Orthrus: A Bimodal Learning Architecture for Malware Classification

Daniel Gibert
Dept. of Computer Science
University of Lleida
daniel.gibert@diei.udl.cat

Carles Mateu
Dept. of Computer Science
University of Lleida
carlesm@diei.udl.cat

Jordi Planes
Dept. of Computer Science
University of Lleida
jordi.planes@diei.udl.cat

***Abstract*—Malware detection and classification is a challenging problem and an active area of research. Traditional machine learning methods depend almost entirely on the ability to extract a set of discriminative features into which characterize malware. However, this feature engineering process is very time consuming. On the contrary, deep learning methods replace manual feature engineering by a system that performs both feature extraction and classification from raw data at once. Despite that, a major shortfall of these methods is their inhability to consider multiple disparate sources of information when performing classification, leading them to perform poorly when compared to multimodal approaches. In this work, we introduce Orthrus, a new bimodal approach to categorize malware into families based on deep learning. Orthrus combines two modalities of data: (1) the byte sequence representing the malware's binary content, and (2) the assembly language instructions extracted from the assembly language source code of malware, and performs automatic feature learning and classification with a convolutional neural network. The idea is to benefit from multiple feature types to reflect malware's characteristics. The experiments carried on the Microsoft Malware Classification Challenge dataset show that our proposed solution achieves higher classification performance than deep learning approaches in the literature and n-gram based methods.**

***Index Terms*—Malware Classification, Convolutional Neural Networks, Deep Learning, Multimodal Learning**

I. Introduction

The detection of malware, malignant computer software designed to infiltrate or damage a computer system without consent of the owner, is an important and challenging problem in cybersecurity. The global malware industry is estimated to be worth millions and grows year after year, with an underground services market which provides malicious software, cybercapabilities, and products to criminals, gangs, and even nation states. Recently, we have seen malware campaigns affecting our daily lives, influencing major elections, and crippling businesses overnight. The most notorious cyberespionage campaign affected the Democratic National Committee computer network and ended up with the release of private and confidential information from party members. In addition, the awareness of the danger of cyber threats increased due to the harm posed by major cyberattacks like WannaCry and Petya campaigns, among others, which held computer systems from all over the globe to ransom.

To limit the impact of cyberattacks it is necessary to improve computer systems' defenses. One essential layer is endpoint protection, specially anti-malware scanners, which is the last layer of defense against malware by preventing, detecting, and removing malicious software. Traditional anti-virus engines use a signature-based approach, where a signature is a set of manually defined rules that can identify a concrete piece of malware or a group with similar characteristics. However, this rules are generally specific, sensitive to small changes, and cannot usually recognize new malware. In consequence, the need for new methods to detect unknown malware is appealing for signature-less machine learning approaches due to their ability to summarize complex relationships and later decision making.

Traditional machine learning solutions perform feature engineering to manually extract a set of features that provide an abstract representation of malware. These features can be obtained from the static and dynamic analysis of malware. On the one hand, static analysis consists of examining the code or structure of a computer program without executing it. On the other hand, dynamic analysis monitors the execution of the program on the system. Indistinctly of the type of analysis, feature-based approaches depend almost entirely on the set of discriminative features used to represent malware. Contrarily, deep learning approaches obviate the need for manual feature engineering by automating the feature learning and classification procedure. Deep learning shifts the burden of feature engineering to an underlying system, typically consisting of a neural network with multiple layers, that jointly perform both feature learning and classification. For instance, E. Raff et al. [1] and D. Gibert et al. [2] trained a neural model by feeding it, as input, a sequence of bytes and a sequence of opcodes (machine language instruction), respectively. Nonetheless, both approaches lack the information from multiple sources of information that is combined before classification in traditional machine learning approaches. Thus, deep learning approaches for malware detection tend to perform poorly in comparison with multimodal approaches.

The primary contribution of this work is the development of the first, to our knowledge, bimodal deep learning architecture for malware classification. Orthrus automatically learns

This research has been partially funded by the Spanish MICINN Projects TIN2015-71799-C2-2-P, ENE2015-64117-C5-1-R, and is supported by the University of Lleida.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

features from two sources of information, (1) the hexadecimal representation of malware's binary content, and (2) the assembly language instructions representing the assembly language source code of malware. The idea is to learn from multiple sources of information to maximize the benefits of multiple features types to reflect the characteristics of malware and, to compensate for the weaknesses inherent in unimodal representations. The generalization performance of our bimodal learning approach has been evaluated on the dataset provided by Microsoft for the Big Data Innovators Gathering Anti-Malware Prediction Challenge [3]. Furthermore, we present a comparison with deep learning methods in the literature. Experiments show that our model successfully takes advantage of both modalities of information to perform significantly better than unimodal deep learning methods.

The rest of the paper is organized as follows. Firstly, we introduce the state-of-the-art approaches to address the problem of malware detection and classification. Afterwards, we describe the bimodal architecture followed by the results of the experimentation. Lastly, we summarize the concluding remarks of our research and proposes some future lines of research.

II. Related Work

Traditional machine learning solutions rely on a set of hand-designed features that provide an abstract representation of the program that is later used for classification. The most common features are byte and opcode n-grams [4], [5]. Byte n-grams are extracted from the hexadecimal representation of malware's binary content whereas opcode n-grams are extracted from the assembly language source code of malware.

To detect the presence of compressed and encrypted segments hidden beneath the executable, security researchers use entropy analysis, as compressed and encrypted segments tend to have higher entropy in comparison with native code [6]. However, simple entropy statistics is not enough to detect sophisticated malware, as packed and encrypted code is often concealed in a way that pass through entropy filters. Thus, researchers started analyzing the structural entropy of executables [7]. The structural entropy consists of a stream of entropy values, where each value describes the amount of entropy over a chunk of code in a specific location of the executable.

A distinct way to represent an executable is to visualize its byte code as a grayscale image [8], where every byte is interpreted as one pixel in the image. Afterwards, features describing the texture of the grayscale image can be extracted such as GIST [8], Haralick [9], Local Binary Patterns [9] and PCA features [10].

In addition, the usage of system functions and libraries is a good indicator of the behavior of malware as they offer information about services and resources provided by the OS [11].

The need for manual feature engineering can be obviated by automated feature learning. Deep learning replaces the feature engineering process by an underlying system which typically consists of a neural network with multiple layers, that performs both feature learning and classification. With deep learning, one can start with raw data as features will be automatically created by the neural network when it learns. The main distinction between deep learning approaches for malware detection and classification lean on what they used as raw data.

D. Gibert et al. [2] and N. McLaughlin et al. [12] feed convolutional networks with the opcode sequences extracted from disassembled Portable Executables (PEs) and Android APKs, to classify malicious software targeting the Windows and the Android operative systems, respectively. The shallow layers of the convolutional networks allow to extract N-gram like features without consuming the exploding amount of computational resources required to extract N-grams for a long N. Alternatively, D. Gibert et al. [13] take advantage of the hierarchical structure of Portable Executable files to build a hierarchical convolutional network that extracts features at the mnemonics-level and at the function-level.

On the contrary, E. Raff et al. [1] presented a detection system trained on raw byte sequences. In their work, each byte of the input sequence is embedded into a fixed length feature vector to avoid representing each byte by its value, as it would imply that certain byte values are closer to each other than other byte values, which is false, as the byte value depends on its context. Afterwards, they combined convolution layer with global max-pooling to obtain the activations regardless of the location of the detected features. This shallow architecture applied filters of width equals to 500 bytes followed by an stride of 500, which allowed to identify interpretable subregions of the binary, mostly from the PE header. Furthermore, M. Krl et al. [14] presented a deeper architecture consisting of 11 layers: the embedding layer, four convolutional and two pooling layers, followed by 4 fully-connected layers.

As the length of the bytes sequence might be up to various million time steps, other works preprocessed the sequence to reduce its size and compress its information. D. Gibert et al. [15] feed a convolutional neural network with the structural entropy representation of malware. Hence, the size of the sequence was diminished from millions to thousands or hundreds, depending on the chunk size. Alternatively, D. Gibert et al. [16] generated an encoded representation of contiguous, non-overlapping chunks using a denoising autoencoder. Afterwards, a residual network extracts features from the compressed sequence and performs classification. Q. Le et al. [17] scaled the file byte code to a fixed target size using a generic image scaling algorithm. After scaling, a malware sample corresponds to one sequence of 10000 values. For classification purposes, they applied recurrent neural network layers on top of the convolutional layers.

D. Gibert et al. [18] takes advantage of the representation of malware as a grayscale image [8] to build a convolutional neural network classifier that automatically extracts discriminant features from the image. Moreover, R. Khan et al. [19] analyzed the performance of the ResNet and GoogleNet architectures for the task at hand.

A further way to represent malware is as an ordered

sequence of API functions invoked during its execution. To capture the long-term dependencies in the API function traces, B. Athiwaratkun et al. [20] examined recurrent neural network architectures. In the first stage, a LSTM or GRU constructs the features associated to a particular API trace and later, a single fully-connected layer or logistic regression with sofmax perform classification.In addition, B. Kolosnjaji [21] constructed a neural network classifier based on convolutional and recurrent layers that combines convolution of n-grams with sequential modeling provided by the recurrent layers.

III. Classification of Malware using a Bimodal Architecture

The main focus of this research is the classification into families of malware targeting the Windows operating system (OS). The most common executable file extension for Windows systems is the Portable Executable (PE) file format. In particular, this file format is used for executables, object code, DLLs, FON Font files, and others in 32-bit and 64-bit versions of the Windows OS. To this end, the method has been evaluated on the Microsoft Malware Classification Challenge dataset [3].

A. N-gram approaches

Static machine learning solutions for malware detection and classification extract features from either the hexadecimal representation of malware's binary content or its assembly language source code counterpart. The hexadecimal representation is a simple way to represent the binary's content of a PE file. Using this representation the binary content is represented as a sequence of bytes (base-16 number representation with digits $[0-9]$ and $[A-F]$). See Figure 1 for an hex view of a PE file. The main advantage of representing malware as a

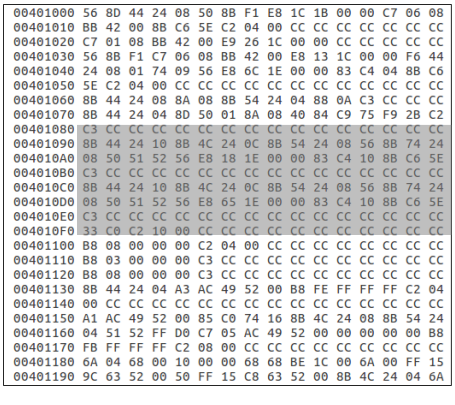

Fig. 1. Hexadecimal view of a PE file.

sequence of bytes is that it is OS resilient, i.e., it could be used to represent malware indistinctively of the target OS and hardware. Alternatively, the assembly language source code contains the symbolic machine code of the executable as well as metadata information such as rudimentary function calls, memory allocation and variable information. See Figure 2 for the assembly view of the grayed area in Figure 1.

```
.text:00401081 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC     align 10h
.text:00401090 8B 44 24 10                                        mov     eax, [esp+10h]
.text:00401094 8B 4C 24 0C                                        mov     ecx, [esp+0Ch]
.text:00401098 8B 54 24 08                                        mov     edx, [esp+8]
.text:0040109C 56                                                 push    esi
.text:0040109D 8B 74 24 08                                        mov     esi, [esp+8]
.text:004010A1 50                                                 push    eax
.text:004010A2 51                                                 push    ecx
.text:004010A3 52                                                 push    edx
.text:004010A4 56                                                 push    esi
.text:004010A5 E8 18 1E 00 00                                     call    _memcpy_s
.text:004010AA 83 C4 10                                           add     esp, 10h
.text:004010AD 8B C6                                              mov     eax, esi
.text:004010AF 5E                                                 pop     esi
.text:004010B0 C3                                                 retn
.text:004010B0                                                    ; ---------------------
.text:004010B1 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC     align 10h
.text:004010C0 8B 44 24 10                                        mov     eax, [esp+10h]
.text:004010C4 8B 4C 24 0C                                        mov     ecx, [esp+0Ch]
.text:004010C8 8B 54 24 08                                        mov     edx, [esp+8]
.text:004010CC 56                                                 push    esi
.text:004010CD 8B 74 24 08                                        mov     esi, [esp+8]
.text:004010D1 50                                                 push    eax
.text:004010D2 51                                                 push    ecx
.text:004010D3 52                                                 push    edx
.text:004010D4 56                                                 push    esi
.text:004010D5 E8 65 1E 00 00                                     call    _memmove_s
.text:004010DA 83 C4 10                                           add     esp, 10h
.text:004010DD 8B C6                                              mov     eax, esi
.text:004010DF 5E                                                 pop     esi
.text:004010E0 C3                                                 retn
.text:004010E0                                                    ; ---------------------
.text:004010E1 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC     align 10h
.text:004010F0 33 C0                                              xor     eax, eax
.text:004010F2 C2 10 00                                           retn    10h
.text:004010F2                                                    ; ---------------------
```

Fig. 2. Assembly view of the grayed part in Figure 1. The first column represents the address, the second column the byte sequence and the third column the mnemonics sequence.

The most common type of features are n-grams. An n-gram is a contiguous sequence of n items from a given sequence of text. N-grams can be extracted from the bytes sequence representing malware's binary content and from the instruction statements extracted from the assembly language source code. By treating a file as a sequence of bytes, byte n-grams are extracted by looking at the unique combination of every n consecutive bytes as an individual feature. On the other hand, n-grams from the assembly language source code refer to the unique combination of every n consecutive opcodes, e.g. the instructions `ADD`, `MUL`, `POP`.

N-gram based methods construct a feature vector representation of malware where each element in the vector indicates the number of appearances of a particular n-gram in the instruction sequence. Thus, the length of the feature vector depends on the number of unique n-grams, which increases exponentially with n. As an example, considering the extraction of bytes n-grams with $n = 3$, the number of possible n-grams is $256^3 = 16,777,216$. Although malware n-grams tend to follow a Zipfian distribution [22], the resulting feature vector is still too large to keep in memory, and even if it is not, you still have to optimize a function with too many input variables, a.k.a. the curse of dimensionality. N-gram based approaches in the literature have reduced this high dimensional input space using feature selection techniques [5], [23] or the hashing trick [24], [25]. On the one hand, feature selection techniques select a subset of relevant features from the initial input space. On the other hand, feature hashing, a.k.a. the hashing trick, is a method for handling sparse, high-dimensional features by using a hash function to determine the feature's location in a vector of lower dimension. It can be seen as a random projection of the the input space $A \in \mathbb{R}^n$ to a low dimensional space $B \in \mathbb{R}^m$, where $m \ll n$. In our case, an array of size N that counts the number of times each n-gram occurred, and a hash function map each n-gram to a location in a lower dimensional array, which will be later used for training a classification algorithm.

In spite of the technique, both feature selection and feature

hashing require to exhaustively enumerate a large number of n-grams during training. To overcome this limitation, D. Gibert et al. [2] explored the application of convolutional networks to malware classification by the assembly language language instructions as a text to be analyzed. This approach has the advantage that the features are automatically inferred from raw data and hence, it removes the feature extraction and selection steps. Similarly, E. Raff et al. [1] and M. Krl et al. [14] presented convolutional neural network architectures to detect malware from raw byte sequences. The main drawback of the aforementioned deep learning approaches is that they focus on only one source of information, either the opcode or the bytes sequence representation, and malware authors can exploit this information to easily bypass detectors [26]. As a result, the most accurate Machine Learning systems for malware detection and classification are still those that are able to extract and combine subsets of features from various sources of information [27].

B. Network Architecture

To overcome the current limitations of deep learning systems in this paper we present Orthrus, a baseline learning system to categorize malware into families that involves various modalities of data. The main idea is to learn from various sources of information to maximize the benefits of multiple feature types to reflect the characteristics of malware. To obviate manual feature engineering, a neural network is used to perform both feature learning and classification. As a result, the network receives as input (1) the sequence of hexadecimal values representing malware's binary content and (2) the sequence of assembly language instructions from the assembly language source code. The automatic feature learning process is carried through a convolutional layer that extracts N-gram like features from both input sequences. Afterwards, the most discriminative features learnt by the filters are combined to produce a final decision, whether the given executable belongs to one family or another. The process of merging intermediate features from the modalities of information is known as intermediate fusion. The overall architecture is presented in Figure 3. It comprises the following layers:

- Bytes input layer. Instead of taking as input an executable represented as a bytes sequence, bytes were grouped into subsequences representing the bytes content of its assembly language source code counterpart. For instance, taking the assembly view in Figure 2 as example, bytes were grouped as: [8B, 44, 24, 10], [8B, 4C, 24, 0C], [8B, 54, 24, 08], [56], [8B, 74, 24, 08], and so on. The maximum sequence length is 16. All subsequences with lesser length were filled with PAD tokens. Initially we considered to use one of the aforementioned architectures [1], [14] but they performed poorly due to the size of the filters and the limited number of samples regarding some families. See Figures 7 and 8. Thus, by grouping the bytes into subsequences and by reducing the size of the kernels we facilitated the learning of simpler and discriminant features.
- Mnemonics input layer. This layer takes as input an executable represented as a sequence of mnemonics. A mnemonic is simply the name of the assembly language instruction. In other words, the parameters of the instruction are removed. For example, the instruction *mov eax, [esp + 10h]* is reduced to *mov*. The maximum number of mnemonics per executable is determined by N, which is set to 10000.
- Embedding layers. As the network cannot be fed with just text strings, each token (either byte or mnemonic) is represented as a low-dimensional vector of real values, also known as word embedding, of size E. In our experiments E has been set to 4. We tried various values [4, 8, 16, 32] for the embedding size and we saw that increasing E does not lead to an increase in accuracy. In addition, increasing the embedding size also increases the memory requirements and in the case of the hexadecimal sequence, it is prohibitive in terms of resources and computational time.
- Bytes convolutional layer. This layer is responsible for convolving various filters over the bytes input and learn filters that activate when a particular feature is found. The size of each filter is $h \times 16 \times E$ where $h \in \{3, 5, 7\}$. Thus, filters are applied to encompass 3, 5 and 7 subsequences at once.
- Mnemonics convolutional layer. This layer convolves various filters over the mnemonics sequence to extract N-gram like features from it. The size of each filter is $h \times E$, where $h \in \{3, 5, 7\}$. As a result, filters are applied to sequences of 3 to 7 mnemonics. The aim behind having filters of various sizes is to allow the network to detect discriminant subsequences that have variations in size.
- Global max-pooling layer. Global max-pooling is applied to extract the maximum activation of each of the feature map activations passed from the convolutional layer.
- Softmax layer. Lastly, the softmax layer combines the features learned and applies the softmax function to output the probability distribution over malware families.

In our experiments we observed that taking both modalities of information as input is suboptimal, since it leads to overfitting one subset of features belonging to one modality and underfitting the features belonging to the other. To address this issue we separately pretrain each subnetwork and optimize their hyperparameters for each subtask. Afterwards, the learned weights of each subnetwork are used to initialize the bimodal network and thus, the knowledge learned by each model is transferred into the bimodal network to save training time and help the network converge faster. Furthermore, to make the network less sensitive to a particular modality it is applied modality dropout, which randomly drops one data modality during training. In addition, dropout has been applied in the softmax layer with a dropout rate equal to 0.5. The nonlinearity function adopted is the Exponential Linear Unit (ELU) [28].

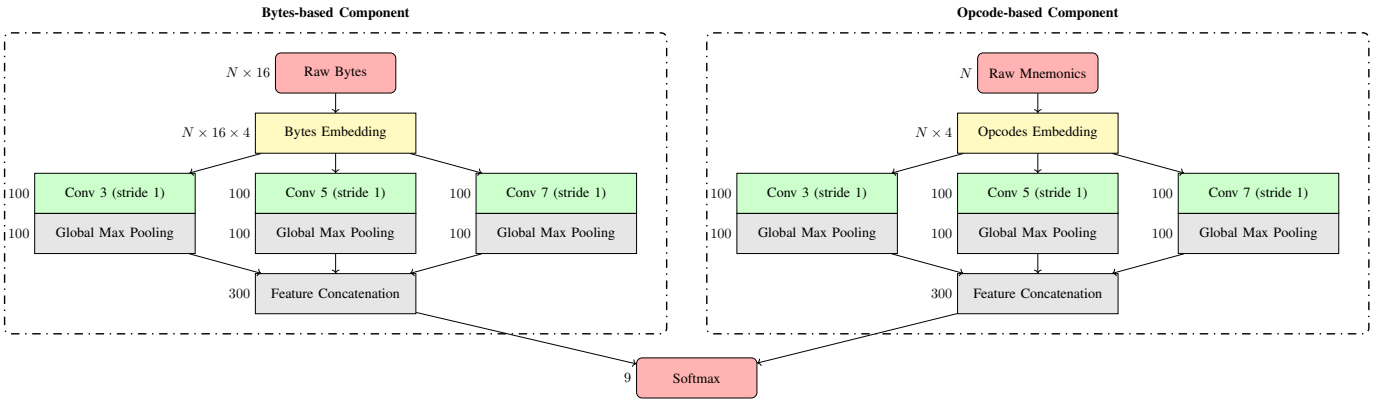

Fig. 3. Bimodal architecture. The letters and the figures at the left side of the layers represent their sizes.

IV. Evaluation

The method has been evaluated on the Microsoft Malware Classification Challenge dataset [3], a standard benchmark for research.

A. Microsoft Malware Classification Challenge

In comparison with other relevant tasks such as image classification, speech recognition, text classification, etc, much of the previous work on malware detection use data not available to public. In consequence, it is not possible to meaningfully compare performance across works as different datasets use different labeling procedures. To simplify comparison and reproducibility we decided to evaluate the performance of our approach on the Microsoft Malware Classification Challenge dataset [3], a standard benchmark for malware research. The dataset is publicly available on Kaggle[1]. It contains the hexadecimal representation of the malware's binary content and its disassembly counterpart. The set of samples represent 9 different families. Cf. Table I. One particularity of the dataset is that the distribution of samples per family is not balanced, i.e., there are some classes with a considerably greater number of samples in comparison with others. Additionally, the average number of bytes and opcodes differs greatly for each class. See Figures 4 and 5. You can observe that those classes with greater number of opcodes do not necessarily coincide with those with the greater number of bytes. This is because the bytes representation includes information of several PE sections, e.g. `.data`, `.edata`, `.idata`.

TABLE I
Class distribution in the Microsoft dataset

Family	#Instances	Type
Ramnit	1541	Worm
Lollipop	2478	Adware
Kelihos_ver3	2942	Backdoor
Vundo	475	Trojan
Simda	42	Backdoor
Tracur	751	TrojanDownloader
Kelihos_ver1	398	Backdoor
Obfuscator.ACY	1228	Obfuscated malware
Gatak	1013	Backdoor

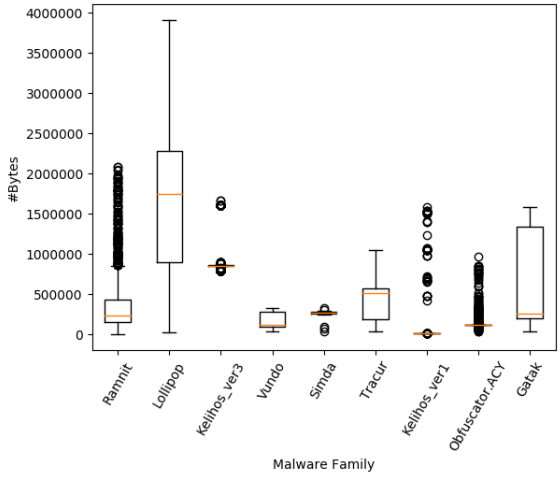

Fig. 4. Distribution of bytes per class.

B. Experimental Setup

The experiments were run on a computer with the following hardware specifications: Intel i7-7700K, 32 GB RAM, 2xNvidia GTX 1080Ti. This allowed us to take advantage of the multi-GPU setup during training to distribute some parts of the model to different GPU instances. That is, each subcomponent of the network was distributed on a different GPU instance.

The generalization performance of our approach has been estimated using 10-fold cross validation. However, instead of evaluating the model with accuracy alone, we selected the best model according to the F1-score. This is because accuracy can be a misleading measure in datasets were there exist a large class imbalance. For instance, a model can correctly predict the value of the majority class for all predictions and achieve a high classification accuracy while making mistakes on the minority and critical classes. In our case, a model can achieve a very high accuracy on the Microsoft dataset by correctly classifying the majority classes and misclassifying samples belong to the `Simda` family. The F1-score metric penalizes

[1]https://www.kaggle.com/c/malware-classification/

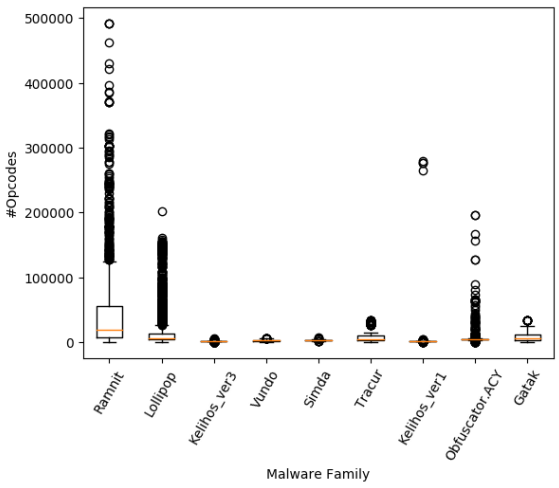

Fig. 5. Distribution of opcodes per class.

this kind of behavior and best meet the requirements of the dataset. Alternatively, we unsuccessfully tried the balanced cross-entropy loss. The results obtained were slightly worse.

C. Comparison with the State-of-the-Art

The 10-fold cross validation confusion matrix is presented in Figure 6. Notice that the percentage of errors in the minority classes does not differ from the number of errors on the majority classes. All classes are classified with more than 97% of accuracy with the exception of the `Simda` family which failed to classify 3 of the 42 samples during 10-fold cross validation. On the other hand, the major contributor to misclassifications is the `Obfuscator.ACY` family, which according to Microsoft, is malware that uses a combination of obfuscation techniques such as encryption, compression, anti-debugging, anti-emulation, etc, to hide its purpose, and thus, are way harsher to classify correctly.

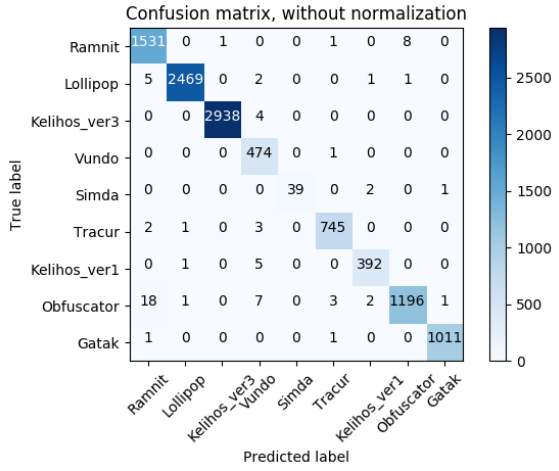

Fig. 6. Orthrus confusion matrix

To evaluate the performance of our bimodal approach, we compared our model with state-of-the-art methods in the literature that have evaluated their model on the Microsoft Malware Classification Challenge dataset. The results are shown in Table II. Existing deep learning approaches for malware classification can be categorized into various groups depending on their corresponding input. With the exception of M. Mays et al. [30], these approaches take as input a single modality of information and perform feature extraction and classification altogether. D. Gibert et al. [18] and J. Kim et al. [29] take as input the grayscale representation of the malware's binary content. D. Gibert et al. [15] represents the content of a malicious program as an entropy stream, where each value describes the amount of entropy of a small chunk of code in a specific location of the file. E. Raff et al. [1] and M. Krl et al. [14] treat each byte as a unit in a sequence and thus, presented architectures to process raw byte sequence of over a few million steps. On the contrary, D. Gibert et al. [16] and Q. Le et al. [17] preprocessed the byte sequence and reduced the size of the input with autoencoders and data compression techniques, respectively. Furthermore, N. McLaughlin et al. [12] and D. Gibert et al. [13] extract N-gram like features from the assembly language instructions of the assembly language content using one or various convolutional layers. Lastly, M. Mays et al. [30] learn two distinct models, one taking as input a grayscale representation of the malware's binary content and the second taking as input a feature vector indicating the presence of particular opcode N-grams. Afterwards, an ensemble classifier returns the final prediction. In addition, we implemented various n-gram classification systems using the hashing trick as baselines. The classification algorithms implemented are logistic regression (LR) and feed-forward neural networks (NN) with one or two hidden layers. The number of hidden neurons is [256] and [256,128] for the neural networks with one and two hidden layers, respectively. The non-linearity applied is the ReLU function. Furthemore, dropout was applied between layers. With the exception of the unigram models, the 2-gram and 3-gram based classification systems apply the hashing trick to map every n-gram into a lower dimensional vector of size 500. The hashing trick has been indispensable to reduce the high dimensionality of the input space. Cf. Table III. As it can be observed in Table II, the bimodal approach outperforms by some margin the existing deep learning methods. Notice that each subnetwork achieves higher accuracy and F1-score than those methods that take as input either the grayscale image representation of malware, its structural entropy or the raw byte sequence. Moreover, the intermediate fusion of features from both the opcode and byte sequences achieve better performance than opcode-based methods. This is because there are some malware instances in the dataset that have been obfuscated with compression and encryption techniques and have very few instructions or none. Thus, the features from the byte sequence provide helpful information and boost the classifier. Additionally, the bytes subnetwork overcome methods [1], [14] for various reasons. First, they have higher complexity (more layers, bigger filter sizes) which make their architectures not suitable for small-size datasets. Second, the 2-dimensional representation of the

TABLE II
STATE-OF-THE-ART COMPARISON OF DEEP LEARNING METHODS FOR MALWARE CLASSIFICATION.

Method	Input	Accuracy	F1-score
LR	Byte 1-Gram	0.8785	0.7549
NN 1H	Byte 1-Gram	0.9718	0.9503
LR	Opcode 1-Gram	0.9911	0.9867
NN 1H	Opcode 1-Gram	0.9932	0.9833
NN 2H	Opcode 1-Gram	0.9865	0.9764
LR	Opcode 2-Gram	0.9729	0.9518
NN 1H	Opcode 2-Gram	0.9857	0.9761
NN 2H	Opcode 2-Gram	0.9871	0.9782
LR	Opcode 3-Gram	0.9545	0.9075
NN 1H	Opcode 3-Gram	0.9758	0.9530
NN 2H	Opcode 3-Gram	0.9650	0.9415
D. Gibert et al. [18]	Grayscale images	0.9750	0.9400
J. Kim et al. [29]	Grayscale images	0.9639	–
D. Gibert et al. [15]	Structural Entropy	0.9828	0.9314
E. Raff et al. [1]	Bytes sequence	0.9641	0.8902
M. Krl et al. [14]	Bytes sequence	0.9756	0.9071
Q. Le et al. [17]	Bytes sequence	0.9820	0.9605
D. Gibert et al. [16]	Bytes sequence	0.9828	0.9636
N. McLaughlin et al. [12]	Opcodes sequence	0.9903	–
D. Gibert et al. [13]	Opcodes sequence	0.9913	0.9830
M. Mays et al. [30]	Grayscale images + Opcode N-grams	0.9770	–
Mnemonics subnetwork	Opcodes sequence	0.9893	0.9802
Bytes subnetwork	Bytes sequence	0.9885	0.9774
Bimodal network	Opcodes+Bytes sequences	**0.9924**	**0.9872**

TABLE III
NUMBER OF UNIQUE N-GRAMS IN THE MICROSOFT MALWARE CLASSIFICATION CHALLENGE DATASET.

	Bytes	Opcodes
1-gram	256	400
2-gram	65536	21036
3-gram	16777216	197442

byte sequence presented in this work allows to group some of the bytes per default, and provides some insights about their function to the network.

V. CONCLUSIONS AND FUTURE WORK

To the best of our knowledge, this research is the first application of multimodal deep learning for PE malware classification that uses intermediate fusion to merge features from various modalities of information. The multimodal approach combines two sources of information through a simple architecture, (1) the byte sequence representing malware's binary content and (2) the mnemonic sequence representing malware's assembly language source code. This architecture extracts n-gram like features from both input sequences to build a robust classifier than existing deep learning approaches. Experiments demonstrate that our model takes advantage of both modalities of information to perform significantly better than state-of-the-art methods on a standard benchmark, the Microsoft Malware Classification Challenge dataset.

A future line of research might be to explore more modalities of data to build a stronger malware classifier. More specifically, research on the combination of wide and deep models [31] to combine the strength of both approaches, memorization (wide models) and generalization (deep models).

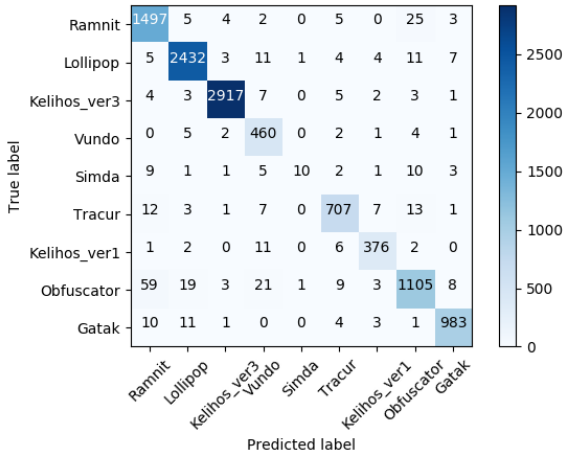

Fig. 7. E. Raff et al. [1] confusion matrix.

REFERENCES

[1] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole EXE," in *The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018.*, 2018, pp. 268–276. [Online]. Available: https://aaai.org/ocs/index.php/WS/AAAIW18/paper/view/16422

[2] D. Gibert, J. Béjar, C. Mateu, J. Planes, D. Solis, and R. Vicens, "Convolutional neural networks for classification of malware assembly code," in *Recent Advances in Artificial Intelligence Research and Development - Proceedings of the 20th International Conference of the Catalan Association for Artificial Intelligence, Deltebre, Terres de l'Ebre, Spain, October 25-27, 2017*, 2017, pp. 221–226. [Online]. Available: https://doi.org/10.3233/978-1-61499-806-8-221

[3] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft Malware Classification Challenge," *ArXiv e-prints*, Feb. 2018.

[4] S. Jain and Y. K. Meena, "Byte level n–gram analysis for malware detection," in *Computer Networks and Intelligent Computing*, K. R.

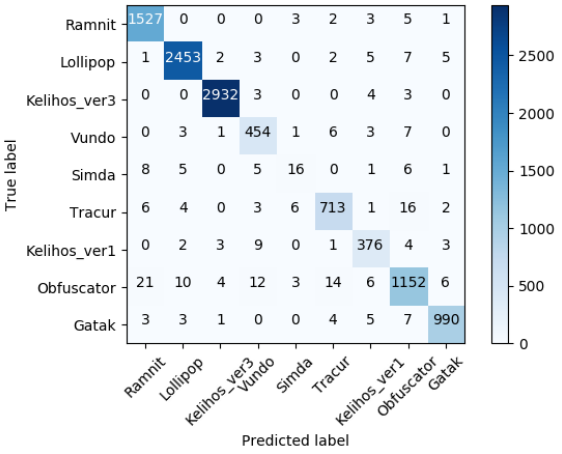

Fig. 8. M. Krl et al. [14] confusion matrix.

Venugopal and L. M. Patnaik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 51–59.

[5] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Information Sciences*, vol. 231, pp. 64–82, 2013, data Mining for Information Security. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025511004336

[6] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," *IEEE Security Privacy*, vol. 5, no. 2, pp. 40–45, March 2007.

[7] I. Sorokin, "Comparing files using structural entropy," *Journal in Computer Virology*, vol. 7, no. 4, p. 259, Jun 2011.

[8] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, ser. VizSec '11. New York, NY, USA: ACM, 2011, pp. 4:1–4:7.

[9] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "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*, ser. CODASPY '16. New York, NY, USA: ACM, 2016, pp. 183–194.

[10] B. N. Narayanan, O. Djaneye-Boundjou, and T. M. Kebede, "Performance analysis of machine learning and pattern recognition algorithms for malware classification," in *2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS)*, July 2016, pp. 338–342.

[11] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining api calls," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC '10. New York, NY, USA: ACM, 2010, pp. 1020–1025. [Online]. Available: http://doi.acm.org/10.1145/1774088.1774303

[12] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé, and G. Joon Ahn, "Deep android malware detection," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, ser. CODASPY '17. New York, NY, USA: ACM, 2017, pp. 301–308.

[13] D. Gibert, C. Mateu, and J. Planes, "A hierarchical convolutional neural network for malware classification," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019.

[14] M. Krl, O. vec, M. Blek, and O. Jaek, "Deep convolutional malware classifiers can learn from raw executables and labels only," in *Workshop in the Sixth International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=HkHrmM1PM

[15] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Classification of malware by using structural entropy on convolutional neural networks," in *The Thirtieth AAAI Conference on Innovative Applications of Artificial Intelligence (IAAI-18)*, 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16133

[16] D. Gibert, C. Mateu, and J. Planes, "An end-to-end deep learning architecture for classification ofmalware's binary content," in *Artificial Neural Networks and Machine Learning – ICANN 2018*, V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis, Eds. Cham: Springer International Publishing, 2018, pp. 383–391.

[17] Q. Le, O. Boydell, B. M. Namee, and M. Scanlon, "Deep learning at the shallow end: Malware classification for non-domain experts," *Digital Investigation*, vol. 26, pp. S118 – S126, 2018.

[18] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Using convolutional neural networks for classification of malware represented as images," *Journal of Computer Virology and Hacking Techniques*, Aug 2018. [Online]. Available: https://doi.org/10.1007/s11416-018-0323-0

[19] R. U. Khan, X. Zhang, and R. Kumar, "Analysis of resnet and googlenet models for malware detection," *Journal of Computer Virology and Hacking Techniques*, Aug 2018.

[20] B. Athiwaratkun and J. W. Stokes, "Malware classification with lstm and gru language models and a character-level cnn," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2017, pp. 2482–2486.

[21] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *AI 2016: Advances in Artificial Intelligence*, B. H. Kang and Q. Bai, Eds. Cham: Springer International Publishing, 2016, pp. 137–149.

[22] E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, A. Tracy, M. McLean, and C. Nicholas, "An investigation of byte n-gram features for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 1, pp. 1–20, Feb 2018. [Online]. Available: https://doi.org/10.1007/s11416-016-0283-1

[23] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, and Y. Elovici, "Unknown malcode detection via text categorization and the imbalance problem," in *2008 IEEE International Conference on Intelligence and Security Informatics*, June 2008, pp. 156–161.

[24] E. Raff and C. Nicholas, "Hash-grams: Faster n-gram features for classification and malware detection," in *Proceedings of the ACM Symposium on Document Engineering 2018*, ser. DocEng '18. New York, NY, USA: ACM, 2018, pp. 22:1–22:4. [Online]. Available: http://doi.acm.org/10.1145/3209280.3229085

[25] X. Hu, K. G. Shin, S. Bhatkar, and K. Griffin, "Mutantx-s: Scalable malware clustering based on static features," in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*. San Jose, CA: USENIX, 2013, pp. 187–198. [Online]. Available: https://www.usenix.org/conference/atc13/technical-sessions/presentation/hu

[26] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," *CoRR*, vol. abs/1803.04173, 2018. [Online]. Available: http://arxiv.org/abs/1803.04173

[27] M. Ahmadi, G. Giacinto, D. Ulyanov, S. Semenov, and M. Trofimov, "Novel feature extraction, selection and fusion for effective malware family classification," *CoRR*, vol. abs/1511.04317, 2015. [Online]. Available: http://arxiv.org/abs/1511.04317

[28] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *CoRR*, vol. abs/1511.07289, 2015. [Online]. Available: http://arxiv.org/abs/1511.07289

[29] J.-Y. Kim, S.-J. Bu, and S.-B. Cho, "Malware detection using deep transferred generative adversarial networks," in *Neural Information Processing*, D. Liu, S. Xie, Y. Li, D. Zhao, and E.-S. M. El-Alfy, Eds. Cham: Springer International Publishing, 2017, pp. 556–564.

[30] M. Mays, N. Drabinsky, and S. Brandle, "Feature selection for malware classification," in *Proceedings of the 28th Modern Artificial Intelligence and Cognitive Science Conference 2017, Fort Wayne, IN, USA, April 28-29, 2017.*, 2017, pp. 165–170.

[31] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "Wide & deep learning for recommender systems," in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, ser. DLRS 2016. New York, NY, USA: ACM, 2016, pp. 7–10. [Online]. Available: http://doi.acm.org/10.1145/2988450.2988454