value is, the better model is. As one can see, our ensemble framework achieves the highest TPR and lowest FNR in testing set.

Fig. 8a and Fig. 8b show TNR and FPR values of our different models. The higher TNR value is, the better model is. Meanwhile, the lower FPR value is, the better model is. According to these measurements, the ensemble framework still yields the best performance compared to all single model.

4 CONCLUSION

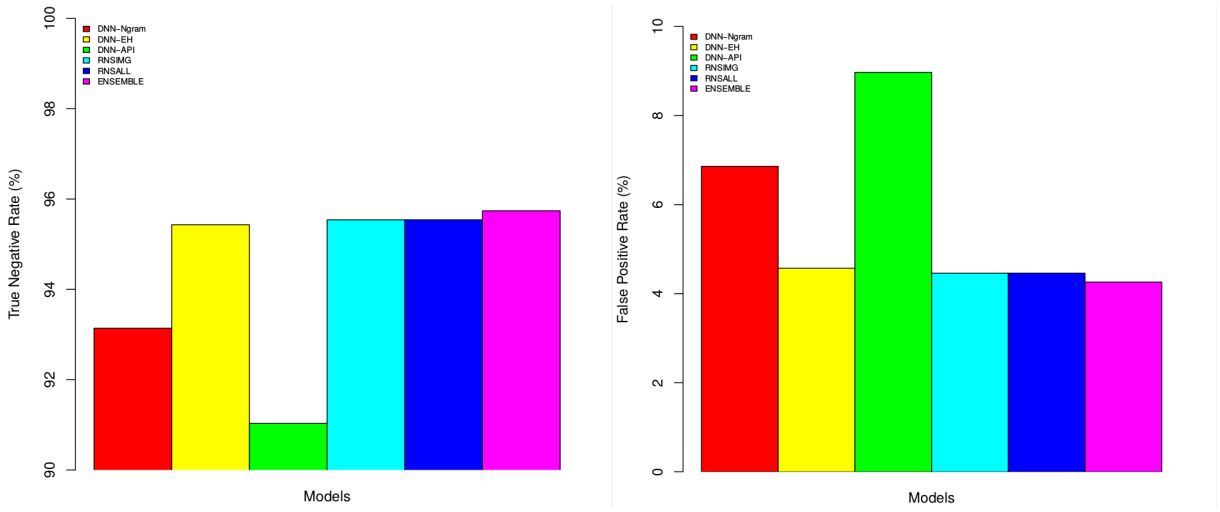
In this paper, we propose different effective deep learning models for malware detection. We use both hand-crafted features such as n-gram, entropy histogram, application programming interface and image representation, as well as deep features extracted from deep neural networks to fit the models. Finally, we combine various single models to make an overall effective ensemble deep learning framework for malware detection. The experimental result shows that our proposed framework yields promising accuracy.

In the future, we would like to try some recent advanced network architectures like DenseNet or NASNet, as well as test our models on larger datasets to improve accuracy and make the framework deployable in real security systems.



(a) True Positive Rate **(b) False Negative Rate**

Figure 7: True Positive Rate, False Negative Rate of our proposed models



(a) True Negative Rate **(b) False Positive Rate**

Figure 8: True Negative Rate and False Positive Rate of our proposed models

REFERENCES

[1] VirusShare. https://virusshare.com/. (2011).

[2] Top maliciously used apis. https://www.bnxnet.com/top-maliciously-usedapis/. (2015).

[3] Tony Abou-Assaleh, Nick Cercone, Vlado Keselj, and Ray Sweidan. 2004. N-gram-based detection of new malicious code. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, Vol. 2. IEEE, 41–42.

[4] Răzvan Benchea and Dragoş Teodor Gavriluţ. 2014. Combining restricted boltzmann machine and one side perceptron for malware detection. In *International Conference on Conceptual Structures*. Springer, 93–103.

[5] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. 2013. Large-scale malware classification using random projections and neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 3422–3426.

[6] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. 2012. A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)* 44, 2 (2012), 6.

[7] Dan Fleck, Arnur Tokhtabayev, Alex Alarif, Angelos Stavrou, and Tomas Nykodym. 2013. Pytrigger: A system to trigger & extract user-activated malware behavior. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*. IEEE, 92–101.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European conference on computer vision*. Springer, 630–645.

[10] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. 2016. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993* (2016).

[11] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*. 448–456.

[12] Jeffrey O Kephart, Gregory B Sorkin, William C Arnold, David M Chess, Gerald J Tesauro, Steve R White, and TJ Watson. 1995. Biologically inspired defenses against computer viruses. In *IJCAI (1)*. 985–996.

[13] Andreas Moser, Christopher Kruegel, and Engin Kirda. 2007. Limits of static analysis for malware detection. In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*. IEEE, 421–430.

[14] Lakshmanan Nataraj, Sreejith Karthikeyan, Gregoire Jacob, and BS Manjunath. 2011. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*. ACM, 4.

[15] Joshua Saxe and Konstantin Berlin. 2015. Deep neural network based malware detection using two dimensional binary program features. In *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*. IEEE, 11–20.

[16] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[17] Michael Weber, Matthew Schmid, Michael Schatz, and David Geyer. 2002. A toolkit for detecting and analyzing malicious software. In *null*. IEEE, 423.