International Journal of Digital Crime and Forensics
Volume 12 • Issue 1 • January-March 2020

A Deep Learning Framework for Malware Classification

Mahmoud Kalash, University of Manitoba, Winnipeg, Canada
Mrigank Rochan, University of Manitoba, Winnipeg, Canada
Noman Mohammed, University of Manitoba, Winnipeg, Canada
Neil Bruce, Ryerson University, Toronto, Canada
Yang Wang, University of Manitoba, Winnipeg, Canada
Farkhund Iqbal, Zayed University, Abu Dhabi, UAE

ABSTRACT

In this article, the authors propose a deep learning framework for malware classification. There has been a huge increase in the volume of malware in recent years which poses serious security threats to financial institutions, businesses, and individuals. In order to combat the proliferation of malware, new strategies are essential to quickly identify and classify malware samples. Nowadays, machine learning approaches are becoming popular for malware classification. However, most of these approaches are based on shallow learning algorithms (e.g. SVM). Recently, convolutional neural networks (CNNs), a deep learning approach, have shown superior performance compared to traditional learning algorithms, especially in tasks such as image classification. Inspired by this, the authors propose a CNN-based architecture to classify malware samples. They convert malware binaries to grayscale images and subsequently train a CNN for classification. Experiments on two challenging malware classification datasets, namely Malimg and Microsoft, demonstrate that their method outperforms competing state-of-the-art algorithms.

KEYWORDS

Convolutional Neural Networks, Deep Learning, Framework, Malware Classification

INTRODUCTION

Malware is malicious software (e.g. viruses, worms, Trojan horses, and spyware) that damages or performs harmful actions on computer systems (Malware Definition, 2017). In this Internet-age, many malware attacks happen that pose serious security threats to financial institutions and everyday users. Prior studies also highlight that malware analysis is crucial for digital forensic investigation (Kaur & Nagpal, 2012). Figure 1 represents the number of malwares spotted in a year. It is clear that the total number of instances of malware has drastically increased over the years. For example, Symantec reported that more than 357 million new variants of malware were observed in 2016 (Internet Security Threat Report, 2017). One of the main reasons for this high volume of malware samples is the extensive use of obfuscation techniques by malware developers, which means that malicious files from the same malware family (i.e. similar code and common origin) are constantly modified and/or obfuscated. In order to cope with the rapid evolution of malware, it is essential to develop robust

DOI: 10.4018/IJDCF.2020010105

malware classification techniques that are tolerant of variants of malware files that belong to same family. Towards this endeavor, we propose a deep learning architecture for malware classification.

Conventional methods use binary signatures of malware for analysis. Malware typically carries a uniquely identifiable signature. Signature-based methods were extensively used in the past in anti-virus software. Given the exponential increase in malware files and degree of variation, these signature-based methods are not scalable. Other methods for malware analysis include static and dynamic code analysis (Nataraj, Karthikeyan, Jacob, & Manjunath, 2011). In static analysis, the malware code is disassembled to find malicious patterns. In contrast, dynamic analysis is done by executing the malicious program in a virtual environment and its behavior is analyzed based on execution trace. Dynamic analysis is more effective than static as it does not require disassembling, but it is time consuming and resource intensive. Also, it is possible that during the dynamic analysis malicious behaviors go unnoticed because the virtual environment may not be able to simulate the exact real conditions (Nataraj, Karthikeyan, Jacob, & Manjunath, 2011).

Previous research on malware classification suggest that malware samples typically fall into a family that share common behavior. Most new malware are variants of existing ones (Nataraj, Karthikeyan, & Manjunath, 2015). Hence, the prospect of building a method that can efficiently classify malware based on its family irrespective of being a variant, seems especially fruitful and a means of dealing with the rapid growth of malware.

In this paper, we take a completely different approach to analyze and classify malware compared with traditional methods. We use a Convolutional Neural Network (CNN), a deep learning architecture, to tackle this problem.

Recently, CNNs have produced state-of-the-art performance on the image classification task in the field of computer vision. Motivated by this success, we translate the malware classification problem into the image classification problem to be addressed using CNNs. We firstly represent each malware binary file as a grayscale image and then train a CNN architecture to perform classification. Previous work (Nataraj, Karthikeyan, Jacob, & Manjunath, 2011) showed that malware belonging to same family are visually similar, which is beneficial with respect to the capacity for a CNN to detect relevant patterns. This is especially true given that the same or similar code is usually used to generate variants of malware. However, the method proposed in (Nataraj, Karthikeyan, Jacob, & Manjunath, 2011) have several shortcomings (See the Related Work section).

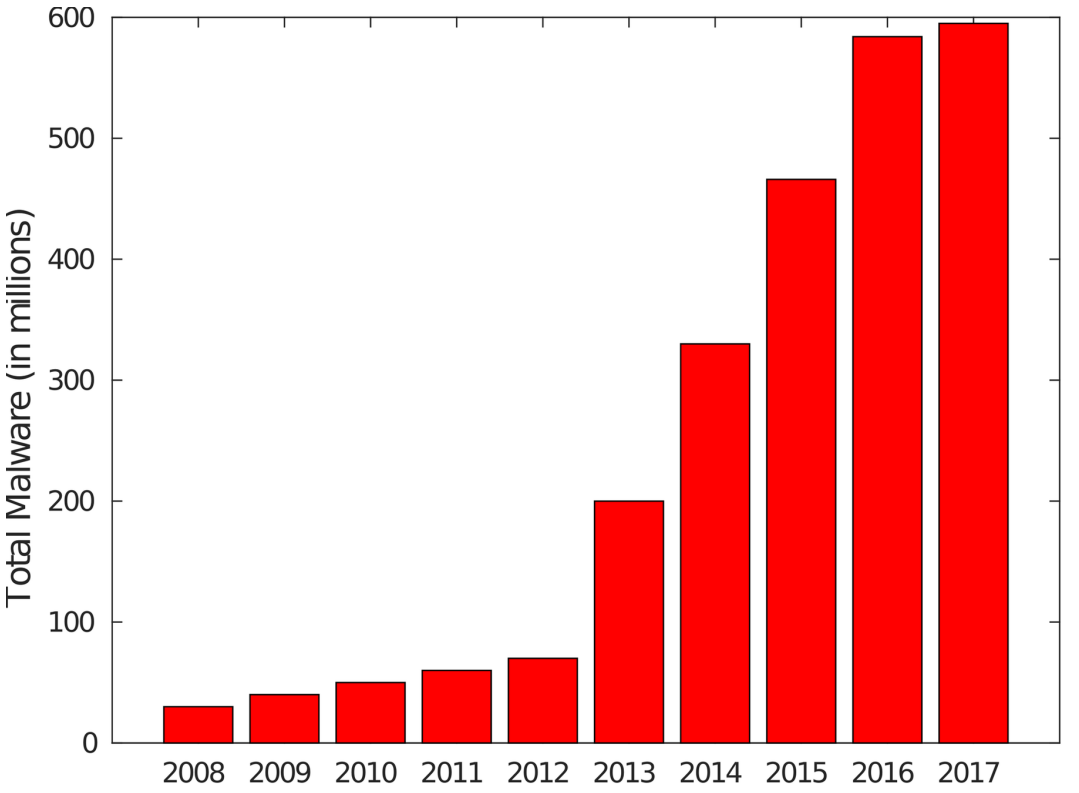
There is a recent work (Gibert Llauradó, 2016) that uses CNN for malware classification, but it is still very shallow in architecture. In computer vision, researchers have shown that deeper CNN architectures (e.g. (Simonyan & Zisserman, 2014)) are helpful in minimizing error for image classification tasks. In this paper, we take the approach of vision researchers and adopt their technique for malware classification.

CONTRIBUTIONS

We make the following main contributions in this paper:

1. We develop a deep convolutional neural network (CNN) architecture for malware classification, which is generic in nature, unlike traditional methods. Existing techniques that achieve high accuracy are often tailored for a specific dataset. In contrast, the proposed approach is data independent and learns the discriminative representation from the data itself rather than depending on hand-crafted feature descriptors.
2. We show that malware classification with higher accuracy is possible even if only a portion of malware sample is available. As far as we know, we are the first to develop CNN-based method that can classify malware samples using only partial knowledge of the properties of samples. In addition, there are a number of advantages of the proposed approach that are discussed in the Advantages section.

Figure 1. Last 10 years malware statistics (Total Malware, 2017). Total volume of malware has increased drastically over the last 10 years.



3. We perform extensive experiments on two benchmark datasets (Malimg (Nataraj, Karthikeyan, Jacob, & Manjunath, 2011) and Microsoft (Microsoft, 2017)) which demonstrate that our approach outperforms state-of-the-art methods. We obtain 99.97% accuracy on the Microsoft dataset while the winning team (Wang, Liu, & Chen, 2017) of the Microsoft Malware Classification Challenge (BIG 2015) (Microsoft, 2017) achieved an accuracy of 99.87% on the same dataset. Note that a preliminary version of this work has also appeared in (Kalash, et al., 2018).

The remainder of the paper is organized as follows. We present related work on malware classification in the Related Work section. The Background section discusses relevant background for this work. The Approach section describes our proposed deep learning framework. We present experimental settings and evaluation on different datasets in the Experiments section. The Advantages section lists the advantages of our proposed method. Finally, we conclude and discuss some interesting paths for future work in the Conclusion section.

RELATED WORK

Previous work on malware classification can be broadly classified into two categories: non-machine learning methods and machine learning-based methods

Non-Machine Learning Methods: In the past, malware was detected using static or dynamic signature-based techniques (Idika & Mathur, 2007). Static analysis uses syntax or structural properties of the program in order to detect malware even before the program under inspection executes. However,

malware developers use various encryption, polymorphism and obfuscation techniques (You & Yim, 2010) (Sung, Xu, Chavez, & Mukkamala, 2004) to overcome these detection algorithms. In the dynamic approach, malware is executed in a virtual environment and its behavior is analyzed in order to detect harmful actions during or after the program execution. Although dynamic analysis of malware is a promising approach, it is still very complex and time consuming (Nataraj, Karthikeyan, Jacob, & Manjunath, 2011). The major drawback of classical signature-based detection is that it is not scalable and its effectiveness can be undermined with the growing variants of malware (Rieck, Holz, Willems, Düssel, & Laskov, 2008). Therefore, we adopt another approach which is based on intelligent machine learning algorithms.

Machine Learning-Based Methods: In order to address the limitations of the aforementioned methods and inspired by the fact that variants of malware families typically share similar behavior patterns (Nataraj, Karthikeyan, Jacob, & Manjunath, 2011), anti-malware organizations started to develop more sophisticated classification methods based on data mining and machine learning techniques (Siddiqui, Wang, & Lee, 2008). These techniques use different feature extraction (i.e. data representation) methods to build more intelligent malware detection systems.

Rieck et al. (2008) proposed a learning-based approach for automatic classification of malware. They analyzed the behavior of 10,072 malware samples that were labeled and divided into 14 malware families by an anti-virus software. Based on their analysis, a string feature vector (i.e. frequency of some specific strings) was generated for each malware sample. Then, an SVM-based classifier was trained that produced a classification accuracy of 88% on the dataset. Schultz et al. (2001) proposed a Naive Bayes classifier for malware classification. Their approach makes use of strings and byte sequences as features. They were able to achieve as much as twice the accuracy when compared with the traditional signature-based methods. The drawback of these methods is that using extracted strings from a malware executable as features does not provide robust performance because malware authors can easily change them. Kolter et al. (Kolter & Maloof, 2004) tried to classify malicious executable using byte-sequence n-grams (i.e. a contiguous sequence of n hexadecimal values from a given malware file) as features. They trained different types of classifiers (Naive Bayes, decision trees, support vector machines, etc.) and then evaluated and compared each classifier performance for different sizes of n-grams. However, these shallow learning techniques are not very scalable with the growing number of malware samples. Another drawback is that these methods are not fully automated, which implies that it is first necessary to determine the best feature representation based on the available malware file types and then design feature representations (or extraction by hand), which is followed by retraining of an associated classifier. In order to tackle these problems, we develop a deep learning architecture that is robust and more general in nature.

Nataraj et al. (2011) present a strategy closely related to our proposed method, in particular in the way that the malware files are pre-processed. This method uses a binary vector (a binary string of zeros and ones) that represents a malware executable to generate a grayscale image. We also convert the malware binary files to grayscale images following their method. In order to characterize and classify the generated malware images, they used a standard image feature descriptor, GIST (Oliva & Torralba, 2001) (Torralba, Murphy, Freeman, Rubin, & others, 2003), to compute texture features. Then, the grayscale malware images are classified using k-nearest neighbor algorithm. Since their feature extractor only captures the global image texture, malware developers can easily attack this approach by swapping sections in malware binary or by adding some redundant data. Another problem with their approach is that it is not scalable as their learning algorithm complexity grows with the increase in number of malware samples.

Drew et al. (Drew, Moore, & Hahsler, 2016) performed malware classification on the Microsoft Malware dataset (Microsoft, 2017) using modern gene sequence classification tool and achieved 97.42% accuracy. They expanded upon their work later (Drew, Hahsler, & Moore, 2017) to include new feature extraction and ensemble techniques for the files generated by the Interactive Disassembler Tool (IDA) in the dataset and achieved higher accuracy of 98.59%. Ahmadi et al. (2016) extracted

and fused 13 different groups of features that specifically target the Microsoft Malware Dataset. They trained a classifier based on the XGBoost (Chen & Guestrin, 2016) technique and achieved an accuracy of 99.77% on the training set of the Microsoft malware classification dataset. The winning team (Wang, Liu, & Chen, 2017) of the Microsoft Malware Classification Challenge (BIG 2015) (Microsoft, 2017) used a highly complex combination of features and trained a classifier based on the XGBoost (Chen & Guestrin, 2016) technique that achieved an accuracy of 99.87% on the training set of the Microsoft malware dataset. Although these works achieve high performance on Microsoft malware dataset, they are designed to target higher performance on specific datasets. In contrast, our aim in this work is to build a more generic framework (i.e. a technique that is not dataset specific) which can be used with any type of malware sample.

All the previously discussed methods use conventional machine learning techniques. Even though deep learning has already demonstrated its superiority due to its multilayer deep architecture, it has not been yet well-explored in the area of malware classification. Convolutional neural networks (CNN) are distinguished from some existing approaches in that they are able to learn the feature representation from the data itself. This approach of learning the feature representation from the data is a more intuitive way to address the challenge posed by the huge rise in the number of malware classes and degree of variability. Hardely et al. (2016) proposed a deep learning framework for malware detection (i.e. to say a file is a malware or not). Their method achieved very high detection accuracy (95.64%) which demonstrates the usefulness of deep architectures for malware analysis. However, in this paper we are interested in the more challenging problem of malware classification, which is essential to analyze the behavior of different malware families in order to deal with the ever-growing body of malware in the wild.

A recent method in (Gibert Llauradó, 2016) is also closely related to our proposed method. This work applies a CNN for malware classification. The author experimented with 3 different architectures by adding an extra block (a block consists of a convolutional layer followed by a Max-pooling layer) each time to its base model. However, their model is still very shallow in nature. In computer vision, researchers have shown that deeper architectures (i.e. network with many layers and huge number of parameters) increases the image classification performance (Simonyan & Zisserman, 2014). Another closely related work by (Raff et al., 2017) proposed a neural network for malware classification that operates on the raw bytes sequences. Different from them, we consider the approach of converting malware to images. In summary, this paper is an attempt to leverage the existing CNN-based state-of-the-art image classification techniques to solve malware classification.

BACKGROUND

A Convolution Neural Network (CNN) is a feed-forward neural network that is biologically inspired, in particular by the organization of animal visual cortex (Convolutional neural network, 2017). CNN is the current state-of-the-art neural network architecture for image classification problem. CNN is comprised of neurons with learnable weights and biases. CNNs mainly consist of the following three components (Intro to Convolutional Neural Networks, 2017):

1. *Convolutional layers*: These layers apply a certain number of convolution operations (linear filtering) to the image in sequence. Typically these filters extract edge, color, and shape information from the input image. Basically the filters operate on subregions of an image and perform computation such that it produces a single value as output for each subregion. The output (say x) of this layer is typically forwarded to a non-linear function (called Relu activation) which is defined as $f\left(x\right) = \max\left(0, x\right)$.
2. *Pooling layers*: This layer is responsible for downsampling (i.e. reducing the spatial resolution of the input layers) the data produced from convolution layers so that processing time can be

reduced, and computational resources can handle the scale of the data. It is the result of pooling operation that the number of learnable parameters is reduced in the subsequent layers of the network. Max pooling is a commonly used pooling technique that keeps the maximum value in a region (e.g. 2×2 non-overlapping regions of data) and discards the remaining values.

3. *Fully connected layers*: This layer performs classification on the output generated from convolution layers and pooling layers. Every neuron in this layer is connected to every neuron present in the previous layer. This type of layer is typically followed by a Dropout layer that improves the generalization capability of the model by preventing over-fitting which is commonly occurring problem in deep learning domain.

   Our proposed CNN is stack of multiple convolutional layers, pooling layers and fully connected layers. Given a malware sample as an input, the CNN predicts N scores (N = number of malware classes) where each score indicates how likely the malware sample belongs to a particular malware class. The class with highest score is the predicted or final class label for the given malware sample.

APPROACH

In this section, we discuss the problem definition and proposed solution methodology.

Problem Definition

The problem tackled in this paper can be divided in the following two parts: i) Given a dataset D of malware files, assign a class label *c* to each unlabeled malware file; ii) Given a part of malware file, assign class label *c* to it.

Although shallow learning methods, such as Support Vector Machines (SVM), Decision Trees (DT), and Artificial Neural Networks (ANN) can be used to handle the malware classification task, Convolutional Neural Networks (CNN) could help us achieve much superior performance on this task. In this paper, we explore CNN-based architecture for malware classification.

Visualizing Malware as Image

Malware authors usually change a portion of the previously available code to produce new malware (Nataraj, Karthikeyan, & Manjunath, 2015). If we represent malware as an image, then these small changes can be easily tracked. Inspired by this and previous work (Nataraj, Karthikeyan, Jacob, & Manjunath, 2011), we visualize malware binary files as grayscale images. Figure 2 demonstrates the process of converting malware binary files to grayscale image.

Firstly, a given malware binary file is read in a vector of 8-bits unsigned integers. Secondly, the binary value of each component of this vector is converted to its equivalent decimal value (e.g. the decimal value for [00000000] in binary is [0] and for [11111111] is [255]) which is then saved in a new decimal vector representing the malware sample. Finally, the resulting decimal vector is reshaped to a 2D matrix and visualized as a grayscale image. Selecting width and height of the 2D matrix (i.e. the spatial resolution of the image) mainly depends on the malware binary file size. We use the spatial resolution provided by Nataraj et al. (2011) while reshaping the decimal vectors. Figure 3 visualizes some examples of generated grayscale malware images for malware belonging to different families.

MODEL OVERVIEW

We use a Convolution Neural Network (CNN) for malware classification which we refer to as M-CNN throughout the remainder of the paper. Our model architecture is based on VGG-16 (Simonyan & Zisserman, 2014). Input to our network is a malware image and output is set of scores/confidences

Figure 2. Overview of malware visualization process. Firstly, the malware binary file is divided into 8-bit sequences which are then converted to equivalent decimal values. Finally, this decimal vector is reshaped and a grayscale image is generated that represents the malware sample.

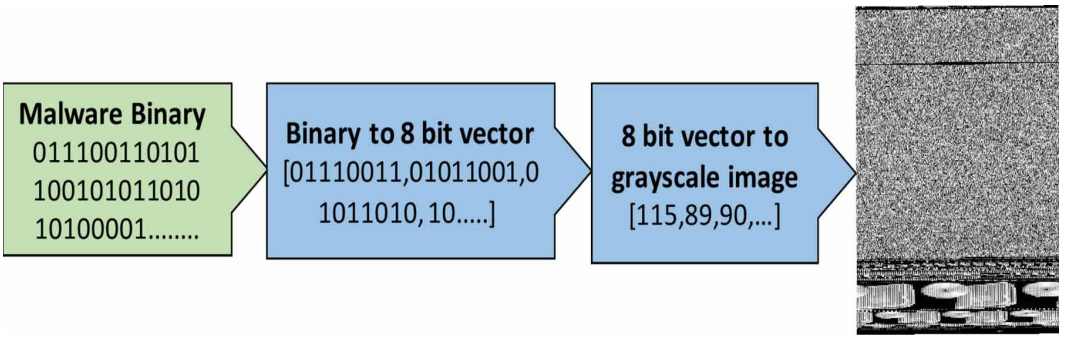
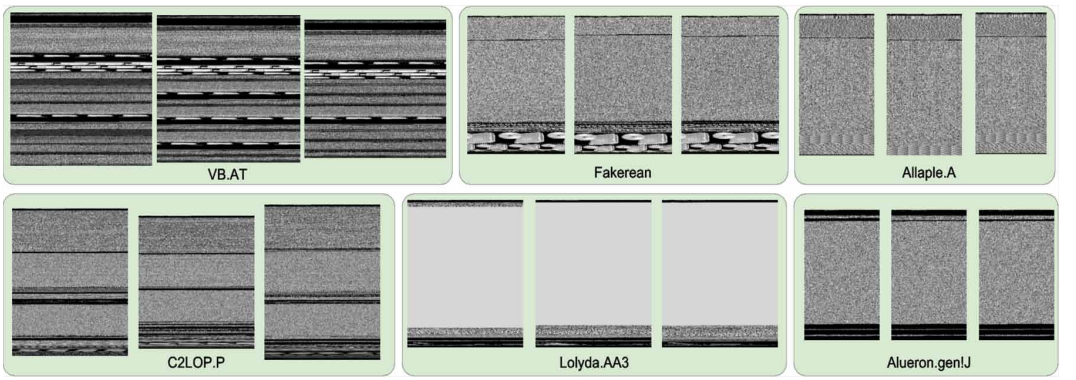


Figure 3. Examples of malware grayscale images belonging to different malware families. These images are acquired from Malimg Dataset. We can see that malware samples from same family are visually similar. Note that images shown above are rescaled for better visualization.



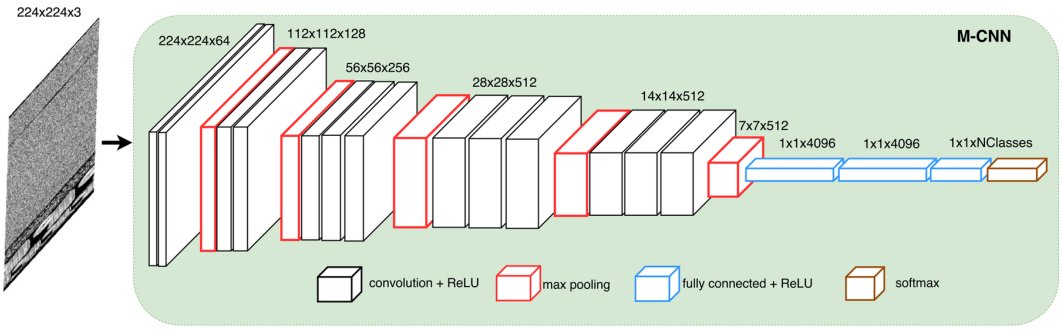
for various malware classes. We finally select the class with maximum score as our prediction for the given malware sample.

Following (Simonyan & Zisserman, 2014), our CNN architecture consists of 16 weight layers including 13 convolution layers and 3 fully connected layers. The convolutional layers have filters of size 3x3 and are divided into 5 groups where each group is followed by a 2x2 max-pooling layer that down samples the output. The number of filters in the first group of convolutional layers is 64 which increases by a multiple of 2 after each max-pooling layer, resulting in 512 filters in the last group of convolution layers. We then have 3 fully connected layers with dropout layers. The last fully connected layer produces output of a size that matches with the number of malware classes in the dataset. Figure 4 shows an overview of our proposed CNN architecture (M-CNN).

Learning

Since malware is labeled with a class (i.e. family) name, we employ a learning method that optimizes a classification loss. We use cross-entropy loss to train our network. The loss L for a training data y is defined as follows:

Figure 4. Overview of our proposed CNN architecture (M-CNN). The network takes a malware image as an input and produces a set of scores of equal size to the number of malware classes (NClasses) in the dataset as an output at the end. We pick the top-scoring class as the predicted class label for the given malware sample provided as input.



$$L = -\log \frac{\exp\left(f_{yi}\right)}{\sum_{j} \exp\left(f_{yi}\right)}$$

Where f_{yj} is the score for the j[th] class and f_{yi} is the score for the correct class of data. The parameters of the model are learned using stochastic gradient descent (SGD), which tries to minimize the loss incurred on the training data.

EXPERIMENTS

In this section, we discuss the malware datasets, experiments and the evaluation scheme.

Datasets and Experimental Setup

We conduct experiments on the following two challenging malware datasets:

1. **Malimg Dataset:** This dataset (Nataraj, Karthikeyan, Jacob, & Manjunath, 2011) has a total of 9,339 malware samples that are represented as grayscale images. Each malware sample in the dataset belongs to one of the 25 malware families. Also, the number of samples belonging to a malware family vary across the dataset. In our experiments, we randomly select 90% of malware samples in a family for training and the remaining 10% for testing. At the end, we have 8,394 malware samples for training and 945 samples for testing.
2. **Microsoft Malware Dataset:** In 2015, Microsoft hosted a Kaggle competition for malware classification (Microsoft, 2017). In this challenge, Microsoft released a huge dataset (almost half a terabyte when uncompressed) consisting of 21,741 malware samples. This dataset is divided in two parts, 10,868 samples for training and the other 10,873 samples for testing. Each malware sample belongs to one of 9 different malware families (i.e. Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator.ACY and Gatak). Like the Malimg dataset, the distribution of malware samples over classes in the training data is not uniform and the number of malware samples of some families significantly outnumbers the samples of other families. Each file has a class label where each label is represented by an integer from 1 to 9, where '1' represents the first malware family in the above list, and '9' the last one. There are two files that represent each malware sample, .bytes file that contains the raw hexadecimal representation of the file's binary content with the executable headers removed and .asm file that contains the

disassembled code extracted by the IDA disassembler tool. In our experiments, we only use the .bytes files to generate the malware grayscale images. We experiment with two different setups involving this dataset: 1) Setup-A: following previous work (Drew, Moore, & Hahsler, 2016) (Drew, Hahsler, & Moore, 2017), the original training set (10,868 malware samples) is class-wise randomly divided into two subsets where the first set consists of 90% malware files (9,776) and the rest 10% (1092) is used for testing; 2) Setup-B: we follow the original train-test split which is provided by Microsoft (i.e. 10,868 samples for training and the other 10,873 samples for testing).

All the experiments are conducted on 64-bit Ubuntu 14.04 Intel Core i7-5820K CPU (3.30GHz) with 64GB RAM and a NVIDIA Titan X GPU with 12GB memory. We implement our framework using the Torch machine learning library (Collobert, Kavukcuoglu, & Farabet, 2011). We set the initial learning rate to be 0.001 which is reduced by a factor of 10 every 20 epochs where 1 epoch means exposing a learning algorithm to the entire set of training data once. We set weight decay and momentum to 0.0005 and 0.9, respectively. We initialize the parameters in M-CNN network with the VGG-16 (Simonyan & Zisserman, 2014) pre-trained weights. We also randomly shuffle the training data in every epoch.

EXPERIMENTS AND EVALUATION

Our experiments are divided into two categories: 1) when we have access to whole malware image; 2) when only part of the malware image is available. The goal of the first set of experiments is to determine classification performance when whole malware data is fed to the model, whereas the second set of experiments is conducted to determine whether or not our proposed network can identify the class of a malware sample by seeing only a portion of it. Note that we convert all the malware binary files in the datasets to grayscale images using the method described in Section APPROACH. For the Microsoft malware dataset, we only used the .bytes files to generate the grayscale malware images. To evaluate the performance of the proposed model, we report the performance of different methods in terms of accuracy which simply refers to the percentage of malware samples that are labeled correctly.

1. Classification with whole malware image

We evaluate our proposed network when the input is the whole malware image. We also compare the performance with several baseline methods.

Apart from comparing with some previous works, we also implement our own baseline. We extract GIST (Oliva & Torralba, 2001) (Torralba, Murphy, Freeman, Rubin, & others, 2003) features (a handcrafted feature that computes texture features) from each malware image. Next, we train a multi-class Support Vector Machine (SVM) classifier on both of the datasets. We call this baseline GIST+SVM, which achieves accuracy of 93.52% on the Malimg datasets and 88.74% on the *Setup-A* of Microsoft malware dataset. These results are shown in Table 1.

Our proposed network M-CNN is trained for 25 epochs with a batch size (a set of training data that is forwarded to the model at once) of 6 for the Malimg dataset and a batch size of 8 when training on Microsoft malware dataset. M-CNN achieved the best performance with a classification accuracy of 98.52% on the testing set of Malimg dataset and 98.99% on the Setup-A of Microsoft malware dataset. Our method outperforms several baseline methods by a huge margin on both of the datasets as can be seen in Table 1.

In addition, when evaluated on the training set of Microsoft malware dataset with Setup-B, our M-CNN model achieves very high classification accuracy of 99.97% where only 3 samples where

Table 1. Classification performance on whole malware images

Malimg Dataset		**Microsoft Dataset: Setup-A**		**Microsoft Dataset: Setup-B**	
Method	*Accuracy*	*Method*	*Accuracy*	*Method*	*Accuracy*
Nataraj et al. (2011)	97.18%	Drew et al. (2016)	97.42%	Gibert (2016)	99.76%
GIST+SVM (ours)	93.23%	Drew et al. (2017)	98.59%	Ahmadi et al. (2016)	99.77%
M-CNN (ours)	**98.52%**	GIST+SVM (ours)	88.74%	Winner (Wang, Liu, & Chen, 2017)	99.87%
		M-CNN (ours)	**98.99%**	M-CNN (ours)	**99.97%**

misclassified. This performance is 0.1% higher than the accuracy of the Microsoft Kaggle challenge winner's solution which had 15 misclassified malware samples. See Table 1 for details.

Figure 5 shows the behavior of the network on the two datasets during training and testing. We compute the performance of the network at various epoch on both training and testing set. The plots suggest that the network performance improves with the increase in the number of training epochs. Figure 6 and Figure 7 show the confusion matrix for Microsoft dataset with Setup-B and Malimg, respectively.

Lastly, we evaluate M-CNN on the test set of Microsoft malware dataset, which is same as the test set of *Setup-B*. We submitted our predictions to the evaluation server which evaluates the submissions based on multi-class logarithmic loss

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log\left(p_{i,j}\right)$$

where N is the number of malware samples, m is the number of malware classes, $y_{i,j}$ is 1 if the prediction is correct and 0 otherwise, and $p_{i,j}$ is the predicted probability (Microsoft, 2017). Table 2 shows the performance of different methods in terms of logloss. note that our performance is not

Figure 5. Accuracy on different datasets at different epochs. The curve in blue denotes test accuracy, whereas the curve in red denotes training accuracy. Left: M-CNN performance on the Malimg dataset. Right: M-CNN performance on the Microsoft dataset with Setup-A.

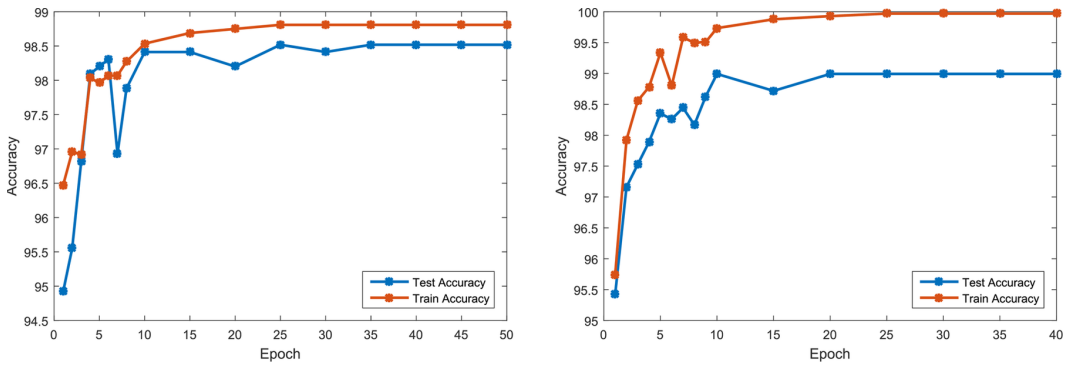
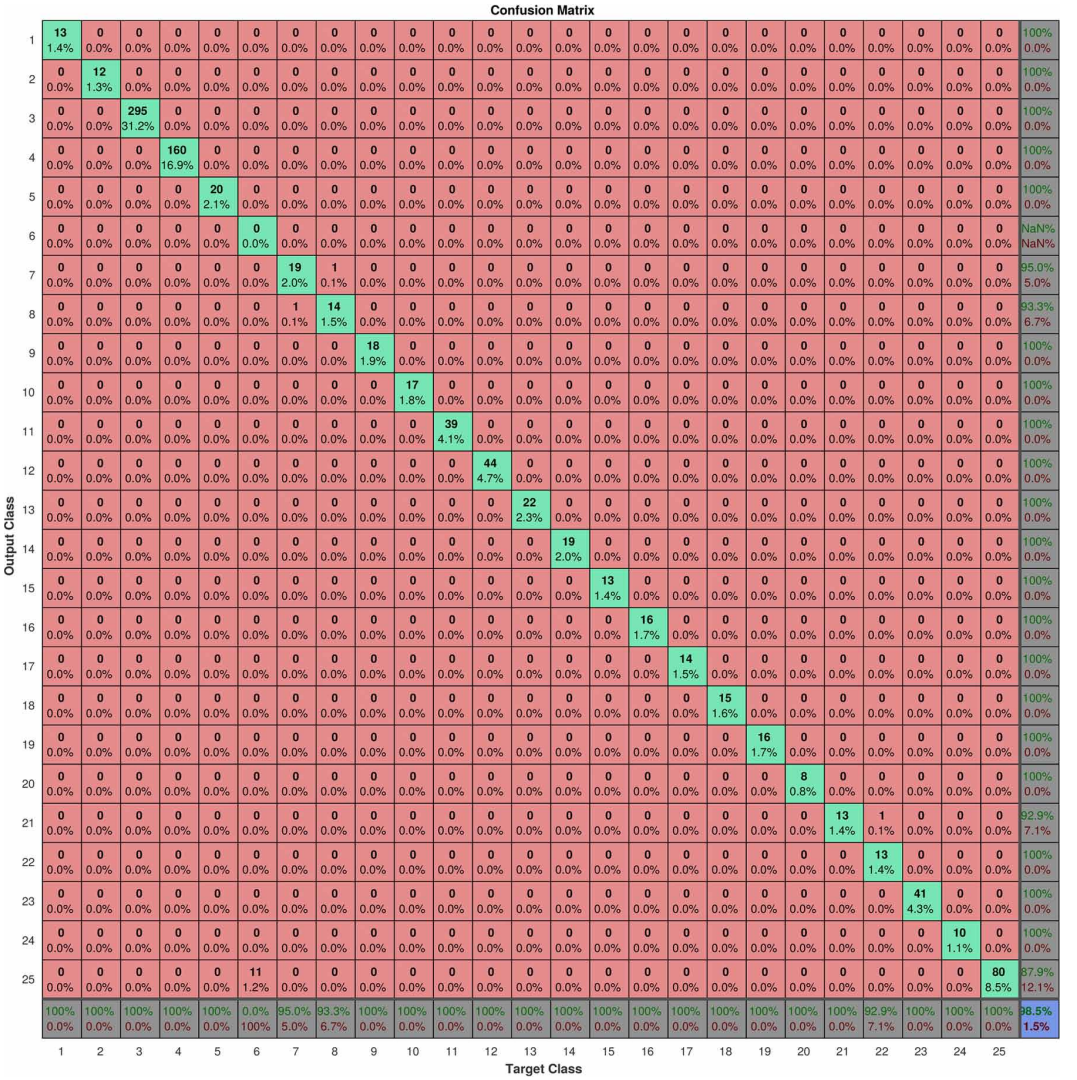


Figure 6. Confusion matrix of our M-CNN network when tested on the Microsoft dataset with Setup-B.

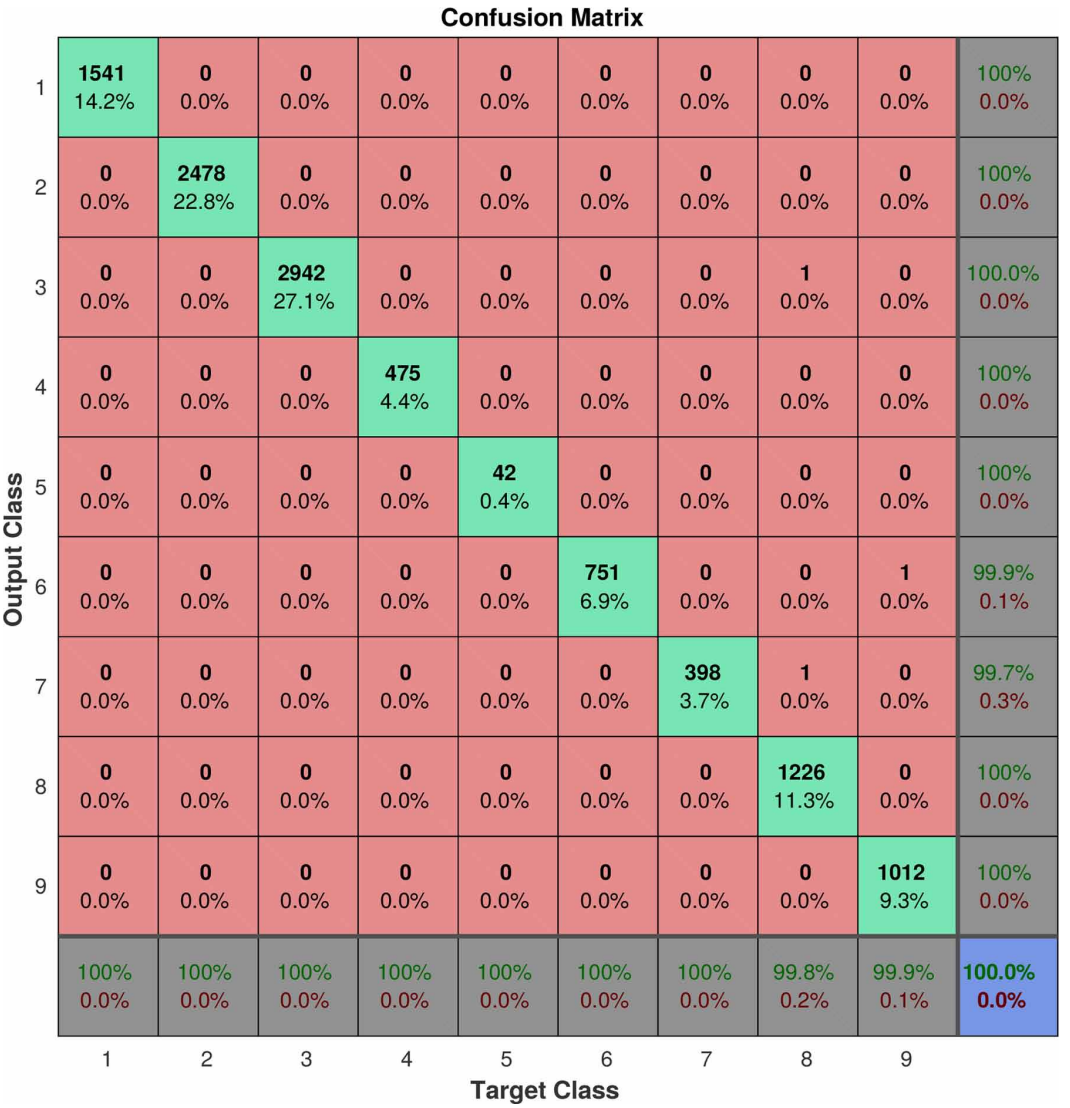


directly comparable with the best method as they use both .bytes and .asm files in their method, whereas we only use .bytes files for training our model. we achieve the best performance when compared with methods that only use .bytes files.

2. Classification with part of the malware image

We perform another set of experiments to examine the performance of M-CNN when only part of the malware image is given as input. We crop each malware image at a random position to generate several cropped versions of the same malware image. The cropping size is randomly selected in such a way that the final cropped image represents 50% or more of the original malware image. Figure 8 shows an example of a malware image and its cropped versions as a result of random cropping at different positions. During the training process, all of the cropped versions of a malware image are resized to the original malware image size (i.e. 224 × 224) and fed to the network.

Figure 7. Confusion matrix of our M-CNN network when tested on the Malimg dataset. Note that the number along the diagonal represents number of samples correctly classified in a particular class, whereas the % represents the percentage of malware samples belonging to that class in the dataset. The last column shows the class-wise accuracy achieved by our model.



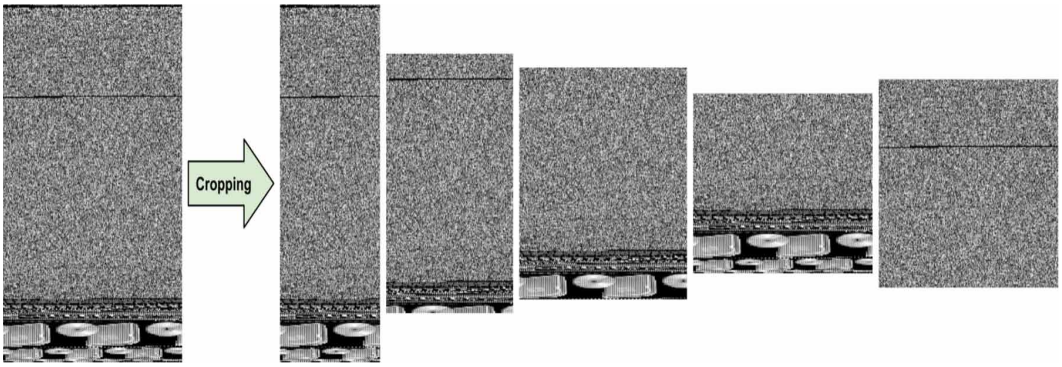
We modify our training procedure so that our M-CNN model can correctly classify cropped malware images. We call this modified model M-CNN-C. We firstly train M-CNN-C model for 5 epochs on the whole malware images and then when the model is partially trained, we start training with randomly cropped malware images. As mentioned above, these cropped images are rescaled to the original malware image size before being forwarded to the model. From epoch 6 onwards, we train network with 25% randomly selected original training malware images (i.e. these are not cropped) and 75% randomly cropped and resized images. We train our M-CNN-C for 40 epochs on the Malimg dataset and for 42 epochs on the Microsoft malware dataset.

Table 3 shows the performance of our M-CNN and M-CNN-C in different cropping settings. We experiment with 3 different image configurations: 1) No Cropping: where both the models are

Table 2. Performance of different methods on original test

Method	Files used	Logloss on test
Gibert (2016)	.bytes	0.1176
Drew et al. (2016)	.bytes	0.2228
Drew et al. (2017)	.bytes + .asm	0.0479
Ahmadi et al. (2016)	.bytes + .asm	0.0063
Winner (Wang, Liu, & Chen, 2017)	.bytes + .asm	0.0028
M-CNN (ours)	.bytes	0.0571

Figure 8. Examples of random cropping on a malware image



evaluated on original malware images; 2) Random Cropping: where models are tested on malware images with random cropping sizes; and 3) 50% Cropping: where models are evaluated on malware images with strictly 50% of random cropping. M-CNN trained on whole malware image performs poorly on both the datasets when tested on randomly cropped image (2nd last row in the table). M-CNN performance further drops when tested with half-cropped malware images (last row in the table). M-CNN-C improves the classification accuracy by more than 25% compared with M-CNN when tested with cropped malware images from Malimg dataset. M-CNN-C also performs significantly well compared to M-CNN on Microsoft malware dataset with Setup-A.

We believe the performance gain by M-CNN-C over M-CNN on cropped malware images can be attributed to two factors. First, the cropping procedure creates variants of malware in the same class which helps the model learn a more discriminative representation of malware originating from

Table 3. Quantitative results for classification accuracy of various methods tested with cropped malware images. M-CNN refers to our base network which is trained with original malware images, whereas M-CNN-C refers to M-CNN trained with cropped malware images.

Image configurations	Malimg Dataset		Microsoft Dataset Setup-A	
	M-CNN	M-CNN-C	M-CNN	M-CNN-C
No Cropping	98.52%	98.41%	98.99%	99.08%
Random Cropping	72.80%	97.35%	78.57%	96.34%
50% Cropping	32.17%	94.39%	50.82%	86.36%

different classes. Second, parallel training with whole original images is helpful and important because it enables the model to keep track of the original overall structure of the malware family.

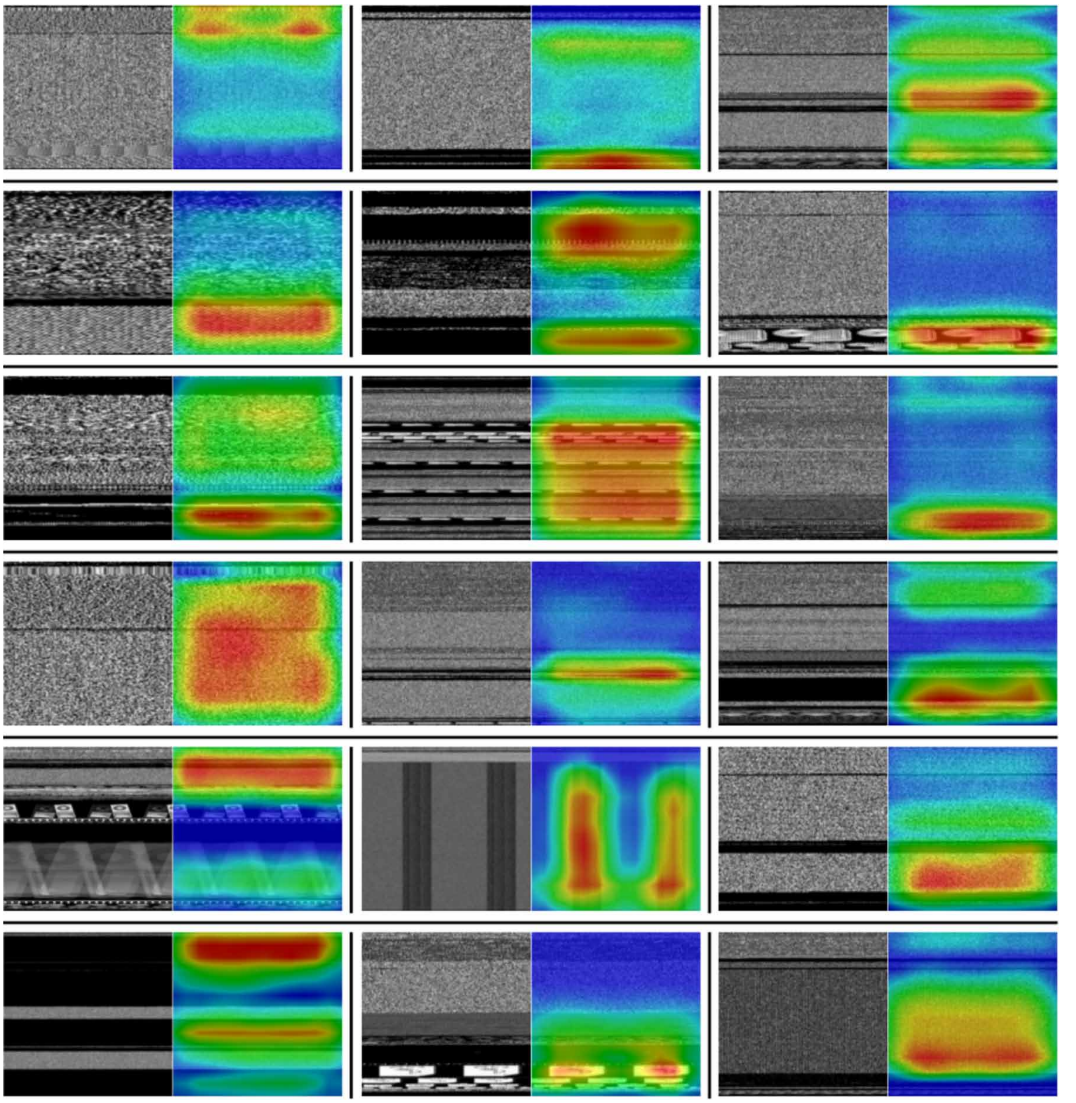
Furthermore, we perform visual analysis of the M-CNN model to show the most important regions of malware images that the model is using to correctly classify that image. Inspired by (Selvaraju et al., 2016), we show in Figure 9 the class activation maps for some malware images from both the datasets. These class activation maps highlight the subregions of the malware image where the model focuses in order to predict its class. For example, when the model is given a malware sample that has vertical dark stripes (5th row and 3rd column in the figure), it looks at the distinctive stripes it contains in order to predict its class. For some of the class activation map examples (such as in 4th row and 2nd column), we see that the model is focusing on large area of the malware image in order to classify it. In this case, since the model could not find any finer discriminative regions like the previous example, it looks at the broader region to determine the class of the malware sample. In summary, this class activation maps indicate the different distinctive regions where the CNN looks in performing the classification, and also shows that in some instances these are highly localized while in others they are more diffusely located requiring broader analysis of the file contents.

ADVANTAGES

There are multiple advantages of the proposed framework over previous methods which are as follows:

1. This is the first attempt in the literature concerning malware classification to perform classification even if only a part of the malware sample is given. One very interesting application of this can be seen in network security. When sharing a file through any network, the receiver does not typically receive the file at once, instead it is transmitted in parts. Thus, using our framework, we can detect and classify whether or not the file contains malware even before having it completely transmitted to the system. If malware is found then we can immediately stop the file transfer and prevent the attack.
2. Even though our model is trained with only half of the available training data (we use only the .bytes files and do not use the .asm files for the Microsoft malware dataset (see the Experiments section for more details), it achieves superior performance which demonstrates that our proposed model is a quick learner, i.e., our proposed model can learn discriminative features of malware families with less information as compared to other approaches that use both files in their methods.
3. Our framework is a generic in nature. Existing methods are highly specialized and tailored specifically for one system or task. For example, most of the other methods based on the Microsoft malware dataset extracted features manually using all of the available files which may not be generalize to other malware datasets, and their extraction methods were also focused on the given dataset. However, our framework is general and can operate on any malware file as long as the binary representation of that file can be acquired, which is usually the case irrespective of operating system.
4. The generated images significantly reduce the space needed for storing large datasets of malware. For example, the original Microsoft malware dataset .bytes files have a total size of almost 51 GB, whereas the corresponding grayscale images use only 615 MB of disk space.
5. Our model is very fast in predicting a class label for a given malware sample. The usual time required to classify a new malware sample is significantly less than most existing methods. Only a fraction of a second is needed to forward a new malware image to the model and obtain the predicted class label.

Figure 9. Class activation maps from M-CNN. The final class score is mapped back to the previous layers to generate the class activation maps. These activation maps highlight the malware class-specific discriminative image regions.



CONCLUSION

Malware is increasingly posing a serious security threat to computer systems. It is essential to analyze the behavior of malware and categorize samples so that robust programs to prevent malware attacks can be developed. Towards this endeavor, we have proposed a deep convolutional neural network (CNN) architecture for malware classification. We first convert malware samples to grayscale images in order to train a CNN for classification. Experimental results on two benchmark malware classification datasets demonstrate that our proposed model performs better than state-of-the-art methods. We trained our CNN such that it can correctly classify samples even if only portion of a malware sample is provided, which we believe we are the first to do in the body of work addressing malware detection. Through extensive experiments on two malware datasets, we showed the effectiveness of proposed CNN for malware classification.

In future work, we plan to incorporate Spatial Transformer Networks (STN) (Jaderberg, Simonyan, Zisserman, & others, 2015) in the proposed neural architecture. The motivation behind this is that it may not be necessary that whole malware images carry discriminative information that can help in classification, i.e., some region in a malware sample can help in distinguishing their family. STN is a tool that can enable capture of such discriminative information which will result in further improvements to classification accuracy, including handling more extensive changes to existing malware categories.

REFERENCES

Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., & Giacinto, G. (2016). Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy* (pp. 183-194). doi:10.1145/2857705.2857713

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. E. (2014). *DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket*. NDSS.

Bayer, U., Comparetti, P. M., Hlauschek, C., Kruegel, C., & Kirda, E. (2009). *Scalable, Behavior-Based Malware Clustering*. NDSS.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794). ACM. doi:10.1145/2939672.2939785

Collobert, R., Kavukcuoglu, K., & Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In *Proceedings of the BigLearn, NIPS Workshop.* Academic Press.

Convolutional neural network. (2017).

Drew, J., Hahsler, M., & Moore, T. (2017). Polymorphic malware detection using sequence classification methods and ensembles. *EURASIP Journal on Information Security*, (1), 2. doi:10.1186/s13635-017-0055-6

Drew, J., Moore, T., & Hahsler, M. (2016). *Polymorphic Malware Detection Using Sequence Classification Methods* (pp. 81–87). Security and Privacy Workshops. doi:10.1109/SPW.2016.30

Gibert Llauradó, D. (2016). *Convolutional neural networks for malware classification*. Universitat Politècnica de Catalunya.

Hardy, W., Chen, L., Hou, S., Ye, Y., & Li, X. (2016). DL4MD: A Deep Learning Framework for Intelligent Malware Detection. In *Proceedings of the International Conference on Data Mining (DMIN).*

Huang, W., & Stokes, J. W. (2016). MtNet: a multi-task neural network for dynamic malware classification. In Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 399-418). Springer. doi:10.1007/978-3-319-40667-1_20

Idika, N., & Mathur, A. P. (2007). A survey of malware detection techniques. Purdue University.

Internet Security Threat Report. (2017, April). Symantec. Retrieved from https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf

Intro to Convolutional Neural Networks. (2017). Retrieved from https://www.tensorflow.org/tutorials/layers

Jaderberg, M., Simonyan, K., & Zisserman, A. et al. (2015). Spatial transformer networks. *Advances in Neural Information Processing Systems*, 2017–2025.

Kalash, M., Rochan, M., Mohammed, N., Bruce, N., Wang, Y., & Iqbal, F. (2018). Malware Classification with Deep Convolutional Neural Networks. In *Proceedings of the IFIP NTMS International Workshop on Cybercrime Investigation and Digital forensics (CID2018).* Academic Press. doi:10.1109/NTMS.2018.8328749

Kaur, G., & Nagpal, B. (2012). Malware analysis & its application to digital forensic. *International Journal on Computer Science and Engineering*.

Kolter, J. Z., & Maloof, M. A. (2004). Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 470-478). Academic Press. doi:10.1145/1014052.1014105

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 1097–1105.

Maiorca, D., Ariu, D., Corona, I., Aresu, M., & Giacinto, G. (2015). Stealth attacks: An extended insight into the obfuscation effects on android malware. *Computers & Security, 51*, 16-31.

Malware Definition. (2017). Retrieved from https://techterms.com/definition/malware

Microsoft. (2017). *Microsoft Malware Classification Challenge (BIG 2015)*. Retrieved from https://www.kaggle.com/c/malware-classification

Microsoft, M. C. C. (2015). (n.d.). *BIG*.

Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011). Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security* (p. 4). Academic Press.

Nataraj, L., Karthikeyan, S., & Manjunath, B. S. (2015). SATTVA: SpArsiTy inspired classificaTion of malware VAriants. In *Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security* (pp. 135-140). ACM. doi:10.1145/2756601.2756616

Oliva, A., & Torralba, A. (2001). Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, *42*(3), 145–175. doi:10.1023/A:1011139631724

Park, Y., Reeves, D., Mulukutla, V., & Sundaravel, B. (2010). Fast malware classification by automated behavioral graph matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, (p. 45). doi:10.1145/1852666.1852716

Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., & Nicholas, C. (2017). Malware Detection by Eating a Whole EXE.

Rieck, K., Holz, T., Willems, C., Düssel, P., & Laskov, P. (2008). Learning and classification of malware behavior. In *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 108-125). Springer. doi:10.1007/978-3-540-70542-0_6

Schultz, M. G., Eskin, E., Zadok, F., & Stolfo, S. J. (2001). Data mining methods for detection of new malicious executables. In *Proceedings of the IEEE Symposium on Security and Privacy* (pp. 38-49). IEEE. doi:10.1109/SECPRI.2001.924286

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2016). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization.

Siddiqui, M., Wang, M. C., & Lee, J. (2008). A survey of data mining techniques for malware detection using file features. In *Proceedings of the 46th Annual Southeast Regional Conference on XX* (pp. 509-510). Academic Press. doi:10.1145/1593105.1593239

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

Sung, A. H., Xu, J., Chavez, P., & Mukkamala, S. (2004). Static analyzer of vicious executables (save). In *Proceedings of the Annual Computer Security Applications Conference* (pp. 326-334). Academic Press. doi:10.1109/CSAC.2004.37

Torralba, A., Murphy, K. P., Freeman, W. T., & Rubin, M. A. et al. (2003). Context-based vision system for place and object recognition. *International Conference on Computer Vision,* 3, pp. 273-280. doi:10.1109/ICCV.2003.1238354

Total Malware. (2017). AV Test. Retrieved from https://www.av-test.org/en/statistics/malware/

Wang, X., Liu, J., & Chen, X. (2017). *Microsoft Malware Winners' Interview: 1st place, "NO to overfitting!"* Retrieved from https://github.com/xiaozhouwang/kaggle_Microsoft_Malware/blob/master/Saynotooverfitting.pdf

You, I., & Yim, K. (2010). Malware obfuscation techniques: A brief survey. In *Proceedings of the International Conference on Broadband, Wireless Computing, Communication and Applications* (pp. 297-300). Academic Press.

Mahmoud Kalash received an M.Sc. in Computer Science from University of Manitoba and a B.Eng. from Damascus University. His research interests lie in computer vision and machine learning.

Mrigank Rochan is currently a Ph.D. student in Computer Science at the University of Manitoba, Canada. Previously, he received an M.Sc. degree in Computer Science from the University of Manitoba and a B.Tech. degree in Computer Science and Engineering from Amrita Vishwa Vidyapeetham University, India. His research interests include computer vision and machine learning.

Noman Mohammed received a Ph.D. degree in Computer Science from Concordia University in 2012. He is currently an Assistant Professor in the Department of Computer Science at University of Manitoba, Manitoba, Canada. Before coming to UofM, he was an NSERC postdoctoral fellow in the School of Computer Science at McGill University and a member of the Cryptography, Security, & Privacy (CrySP) Research Group at the University of Waterloo. His research interests include private data sharing, privacy-preserving data mining, secure distributed systems, and applied cryptography.

Neil D.B. Bruce is currently an Assistant Professor in the Department of Computer Science at Ryerson University, Canada and an Associate Professor in Computer Science at the University of Manitoba. He has been a postdoctoral fellow at the Centre for Vision Research at York University, and at INRIA Sophia Antipolis in France. He holds a Ph.D. in Computer Science (York University, 2008), M.A.Sc in System Design Engineering (University of Waterloo, 2003), and an Honours B.Sc. with Majors in Computer Science and Mathematics (University of Guelph, 2001).

Yang Wang is currently an Associate Professor in the Department of Computer Science, University of Mantioba. He received his PhD from Simon Fraser University, MSc from University of Alberta, and BSc from Harbin Institute of Technology. He was an NSERC postdoc fellow at the Department of Computer Science, University of Illinois at Urbana-Champaign. His research interests lie in computer vision and machine learning.

Farkhund Iqbal is an Associate Professor and Director Advanced Cyber Forensics Research Laboratory in the College of Technological Innovation, Zayed University, United Arab Emirates. He holds a Master (2005) and a Ph.D. degree (2011) from Concordia University, Canada. He is using machine learning and Big Data techniques for problem solving in healthcare, cybersecurity and cybercrime investigation in smart and safe city domain. He has published more than 80 papers in high ranked journals and conferences. He is an affiliate professor in school of information studies, McGill university, Canada and Adjunct professor in Adjunct Professor, Faculty of Business and IT, University of Ontario Institute of Technology, Canada. He is the recipient of several prestigious awards and research grants. He has served as a chair and TPC member of several IEEE/ACM conferences and is the reviewer and guest editor of high rank journals.