End-to-end Multimodel Deep Learning for Malware Classification

Elijah Snow*, Mahbubul Alam†, Alexander Glandon†, Khan Iftekharuddin†
* University of Texas at Dallas (elijah.snow@utdallas.edu)
† Electrical and Computer Engineering, Old Dominion University (malam001, aglan001, kiftekha @odu.edu)

***Abstract*— Malicious software (malware) is designed to cause unwanted or destructive effects on computers. Since modern society is dependent on computers to function, malware has the potential to do untold damage. Therefore, developing techniques to effectively combat malware is critical. With the rise in popularity of polymorphic malware, conventional anti-malware techniques fail to keep up with the rate of emergence of new malware. This poses a major challenge towards developing an efficient and robust malware detection technique. One approach to overcoming this challenge is to classify new malware among families of known malware. Several machine learning methods have been proposed for solving the malware classification problem. However, these techniques rely on hand-engineered features extracted from malware data which may not be effective for classifying new malware. Deep learning models have shown paramount success for solving various classification tasks such as image and text classification. Recent deep learning techniques are capable of extracting features directly from the input data. Consequently, this paper proposes an end-to-end deep learning framework for multimodels (henceforth, multimodel learning) to solve the challenging malware classification problem. The proposed model utilizes three different deep neural network architectures to jointly learn meaningful features from different attributes of the malware data. End-to-end learning optimizes all processing steps simultaneously, which improves model accuracy and generalizability. The performance of the model is tested with the widely used and publicly available Microsoft Malware Challenge Dataset and is compared with the state-of-the-art deep learning-based malware classification pipeline. Our results suggest that the proposed model achieves comparable performance to the state-of-the-art methods while offering faster training using end-to-end multimodel learning.**

Keywords— Deep Learning, Convolutional Neural Network, Recurrent Neural Network, End-to-end learning, Multimodel, Malware Classification

I. INTRODUCTION

Malware is short for malicious software and refers to software whose purpose is to cause damage to a computer [1] for reasons including stealing or ransoming information, stealing processor power, or causing system failure [2]. Now that society relies heavily on computers, combating malware is very important. There are several current techniques for malware detection. Signature based detection is based on matching bytecodes over a known set of malicious signature bytecodes [3]. The signature method is fast but purely reactionary and unable to recognize new malware. Behavioral analysis executes the program in a controlled environment and observes it for malicious behavior, which is slow and risky [4]. Heuristic analysis uses features of a file to determine whether or not it is malware [5]. Heuristic analysis is limited by the effectiveness of its classification model, since it is designed by hand. Because deep learning can be used to create a classification model, using deep learning to extend heuristic analysis has the potential to drastically improve performance over heuristic analysis. Deep learning can surpass the weaknesses of hand-designed models by cutting down the time for model design and by using information that may be beyond the comprehension of the programmer.

Deep learning has already shown paramount success in various application domain such as computer vision, medical image analysis, autonomous driving, etc. Consequently, deep learning techniques are utilized for solving intricate cyber security problems such as malware classification. Wang et al. propose a malware classification technique which utilizes XGBoost. The ensemble outputs are combined using geometric mean and grid search to obtain a classification results. . The proposed technique is tested on the Microsoft Malware Classification Challenge (MMCC) dataset and achieves a top classification accuracy of 99.83% [6, 7]. This specific method ranked number one in the Microsoft Malware Classification challenge in Kaggle. The 2nd and 3rd place teams used similar methods, using random forests to determine feature importance [8, 9]. The downside of Wang et al.'s approach is that their model took two days to train [7]. Kalash et al. used a deep convolutional network and transfer learning from the VGG-16 model [10] to classify malware based on input conversion to grayscale image representation. They are able to achieve a 98.99% validation accuracy without using any other features of the malware, and are notably the only strong result that does not use ensembling to boost their accuracy [11]. Drew et al. approach the problem from a gene sequencing perspective and use ensembling method. Their classifier, Strand, is given the file's byte sequence and opcode sequence as input and is able to achieve 98.59% classification accuracy for the MMCC dataset [12]. Yan et al. also use an ensemble method with a deep convolutional network based on VGG-16 [10] in conjunction with a recurrent network composed of LSTM layers. They use truncated backpropagation through time and a data augmentation strategy based on a sliding window. The convolutional and recurrent networks are trained separately, and their results are combined with a second level classifier and achieves a classification accuracy of 99.36% [13].

The above mentioned models rely on using an ensemble of models or on using a pre-trained network as a starting point to

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

boost their accuracy. Conversely, this paper proposes an efficient end-to-end multimodel deep learning architecture for solving the malware classification task. End-to-end learning multimodel leverages automatic feature learning from the training data, eliminating the need for manual feature engineering. Moreover, our proposed method incorporates different deep learning architectures and jointly learns to capture relevant features from different meta information of the malware data. Additionally, our proposed method inherently achieves the benefit of ensembling due to the use of multiple deep learning models in a pipeline. Our proposed method shows competitive accuracy when compared to the state-of-the-art methods.

To evaluate the performance of our proposed method we use the challenging MMCC dataset which was publicly released under a contest run on Kaggle. MMCC is composed of 10868 labeled pieces of malware from 9 different classes, and 10873 unlabeled pieces of malware. The structure of this dataset is described in the testing methodology section of this paper and has been released to be used freely for research purposes [14]. The use of this dataset has made it more straightforward to compare different kinds of malware classification methods, including other state-of-the art work. Our results suggest that our end-to-end multimodel learning achieves accuracy comparable to other state-of-the-art deep learning models with very good training time. The model is able to achieve a best 4-fold classification accuracy of 99.23% in only 35 minutes of training.

The remaining of the paper is organized as follows. Section II covers the required background information for this model. Section III discusses the testing methodology. Section IV shows our results and comparison to other methods. Section V concludes and discusses future work.

II. Background

This section discusses the necessary background required to understand our proposed end-to-end multimodel learning malware classification technique.

A. Convolutional Neural Network

A Convolutional Neural Network (CNN) is a neural network architecture that is not fully connected. Any given node in a CNN is connected to a subset of the previous layer, rather than the entire layer as would be the case with a densely connected network. These connections are governed by the convolution operation. In convolution, a kernel, which is a matrix of weights and windows of the data of the previous layer at combined in a dot product. When this dot product is applied by sliding the window over a previous layer, this implements a convolution. Kernel weights are adjusted in the same way that a connection's weights would be adjusted during the backpropagation of dense network. A kernel may also be referred to as a filter. For example, with a 2-dimensional input, a kernel of size 3x3 would mean that each neuron in the output layer is connected to a 3x3 window of the previous layer. This technique takes advantage of the spatial relationship of the data and as a result is particularly useful for image processing. Typically, several kernels are applied in convolution to the same input, producing multiple outputs, called feature maps. For instance, an input of 128x128 with 32 kernels would produce 32 feature maps, resulting in an output of size 128x128x32. Convolutional layers are typically used in conjunction with pooling layers, which downsample a 2D input. With a 2x2 max-pooling layer, each output neuron is selected by taking the maximum value of a 2x2 area of the input, resulting in an output layer with dimensions that are half of the input dimensions. Other types of pooling are possible but less common [15, 16]. Unlike a convolutional layer, in pooling the areas of a layer that map to the next layer do not overlap. The convolution and pooling are described below.

$$Conv_{Out}(x,y) = \sum_{\Delta x}\sum_{\Delta y} Conv_{In}(x-\Delta x, y-\Delta y) Filter(\Delta x, \Delta y) \quad (1)$$

$$Pool_{Out}(x,y) = Max\begin{Bmatrix} Conv_{Out}(2x,2y), Conv_{Out}(2x,2y+1), \\ Conv_{Out}(2x+1,2y), Conv_{Out}(2x+1,2y+1) \end{Bmatrix} \quad (2)$$

B. Long Short Term Memory

LSTM (Long Short Term Memory) is a special form of recurrent neural network designed to remember data for longer periods of time than a normal recurrent neural network and was introduced by Hochreiter and Schmidhuber. LSTMs differ from regular recurrent neural networks in that they use "gates" to control what the network remembers more effectively. Neurons in a normal RNN use their previous value and an input to produce their new value. LSTMs decide what their state should be using a gate that controls the input value and a gate that controls the previous value. These gates allow for better control over what the network remembers. LSTMs also use an output gate to determine what parts of their stored value should be sent along to the next neuron. LSTM is suitable for sequences processing including sentences or code since their meanings are dependent on long range contextual information [17]. LSTM operation is governed as follows.

For X∈ {Input, Forget, Output}, W_x is a set of weights applied to outside input and U_x is a set of weights applied to recurrent input. it is the set of external inputs to the gate at time t and mt is the set of recursive inputs from time t. σ is the sigmoid function.

$$f_t = \sigma\left(W_f i_t + U_f m_{t-1}\right) \quad (3)$$

$$i_t = \sigma\left(W_i i_t + U_i m_{t-1}\right) \quad (4)$$

$$o_t = \sigma\left(W_o i_t + U_o m_{t-1}\right) \quad (5)$$

Each gate receives input from itself 1 time step ago and outside input, and takes their weighted sum. A constant bias may also be added to each value. The memory value c_t of the cell is calculated using these values in (6). Note that ∘ denotes an element-wise product.

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

$$c_t = c_{t-1} \circ f_t + i_t \circ tahn(W_c i_t + U_c m_{t-1}) \quad (6)$$

Output of the LSTM cell is as follows:

$$h_t = o_t \circ \tanh(c_t) \quad (7)$$

C. End-to-end Training

End-to-end is a technique where an entire model is trained simultaneously, rather than training its parts individually [18]. End-to-end is designed to reduce human interference in the training of a model and eliminate the need of a separate scheme to combine multiple trained networks. End-to-end models optimize all processing steps simultaneously, and this has been shown to improve performance of the models. A major principle of end-to-end training is that the model, not the programmer, decides what features are important. For example, Dieleman et al. [19] use end-to-end models to autonomously extract features from raw audio input for input representation understanding. They note certain invariants are preserved in the features that their model discover. End-to-end models also exhibit improved accuracy [18, 20] and generalizability of the model to more challenging datasets [21]. Also models that are trained end-to-end, are more compact with smaller number of parameters [18]. Typically, separate models are trained and combined in ensemble postprocessing [7, 13], but the need for this is subverted by training the model end-to-end. End-to-end training has been used primarily in image and language processing [22, 23] and a goal of this paper is to determine whether it can be applied successfully to the challenge of malware classification, and whether we can obtain competitive performance by doing so. The top three teams in the MMCC used Random Forest to determine important features [7-9], and end-to-end training in this paper serves as an alternative for determining important features without having to use any ensembling method, and without having to weigh the importance of different kinds of networks.

III. Methodology

A. End-to-end multimodel deep learning model

Our end-to-end model multimodel learning design is composed of three types of networks in parallel branches: a dense network, a CNN, and an RNN using LSTM. To best facilitate this end-to-end learning, our multimodel is provided with a wide array of features. The architectures are selected to efficiently handle different attributes of the data and to capture important features during training. Each of these neural network architectures works optimally for different kinds of data, and many kinds of data are available from a malware file. By using the different architectures in conjunction with each other, we seek to ensure that all the different kinds of data available from the malware can be used as input to the neural network architecture that is best suited for them. The structures of each type of network are quite simple and are expanded upon in detail in Table 1. In short, the dense network is composed of four fully connected layers, the convolutional network of three convolutional layers with pooling layers between them, and the recurrent network of one LSTM layer. The three network architectures are then connected in parallel and attached to four fully connected layers as shown in Fig. 1.

The model is constructed using Keras [24] with Tensorflow [25] as the backend and trained on a system using an Intel i7 6700k processor, Nvidia GTX 1070 GPU, 16GB of memory, and Windows 10. The model is trained with an Adam [26] optimizer minimizing training loss with learning rate 5*10^-4

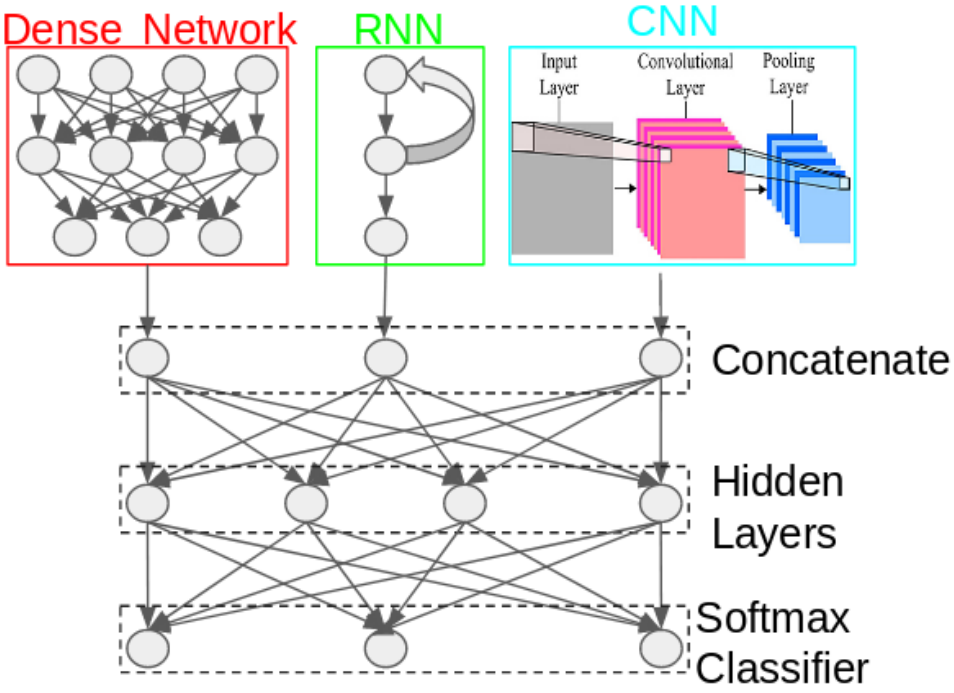

Figure 1. Architecture of the end-to-end multimodel deep learning based malware classification technique.

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

and a learning rate decay of 10^-6 per epoch. Training occurred in 20 epochs with a batch size of 16 at a mean of 105 seconds per epoch resulting in a total training time of 35 minutes. Algorithm 1 details the training and testing procedure for our model using Adam optimization and 4-fold cross-validation.

The layers that compose the model and their properties are detailed in Table 1. Dense(x,y) refers to a fully connected layer with x neurons, using y activation function. Conv(x,(y1x y2),(z1xz2)) refers to a convolutional layer producing x feature maps using a kernel of size (y1,y2) and stride size (z1,z2) with padding and ReLU activation in all instances. MaxPool refers to a max pooling operation with a pooling size of 2x2 and a stride size of 2x2. LSTM(x) refers to an LSTM layer with x neurons. For dropout [27], a standard dropout-rate of 0.5 is used in all cases. Any parameters not mentioned used Keras default values.

Algorithm 1: Training End-to-End Multimodel Learning using cross validation fold

1: Input: Metadata, 64x64 Byte Code Images, 60 step Op Code sequences

2: generate concatenated training vectors V_{All} from metadata

3: split vectors into 4 folds with 25% vectors removed to comprise each training set *T1,T2,T3,T4*

4: for *fold* = 1:4

5: $\alpha_1 = 5*10^{-4}$ **(set learning rate for initial epoch)**

6: for *epoch* = 1:20

7: for *batch* = 1:batches per fold

8: stochastically select batch V_{batch} from V_{All}

9: $g_{batch} = \nabla_\theta f(V_{batch}, \theta_{batch-1})$
calculate end-to-end gradient w.r.t. to all parameters for given batch V_{batch}

10: calculate Adam "bias-corrected" moments m_{batch} (first moment) and v_{batch} (second moment)

11: $\theta_{fold,\ batch} = \theta_{fold,\ batch-1} - \alpha_{epoch} \cdot m_{batch}/(\sqrt{\boldsymbol{v_{batch}}} + \varepsilon)$
update weights with using moments
(ε divide-by-zero guard)

12: $\alpha_{epoch+1} = (1\text{-}10^{-6}) \cdot \alpha_{epoch}$ (decay learning rate)

13: Output: θ_{fold} (trained weights for end-to-end multimodel for each fold)

TABLE I. DETAILED ARCHITECTURE OF THE PROPOSED MODEL

Dense Network	Convolutional Network
Dense(4096,ReLU) Dropout Dense(4096,ReLU) Dropout Dense(4096,ReLU) Dropout Dense(9,softmax)	Conv(32,(5x 5),(2x2)) MaxPool Conv(64,(5x 5),(1x1)) MaxPool Conv(96,(3x 3),(1x1)) MaxPool Dropout
Recurrent Network	**End Layers**
LSTM(64) Dropout	Dense(4096,ReLU) Dropout Dense(4096,ReLU) Dropout Dense(1024,ReLU) Dropout Dense(9,softmax)

B. MMCC Data description and pre-processing

The MMCC dataset is composed of 10868 pieces of malware that are labeled in accordance to membership in one of nine malware classes. Additionally, the set contains 10873 unlabeled pieces of malware, but they are not used for this work. For each piece of malware in the dataset, two files are provided. First, for each instance of malware there is a .bytes file containing the bytecode of the malware executable, with the portable executable header removed so that the malware cannot run and contaminate the system. Second, a .asm file is provided, having been generated by the IDA decompiler [23], that contains metadata about the malware such as its starting address and the opcodes it calls [14]. For training our model, the following metadata information from the files is collected.

- The sizes of the two files
- The length of the different sections of the .asm file
- Opcode 1, 2, and 3-grams
- Byte 4-grams

This metadata information is used as input to the dense portion of the network. Because the number of possible n-grams is very large, less frequent examples are ignored. More specifically, for opcode 1-grams, the 66 most common are used, and the 50 most common opcode 2-grams and 3-grams are used. For the byte 4-grams, the 100 most common are used. These values are combined into a single vector for use as input to the model, and the choice of metadata is intentionally kept simple; we rely on the deep model to extract the features that provide power in classification.

For input to the convolutional portion of the network, grayscale image conversions of the malware files are used as input. The creation of grayscale images from the malware is based on the concept that a byte can be interpreted as the intensity of a pixel, and the conversion is simple. For each byte in the bytecode of a piece of malware, the intensity of the corresponding pixel in the output image is set to match it. Some

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

of the bytecode files contain '??' in place of some bytes, and these are interpreted as a value of 0 (a black pixel). Interestingly, there are images embedded in some of the malware files which become visible after this conversion. Each image is resized to 64x64 using bicubic subsampling to decrease computation time. For data augmentation, a random shear of up to 10 degrees and a random shift of up to 64 pixels are used before the image is given to the model. Example bytecode images are shown in Fig 2.

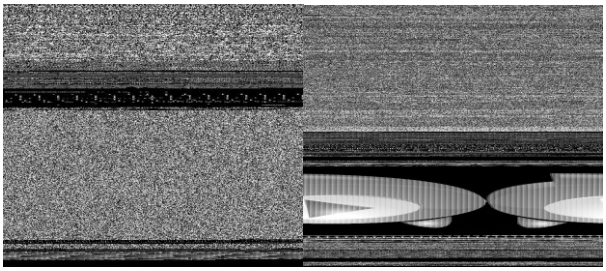

Fig. 2. Example grayscale images from the MMCC dataset

To obtain the input to the RNN, the first 60 opcodes in each file are used. There appears to be little or no improvement in accuracy for a higher number of opcodes, despite an increase in run time. However, lowering the number of opcodes used did decrease the accuracy. Additionally, using the first 60 opcodes in a file showed better results than opcodes from arbitrary locations. For files with fewer than 60 opcodes, a special "zero" opcode is inserted until there are 60. Opcodes are suitable for use with LSTM because they occur in an order corresponding to an execution sequence. Embedding is used to make the data formatted suitably for input to LSTM. No further adjustments are made to the data.

IV. Results and Discussion

We performed the following malware classification experiment with our end-to-end multimodel deep learning technique. For each malware example corresponding to a bytecode and metadata/opcode file pair, we extract 3 raw data structures. These include first the grayscale image matrix from bytecodes, second the opcode sequence, and third the metadata concatenated into a vector. This feature information is compiled into a dataset including associated malware family target class labels per each example. Then we perform 4-fold cross validation given the features and labels. This involves splitting vectors into 4 folds with 25% vectors removed to comprise each training set T1, T2, T3, T4 as described in algorithm 1, line 3. Then for each fold, Adam optimization is performed as described in algorithm 1, lines 6-12. Adam optimization is an advanced modification of gradient descent learning and updates the parameters for the end-to-end multimodel for each fold. After training, each fold is tested, and the accuracies for each fold are saved. Average and best accuracy are considered as discussed below and compared to results from literature.

The model achieves a mean fold accuracy of 98.35% and a best validation accuracy of 99.23% as shown in Table 2. Extraction of features from the given files takes a mean of 1.674 seconds and varies proportionately with the size of the files. Once the model is constructed, using it to classify a new malware input takes about 0.03 seconds. Some works dealing with the MMCC dataset use mean cross-validation classification accuracy to measure performance, and some simply use some of the training data for validation to test accuracy. Since using some of the training data for validation is equivalent to a single fold out of a k-fold cross-validation, we use our best single fold result for accurate comparison to these works, and mean cross-validation accuracy for comparison to those who tested their own accuracy with cross-validation. The confusion matrix for our 4-fold cross-validation is shown in Fig. 3.

TABLE II. 4-Fold cross-validation accuracy of the end-to-end multimodel Learning using the MMCC dataset

Fold 1	Fold 2	Fold 3	Fold 4	Mean Accuracy	Best Accuracy
98.21%	98.29%	97.67%	99.23%	98.35%	99.23%

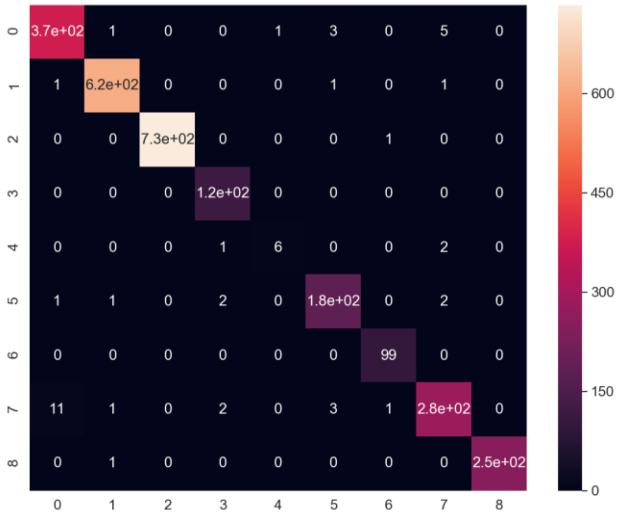

Fig. 3. Confusion matrix of classification results averaged from 4 folds.

We compared our method to other works trained and tested on MMCC dataset. With respect to heavily feature engineered works like Wang et al. [7], our approach has lower classification accuracy but runs far faster. Their model takes about 2 days to generate [7]. Our model can be generated in 5 hours of feature extraction about 30 minutes of training. Since their feature engineering is heavily tailored to the dataset, new data would not only take longer to feed through their model but would also likely have lower accuracy than the current dataset. Ahmadi et al. apply expert knowledge to more efficiently choose features

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

compared to Wang et al. and achieve comparable results in a shorter training time [28].

Our model slightly outperforms Kalash et al. [11] who achieve an accuracy of 98.99% using 90% of the MMCC dataset for training and 10% for validation when compared to our 99.23% accuracy. Their model utilizes a deep convolutional network, which is the only other non-ensembled method mentioned here and is notable for using only a grayscale image conversion of the malware [11]. Their method however uses a pretrained VGG network for initial layers. Our model foregoes ensembling and use of pretrained networks, which all the other models here (besides that of Kalash et al.) use to boost their accuracy. Again, Drew et al. [12] with their gene sequencing style model is able to achieve better accuracy compared to our method, however, requires more training time and utilizes manual ensemble technique. Table 3 below summarizes the comparison of our method to other state-of-the-art malware classification methods.

Yan et al. achieved a 99.36% classification accuracy on the training data using 90% of their dataset for training and 10% for validation. Their dataset is composed of the Microsoft Malware Classification Challenge dataset plus a set of benign files of around the same size, and 90% of this dataset is used for training. Their accuracy score is comparable to our score of 99.23% classification accuracy using a 75% of the MMCC dataset for training and 25% for validation. Classification using their model and ours takes 0.03 seconds, but ours uses features that take longer to extract. They utilize a special data augmentation strategy to balance out the representation of each class [13] since the classes are not equally represented in the training data. For example, one particular class comprises only 41 out of the 10868 pieces of malware in the MMCC training set [14].

Our model has two benefits that allow it to generalize to other datasets. First, our multimodel learning uses covers the widest array of features that will apply to a different dataset as opposed to a model which uses only a narrow scope of features. Second, the end-to-end training avoids excessive tailoring of a model to a dataset and should be resilient to a generalization in dataset as noted in [21] for end-to-end learning. Approaches like those of Wang et al. [7] and Ahmadi et al. [28] which rely heavily on feature engineering are not suitable to a change in dataset because they are designed with the unique elements of the MMCC dataset in mind. Yan et al.'s [13] approach uses a similar breadth of features but in the event that retraining is necessary for a new dataset their model would be less appropriate since multiple networks must be trained separately versus just one for our model. Retraining will be necessary if a dataset has different malware families than the MMCC dataset, but if the new dataset has the same families of malware, then our model should still have excellent performance even without retraining.

TABLE III. COMPARISON OF OUR PROPOSED END-TO-END MULTIMODEL DEEP LEARNING PIPELINE WITH STATE-OF-THE-ART MODELS

	Validation Accuracy	Training Time (Hours)	Ensembling	Measurement Scheme
End-to-end multimodel (Our proposed)	98.35%	0.58	No	4-fold CV
End-to-end multimodel (Our proposed)	99.23%	0.58	No	75% training 25% validation
Wang et al.[7]	99.83%	48	Yes	4-fold CV
Ahmadi et al.[28]	99.77%	N/A	Yes	5-fold CV
Yan et al.[13]	99.36%	2.91	Yes	90% training 10% validation
Kalash et al.[11]	98.99%	N/A	No	90% training 10% validation
Drew et al.[12]	98.59%	0.75	Yes	10-fold CV

V. CONCLUSION

This paper proposes end-to-end multimodel deep learning for malware classification. We demonstrate that end-to-end trained models are viable for malware classification and that they achieve competitive performance with fast training. End-to-end learning is useful for improving our model's accuracy and generalizability. Our proposed end-to-end model incorporates three different types of deep neural network architectures to capture diverse features from the meta information of the malware data. We conduct a cross validation experiment and achieve an average 98.35% classification accuracy with 4-fold cross-validation using the MMCC dataset which is comparable to the state-of-the-art methods trained on the MMCC dataset. Moreover, our method shows significant training time improvement when compared to the state-of-the-art methods trained using the MMCC dataset.

Our future plan is to deploy our model in a live cyber environment to study its performance in real time. Furthermore, we plan to generalize the proposed multimodel learning architecture for different types of multimodal data including features extracted from malware metadata, byte codes, and system calls.

VI. ACKNOWLEDGEMENT

This work is partially supported by NSF under grant CNS-1659795.

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

REFERENCES

[1] R. Moir. "Defining Malware: FAQ." https://technet.microsoft.com/en-us/library/dd632948.aspx (accessed 7/23/18.

[2] M. J. Schwartz. "Cryptojacking Displaces Ransomware as Top Malware Threat." https://www.bankinfosecurity.com/cryptojacking-displaces-ransomware-as-top-malware-threat-a-11165 (accessed 7/23/18.

[3] I. Santos, Y. K. Penya, J. Devesa, and P. G. Bringas, "N-grams-based File Signatures for Malware Detection," ICEIS (2), vol. 9, pp. 317-320, 2009.

[4] I. Firdausi, A. Erwin, and A. S. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," in 2010 second international conference on advances in computing, control, and telecommunication technologies, 2010: IEEE, pp. 201-203.

[5] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, "A survey on heuristic malware detection techniques," in The 5th Conference on Information and Knowledge Technology, 28-30 May 2013 2013, pp. 113-120, doi: 10.1109/IKT.2013.6620049.

[6] K. Team, "Microsoft Malware Winners' Interview: 1st place, "NO to overfitting!"," No Free Hunch, 2015.

[7] X. Wang, J. Liu, and X. Chen, "Microsoft Malware Classification Challenge (BIG 2015) First Place Team: Say No to Overfitting," 2015. [Online]. Available: https://github.com/xiaozhouwang/kaggle_Microsoft_Malware/blob/master/Saynotooverfitting.pdf.

[8] M. Trofimov, D. Ulyanov, and S. Semenov, "Microsoft Malware Classification Challenge third place solution," 2015. [Online]. Available: https://github.com/geffy/kaggle-malware/blob/master/description.pdf.

[9] M. Michailidis and G. Jacobusse, "Microsoft Malware Classification Challenge 2nd place solution documentation," 2015. [Online]. Available: https://kaggle2.blob.core.windows.net/forum-message-attachments/75478/2389/documentation.pdf.

[10] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," CoRR, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556.

[11] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal, "Malware Classification with Deep Convolutional Neural Networks," in 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 26-28 Feb. 2018 2018, pp. 1-5, doi: 10.1109/NTMS.2018.8328749.

[12] J. Drew, M. Hahsler, and T. Moore, "Polymorphic malware detection using sequence classification methods and ensembles," EURASIP Journal on Information Security, journal article vol. 2017, no. 1, p. 2, January 23 2017, doi: 10.1186/s13635-017-0055-6.

[13] J. Yan, Y. Qi, and Q. Rao, "Detecting Malware with an Ensemble Method Based on Deep Neural Network," Security and Communication Networks, vol. 2018, p. 16, 2018, Art no. 7247095, doi: 10.1155/2018/7247095.

[14] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft Malware Classification Challenge," CoRR, vol. abs/1802.10135, 2018. [Online]. Available: http://arxiv.org/abs/1802.10135.

[15] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in The handbook of brain theory and neural networks, A. A. Michael Ed.: MIT Press, 1998, sec. 303704, pp. 255-258.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," presented at the Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, Lake Tahoe, Nevada, 2012.

[17] S. Hochreiter, J\, \#252, and r. Schmidhuber, "Long short-term memory," Neural Comput., vol. 9, no. 9, pp. 1735-1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[18] M. Bojarski et al., "End to End Learning for Self-Driving Cars," CoRR, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316.

[19] S. Dieleman and B. Schrauwen, "End-to-end learning for music audio," in 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014: IEEE, pp. 6964-6968.

[20] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4490-4499.

[21] D. Amodei et al., "Deep speech 2: End-to-end speech recognition in english and mandarin," in International conference on machine learning, 2016, pp. 173-182.

[22] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," presented at the Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, Beijing, China, 2014.

[23] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification," ACM Trans. Graph., vol. 35, no. 4, pp. 1-11, 2016, doi: 10.1145/2897824.2925974.

[24] F. Cholet, "Keras," 2015. [Online]. Available: https://keras.io.

[25] Mart et al., "TensorFlow: a system for large-scale machine learning," presented at the Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation, Savannah, GA, USA, 2016.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," The journal of machine learning research, vol. 15, no. 1, pp. 1929-1958, 2014.

[28] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification," presented at the Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, Louisiana, USA, 2016.

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