

An ensemble approach for imbalanced multiclass malware classification using 1D-CNN

Binayak Panda¹, Sudhanshu Bisoyi^{Corresp., 2}, Sidhanta Panigrahy³

¹ Department of Computer Science and Engineering, Institute of Technical Education and Research, Siksha 'O' Anusandhan (Deemed to be) University, Bhubaneswar-751030, Odisha, India

² Department of Computer Science and Information Technology, Institute of Technical Education and Research, Siksha 'O' Anusandhan (Deemed to be) University, Bhubaneswar-751030, Odisha, India

³ Haas School of Business, University of California, Berkeley, 2220 Piedmont Ave, Berkeley CA 94720, United States of Amerika

Corresponding Author: Sudhanshu Bisoyi
Email address: sudhanshu.bisoyi@gmail.com

Dependence on the internet and computer programs demonstrates the significance of computer programs in our day-to-day lives. Such demands motivate malware developers to create more malware, both in terms of quantity and variety. Researchers are constantly faced with hurdles while attempting to protect themselves from potential hazards and risks due to malware authors' usage of code obfuscation techniques. Metamorphic and polymorphic variations are easily able to elude the widely utilized signature-based detection procedures. Researchers are more interested in deep learning approaches than machine learning techniques to analyze the behavior of such a vast number of virus variants. Researchers have been drawn to the categorization of malware within itself in addition to the classification of malware against benign programs to examine the behavioral differences between them. In order to investigate the relationship between API calls throughout API sequences and classify them, this work uses the CNN model to solve a multiclass classification problem. On API sequences, feature vectors for distinctive APIs are created using the Word2Vec word embedding approach and the skip-gram model. The One-vs.-rest approach is used to train 1D-CNN (One-Dimensional Convolutional Neural Network) models to categorize malware, and all of them are then combined with a suggested ModifiedSoftVoting algorithm to improve classification. On the benchmark dataset Mal-API-2019 released by Ferhat Ozgur et al. (2020), the suggested ensembled 1D-CNN architecture captures improved evaluation scores with an accuracy of 0.90, a weighted average F1-score of 0.90, and an AUC score of more than 0.96 for all classes of malware.

An Ensemble approach for imbalanced Multiclass Malware Classification using 1D-CNN

Binayak Panda¹, Sudhanshu Shekhar Bisoyi^{2*}, and Sidhanta Panigrahy³

¹Department of Computer Science and Engineering, Institute of Technical Education and Research, Siksha 'O' Anusandhan (Deemed to be) University, Bhubaneswar-751030, Odisha, India

²Department of Computer Science and Information Technology, Institute of Technical Education and Research, Siksha 'O' Anusandhan (Deemed to be) University, Bhubaneswar-751030, Odisha, India

³Haas School of Business, UC Berkeley, 2220 Piedmont Ave, Berkeley, CA 94720, USA

Corresponding author:
Sudhanshu Shekhar Bisoyi²

Email address: sudhanshu.bisoyi@gmail.com

ABSTRACT

Dependence on the internet and computer programs demonstrates the significance of computer programs in our day-to-day lives. Such demands motivate malware developers to create more malware, both in terms of quantity and variety. Researchers are constantly faced with hurdles while attempting to protect themselves from potential hazards and risks due to malware authors' usage of code obfuscation techniques. Metamorphic and polymorphic variations are easily able to elude the widely utilized signature-based detection procedures. Researchers are more interested in deep learning approaches than machine learning techniques to analyze the behavior of such a vast number of virus variants. Researchers have been drawn to the categorization of malware within itself in addition to the classification of malware against benign programs to examine the behavioral differences between them. In order to investigate the relationship between API calls throughout API sequences and classify them, this work uses the CNN model to solve a multiclass classification problem. On API sequences, feature vectors for distinctive APIs are created using the Word2Vec word embedding approach and the skip-gram model. The One-vs.-rest approach is used to train 1D-CNN (One-Dimensional Convolutional Neural Network) models to categorize malware, and all of them are then combined with a suggested ModifiedSoftVoting algorithm to improve classification. On the benchmark dataset *Mal-API-2019* released by Ferhat Ozgur et al. (2020), the suggested ensemble 1D-CNN architecture captures improved evaluation scores with an accuracy of 0.90, a weighted average F_1 -score of 0.90, and an AUC score of more than 0.96 for all classes of malware.

Keywords: Malware Classification, Dynamic Analysis, API Sequence, 1D-CNN, Skip-gram, Ensemble Learning.

INTRODUCTION

Information technology has a significant impact on our daily lives in the modern day. People of all ages use e-commerce, e-banking, e-healthcare, and other online services frequently to meet their everyday needs as the internet's accessibility has increased. Due to the sheer number of people who are exposed to the internet, malicious coders are motivated to explore all avenues for emotionally and financially exploiting victims. Malware (a malicious application) is the main tool used in cyberattacks to carry out hostile actions on the computers of targeted or harmed users (Aslan and Samet, 2020).

The number and variety of malicious programs continue to grow, posing an ongoing challenge to antimalware vendors and researchers. According to statistics from the Kaspersky Security Bulletin Report 2021, there were 64,559,357 distinct malicious programs discovered between November 2020 and October 2021, consisting 20 different categories such as Backdoor, Trojan, etc. (kas, 2021). According to

SonicWall's cyber threat analysis, there are now 5.5 billion known instances of malware, growing 2% annually by 2022 (Son, 2023). It would be accurate to claim that the number of malware programs is increasing tremendously, but not the variety. Again, the resulting harms to people and businesses are getting worse every day. Therefore, research aimed at classifying malware can help in malware detection and mitigation.

The most popular method of malware detection is signature-based detection, which involves searching for a certain signature in a previously built signature store in order to label a program as malicious (Shijo and Salim, 2015). The majority of anti-malware vendors employ this approach, which stores signatures of previously identified malware for malware detection, albeit the signature store may be updated often. However, it is possible for both newly obfuscated malware and previously identified malware to go undetected (Ucci et al., 2019). The other way for identifying malware by looking at the execution time parameters and related behaviors is behavior-based detection. The identification of both old and new unknown malware is better with behavior-based method than with signature-based method, albeit there may be a time-space trade-off between the two (Gibert et al., 2020). In these situations, the researchers are compelled to look for rational, practical, and cutting-edge methods in order to identify unidentified malware. A behavioral study using several machine learning algorithms has demonstrated the potential for improving malware identification and classification (Ucci et al., 2019; Tayyab et al., 2022). Compared to machine learning algorithms Deep learning models are becoming more and more popular, but the time and resource requirements for model training continue to be a problem (Liu et al., 2017; Alom et al., 2019). Deep learning also performs well in the domain of information security, in addition to applications for image analysis and language processing (Tekerek, 2021).

In this study, a multiclass classification problem is addressed utilizing an imbalanced data set, and the characteristics used to examine the categorization of each form of malware are the API sequences from various malware classes. To combine the output of various 1D-CNN classifiers trained using the One-vs-rest principle, a 1D-CNN based ensembled architecture is presented. For training and testing purposes, the data set *Mal-API-2019* published by Ferhat Ferhat Ozgur et al. (2020) is used.

The remainder of the paper's contents are organized as follows: Works by other researchers that are connected to the study are discussed in related work. The preprocessing of the data set, followed by the algorithm, and other specifics of the suggested design are explained in the ensembled 1D-CNN architecture section. Comparison of the results and performance-related graphs are shown in the experimental setup and results section. The conclusions are highlighted in the final section.

RELATED WORK

The discipline of malware analysis has benefited from the work of numerous researchers. For this area of study, there is an enormous amount of literature. The suggested method uses CNN to classify malware into multiple classes. The relevant literature is examined in relation to the classification of malware into several families using CNN and other techniques.

Vinod et al. (2010) performed dynamic analysis on four types of metamorphic malware and produced malware signatures tracing their API sequences. In their experimental setting, they employed the API sequences of a total of 80 viruses from 4 families and 20 benign programs. They have calculated the degree of membership of a malware to a malware class using the Chi-Square test. They achieved accuracy of 80%, 80%, 75%, and 75% for the respective families G2, MPCGEN, IL SMG, and NGVCK using this method. Additionally, they suggested that greater precision may be attained by increasing the number of samples. The method of signature matching is vulnerable to newly discovered malware samples.

Control Flow Graphs and API call graphs were extracted by Mehra et al. (2015) using 600 instances of malware and 150 benign samples. The desired features have been extracted from API call graphs using their suggested Gourmand Feature Selection technique. They performed classification using the WEKA tool and achieved accuracy of 89%, 92.24%, 94.56%, 99.10%, and 91.08%, respectively, using the KNN, VP, NB, J-48, and SMO classifiers. They didn't parse API sequences sequentially; instead, they exclusively used portable executables.

Zhang et al. (2016) suggested a simple malware classification system using ensemble learning, using data from the Microsoft malware classification challenge of kaggle(big 2015). It successfully assigned malware samples from the unbalanced training dataset to the appropriate family. Kolosnjaji et al. (2016) considered system call sequences to classify malware. They have extracted best features using

convolutional and recurrent network layers. They have achieved average precision of 85.6% and recall of 89.4% using this hybrid neural network architecture.

Han et al. (2019) used the TF-IDF technique to examine the relationship between API calls in API sequence on 807 benign and 3027 malicious (both packed and unpacked variation) samples. In order to identify and categorize malware, they applied the machine learning techniques Random Forest, Decision Tree, KNN, and XGBoost on their developed explainable malware detection framework (MalDAE). They used static, dynamic, and fused API sequences. They were able to reach accuracy of 84.96%, 79.65%, 74.74%, and 83.15% using dynamic API sequence, which is better than static API sequence. However, with fused API sequence, the accuracy increased to 94.39%, 88.42%, 85.26%, and 93.33%, respectively. Due to the difficulty in dealing with the large number of malware variations, they claimed that Deep Learning approaches could increase productivity. Using system call sequences from malicious and benign Android applications, Xiao et al. (2019) showed high recall of 96.6% and low FPR of 9.3%.

A data set of 7107 API sequences for 8 different classes of malware was produced and published by Ferhat Ozgur et al. (2020). In order to do multiclass classification, they have additionally trained Single Layer LSTM, Two Layers LSTM, DT, KNN, RF, and SVM on the data set. In comparison to all other models, they achieved Recall and Precision of 0.47 using Single-Layer LSTM. Li and Zheng (2021) used this dataset to classify malware utilizing API calls using LSTM and GRU (Gated Recurrent Unit) models. They have achieved recall of 0.58 and 0.59 for LSTM and GRU, respectively, with precision of 0.56 for both approaches. Demirkiran et al. (2022) classified malware families using the same benchmark dataset as Ferhat Ozgur et al. (2020). They have contrasted the Transformer, CANINE-S, and BERT models with their proposed model RTF. With an F1-score of 0.61 and an AUC score of 0.88, the RTF model outperformed all other models.

Vasan et al. (2020) performed image-based malware classification using an ensemble of pretrained CNN models, VGG16 and ResNet-50, and the open dataset Maling. Comparing the ensembling approach to conventional ML-based models, it demonstrated excellent accuracy. To solve the uneven size of the malware files of the employed data sets BIG2015 and DumpWare10, Tekerek (2021) created a method called CycleGAN and the B2IMG method to convert binaries to images. Their experimental findings demonstrate greater accuracy compared to other CNN-based algorithms. Catak FO (2021) created images from binaries using their dataset. The final dataset is applied to CNN for classification after dataset enhancement with additive noise approaches and picture augmentation. According to experimental findings, a dataset that includes noise factor has a classification accuracy of 0.96 for malware classes, which is higher than the accuracy of 0.83 for a dataset that does not include noise factor. By using RNN and CNN, Sun and Qian (2021) have performed static analysis of the visual malware images used in their RMVC approach. Even with a small training dataset, they discovered accuracy greater than 92%. They mentioned testing their method's efficacy in dynamic analysis as a potential future direction. Using malware images made from malware binaries, Hammad et al. (2022) performed malware classification. The Maling dataset is used to train and evaluate KNN, SVM, and ELM (Extreme Learning Classifier) models. Features are retrieved using GoogleNet (a deep learning model) and Tamuar (a texture feature that correlates to human visual perception). They discovered that ELM performed better than any other model. They have recommended that data augmentation be used, which could enhance classification outcomes.

The Word2Vec embedding technique with the Skip-gram model is used in the proposed ensembled 1D-CNN architecture to look into the semantic relationships between APIs in API sequences. A few 1D-CNN classifiers are trained on the Mal-API-2019 utilizing the One-vs-rest notion for classifying the various classes of malware. A suggested ensembling method combines each of these results to explore efficiency improvement.

METHODOLOGY

Proposed Ensembled 1D-CNN Architecture

In this study, the suggested architecture shown in Figure 1 ensemble eight independently trained 1D-CNN models to address the multiclass malware classification problem. The aforementioned architecture has three phases and conducts training and testing using the *Mal-API-2019* data set. The dataset is vectorized in the first phase, independent 1D-CNN models are trained as One-vs-rest classifiers in the second phase, and in the final phase, an ensembled model using the *ModifiedSoftVoting* algorithm is created to address the multiclass malware classification problem.

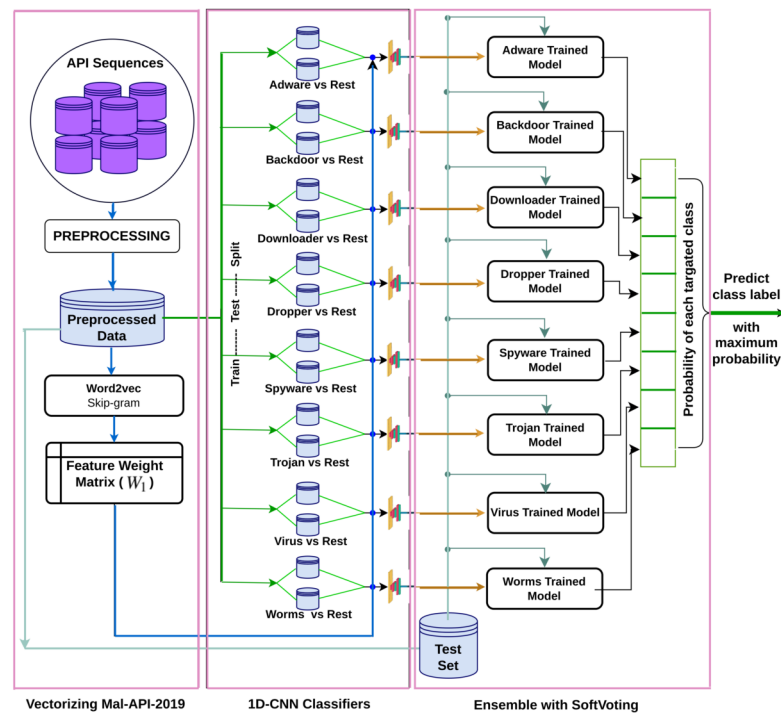


Figure 1. Proposed Ensembled 1D-CNN Architecture

Phase-I: Vectorizing Mal-API-2019

Dataset Description: The Mal-API-2019 consists of API sequence records of 7107 malwares of 8 different classes. The Figure 2 speaks about the frequency distribution of each class and Figure 3 shows the frequency of malware samples considering the API sequence length before preprocessing.

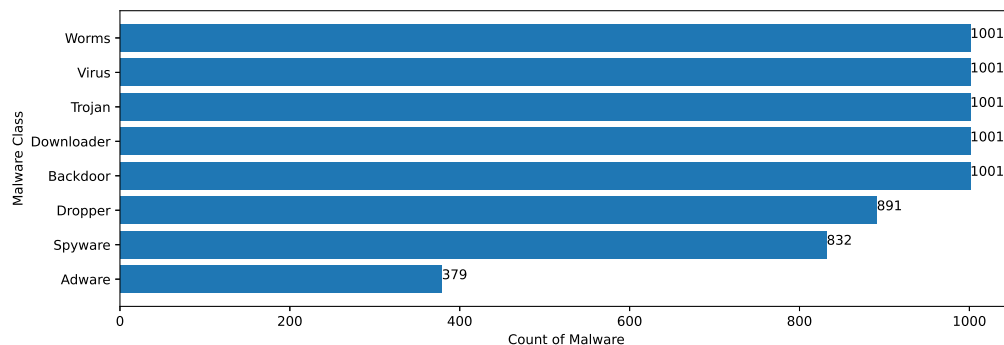


Figure 2. Malware Distribution Class Wise

The Figure 2 and Figure 3 shows that Mal-API-2019 is highly imbalanced. Such diverse distribution has adverse effect on the success of classification. The said dataset is interpreted as having two attributes named API_{seq} and M_{class} . The former represents the sequence of APIs called by the malware and the latter represents the class of that malware. Each entry in API_{seq} is interpreted as a sentence of type M_{class} made up of finite number of words i.e. APIs with repetitions from the $API_{Vocabulary} = \{API_0, API_1, API_2, \dots, API_n\}$. The M_{class} ranges from 0 to 7 representing 8 different type of sentences mapping to 8 malware classes. For the representation of textual documents in a multidimensional vector space, the vector space model is particularly popular. In one of the previous work TF-IDF vectorization technique has been used on DLL sequences for host based anomaly detection (Panda and Tripathy, 2020). In a comparable manner, word embedding vector of each distinct API of $API_{Vocabulary}$ is created using Skip-gram model of Word2Vec embedding technique.

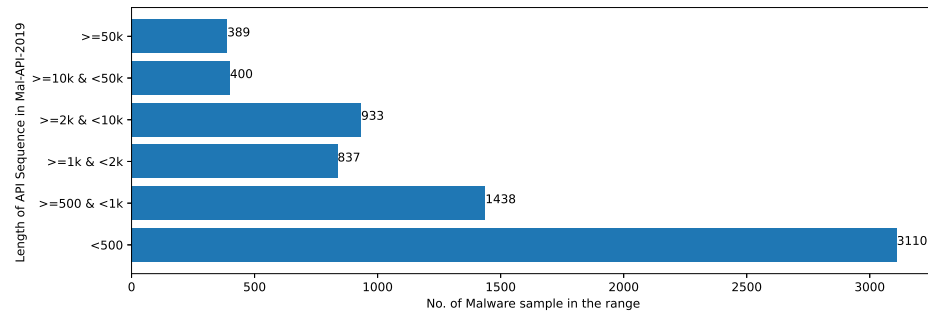


Figure 3. Sample Distribution before Preprocessing

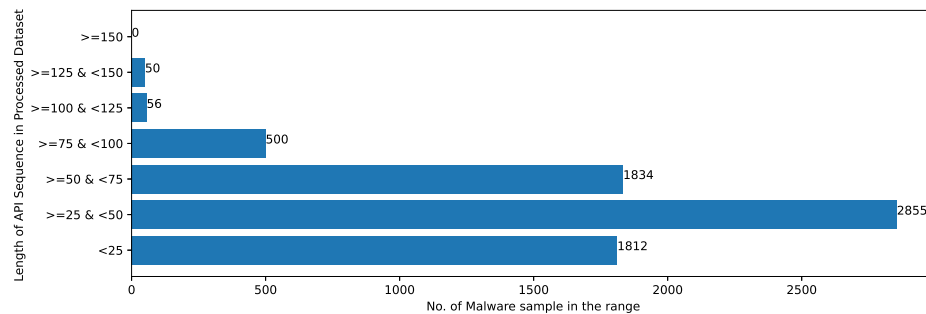


Figure 4. Sample Distribution after Preprocessing

Finding Word Embeddings for each API The word embeddings are discovered using Skip-gram by taking into account all the records of *API_{seq}* for each unique API. Skip-gram does well with small datasets and depicts less common words better than more frequent ones (Mikolov et al., 2013). With the aim of capturing the embeddings (feature vectors) of each unique API that qualify the meaning of the API they represent, a skip-gram neural network model, as shown in Figure 5, is utilized. Such feature vectors can be quite useful in describing the *Malclass* type. For each target API, the skip-gram technique takes into account all windows of size ℓ in order to extract the semantics of the APIs into embeddings. The embedding matrix W_1 shown in Figure 5, which is given to the 1D-CNN models for training purposes in Phase-II of the proposed architecture, serves as the word embeddings for all different APIs. The weight matrix W_2 can be employed to predict the likelihood of various words given a context word. Since the main goal of this effort is to obtain the word embeddings, W_2 is not utilized. The redundant API occurrences are eliminated in order to address the dataset's variability with regard to the length of each entry. The distribution of records in the Processed Dataset against record length following duplicate removal is shown in Figure 4.

Phase-II: Training 1D-CNN Models

Due to their effectiveness, deep learning models are becoming more and more popular. Convolution Layer, Pooling Layer, and Fully Connected Artificial Neural Network Layer (Dense Layer) are the three basic layers that make up the One Dimensional Convolutional Neural Network (1D-CNN), a deep learning model. These models use convolution and pooling operations to learn features from sequential input, such as texts, and then conduct binary or multiclass classification in the dense layer. It performs the convolution operation with various kernels on the spatial input texts in the convolution layer to produce corresponding one-dimensional feature maps.

The operation of the one-dimensional convolution layer is as described in Eq.1.

$$x_j^l = f \left(\sum_{i=1}^M x_i^{l-1} * k_{ij}^l + b_j^l \right) \quad (1)$$

where k and j represents convolution kernels and number of kernels respectively. M denotes channel

number in input x^{l-1} with b as the bias to the corresponding kernel. The $*$ is the convolution operator and $f()$ is the activation function. The Pooling layer uses the average pooling or max pooling method with a predetermined window to reduce the feature dimension produced by the convolution operation. Output of the final pooling layer $l + 1$ is given as input to the dense layer and the output of the dense layer is evaluated as described in Eq.2

$$h(x) = f(w^{l+1} \cdot x^{l+1} + b^{l+1}) \quad (2)$$

where w and b denotes weight and bias respectively. Figure 6 depicts the architecture and summary of the proposed 1D-CNN model used in this work for training and testing purpose. The steps defined in Algorithm-1 trains multiple 1D-CNN classifiers denoted as $model_c$ with their corresponding intermediate dataset Mal_API_c to classify with One vs. rest principle. $Over_Mal_API_Datasets$ will contain intermediate datasets corresponding to the list of individual classifiers in $1D_CNN_Classifier_List$ with 'class of interest' labeled as 1 i.e. positive and 'rest all' labeled as 0 i.e. negative. This algorithm returns $1D_CNN_Classifier_List$, which contains individually trained 1D-CNN classifiers as depicted in Figure 1.

Phase-III: Ensemble with SoftVoting

Using the modified soft voting method outlined in Algorithm-2, the classification abilities of independently trained classifiers in the $1D_CNN_Classifier_List$ are ensembled. Based on the outcomes of all the trained classifiers for each record in the test set, this algorithm determines the best predicted class. In the $1D_CNN_Classifier_List$, each classifier ($model_c$) predicts two probability scores of each record in the X_{test} as 'not in class of interest' and 'in class of interest' respectively. $ClassificationScore(c_{score})$ is a two dimensional array with a size of $(m \times 2)$ where m represents count of records in X_{test} . The i^{th} row of c_{score} represents probability score for the i^{th} record of X_{test} as 'not in class of interest' and 'in class of interest' at index 0 and 1 respectively.

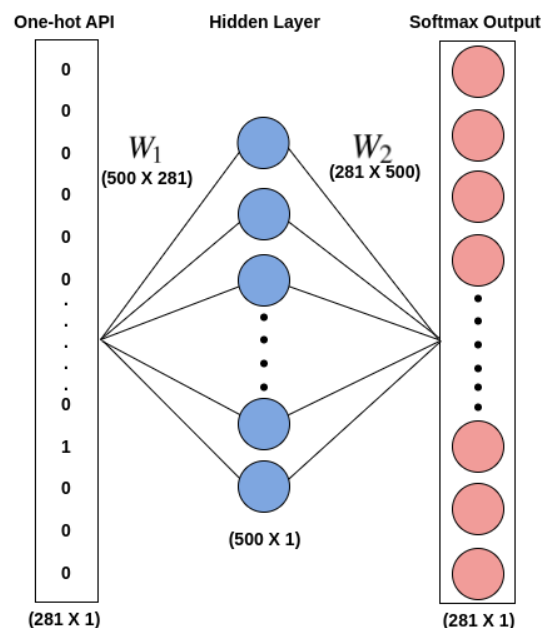


Figure 5. Word2Vec Skip Gram Model

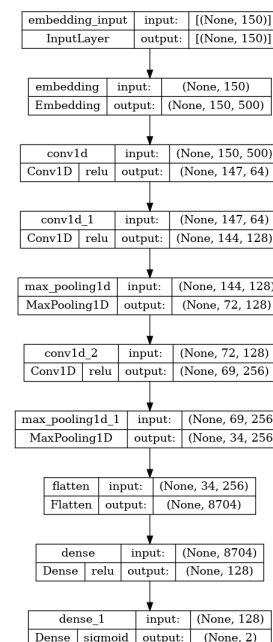


Figure 6. Summary of the 1D-CNN Model

Algorithm 1 1D-CNN Classifiers

Require: W_1 , ProcessedDataset $\{API_{seq}, M_{class}\}$

Ensure: 1D-CNN-Classifiers-List

```

1: EmbeddingMatrix =  $W_1$ 
2: OvR_MalAPI_Datasets =  $\phi$ 
3: for each distinct class  $c \in M_{class}$  do
4:    $Mal\_API_c = \phi$ 
5:   for each Record  $\{API_{seq}, M_{class}\} \in ProcessedDataset\{API_{seq}, M_{class}\}$  do
6:     if ( $M_{class} == c$ ) then
7:        $Mal\_API_c = Mal\_API_c \cup \{API_{seq}, 1\}$ 
8:     else
9:        $Mal\_API_c = Mal\_API_c \cup \{API_{seq}, 0\}$ 
10:    end if
11:  end for
12:   $OvR\_MalAPI\_Datasets = OvR\_MalAPI\_Datasets \cup \{Mal\_API_c\}$ 
13: end for
14: 1D-CNN-Classifiers-List =  $\phi$ 
15: for each  $Mal\_API_c \in OvR\_MalAPI\_Datasets$  do
16:    $Xtrain_c, Xtest_c, Ytrain_c, Ytest_c = train\_test\_split(Mal\_API_c[API_{seq}], Mal\_API_c[M_{class}], 0.8)$ 
17:    $model_c = Conv1D\_Model(Xtrain_c, Ytrain_c, EmbeddingMatrix, validation\_data = (Xtest_c, Ytest_c))$ 
18:   1D-CNN-Classifiers-List = 1D-CNN-Classifiers-List  $\cup \{model_c\}$ 
19: end for
20: Return 1D-CNN-Classifiers-List

```

Algorithm 2 ModifiedSoftVoting

Require: 1D-CNN-Classifiers-List as C_{list} , ProcessedDataset $\{API_{seq}, M_{class}\}$

Ensure: confusion_matrix as cm

```

1: Set  $cm[n][n] = 0$  ▷ n: Number of distinct class in  $M_{class}$ 
2:  $Xtest, Ytest = test\_split(ProcessedDataset\{API_{seq}, M_{class}\})$  ▷ Test Set of multiclass records
3: for each  $c \in C_{list}$  do
4:    $c_{index} = C_{list}.index(c)$  ▷  $c_{index}$ : Index of classifier  $c$  in  $C_{list}$ 
5:    $c_{score} = c.predict(Xtest)$  ▷  $c_{score}$ : Predicted ClassificationScore
6:    $c_{proba}[:, c_{index}] = c_{score}[:, 1]$  ▷ Column 1 of  $c_{score}$  is assigned to column  $c_{index}$  of  $c_{proba}$ 
7: end for
8: for each  $r \in c_{proba}$  do ▷ Row wise traversal on  $c_{proba}$ 
9:    $r_{index} = c_{proba}.index(r)$  ▷  $r_{index}$  is the row index of  $r$  in  $c_{proba}$ 
10:   $c_{plabel} = indexof(max(r))$  ▷  $c_{plabel}$ : Predicted label as index of largest element in  $r$ 
11:   $c_{tlabel} = Ytest[r_{index}]$  ▷  $c_{tlabel}$ : Actual label at  $r_{index}$  in  $Ytest$ 
12:   $cm[c_{tlabel}, c_{plabel}] = cm[c_{tlabel}, c_{plabel}] + 1$ 
13: end for
14: Return  $cm$ 

```

214 $ClassProbability(c_{proba})$ is a two dimensional array of size $(m \times 8)$ where m represents count
215 of records in $Xtest$. The j^{th} column of c_{proba} will contain m probability scores of m records of
216 $Xtest$ as ‘in class of interest’ using j^{th} classifier $model_j$ in 1D-CNN-Classifiers-List. The i^{th} row of
217 c_{proba} represents probability scores of i^{th} record of $Xtest$ as ‘in class of interest’ for all classifiers in
218 1D-CNN-Classifiers-List respectively.

219 Statements 8-13 in Algorithm-2 constructs the confusion matrix(cm) by ensembling all the predictions.
220 The cm is a square matrix of dimension $(n \times n)$ where n is the count of distinct classes in M_{class} . It
221 contributes to the estimation of several performance metrics as mentioned in Eq.(3, 4, 5, and 6).

222 The fundamental parameters required to calculate various performance metrics are TP , FN , FP , and
223 TN . These parameters for a specific class of interest is interpreted in the cm as a case of

224 a) **TP (True Positive):** When Malware of “Class of Interest” is predicted as “Class of Interest”

- 225 b) **FN (False Negative)**: When Malware of “Class of Interest” is predicted as some other Class.
- 226 c) **FP (False Positive)**: When Some other Class of malware is predicted as malware of “Class of
- 227 Interest”
- 228 d) **TN (True Negative)**: When Malware of other Class is predicted as malware of other Class.

229 Table 1 and Table 2 represents two exemplary cases of confusion matrix and TP, FP, FN, and TN parameters

230 for specific “Class of Interest” C_3 and C_2 respectively in the context of multiclass problem.

Table 1. When Class of Interest is Class-3

| | | | | | |
|--------|-------|-----------|-------|-------|-------|
| Actual | C_1 | TN | | FP | TN |
| | C_2 | | | FP | |
| | C_3 | FN | FN | TP | FN |
| | C_4 | TN | | FP | TN |
| | | C_1 | C_2 | C_3 | C_4 |
| | | Predicted | | | |

Table 2. When Class of Interest is Class-2

| | | | | |
|--------|-------|-----------|-------|-------|
| Actual | C_1 | TN | FP | TN |
| | C_2 | FN | TP | FN |
| | C_3 | | FP | |
| | C_4 | TN | FP | TN |
| | | C_1 | C_2 | C_3 |
| | | Predicted | | |

231 **Accuracy** of the model is estimated as count of correct predictions divided by total count of predictions.

232 Eq-3 mathematically represents the calculation of accuracy. Sometimes accuracy may mislead, hence the

233 performance is ensured by calculating average of precision, recall and F_1 score respectively for all classes

234 in multiclass class classification problem.

$$235 \quad Accuracy = \frac{\sum_{i=1}^n cm[i, i]}{\sum_{i=1}^n \sum_{j=1}^n cm[i, j]} \quad (3) \quad Precision_c = \frac{cm[c, c]}{\sum_{i=1}^n cm[i, c]} \quad (4)$$

$$236 \quad Recall_c = \frac{cm[c, c]}{\sum_{i=1}^n cm[c, i]} \quad (5) \quad F_{1c} = \frac{2}{\frac{1}{Recall_c} + \frac{1}{Precision_c}} \quad (6)$$

237 Precision for a specific class c (**Precision_c**) is estimated to see the impact of FP as higher concern

238 than FN , as explained in Eq-4. It is estimated as the number of true positives divided by the number of

239 predicted positives. Recall for a specific class c (**Recall_c**) is estimated to see the impact of FN as higher

240 concern than FP . It is estimated as the number of true positives divided by total number of actual positives.

241 F_1 -Score of a specific class c (**F_{1c}**) is the harmonic mean of **Precision_c** and **Recall_c**. It is used to ensure

242 high precision against high recall. Weighted and macro average of precision, recall and F_1 -score are used

243 as performance metrics for multiclass problems. Unweighted mean of each of these performance metrics

244 are referred as macro average measure. Weighted mean of each of these performance metrics are referred

245 as weighted average measure using count of samples of each class as the weight.

246 EXPERIMENTAL SETUP AND RESULTS

247 The experimental work for the described “Ensembled 1D-CNN architecture” is carried out using “Intel(R)

248 Xeon(R) CPU E5-2620 v4 @ 2.10GHz” HPC with 128 GB of RAM, “NVIDIA Corporation GP102

249 [GeForce GTX 1080 Ti]” GPU, Ubuntu-18.04 LTS, and Python 3.8.

250 In Phase-I of the architecture, the word embedding matrix W_1 [500 X 280] is constructed for a total of

251 281 distinct APIs using Skip-gram model. The final word embeddings of each distinct API in embedding

252 matrix W_1 is found using window size $\ell = 10$, vector_size = 500 after considering several combinations

253 of window size and vector_size.

Table 3. Classification Report of 1D-CNN Models

| OvR_Mal_API_Datasets | 1D_CNN_Classifiers_List | | precision | recall | f1-score | accuracy |
|----------------------|-------------------------|---------------|-----------|--------|----------|----------|
| MAL_API_0 | $model_0$ | Rest(0) | 0.99 | 0.99 | 0.99 | 0.98 |
| | | Adware(1) | 0.88 | 0.83 | 0.85 | |
| MAL_API_1 | $model_1$ | Rest(0) | 0.91 | 0.95 | 0.93 | 0.88 |
| | | Backdoor(1) | 0.61 | 0.46 | 0.52 | |
| MAL_API_2 | $model_2$ | Rest(0) | 0.95 | 0.97 | 0.96 | 0.93 |
| | | Downloader(1) | 0.78 | 0.66 | 0.72 | |
| MAL_API_3 | $model_3$ | Rest(0) | 0.95 | 0.95 | 0.95 | 0.91 |
| | | Dropper(1) | 0.65 | 0.63 | 0.64 | |
| MAL_API_4 | $model_4$ | Rest(0) | 0.93 | 0.95 | 0.94 | 0.89 |
| | | Spyware(1) | 0.52 | 0.43 | 0.47 | |
| MAL_API_5 | $model_5$ | Rest(0) | 0.90 | 0.93 | 0.91 | 0.85 |
| | | Trojan(1) | 0.46 | 0.36 | 0.40 | |
| MAL_API_6 | $model_6$ | Rest(0) | 0.95 | 0.96 | 0.95 | 0.92 |
| | | Virus(1) | 0.73 | 0.70 | 0.72 | |
| MAL_API_7 | $model_7$ | Rest(0) | 0.93 | 0.94 | 0.93 | 0.89 |
| | | Worm(1) | 0.60 | 0.56 | 0.58 | |

To address the imbalanced multiclass malware classification problem, 8 number of 1D-CNN models are trained and validated with One vs. Rest classification principle in Phase-II of the architecture. Figure 6 depicts the best configuration of the 1D-CNN model, which is decided by working around several way of consideration of Convolution layers, MaxPolling layers, and Dense layers with various parameters such as filters, kernel size, pool/window size, batch size, activation functions. *Adam* is found as the best optimizer after working around multiple optimizers like *Adam*, *Adaboost* and *Adadelata*. The training and validation of each $model_c$ respective to M_{class} is done with 80:20 stratified split ratio of their respective data set MAL_API_c from *OvR_MalAPI_Datasets*. *OvR_MalAPI_Datasets* is a set of 8 datasets corresponding to 8 models constructed using Algorithm-1. The performance metrics of each $model_c$ respective to malware classes Adware, Backdoor, Downloader, Dropper, Spyware, Trojan, Virus, and Worms are mentioned in Table 3.

Figure 7 depicts the Accuracy-Loss plot during traing and valiodation, ROC plot, and Precision-Recall plot with AUC score of all the 8 individual classifiers.

The classification capabilities of all these 8 trained classifiers are ensembled using the proposed *ModifiedSoftVoting* algorithm as described in Algorithm-2 and used in PHASE-III of the architecture. With a stratified split rate of 20%, the multiclass “Test Set” is obtained from $ProcessedDataset\{API_{seq}, M_{class}\}$ created in PHASE-I. *ModifiedSoftVoting* algorithm gets the best predicted class for each record of the multiclass Test Set, considering the predicted results of all the 8 trained classifiers. Figure-8 depicts the confusion matrix of the classification statistics of the *Test Set* after ensembling. Table-4 mentions the classification report of proposed *ModifiedSoftVoting* model by ensembling all the 8 classifiers.

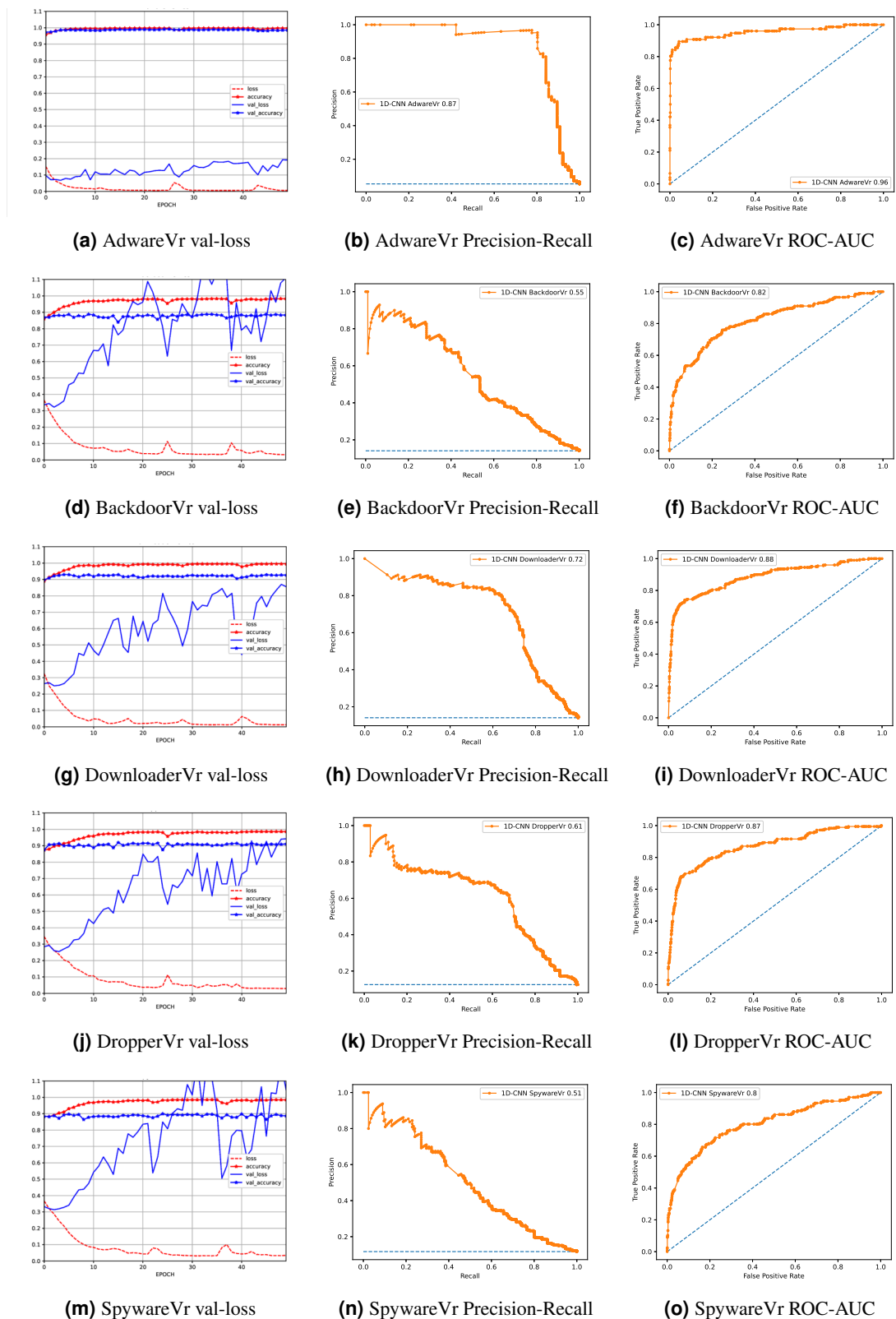


Figure 7. Performance Plots of Individual 1D-CNN Classifiers. (*cont.*)

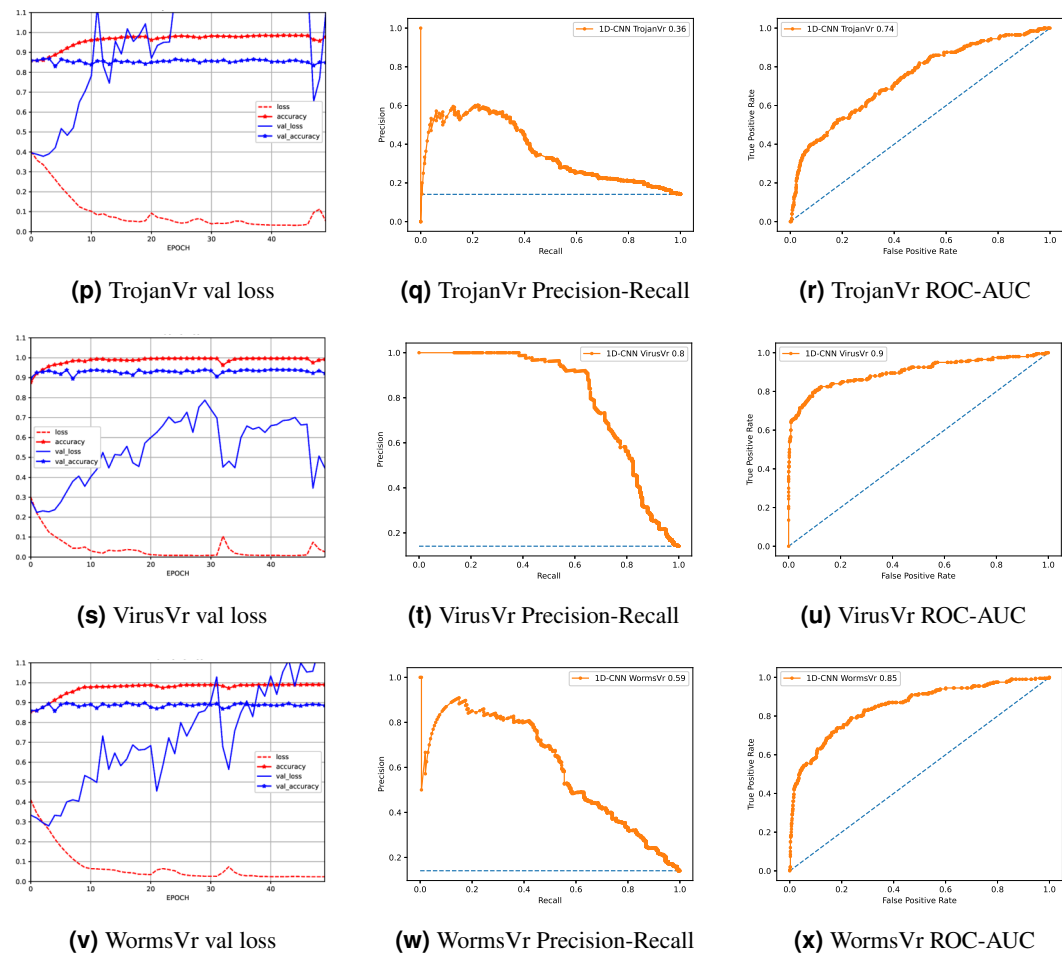


Figure 7. Performance Plots of Individual 1D-CNN Classifiers.

Table 4. Classification Report of Ensembled ModifiedSoftVoting Model

| | precision | recall | f1-score |
|---------------------|-----------|--------|----------|
| Adware | 0.95 | 0.95 | 0.95 |
| Backdoor | 0.87 | 0.86 | 0.87 |
| Downloader | 0.99 | 0.92 | 0.95 |
| Dropper | 0.97 | 0.89 | 0.93 |
| Spyware | 0.77 | 0.87 | 0.82 |
| Trojan | 0.85 | 0.89 | 0.87 |
| Virus | 0.89 | 0.95 | 0.92 |
| Worms | 0.97 | 0.90 | 0.93 |
| accuracy | | | 0.90 |
| macro avg | 0.91 | 0.90 | 0.90 |
| weighted avg | 0.90 | 0.90 | 0.90 |

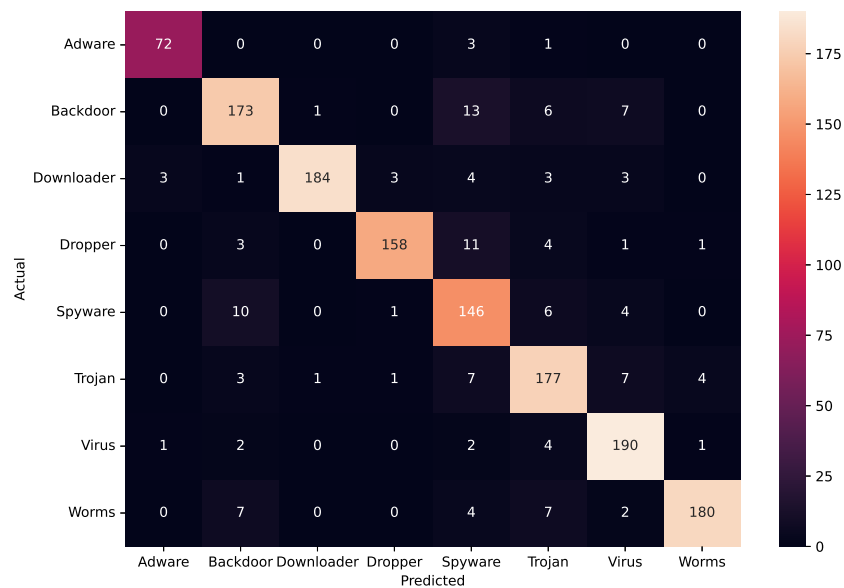
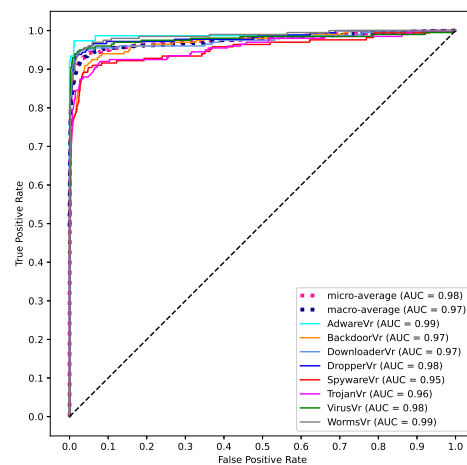
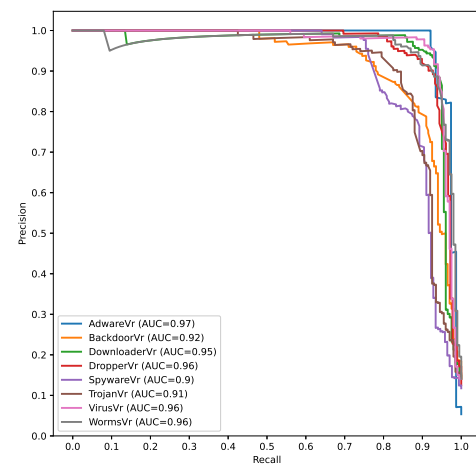


Figure 8. Confusion Matrix of ModifiedSoftVoting Model



(a) ROC-AUC



(b) Precision-Recall

Figure 9. Performance plots of Ensembled ModifiedSoftVoting Model

Figure-9a shows the ROC with AUC Score and Figure-9b shows the Precision-Recall plot with AUC score of all the 8 classes of malware present in *Mal-API-2019* using the proposed ensembled ModifiedSoftVoting model.

The Table-5 shows the performance comparison of the proposed ensembled ModifiedSoftVoting model with models proposed by several authors in their work using the same data set *Mal-API-2019*.

Table 5. Comparison of Result with works of other Authors

| | Accuracy | Macro-avg Precision | Macro-avg Recall | Macro-avg F1-Score |
|--|-------------|------------------------|---------------------|-----------------------|
| Single Layer LSTM(Ferhat Ozgur et al., 2020) | - | 0.50 | 0.47 | 0.47 |
| s Two Layer LSTM(Ferhat Ozgur et al., 2020) | - | 0.40 | 0.41 | 0.39 |
| LSTM with Case2(Li and Zheng, 2021) | 0.55 | 0.56 | 0.58 | 0.57 |
| GRU with Case2(Li and Zheng, 2021) | 0.55 | 0.56 | 0.59 | 0.57 |
| RTF Model (Demirkiran et al., 2022) | 0.60 | - | - | 0.61 |
| Proposed Ensembled Model | 0.90 | 0.91 | 0.90 | 0.90 |

CONCULSION AND FUTURE DIRECTIONS

API call sequences are becoming recognized as a key characteristic for categorizing malware. To classify eight extremely unbalanced malware classes, the proposed Ensembled architecture of separately trained 1D-CNN models has demonstrated good results in this work. The experimental set-up using the Mal-API-2019 benchmark dataset has demonstrated significant improvement in classification accuracy, which is now 90%. The macro averaged precision, recall, and F_1 score for all classes are calculated to be 91%, 90%, and 90%, respectively. In the ROC plot and Precision-Recall plot, the AUC score is found to be greater than 95% and 90%, respectively, for all classes. Compared to past studies' findings, which had a maximum macro average F_1 score of 61%, this result clearly represents a significant improvement. The impact of the data augmentation strategy on the classification outcomes for such imbalanced classes is not investigated in this work. Another approach is to examine the effects of statistical feature engineering approaches on classification outcomes, such as PCA and duplicate subsequence removal techniques.

REFERENCES

- (2021). Kaspersky security bulletin.
- (2023). Sonicwall, cyber threat report, tech. rep., sonicwall.
- Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Hasan, M., Van Essen, B. C., Awwal, A. A. S., and Asari, V. K. (2019). A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8(3).
- Aslan, O. A. and Samet, R. (2020). A comprehensive review on malware detection approaches. *IEEE Access*, 8:6249–6271.
- Catak FO, Ahmed J, S. K. K. Z. (2021). Data augmentation based malware detection using convolutional neural networks. *PeerJ Comput Sci.*, 7:e346.
- Demirkiran, F., Cayir, A., Unal, U., and Dag, H. (2022). An ensemble of pre-trained transformer models for imbalanced multiclass malware classification. *Computers and Security*, 121:102846.
- Ferhat Ozgur, C., Ahmet Faruk, Y., Ogerta, E., and Javed, A. (2020). Deep learning based sequential model for malware analysis using windows exe api calls. *PeerJ Comput Science* 6:e285.
- Gibert, D., Mateu, C., and Planes, J. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526.
- Hammad, B. T., Jamil, N., Ahmed, I. T., Zain, Z. M., and Basheer, S. (2022). Robust malware family classification using effective features and classifiers. *Applied Sciences*, 12(15).
- Han, W., Xue, J., Wang, Y., Huang, L., Kong, Z., and Mao, L. (2019). Maldae: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *Computers and Security*, 83:208–233.
- Kolosnjaji, B., Zarras, A., Webster, G., and Eckert, C. (2016). Deep learning for classification of malware system call sequences. In Kang, B. H. and Bai, Q., editors, *AI 2016: Advances in Artificial Intelligence*, pages 137–149, Cham. Springer International Publishing.

- 316 Li, C. and Zheng, J. (2021). Api call-based malware classification using recurrent neural networks.
317 *Journal of Cyber Security and Mobility*, 10(3):617–640.
- 318 Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., and Alsaadi, F. E. (2017). A survey of deep neural network
319 architectures and their applications. *Neurocomputing*, 234:11–26.
- 320 Mehra, V., Jain, V., and Uppal, D. (2015). Dacomm: Detection and classification of metamorphic malware.
321 In *2015 Fifth International Conference on Communication Systems and Network Technologies*, pages
322 668–673.
- 323 Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in
324 vector space. *CoRR*, abs/1301.3781.
- 325 Panda, B. and Tripathy, S. N. (2020). Detection of anomalous in-memory process based on dll sequence.
326 *International Journal of Advanced Computer Science and Applications*, 11(10).
- 327 Shijo, P. and Salim, A. (2015). Integrated static and dynamic analysis for malware detection. *Procedia*
328 *Computer Science*, 46:804–811. Proceedings of the International Conference on Information and
329 Communication Technologies, ICICT 2014, 3-5 December 2014 at Bolgatty Palace & Island Resort,
330 Kochi, India.
- 331 Sun, G. and Qian, Q. (2021). Deep learning and visualization for identifying malware families. *IEEE*
332 *Computer Society Press*, 18(1):283–295.
- 333 Tayyab, U.-e.-H., Khan, F. B., Durad, M. H., Khan, A., and Lee, Y. S. (2022). A survey of the recent
334 trends in deep learning based malware detection. *Journal of Cybersecurity and Privacy*, 2(4):800–829.
- 335 Tekerek, A. (2021). A novel architecture for web-based attack detection using convolutional neural
336 network. *Computers and Security*, 100:102096.
- 337 Ucci, D., Aniello, L., and Baldoni, R. (2019). Survey of machine learning techniques for malware analysis.
338 *Computers and Security*, 81:123–147.
- 339 Vasan, D., Alazab, M., Wassan, S., Safaei, B., and Zheng, Q. (2020). Image-based malware classification
340 using ensemble of cnn architectures (imcec). *Computers and Security*, 92:101748.
- 341 Vinod, P., Jain, H., Golecha, Y., Gaur, M., and Laxmi, V. (2010). Medusa: Metamorphic malware dynamic
342 analysis usingsignature from api. pages 263–269.
- 343 Xiao, X., Zhang, S., Mercaldo, F., Hu, G., and Sangaiah, A. K. (2019). Android malware detection based
344 on system call sequences and lstm. *Multimedia Tools and Applications*, 78:3979–3999.
- 345 Zhang, Y., Huang, Q., Ma, X., Yang, Z., and Jiang, J. (2016). Using multi-features and ensemble
346 learning method for imbalanced malware classification. In *2016 IEEE Trustcom/BigDataSE/ISPA*,
347 pages 965–973.