View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Queen Mary Research Online

Malware Detection Using 1-Dimensional Convolutional Neural Networks

Arindam Sharma
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, UK
a.sharma@se15.qmul.ac.uk

Pasquale Malacaria
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, UK
p.malacaria@qmul.ac.uk

MHR Khouzani
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, UK
arman.khouzani@qmul.ac.uk

***Abstract*—This work introduces a highly accurate and efficient malware detection system based on 1-dimensional convolutional neural networks. The system takes as input a binary file and classifies it as malicious or benign. There is minimal pre-processing of the binaries, with features discovery left to the network during training. A crucial difference with other convolutional neural networks (CNN) based approaches is the use of 1-dimensional convolutions; this methodological choice is shown to have significant positive consequences for the detector. In order to compare the detector with state-of-the-art techniques a TF-IDF based benchmark malware detector is also implemented: experiments show an improved accuracy of the proposed CNN detector while maintaining similar training times. The system is also compared, on a publicly available dataset of 11130 binaries, with an existing embedding based CNN detector. The proposed system outperforms, both in accuracy and training time the embedding based CNN.**

I. Introduction

Malware, or malicious software, is one of the primary threats in digital security. Large resources, both labour and monetary, are invested in anti-malware technologies to tackle this problem [1]. As the efficiency of the anti-malware technologies has improved, so has the sophistication, potency and domain of malware and their impact.

The complex nature of the digital ecosystem leaves many domains for intrusive attacks. These domains are exploited by malicious entities in ways that constantly change. Malware have been particularly effective for intrusive attacks primarily due to their ever evolving nature. This evolutionary nature of modern-day malware is the key limitation of the popular signature-based anti-malware technologies [18], [23]. Moreover, the multitude of available and new software in this ecosystem makes it imperative for anti-malware technologies to have high accuracy, as there is a large indirect cost for even a small false positive rate. The need for high accuracy compounded by the evanescent nature of malware make malware detection a challenging practical problem to solve.

The field of machine learning has made invaluable contributions in almost all major fields of research [9] with its strongest impact on data-driven areas. Considering the nature of the problem, this work aims at utilizing the effectiveness of modern machine learning algorithms as a more robust tool for detection of malware. An attractive property of machine learning techniques is their "generalization", i.e., their ability to effectively tackle a wide range of input even those that they have not seen or trained on. As long as the premise holds that the new malware are in some feature space and with some notion of distance "close" to the previous malware, machine learning tools can effectively generalize to new malware.

While there have been attempts at using these technologies for the analysis of malware, it still remains a domain with a lot of scope for investigation. With the development of new architectures and methodologies, the field has proven to be extremely versatile in terms of applications.

a) Contributions: This work presents a novel, deep learning based detection system, which classifies binaries as 'malicious' or 'benign' in a static manner. The detection model proposed is based on the notion of 1-dimensional convolutional neural network i.e. neural networks with single dimension input and single dimension filters being used for the convolutional layers.

This paper argues 1-dimensional convolutional neural network are a conceptually meaningful choice, as 2-dimensional CNNs have problematic semantic interpretations.

Moreover such choice simplifies the model resulting in a computationally lightweight implementation, taking around 20 minutes to complete training over a publicly available dataset of 11130 binaries, with a resulting accuracy of 99.2%. The detector is compared with the (only) CNN detector for which results on the above dataset are available: the proposed system improves on that detector both in accuracy and precision; moreover there is a (minimum) five fold improvement in the training time.

II. Overview of Malware Detection

This section summarizes the major techniques that are being used for detecting malicious pieces of software. Figure 1 shows a high-level view of the relation between the different approaches.

a) Signature based techniques: The most popular technique that is employed to the task of malware detection is based on static, signature matching ([6]). It, and its variants,

This research was supported by EPSRC grant EP/R004897/1

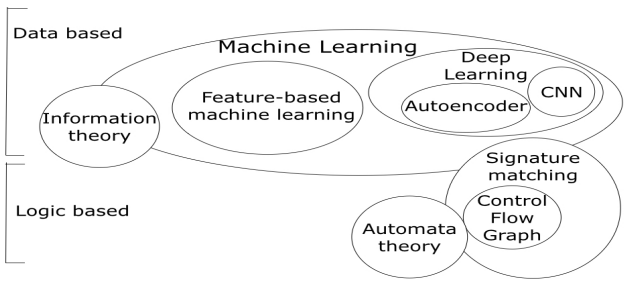

Fig. 1. Overview of techniques for malware detection

serve as the basis of the most popular anti-malware technologies in the market. This methodology follows the principle of "code hoisting" and tries to provide a "confidence-centric" measure of the files malicious potency.

In [14], the file in question is analyzed by the tool, which tries to run the file through a set of known malicious code patterns. The amount of matches it gets is equivalent to the amount of confidence the tool has on the file being of malicious nature. The signatures used by such tools vary from the simple code sequences to more sophisticated regular expression based code samples [2].

b) Feature based techniques: A commonly used technique is based on the idea of extracting distinguishing features from files in question, which could then be used for detecting malicious files.

The features extracted range from something as trivial as the file size to a much more sophisticated feature based on n-grams of files [21]. In [8], the authors worked on extracting n-grams from sample binaries, which were then used as features for training models for the task detecting malwares. The authors from [19] aimed at utilizing the benefits of the n-gram based approaches while trying to minimize the dimensionality of the feature input space by evaluating the relative importance of the various extracted n-grams. [11] used a similar strategy and extract n-grams of operation codes from binaries, which are then used for detection.

The technique from [25] is based on the idea document classification techniques like TF-IDF for feature extraction. These features are then used on various classifiers for detecting malwares.

There have been techniques in the literature that utilize compression-based distance metrics to analyze malicious pieces of code. The work from [1] is based on the notion of normalized information distance (NID) and employ this as a distinguishing feature for training the classifier on.

Such techniques, which usually make use of a set of features, rely on the fact that the classifier should be able to generalize well based on the most distinguishing subset of these features.

c) Other techniques: A popular class of techniques is based on the notion of functional similarity between pieces of code. These techniques are based on the idea of a distance-like metric to quantify the amount of functional similarity between two files.

These include control flow graph based approaches ([2]), along with works like [4] which use such graphs for computing distance measures.

Another approach is based on model checking. In [14] model checking is used for specifying the behaviour of programs and for detecting malicious chunks of code.

There have also been works which involve use of convolutional neural networks (CNNs) [12] and variational autoencoders (VAEs) [26] for the purpose of analyzing and detecting malware.

These methodologies are based on the idea of finding the most relevant features, based on the structure of the file, for detecting malicious behaviour. The robustness of these techniques comes form the ability of deep neural networks to generalize on new, unseen data. Such techniques are gaining popularity due to their ability to utilize the large amounts of data available.

A. Related Works

The successful use of CNNs for something significantly different from image processing and computer vision was demonstrated in [13], which uses it for the purpose of sentence classification in natural language processing. That work influenced a number of methodological choices made in this work.

This paper differs from [13] in multiple ways, one of which is the use of single dimensional filters for convolutional layers, but the overlying concept presented in the work is closely related to it.

Works like [17], [7] and [28] make use of deep learning techniques for detecting malwares. In terms of the approach taken, our work is the most similar to the work of [10], [17] and [20], which in turn, follow similar approaches amongst themselves. The idea being utilized in the works from [10] and [20] is based on principle of a specific variation of convolutional neural networks, where the filters (or feature detectors), are two dimensional convolution which are allowed to move in only one direction (either across x-direction or across the y-direction). While [20] was focused on the input size and computability of such single-dimensional-movement-filter convolutional neural networks, the authors from [10] tried to improve the aforementioned paper's approach, while still using the notion of an 'embedding' space and convolutional neural networks. Our work makes use of convolutional neural network, where the input is one dimensional (and consequently the filters are one dimensional). As discussed in the later sections of this work, our methodology improves upon the performance from [10], while keeping it computationally efficient, the major point of contention for [20].

The authors from [15] use sequences of system calls for malware detection. Extracting system calls associated with

binaries requires the binary to be executed in a controlled environment, which adds to the overhead in terms of computations involved. The model presented here works statically and directly on the disassembled, sample binaries, thereby eliminating the need for extracting system calls from sample binaries.

Malware authors who are cognizant of the functioning of anti-malware signature based techniques are able to circumvent through most checks with relative ease ([18]). One of the primary reasons for this seems to be the stringent signatures that serve as the basis for these techniques, as being used in [14]. This is primarily due to the fact that these techniques do not employ statistical measures and have strict pattern matching rules used in the malware detection process. Our approach is based on statistical learning models, such that the features learned are not as rigid as the ones being used by traditional signature based techniques. Also, the work of [14] relies on the file being represented in an appropriate manner, which adds to the computational overhead of the task. Our model involves minimal pre-processing, and is structured to work on the existent representation of the sample binaries.

In this paper, we compare our deep learning based detector's functionality with a benchmark detector that we implemented, which is closely related, in terms of the principles used, to the works like [25], [11] and [27]. In the work of [24], the authors work with multiple extracted features form binaries, some of which revolve around the notion of capturing structure-related attributes of binaries by collecting n-gram based statistics. Consequently, due to the large number of detection tools built around this principle, this work uses its own detection tool, which is inspired from the aforementioned works from the literature. Also, owing to the lack of public availability of datasets being used in majority of the works presented as observed by the authors for [20], this paper's benchmark tool tries to provide a detector, as explained in Section IV-A, based on the classical techniques, which can be used for assessing the performance of our proposed architecture in a reasonable manner.

Apart from the popular classical techniques, this work also addresses some of the issues with the other methodologies used for malware detection. The graph based techniques, as used in [3], [2] and [4] are primarily aimed at developing more robust malware 'signatures'. Their heavy reliance on graphical representation of code usually do not scale well with files which are large in size. Also, the similarity measures, which happen to be graph based, are computationally expensive to compute, thereby reducing the practicality of such techniques. In this work, we have demonstrated the training and evaluation performance of our model, by using a dataset which has files of varied lengths.

For this work, we tried the implementation of another benchmark detector based on the information theoretic approach for malware detection, similar to the ones used by [1]. We found the computational limitation of such an approach during the attempted implementation. The pre-processing stage was computationally expensive and could not match the practicality of the other approaches.

The deep neural networks based techniques are able to use the large malware samples available and aim to automatically extract relevant features for detecting malware. Given that these neural networks receive the raw file as input, the features that they are able to extract are constrained to be structural in nature. Consequently, these techniques suffer from the common drawbacks associated with deep learning approaches. The techniques that use CNNs [12] for analysing suspected files tend to represent the raw binary as an image, which may not be an accurate representation due to the limitations of representing a binary stream of data as a two dimensional image. The use of more modern techniques, such as the one from [26], need a lot of data for training and the training of VAEs [1] remains a practical challenge.

III. Proposed Methodology for CNN

This section outlines the main components of the proposed approach. We start by discussing the problem with 2 dimensional representations of binaries.

A. Instruction Embedding: the distributional hypothesis

In order for traditional deep learning based CNN approaches to be used for the task of malware detection, the binaries are usually represented in a two dimensional way. This would allow to leverage the benefits of CNN, including the most relevant one which accounts for translational invariance [2]. There are two basic methodologies for representing binaries in a 2 dimensional way: The first treats raw binaries as images: this approach loses the structure of the code, and more worryingly conflates vertical and horizontal proximity in the 2-d space (one meaningful, the other meaningless): for these reasons it seems a poor methodological choice. The second approach, more sophisticated, uses the notion of 'embedding space', a concept borrowed from the NLP community and used in malware detection in [20], [28], [17] and [10]. At a high level, the essence of the embedding space approach is to transform the n length binaries in a $n \times m$ matrix where each row is a vector associated with an instruction. The idea of projecting instructions into a high-dimensional vector space has been borrowed from the field of natural language processing, where the *distributional hypothesis* ("words that occur in similar contexts tend to have similar meanings") is used to justify the approach [16]. The work from [16] shows certain issues with the meaningfulness of embedding vectors in some type of sentences, the prominent one being the problem with contextuality. This problem stems from the fact that the corpus from which the embeddings may be extracted, may contain samples wherein semantically different words occur nearby each other. Consequently, the embeddings extracted would treat these words in a similar manner, given that their context appears to be similar whereas their semantics is in fact significantly different.

[1] Variational Autoencoders

[2] ability to detect features anywhere in the image

Apart from the conceptual issues elaborated above, the use of 1-dimensional inputs to the CNN network has significant performance advantages. This is further substantiated by the results in Section VI, where we attribute the improvement in training time to the decision of using single dimensional inputs, instead of embedding vectors.

Summing up: by eliminating the need for two dimensional representation of binaries, the proposed model is conceptually simpler, able to work in a computationally efficient manner, with no negative effect on the accuracy, and in case of works with publicly available data-sets, even outperforming them, thereby strengthening our assumptions regarding the issues with embedding space based detectors.

B. 1-Dimensional Convolutional Neural Network

CNNs have been one of the major components of the use of artificial neural networks for solving complex computer vision problems.

The basis of CNNs is the convolution operation that is carried out as a part of the convolutional layers. For most of the computer vision tasks, a 2-dimensional version of this operation is used, given that the network usually operates on 2-dimensional images. Typically such convolution can be seen as a matrix of tunable parameters that scans through the image from left to right an top to bottom; this operation is capable to capture visual features (e.g. a straight line) whenever it may occur in the image, in particular for example both vertical, diagonal and straight lines.

A 1-dimensional convolution operation works on the same principle, with the only major difference being the input and the filter dimensions; it is hence a vector scanning the program top to bottom, looking for features in this sequence.

In the example depicted in Fig. 2, the notion of 1-dimensional convolution operation is shown with a 1-dimensional 5×1 input vector, that is being acted upon by another 1-dimensional 3×1 weight vector. The 3×1 vector, moves in strides of 1, working on a 3×1 section of the input, as it moves across it, producing a 3×1 output vector.

Formally, given a weight vector $\vec{W}$, of dimensions $r \times 1$ (also called a filter or 'receptive field') acting on the input vector $\vec{T}$, a convolution with output z is defined as follows:

$$z = \vec{W} * \vec{T}[p : p + r] \tag{1}$$

where *p* is the current position of the weight vector as it strides along the input, to produce a vector of output values. In equation 1 the operation $*$ is the element-wise multiplication of the weight vector with the input, followed by the summation of the values obtained. The pseudo-code below is an implementation of the operation $*$:

```
for i in len(W):
    z = z + W[i]*T[p+i]
```

The values z, obtained as the filter moves across the input together form the input vector to the subsequent layer.

After training the parameters get tuned so that each weight vector is responsive to a specific feature in the binary. These 'receptive fields', when analyzed in the context of instruction sequences extracted form binaries, turn out to work as instruction sub-sequence detectors. The convolutional layers of weight vectors, of dimensions 3×1 in our model, works as a 'tri-gram' detector. Following the training process, the weight vectors get tuned to be activated at certain 'tri-grams', which when present in certain combinations, result in the binary being classifier as 'malicious'.

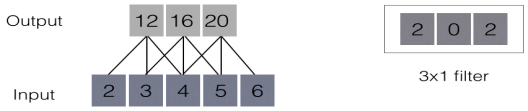

Fig. 2. 1-Dimensional Convolution

The activation layer, conventionally, works on the output of the preceding convolutional layer by applying a specific function to the output vector values, in an element-wise manner.

The detector uses the Rectified Linear Unit or ReLU activation function. Apart from being computationally advantageous, the ReLU activation function is well studied and has known, positive implications on the training process.

The dense layers are placed at the end of architecture, receiving inputs from the previous convolutional and activation layers. In essence, these layers are identical, in terms of functionality, to the hidden layers of a traditional multi-layer perceptron. The neurons in these layers are fully-connected, in terms of the weights, and hence are 'dense' from a connectivity point-of-view.

In case of a 1-dimensional CNN architecture, the dense layers serve as the final stage for parametric operations, before the class probabilities are generated.

IV. Implementation

This section describes the two architectures implemented: the first is a TF-IDF based detector used as benchmark implementation. The second architecture implemented, and main contribution of this work, is based on 1-dimensional convolutional neural networks.

A. TF-IDF n-gram based detector

1) Background on TF-IDF: Term frequency-inverse document frequency, or TF-IDF, is a measure, which has been used extensively in the field of information retrieval. It works on the principle of utilizing a term's relative commonness for associating a weight with the said term. It estimates a term's relative commonness using a two-part formula, the elements of which capture the frequency of the respective term while accounting for its information content.

The TF-IDF measure value for a term *t* in document *d* from the corpus $\mathcal{C}$ is

$$\text{TF-IDF}_{\text{t,d}} = \text{tf}_{\text{t,d}} \cdot \text{idf}_{\text{t},\mathcal{C}} \tag{2}$$

Where $\text{tf}_{\text{t,d}}$ is the *term-frequency* of the term t in document d: Its value is the number of times t appears in the document d divided by the total number of terms in d.

The problem with term-frequency on its own is that common terms (like 'and', 'that' etc.) have high frequency yet are not very important. The $\text{idf}_{\text{t},\mathcal{C}}$ term addresses this problem.

The term $\text{idf}_{\text{t},\mathcal{C}}$ in the formula (2) is the *inverse document frequency* which is defined as the log of the number of documents in the corpus divided by the number of documents containing the term t:

$$\text{idf}_{\text{t},\mathcal{C}} = \log \frac{|\mathcal{C}|}{|\{d \in C \mid t \in d\}|} \tag{3}$$

For terms which do not occur in a large number of documents of the corpus, the $\text{idf}_{\text{t},\mathcal{C}}$ formula assigns a relatively higher value as compared to the more commonly used terms across the corpus. The aforementioned term is, hence, able to improve the weights associated with individual terms that, when encountered in a document, convey more information.

2) TF-IDF document vector: Let $\mathcal{T} = \{t_1, \ldots, t_n\}$ be the set of terms in all documents from the corpus. $\mathcal{T}$ provides a basic standard vocabulary, against which, we can define TF-IDF measure values for each $d \in \mathcal{C}$.

Given a document d, its associated TF-IDF vector is then defined as:

$$\vec{D}_d = [\text{TF-IDF}_{t_1,d} \cdots \text{TF-IDF}_{t_n,d}]$$

This vector is a numerical representation of the structure of d against a base set of terms.

Such vectorial representation of documents holds structural information that can be leveraged in various pattern detection tasks.

The TF-IDF detector involves the use of term frequency-inverse document frequency measures of n-gram sequences for extracting features from the subject binaries.

The detector here implemented has been influenced by works in [21], [8] and [19].

For the purpose of developing a benchmark detector based on classical feature-based techniques, we evaluated the performance of n-gram and TF-IDF based features individually, as summarized in Table I. Based on the results in Table I, our design of using TF-IDF measures of bi-grams seems reasonable, given that it happens to be the best performing methodology.

3) Detector: The TF-IDF detector has the following two components:

a) Feature Engineering: Given a binary file b, seen here as a sequence of assembly instructions, we extract a list of bi-grams. For example given the following snippet of assembly code:

```
.text:00401314  pop     eax
.text:00401315  call    eax
.text:00401317  test    eax, eax
.text:00401319  jnz     short loc_401302
.text:0040131B  pop     edx
.text:0040131C  push    edx
```

We obtain the following list of bi-grams:

```
['pop call', 'call test', 'test jnz',
'jnz pop', 'pop push']
```

The 'corpus', $\mathcal{C}$ of 'documents', is defined as the set of all list of bi-grams, one list for each binary file in the set.

The set $\mathcal{G}$ is the set of all bi-grams appearing at least once in the corpus $\mathcal{C}$. The set of bi-grams thus defined serves as the 'vocabulary' for calibrating the features using TF-IDF measure values.

The corpus $\mathcal{C}$ is then processed in order to compute the TF-IDF measures for each 'document' in $\mathcal{C}$, such that for each $d \in \mathcal{C}$, we have a $|\mathcal{G}|$-length vector of TF-IDF measure values. The process followed is described in IV-A2, where the standard vocabulary set $\mathcal{T}$ as described in section IV-A2 is the set of bi-grams $\mathcal{G}$.

This TF-IDF measure vector serves as the document's input feature that gets fed into the artificial neural network, which uses it for the binary classification task of establishing as to whether or not the input binary is malicious or benign.

b) Neural Network for the TF-IDF detector:: The TF-IDF detector is built on top of a fully-connected artificial neural network, which is trained on the TF-IDF measure based features extracted from the binaries using the feature engineering process.

The network uses a 2 hidden-layered, fully-connected, artificial neural network, where the 2 hidden layers are attached to an input and an output layer. The hidden layers are each 64 neurons wide, with Rectified Linear Unit (ReLU) activation functions and a dropout probability of 50%.

The final layer, which also serves as the output layer of the network, uses the softmax function to produce prediction values. The network uses the standard softmax cross-entropy loss function and the Adam optimizer during the training process.

B. Deep 1-D Convolutional Neural Network based detector

The second technique that the paper proposes for the task of malware detection is based on the notion of convolutional neural networks (CNN).

1) Data pre-processing: A binary file from the dataset is represented as a list of instructions, e.g. `['pop','call','test','jnz','pop','push']` The processed binaries are then padded (with a fresh character not used for the alphabet of instructions) such that the resultant dataset has samples of the same length. This dataset is utilized for training and evaluating the neural network, which is described in the next section.

The pre-processing stage for this methodology differs from the TF-IDF pre-processing as it doesn't extract features from the samples, thereby feeding into the neural network the raw sequences of instructions extracted from sample binaries.

2) 1-D Convolutional Neural Network: The pre-processed binaries, as described in the previous section, are fed into the network, which, after a set of convolutions, activation and matrix multiplication operations, outputs the class probabilities.

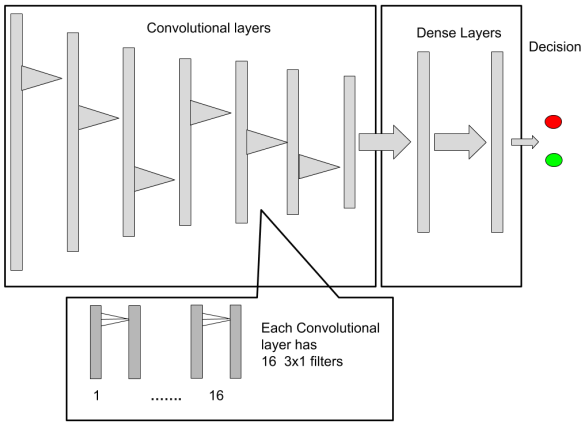

Fig. 3. 1-Dimensional CNN architecture

Method	Accuracy	Train time
Ngram	87.2%	150
TF-IDF	88.0%	180
TF-IDF+Ngram	93.7%	190

TABLE I
METHODOLOGY COMPARISON FOR BENCHMARK DETECTOR. TRAIN TIME IS TIME PER EPOCH (IN SECONDS)

These probabilities are used for deciding whether or not a binary is malicious.

The implementation consists of a neural network composed of six 1-D convolutional layers followed by two fully-connected, dense layer. Following the convention associated with convolutional neural network, this work uses ReLU (Rectified Linear Units) activation layers, in front of each of the six convolutional layers. Figure 3 shows an overview of the architecture.

Our architecture has differences from the other works using CNNs for this task. In contrast with the them, these convolutional layers used employ a single dimension filter on a single dimension output, leading to an architecture with only single dimensional components. Also, the model proposed in this work doesn't make use of any max-pool layers. In works like [10], one of the reasons presented for using max-pool layers is the need for a lower size input for the subsequent layers. Due to the use of 1-dimensional input-filter convolutions for the CNN model proposed here, we managed to achieve the desired accuracy without the need for reducing input sizes.

The architecture uses a set of 16 filters for each of the six convolutional layers. The individual filters have a window size of 3×1 and slide at a stride of 1 across the input vector. The former of the two dense layers produces a 16 dimensional output while the latter serves as the output layer, producing the class probabilities.

Our architecture follows standard conventions, including use of a 50% dropout rate for the neurons in the intermediary layers. The network uses the Adam optimizer at the initial learning rate of 0.0001 for minimizing the categorical cross-entropy loss function during the training process.

V. EXPERIMENTS

The following section is divided into two parts, explaining the datasets and the class distribution. For both datasets the Portable Executable (PE) format file is used.

A. Dataset A

The dataset consists of a set of malware executables available from the Microsoft BIG Malware Dataset [22]. The benign files in the dataset are system executables from Windows 10 installations.

For the malware binaries available from the Microsoft BIG classification challenge [22], the *asm* files were already provided. This dataset from Microsoft consisted of a number of malware samples from 9 different malware families. The most samples available are from Ramnit, Lollipop and Kelihos ver3 families, and hence form the majority of the subset used for experiments. These comprise of computer worms, adwares and botnets. For the set of benign binaries, we used a disassembler to process it. Given that this methodology of collecting benign binaries has been followed by a number of other works, we use it to minimize the differences in the testing scenarios. Table II describes the dataset split.

B. Dataset B

The second dataset is the one made available by the authors from [10]. The dataset contains 11130 binary samples, of which 6066 samples are labelled as malware and the rest 5064 are labelled as 'trusted'. The malware samples in this dataset have been downloaded from VirusShare and have been verified by VirusTotal, making this dataset specific for viruses. The benign samples were collected from application stores (Tencent and Baidu). The dataset is downloadable from the URL provided in [10] [3]

	Count	Size	Variance
Malware	1155	9.5	553.7
Benign	608	6.8	308.3

TABLE II
AVERAGE SIZE AND VARIANCE (IN MB) FOR BINARIES IN DATASET A.

Comparing the two dataset here are some pros and cons:

Dataset A contains binaries of varying sizes and more varied nature, with the number of binaries belonging to the 'benign' class being significantly lower. This allows for the classifier to learn on a more closely modelled real-world dataset, given that binaries of various sizes may appear in practice.

For dataset B, the class sizes are almost the same, and the size of binaries is not as different as in dataset A. This serves as a reasonable dataset due to the much larger number of samples available for the classifier to train on.

VI. RESULTS

Both architectures specified in the previous sections have been implemented using Keras ([5]) with Python 3.5. The network architecture has the capability to be trained on a GPU, if available.

[3] https://github.com/deep-learning-malware/Dataset

For the purpose of training the network, we used the Adam optimizer, at a learning rate of 0.0001, with a batch size of 16 samples, and the entire training set being considered for the epochs.

The training epoch length for our 1-dimensional CNN detector is 10, when run on an Ubuntu virtual machine from Google Cloud's compute engine suite, containing one Nvidia Tesla P100 GPU for use.

The following subsections contain the results obtained during the experiments conducted on the two datasets, with the detectors from this work.

a) Benchmark Detection Tool's Results: Dataset A: In order to compare the performance of our system with the benchmark detector implemented in this work, we split the dataset A in two parts, the training set and the test set, with a ratio of 80:20, respectively, for the two parts. The model gets trained on the 80% training data, and is tested on the remaining 20% of the data.

The performance of the benchmark detector is summarized by the classification report, shown in Table III. The TF-IDF based detector performs well on the 'malware' class, while the performance drops for the samples from the 'benign' class. A possible explanation for this behaviour may be the imbalance between classes' sizes in the dataset A.

The confusion matrix concrete values for the TF-IDF based detector is shown in Table IV. The number of 'False Positives' by the benchmark detector is low, with an overall accuracy of 93.7%.

The ROC curve for this detector, Fig. 4, reflects, graphically, the classification performance of this detector. The curve, whose slope is a representation of the ratio between True Positive Rate and the False Positive Rate for different threshold values, depicts the detector's performance against varying threshold values. It can be seen from the curve, that the classification performance of this detector reflects the classification report values.

The performance of this classifier is comparable with the performance of some of the other detectors in the literature based on n-grams and TF-IDF. This sets for a fair comparison with the CNN detector being proposed in this work.

	precision	recall	f1-score	support
Benign	85%	100%	92%	608
Malware	100%	91%	95%	1155
Average	92.5%	95.5%	93.5%	1763

TABLE III
CLASSIFICATION REPORT:TF-IDF-BASED BENCHMARK DETECTOR: DATASET A

	True diagnosis		
	Positive	Negative	
Positive	1046	2	
Negative	109	606	
Total	1155	608	1763

TABLE IV
CLASSIFICATION REPORT VALUES:TF-IDF-BASED BENCHMARK DETECTOR: DATASET A

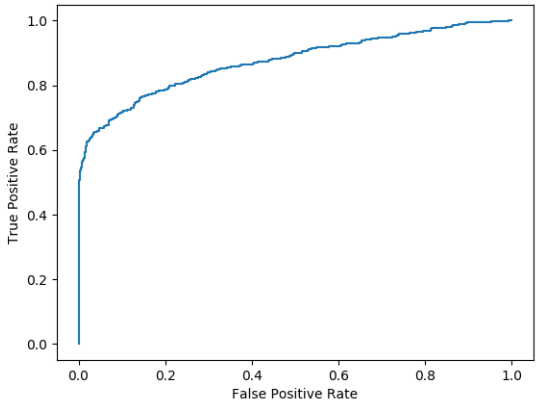

Fig. 4. ROC curve: TF-IDF-based benchmark detector : Dataset A

b) Proposed CNN-based Detector Results: Dataset A: The performance of the 1-dimensional CNN based detector is evaluated on the same dataset and using the same settings as for the TF-IDF based benchmark detector.

The CNN-based detector was trained for a period of 10 epochs, following which the learning loss was seen to have stabilized. Due to the use of one dimensional inputs for the convolutions to work on in the detector as opposed to two dimensional inputs, the training process was computationally very efficient, as described in Section VI-A. The model architecture, which was deeper in terms of the number of layers as compared to the benchmark detector, took a comparable time for training, using the same resources.

We use the standard metrics to assess the classification performance of the detector, including the ROC curve (Fig. 5) and the classification report values (Table VII).

The classification report and the confusion matrix values for the 1-D CNN shown in Table VII and Table VI respectively. Results show that with the same amount of data given for training, the CNN-based detector has a precision value of 94% for the samples belonging to the 'benign' class, as compared to the 85% precision obtained from the TF-IDF based detector.

Fig. 6 shows the performance of the detector on the test set during the training process. The graphical representation further shows the detector's improvements on unseen data, as it goes through the learning process.

The CNN-based detector has an accuracy of 97.51% on dataset A, with relatively low 'False Positive' and 'False Negative values'.

c) Proposed CNN-based Detection Tool's Results: Dataset B: To test the efficiency and the accuracy of the CNN-based model and compare to other works, we use the dataset made available by the authors from [10]. This dataset is described in Section V-B.

Various variations in the architecture of the proposed model, as summarized by Table VIII, have been explored, and the six-layered architecture produces the best results. The depth of the architecture in the proposed CNN model improves

Technique	Accuracy	Precision	Recall	F1-score
Proposed model	**99.2%**	**99.5%**	**99%**	**99%**
EzNet	99%	99.4%	98.4%	98.9%

TABLE V
MODEL COMPARISON: CNN-BASED DETECTOR AND EZNET : DATASET B

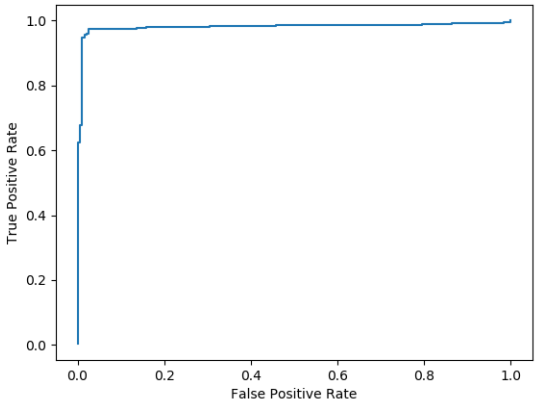

Fig. 5. ROC curve: CNN-based detector : Dataset A

	precision	recall	f1-score	support
Benign	94%	99%	96%	608
Malware	99%	97%	98%	1155
Average	96.5%	98%	97%	1763

TABLE VI
CLASSIFICATION REPORT: CNN DETECTOR: DATASET A

	True diagnosis		
	Positive	Negative	
Positive	1118	7	
Negative	37	601	
Total	1155	608	1763

TABLE VII
CLASSIFICATION REPORT VALUES: CNN-BASED DETECTOR: DATASET A

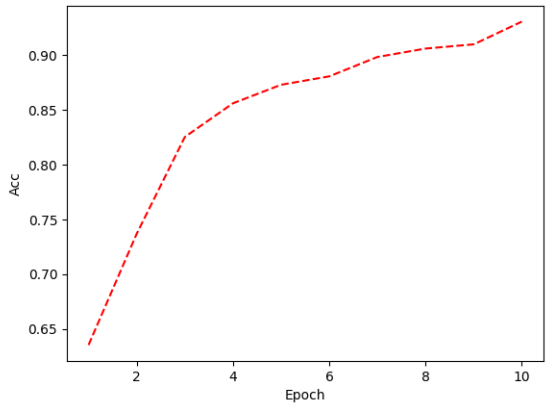

Fig. 6. Validation accuracy curve during training: CNN-based detector: Dataset A

the accuracy measure and doesn't have any adverse impact, computationally, on the training process. The increase in the number of layers provides more parameters thereby increasing the learning capacity of the model. The maximum accuracy is reached by the architecture with 6 convolutional layers, with no max-pool layers. Beyond this, increasing the layers doesn't impact the accuracy and makes the training computationally expensive.

The dataset was split into two parts, 80% containing the training data while the remaining 20% containing the testing data, which was used during training to monitor the detector's learning. The training process lasted for a total of 10 epochs, with the best accuracy being achieved on the 8^{th} epoch.

The evaluation metric results are shown in Table XI holding the classification report and Table XII holding the classification report values.

The detector, on evaluation, has relatively low 'False Positive' and 'False Negative' values. This reflects the high overall accuracy obtained. As expected while training this detector on the smaller dataset A, it generalizes well on a much larger dataset, while staying computationally efficient.

The detector performs with an overall accuracy of 99.2%, with the recall and f1-score values of 99% and an average precision score of 99.5%. Given this, the performance happens to be better than the detector presented in [10] in terms of the overall accuracy, precision, recall and f1-score percentages achieved, as summarized by Table V.

The proposed detector's behaviour after training can be seen in Fig. 7, where we plot of the output of the first 1-dimensional convolutional layer, and in Fig. 8, plotting the the output of the last 1-dimensional convolutional layer. In both plots the x-axis represents the vector output of the convolutional layer, the y-axis the value of the vector at that entry. We consider the trained network and we randomly sampled 500 'benign' binaries and 500 'malware' binaries from dataset B, and pass it through the network. A red dot in the plot is the average value of the 500 'malware' binaries for that vector entry. A green dot is the average value of the benign binaries on that vector entry.

Since the two plots have been generated after the model has been trained, we can see the difference in the activations (from the first filter) produced by 'malware' and 'benign' samples. In particular, both figures show a clear separation between the sections of the layer outputs that get activated by 'malware' and the ones activated by 'benign' samples.

A. *Discussion*

Based on the results from the experiments, the 1-dimensional CNN-based detector here presented is a computationally light and accurate detection mechanism.

Architecture	Best Accuracy	Train time
3-layers	95.2	100
4-layers	95.8	110
5-layers	97.2	114
6-layers	99.2	130
7-layers	98.10	134

TABLE VIII
ARCHITECTURE COMPARISON: CNN-BASED DETECTOR: DATASET B. EACH LAYER HAS 16 FILTERS. TRAIN TIME IS TIME PER EPOCH (IN SECONDS)

	precision	recall	f1-score	support
Benign	97%	98%	98%	5064
Malware	98%	98%	98%	6066
Average	97.5%	98%	98%	11130

TABLE IX
CLASSIFICATION REPORT: TF-IDF BASED BENCHMARK DETECTOR: DATASET B

Due to the unavailability of public datasets for comparisons when it comes to classical feature-based approaches, we implemented our benchmark tool and evaluate it against our CNN proposed model. From the results obtained, the 1-D CNN-based detector outperforms the TF-IDF based detector. Apart from the overall accuracy value, where the CNN detector achieves a 97.51% accuracy as compared to the 93.7% accuracy obtained by the TF-IDF detector on dataset A, the CNN-based detector, although being deeper, in terms of layers, and making use of computationally heavier convolution layers, seems to have training times comparable to the shallower TF-IDF based classifier. In order to make a fair comparison between the two approaches we used the same resources and datasets.

For a more general comparison, we tested our CNN-based detector with the one presented by [10] on their publicly available dataset. Based on the results obtained, our 1-D CNN detector achieves a better accuracy, while being computationally light. As shown in Table IX and Table X, our proposed detector, performs better than the benchmark detector on the publicly available dataset as well. Furthermore, the EzNet model with a single layer has a training time of 5,855.28 sec ([10]), while the model presented in this work achieves the

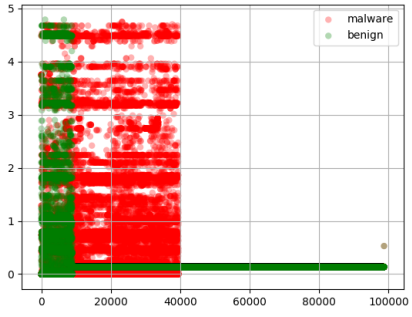

Fig. 7. First layer output visualization for the CNN

	True diagnosis		
	Positive	Negative	
Positive	5933	105	
Negative	133	4959	
Total	6066	5064	11130

TABLE X
CLASSIFICATION REPORT VALUES DATASET B:TF-IDF-BASED BENCHMARK DETECTOR

	precision	recall	f1-score	support
Benign	100%	98%	99%	5064
Malware	99%	100%	99%	6066
Average	99.5%	99%	99%	11130

TABLE XI
CLASSIFICATION REPORT: CNN DETECTOR: DATASET B

said accuracy of 99.2% within a span of 10 epochs, with a per epoch train time of 130 sec, leading to an overall training time of 1,300 sec. We attribute this to use of a single dimension input-filter convolutions, as discussed in Sec. III-A.

Limitations: The detector relies on the availability of the disassembled code from a disassembler. There have been techniques, explained in [18], which discuss obfuscation techniques which make the disassembly of executables difficult. More general machine learning based limitations including constructed adversarial attacks may provide challenges to the detector.

VII. CONCLUSION

Convolutional neural networks have led to impressive advances in the field of machine learning. Their success originated from their proven capability of automatically discovering relevant features in images and their generalization capabilities. This paper introduced a novel detection technique for malware binaries based on convolutional neural networks. A crucial difference with other CNN approaches is the use of

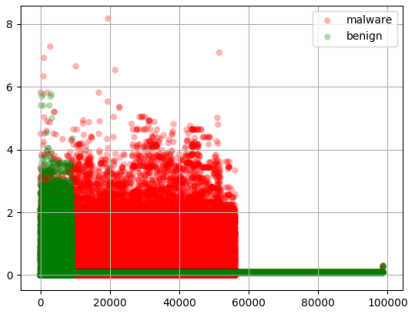

Fig. 8. Final layer output visualization for the CNN

	True diagnosis		
	Positive	Negative	
Positive	6059	86	
Negative	7	4978	
Total	6066	5064	11130

TABLE XII
CLASSIFICATION REPORT VALUES: CNN DETECTOR: DATASET B

1-dimensional convolutions. This methodological choice is justified in terms of its conceptual meaningfulness. The conceptual simplification compared to 2-dimensional convolutions has also practical implications; based on publicly available datasets, it suggests it improves on the state of the art, both in accuracy and training times with an at least five-fold improvement in training time with respect to a comparable detector. Further works of interest would be investigating this methodology for classification of malware families and the integration with static analysis techniques like abstract interpretation to further improve robustness.

REFERENCES

[1] Nadia Alshahwan, Earl T. Barr, David Clark, and George Danezis. Detecting malware with information complexity. *CoRR*, abs/1502.07661, 2015.

[2] Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion. Control Flow Graphs as Malware Signatures. In Eric Filiol, Jean-Yves Marion, and Guillaume Bonfante, editors, *International Workshop on the Theory of Computer Viruses*, TCV'07, Nancy, France, May 2007. Matthieu Kaczmarek; Guillaume Bonfante.

[3] Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion. Control Flow to Detect Malware. In *Inter-Regional Workshop on Rigorous System Development and Analysis 2007*, Nancy, France, October 2007. Pascal Fontaine; Stephan Merz.

[4] Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga. Detecting self-mutating malware using control-flow graph matching. In Roland Büschkes and Pavel Laskov, editors, *Detection of Intrusions and Malware & Vulnerability Assessment*, pages 129–143, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[5] Franois Chollet. keras, 2015.

[6] Mihai Christodorescu, Somesh Jha, Sanjit A. Seshia, Dawn Xiaodong Song, and Randal E. Bryant. Semantics-aware malware detection. In *IEEE Symposium on Security and Privacy*, pages 32–46. IEEE Computer Society, 2005.

[7] Zhihua Cui, Fei Xue, Xingjuan Cai, Yang Cao, Gai-ge Wang, and Jinjun Chen. Detection of malicious code variants based on deep learning. *IEEE Transactions on Industrial Informatics*, 14(7):3187–3196, 2018.

[8] Zhang Fuyong and Zhao Tiezhu. Malware detection and classification based on n-grams attribute similarity. In *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE, jul 2017.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[10] Zeliang Kan, Haoyu Wang, Guoai Xu, Yao Guo, and Xiangqun Chen. Towards light-weight deep learning based malware detection. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, jul 2018.

[11] Boojoong Kang, Suleiman Y. Yerima, Kieran Mclaughlin, and Sakir Sezer. N-opcode analysis for android malware classification and categorization. In *2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security)*. IEEE, jun 2016.

[12] Temesguen Messay Kebede, Ouboti Djaneye-Boundjou, Barath Narayanan Narayanan, Anca Ralescu, and David Kapp. Classification of malware programs using autoencoders based deep learning architecture and its application to the microsoft malware classification challenge (big 2015) dataset. In *Aerospace and Electronics Conference (NAECON), 2017 IEEE National*, pages 70–75. IEEE, 2017.

[13] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751. Association for Computational Linguistics, 2014.

[14] Johannes Kinder, Stefan Katzenbeisser, Christian Schallhart, and Helmut Veith. Detecting malicious code by model checking. In *Proceedings of the Second International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'05, pages 174–187, Berlin, Heidelberg, 2005. Springer-Verlag.

[15] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. Deep learning for classification of malware system call sequences. In *AI 2016: Advances in Artificial Intelligence*, pages 137–149. Springer International Publishing, 2016.

[16] Stephen McGregor, Matthew Purver, and Geraint Wiggins. Words, concepts, and the geometry of analogy. *Electronic Proceedings in Theoretical Computer Science*, 221:39–48, aug 2016.

[17] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, and Gail Joon Ahn. Deep android malware detection. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, pages 301–308, New York, NY, USA, 2017. ACM.

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

[19] Philip O'Kane, Sakir Sezer, Kieran McLaughlin, and Eul Gyu Im. SVM training phase reduction using dataset feature filtering for malware detection. *IEEE Transactions on Information Forensics and Security*, 8(3):500–509, mar 2013.

[20] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. Malware detection by eating a whole EXE. *CoRR*, abs/1710.09435, 2017.

[21] Edward Raff, Richard Zak, Russell Cox, Jared Sylvester, Paul Yacci, Rebecca Ward, Anna Tracy, Mark McLean, and Charles Nicholas. An investigation of byte n-gram features for malware classification. *J. Computer Virology and Hacking Techniques*, 14(1):1–20, 2018.

[22] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. Microsoft malware classification challenge. *CoRR*, abs/1802.10135, 2018.

[23] Imtithal A. Saeed, Ali Selamat, and Ali M. A. Abuagoub. Article: A survey on malware and malware detection systems. *International Journal of Computer Applications*, 67(16):25–31, April 2013. Full text available.

[24] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *MALWARE*, pages 11–20. IEEE, 2015.

[25] Asaf Shabtai, Robert Moskovitch, Clint Feher, Shlomi Dolev, and Yuval Elovici. Detecting unknown malicious code by applying classification techniques on OpCode patterns. *Security Informatics*, 1(1), feb 2012.

[26] Mahmood Yousefi-Azar, Vijay Varadharajan, Len Hamey, and Udaya Kiran Tupakula. Autoencoder-based feature learning for cyber security applications. In *IJCNN*, pages 3854–3861. IEEE, 2017.

[27] Richard Zak, Edward Raff, and Charles Nicholas. What can n-grams learn for malware detection? In *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, oct 2017.

[28] Yi Zhang, Yuexiang Yang, and Xiaolei Wang. A novel android malware detection approach based on convolutional neural network. In *Proceedings of the 2Nd International Conference on Cryptography, Security and Privacy*, ICCSP 2018, pages 144–149, New York, NY, USA, 2018. ACM.