Malware Packet Detections via CNN Ensembles and Optimum Feature Selection

Alex Liew[1], Ping Ji[2], Sos S. Agaian[3]
[1]aliew@gradcenter.cuny.edu,[2]pji@gc.cuny.edu,[3]sos.agaian@csi.cuny.edu
Graduate Center of City University of New York

***Abstract*—Malware packets can disrupt network operations and compromise the integrity of network infrastructure. Malware packet detections aid in discovering malicious activity within a network. This allows organizations to detect and respond to potential security threats before they can cause significant harm. Rapid detection and mitigation of these packets help in maintaining the reliability and performance of the network. This work introduces malware packet detection through convolutional neural networks (CNN) ensembles and feature selections through feature weightings with singular value decomposition. Our experimentation demonstrates that the utilization of CNN ensembles and feature selection techniques can attain an accuracy of 99.76% in identifying malicious packets within the USTC-2016 dataset. Additionally, our experiments include a comparative analysis between performance and computation times study, which illustrates good accuracy within a substantially reduced timeframe.**

***Index Terms*—— Malware packet detection, Convolutional Neural Network (CNN), Feature weights, CNN Ensembles**

I. Introduction

Malware, often disguised as harmless files or links, poses a threat by potentially seizing control of or disrupting targeted computer infrastructures [1]. In 2022, the cybersecurity community reported approximately 5.5 billion malware attacks worldwide [2]. Worms, viruses, ransomware, and trojans were the most common forms of malware, often delivered through phishing emails and websites. Given the these threats, the detection and ongoing surveillance of malware activities are imperative. Current methodologies for identifying and neutralizing malicious software encompass signature-based detection, network-based detection, deployment of anti-malware solutions, and the application of machine learning/deep learning (ML/DL) techniques.

Signature-Based Malware Detection [3] is to identify and prevent known malware from compromising a computer system or network by leveraging a repository of malware signatures. A ”signature” in this context is a unique string of bits or identifiable patterns in the code that distinguishes a specific piece of malware. The primary goal of Network-Based Malware Detection [4] is to monitor, identify, and mitigate malicious activities and threats within network traffic to protect an organization's computing infrastructure. This method focuses on analyzing the data flowing across a network in real-time to detect signs of malware, intrusions, or other security threats based on known patterns of malicious behavior or anomalies in the network traffic. The goal of Anti-Malware Software Detection [5] is to protect computer systems and networks from malware encompassing viruses, worms, trojans, ransomware, spyware, adware, and other malicious programs. This is achieved through the identification, blocking, and removal of malware to prevent data theft, unauthorized access, and to ensure the privacy and integrity of user data. The primary goal of employing ML and DL [6] in malware detection is to enhance cybersecurity defenses with automated systems capable of identifying and responding to threats with minimal human intervention. They offer pattern recognition, adaptive learning, and proactive defense, which improve threat detection speed and accuracy.

The adoption of deep learning techniques in cybersecurity research has grown rapidly, fueled by advances in GPU-based parallel processing, which have enabled efficient implementation of computationally intensive architectures such as convolutional neural networks (CNNs). Studies [8]-[15] have demonstrated the effectiveness of CNNs in network traffic analysis by utilizing the similarity between TCP packet payloads and image data, allowing CNNs to extract high-level features for malware detection. However, CNNs' extensive parameterization presents challenges, including high computational demands and increased latency, particularly for real-time applications. An alternative approach combines CNNs with classical classifiers and ensemble methods, offering some advantages over standalone CNNs. This hybrid model reduces computational complexity by delegating feature extraction to the CNN while relying on classical classifiers for decision-making, resulting in lower latency.

To address these challenges, our paper provides the following contributions.

1. We propose a novel ensemble of weighted Convolutional Neural Networks (CNNs) designed to distinguish between benign and malicious network packets by analyzing their payloads. The ensemble approach leverages the strengths of multiple CNN architectures to enhance detection accuracy and robustness.
2. We refine the weighted CNN ensemble by incorporating Singular Value Decomposition (SVD) and feature weighting techniques, enabling an efficient feature selection process. This approach extracts a small but informative subset of features from the broader feature set generated by the CNN. By focusing on the most relevant features, the method reduces computational overhead.
3. The proposed model is evaluated on the USTC-2016 dataset. Experimental results demonstrate the model's

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

high accuracy in identifying malware. Additionally, we provide a comparative analysis of accuracy versus detection time. It illustrates that the model achieves reduced detection times while maintaining detection performance.

The rest of our paper is organized in the following: In Section II, we provide a literature review that discusses the most recent works in deep learning-based malware packet detection; In Section III, we describe the implementations of our CNN ensembles and feature selection process; In section IV, our dataset and measurement mechanisms are described; In Section V, we present our evaluation results; Section VI summarizes our work.

II. Literature Review

Deep Learning (DL) techniques such as Recurrent Neural Networks (RNN), Long-Short-Term-Memory (LSTM) networks, and CNNs have shown the ability to detect malicious traffic within a network. The following is a brief review of recent works in malware detection using DL models. A model aims to assist network managers in assessing network traffic to identify potential malicious sources or unusual behaviors from log data [9]. Two generative models have been trained to generate malware, which includes behavioral attributes. A three-step validation process is used to confirm the presence of specified behaviors within generated malware data points [10]. A method utilizing CNNs is adopted for image feature extraction of malware traffic, and further utilizes three different types of classifiers and demonstrates the proposed approach meets the accuracy requirements for practical applications [11].

As monitoring is provided to power systems, there is a rising vulnerability to diverse cyber-attacks. A filter extracts time-series relationships by examining the temporal structure of packet data. This relationship is employed by CNN to classify distinct power system events [12]. A recent development compared the following DL models, Long Short-Term Memory, and Convolutional Neural Network for malware detection [13]. Recurrent Neural Networks (RNN) have been proposed to accurately predict whether an executable is malicious or benign [14]. Another hybrid DL method, specifically the CNN-LSTM technique, was employed to identify botnet attacks. Malicious software, which infiltrates a computer system or is installed without the administrators' consent or awareness, was the focus of this study [15].

As the works presented in [8]-[15] aim to effectively distinguish benign and malicious software, they have overlooked the extraction of an optimal feature set that balance both performance and speed. To address this, our work provides CNN feature reduction techniques to shorten testing times of each packet while maintaining high accuracy. For this purpose, we start with preprocessing network traces captured through Wireshark by converting each network packet into image data, regardless whether it is benign or containing malware components.

III. Methodology

A. Preprocessing

Our methodology starts with data preprocessing of each data packet captured by Wireshark. Wireshark records various information of each data packet within a nearby network, such as current time, time to live (TTL), round trip time (RTT), source and destination internet protocols (IP), packet type, and data payload. The focus of our preprocessing is to extract payload information from network data packets. The extraction begins with conversion of Wireshark captured data file to CSV file. The standard length of packet payload is the first 72 bytes recorded into the CSV file. When the payload is less than 72 bytes, zeros are padded. Figure 1. illustrates this process below. The 72 bytes are reorganized into 8 bytes $\times$ 9 bytes *image*, where every 1 hex digit is converted to 4-bit decimal with 16 image intensity levels. Then the 8×9 image is resized to $224 \times 224 \times 3$ to conform with most of off-the-shelf CNNs' input size. In the next subsection, each image extracted from the packet payload is employed for feature extraction.

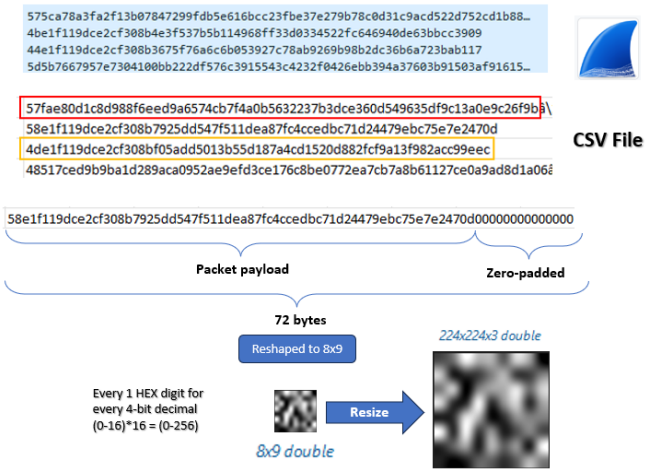

Fig. 1. Data Preprocessing steps of each Packet.

B. CNN Features

The components of a CNN begin with image I, which has dimensions of $n_1 \times n_2 \times 3$, where the '3' indicates the number of channels in the image (RGB). Image, I act as input to feature extractors, where a feature extractor can be represented by a trained CNN. The number of channels increases by $3*N_F$ as image I pass through each convolutional layer, where N_F the number of filters in that layer. The total number of channels becomes K as the image traverses the final convolutional layer. Consequently, the feature map at the output of the average pooling transform from dimensions of $H \times W \times K$ to $H_p \times W_p \times K$. Features extracted $\mathbf{f}$ is given by $\mathbf{f} = [f_1, f_2, f_3, \ldots f_L]$, where L represents the total length of the feature vector, where $L = H_P W_P K$. f is the feature vector generated by the activation extracted at the output of the global pooling.

C. Feature Selection

Principal Component Analysis (PCA) is the commonly adopted method for reducing the dimensions of features. The PCA employed in this paper relies on Singular Value Decomposition (SVD) [19], which is utilized to find optimal features. Consider the following point-to-point product of two matrices $\mathbf{\Gamma}\cdot\mathbf{X}$ given by $\mathbf{X_\Gamma} = \mathbf{\Gamma}\cdot\mathbf{X} = \mathbf{U\Sigma V}^T$ which characterizes the SVD of the product $\mathbf{\Gamma}\cdot\mathbf{X}$. $\mathbf{X}$ comprises feature vectors extracted from a trained CNN which can be written by the following:

$$\mathbf{X} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \dots \\ \mathbf{f}_N \end{bmatrix} = \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} & \dots & f_{1,K} \\ f_{2,1} & f_{2,2} & f_{2,3} & \dots & f_{2,K} \\ \dots & \dots & \dots & \dots & \dots \\ f_{N,1} & f_{N,2} & f_{N,3} & \dots & f_{N,K} \end{bmatrix} \quad (1)$$

where $\mathbf{f}_i$, is the feature vector extracted from an instance, I_i with $\mathbf{I} = I_i, i = 1, ..., N$, N is the number of instances in the dataset, $\cdot$ is the point-to-point multiplication between $\mathbf{\Gamma}$ and $\mathbf{X}$, U represents the reflection of $\mathbf{X_\Gamma}$, the diagonal of $\mathbf{\Sigma}$ represents the variabilities, σ_1, ..., σ_L, up to the L^{th} feature, $\mathbf{V}^T$ is the 'eigen values' of $\mathbf{X_\Gamma}$. $\mathbf{\Gamma}$ is found by incorporating feature weighting through minimum redundancy maximum relevance (MRMR) algorithm [20]. The MRMR algorithm discovers a group of features characterized by maximum dissimilarity. This method minimizes redundancy within the feature set while maximizing their relevance to their labels. The output of the MRMR algorithm is a vector, $\boldsymbol{\rho}$, given by: $\boldsymbol{\rho} = (\rho_1, \rho_2, \dots, \rho_L)$, where the weight, ρ_k, represents the weight importance of the k^{th} feature at the j^{th} data point, $\mathbf{f}_{j,k}$. $\boldsymbol{\rho}$ is then expanded to a matrix $\mathbf{\Gamma}$ by repeating the same features row by row N–1 number of times. $\mathbf{\Gamma}$ given by:

$$\mathbf{\Gamma} = \begin{bmatrix} \rho_1 & \rho_2 & \rho_3 & \dots & \rho_L \\ \rho_1 & \rho_2 & \rho_3 & \dots & \rho_L \\ \dots & \dots & \dots & \dots & \dots \\ \rho_1 & \rho_2 & \rho_3 & \dots & \rho_L \end{bmatrix} \quad (2)$$

where, $\mathbf{\Gamma}$ has N rows. $\mathbf{X_\Gamma}$ weighted by $\boldsymbol{\rho}$ and is written by the following:

$$\mathbf{X_\Gamma} = \begin{bmatrix} \rho_1 f_{1,1} & \rho_2 f_{1,2} & \rho_3 f_{1,3} & \dots & \rho_L f_{1,L} \\ \rho_1 f_{2,1} & \rho_2 f_{2,2} & \rho_3 f_{2,3} & \dots & \rho_L f_{2,L} \\ \dots & \dots & \dots & \dots & \dots \\ \rho_1 f_{N,1} & \rho_2 f_{N,2} & \rho_3 f_{N,3} & \dots & \rho_L f_{N,L} \end{bmatrix}. \quad (3)$$

Next, $\mathbf{X_\Gamma}$ will be subtracted by its mean of each column of X_{mean}, to obtain $X_m = \mathbf{X_\Gamma} - X_{mean}$. We then calculate $\mathbf{X}_P$, which is defined by the following:

$$\mathbf{X}_p = \mathbf{V}^T_{N\times N}\mathbf{X}^T_{m,N\times M} \quad (4)$$

where M is adjusted based on the percentage of variability up to L, $L \geq M$, and the dimension of $\mathbf{X}_P$ is $N \times M$. We can search for the sets of optimal features by starting M_i features of $\mathbf{X}_P$ and iterate by increasing the number of features. We can rewrite our input classification matrix, $\mathbf{X}_P^{(l)}$, at the l^{th} iteration using the following:

$$\mathbf{X}_p^{(l)} = (\mathbf{V}^T_{N\times N}\mathbf{X}^T_{m,N\times M}), M_i \geq M_l \geq L \quad (5)$$

where the first M_1 features are employed at l^{th} iteration and $\mathbf{X}_{m,N\times M_l}$ is the updated feature matrix employing M_l features. At l^{th} iteration, $X_P^{(l)}$ is a classification input matrix into each of the classifiers. The classifiers used during each iteration are support vector machine using Polynomial and Gaussian [28], Adaboost [25], and Random Forest [24]. The classification accuracies, $a(l,h)$, are recorded in each l^{th} iteration at the h^{th} classifier. $a(l,h)$ is defined by the following:

$$a(l,h) = C_h\{X_P^{(l)}\}, h = 1, \dots, 4 \quad (6)$$

where C_h is one of the h^{th} classifiers listed above. The maximum accuracy is found using the following:

$$l_{max} = \underset{l,h}{\text{argmax}}(a(l,h)) \quad (7)$$

$$l_{max} = \underset{l,h}{\text{argmax}}(C_h\{(\mathbf{V}^T_{N\times N}\mathbf{X}^T_{m,N\times M}), M_i \geq M_l \geq L\} \quad (8)$$

where argmax(.) determines the maximum location or accuracy of l^{th} at h^{th} classifier.

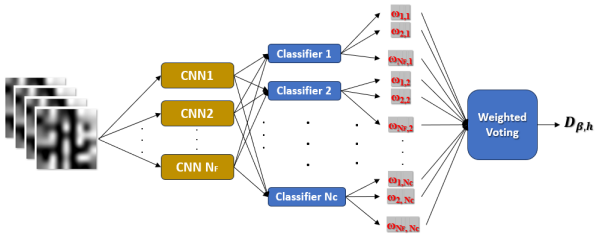

Fig. 2. Illustration of weighted Ensemble Decisions

D. CNN Ensembles

Ensembles combine the strengths of multiple models, leading to more accurate predictions. Each model in the ensemble contributes to the final decision, reducing the impact of errors from individual models. Numerous ensemble methodologies exist within the realm of machine learning [19]. Among the simplest are non-weighted and weighted voting techniques, which involve aggregating predictions from individually trained models across different feature extractors and classifiers.

In our work we explore the use of weighted ensemble fusion. We start by defining $p\{a_\beta(l,h)\}$, which is the class predictions generated for each measurement $a_\beta(l,h)$, where β is a feature extractor which is employed to generate the initial features $\mathbf{f}_1, \mathbf{f}_2, \dots \mathbf{f}_N$. Each feature extractor that is employed is represented numerically, $\beta = 1, 2, \dots, \beta_{max}$, where β_{max} is the number of feature extractor employed. The measurement that produced the highest score (7) by each h_{th} classifier are selected, $p\{a_\beta(l_{\beta,max}, h)\}$ in the ensemble system, where the highest accuracy is obtained at $l_{\beta,max}$ iteration using β feature extractor. The weighted generalized ensemble decision formula given by:

$$D_{\beta,h} = \max_{\beta,h}(\omega_{1,1} \odot p\{a_1(l_{1,max}, 1)\}, \dots, \omega_{\beta_{max},4} \odot p\{a_1(l_{\beta_{max}}, 4)\}) \quad (9)$$

where $D_{\beta,h}$ is the final prediction produced by predictors $p\{a_1(l_{1,max},h)\}$, ... , $p\{a_1(l_{\beta_{max},max},4)\}$ sorted by their accuracies in decreasing order, $\omega_{\beta,h}$ is the weight which corresponds to the β feature extractor and h^{th} classifier, and $\omega_{1,1} \odot p\{a_1(l_{1,max},1)\}$ reads the following: $p\{a_1(l_{1,max},1)\}$ is weighted by a factor $\omega_{1,1}$. $D_{\beta,h}$ selects the class with the highest weight. Each is given by:

$$\omega_{\beta,h} = \log(\frac{a_1(l_{1,max},h)}{1-a_1(l_{1,max},h)}) \tag{10}$$

where more weight is placed on predictions with higher accuracies. An illustration of obtaining the final prediction is shown in figure 2. The pseudo code of selecting the optimum set of features and CNN ensembles are shown in Algorithm 1.

Algorithm 1

Input: I
Split **I** into training, $\mathbf{I}_T$ and testing sets $\mathbf{I}_t$
for $\beta = 1, \ldots, \beta_{max}$ **do**
  Train $CNN_\beta(\mathbf{I}_T)$
  Extract features, $CNN_\beta(\mathbf{I}_t) \leftarrow \mathbf{f}_1, \mathbf{f}_2, \ldots \mathbf{f}_N$
  $\mathbf{X} = [\mathbf{f}_1, \mathbf{f}_2, \ldots \mathbf{f}_N]^T$
  $\boldsymbol{\rho} \leftarrow$ mrmr($\mathbf{X}$), $\boldsymbol{\Gamma} \leftarrow repeat(\boldsymbol{\rho})$
  $\mathbf{X}_\Gamma \leftarrow \mathbf{X} \cdot \boldsymbol{\Gamma}$
  $\mathbf{X}_{mean} \leftarrow$ mean($\mathbf{X}_\Gamma$)
  $\mathbf{X}_m \leftarrow \mathbf{X}_\Gamma - \mathbf{X}_{mean}$
  $\mathbf{U}, \boldsymbol{\Sigma}, \mathbf{V}^T \leftarrow$ SVD($\mathbf{X}_m$)
  $\mathbf{X}_p \leftarrow \mathbf{V}^T_{N\times N}\mathbf{X}^T_{m,N\times M}$
  for $l = 50, \ldots, 512$ **do**
    $\mathbf{X}_p^{(l)} \leftarrow \mathbf{V}^T_{N\times N}\mathbf{X}^T_{m,N\times M_l}$
    $a(l,h) = C_h\{X_P^{(l)}\}, h = 1, \ldots, 4$
  end for
  $l_{max} = \text{argmax}_{l,h}(C_h\{(\mathbf{V}^T_{N\times N}\mathbf{X}^T_{m,N\times M})\})$
  for $h = 1, \ldots, 4$ **do**
    $\omega_{\beta,h} \leftarrow \log(\frac{a_1(l_{h,max},h)}{1-a_1(l_{h,max},h)})$
    Find $\omega_{\beta,h} \odot p\{a_1(l_{1,max},h)\}$
  end for
end for
Find $D_{\beta,h}$

IV. Datasets & Performance Metrics

A. Dataset

The dataset employed in our paper is referred to as USTC-2016 [21]. This dataset was gathered by researchers at the University of Science and Technology of China (USTC) with the purpose of conducting network traffic analysis. Specifically, the Wireshark dataset comprises packet capture (PCAP) files captured through the utilization of Wireshark. The PCAP files include both malware and benign captures. The malware packets employed in our work are Neris, Nsis-ay, and Zeus. These files are some well-known trojan horses used in executables for installations and intended to attack banking institutions. The benign packets employed in our work are Facetime, FTP, and Weibo. Facetime is a popular App for video communication and Weibo is a popular eSports website which hosts gaming competitions. The packets are preprocessed and converted to images. An illustration of the image patterns is shown in figure 3. The count for each packet category is shown in Table I.

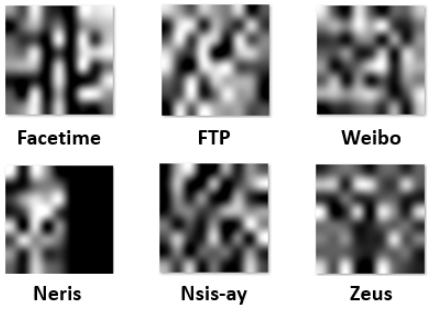

Fig. 3. Illustration of Image Patterns of Facetime, FTP, Weibo, Neris, Nsis-ay, and Zeus.

TABLE I
NUMBER OF IMAGES PER CATEGORY

Facetime	FTP	Weibo	Neris	Nsis-ay	Zeus
6,000	22,475	10,000	10,000	10,000	2,350

B. Feature Extractor and Classifier Selections

Our ensemble model incorporates the following feature extractors, classifiers, and parameters. The feature extractors employed include ResNet18, ResNet50 [16], and GoogleNet [17], chosen for their low parameter count. These CNNs are trained on top of ImageNet weights with an Adam Optimizer. The initial learning rate of 10e-4 employed along with multi-class cross-entropy function. Data is split into an 80-20 ratio for training and testing, with the learning rate decreasing by a factor of 0.2 if training accuracy doesn't improve after 4 epochs. The maximum number of epochs is set at 100, with a batch size of 16 images per iteration. Image augmentation techniques such as reflections and translations are applied randomly. After the CNNs are trained, the activations are extracted from the output of the global average pooling. These features are employed to trained our classifiers. Classifiers employed include Adaboost, Random Forest, support vector machines using Polynomial and Gaussian kernels. Training of selected CNN models is conducted on a laptop equipped with an RTX 3070 and an Intel Core™ i7-10750H CPU @ 2.60MHz with 6 cores. Our implementation is carried out in MATLAB 2021a, utilizing deep learning toolbox.

C. Performance Metrics

The standard performance metrics of a classification model using different parameters are accuracy, sensitivity (recall), specificity, precision, and F1-score:

$$accuracy = \frac{T_P + T_N}{T_P + T_N + F_P + F_N} \tag{11}$$

$$Sensitivity = Recall = \frac{T_P}{T_P + F_N} \tag{12}$$

$$Specificity = \frac{T_N}{T_N + F_P} \tag{13}$$

$$Precision = \frac{T_P}{T_P + F_P} \tag{14}$$

$$F1 - Score = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{15}$$

where, T_P and T_N are the true positive and true negative and F_p and F_n are the false positive and true positive, respectively. By evaluating accuracy, precision, sensitivity, specificity, and F1 score, we can understand the impact of classification errors on network operations.

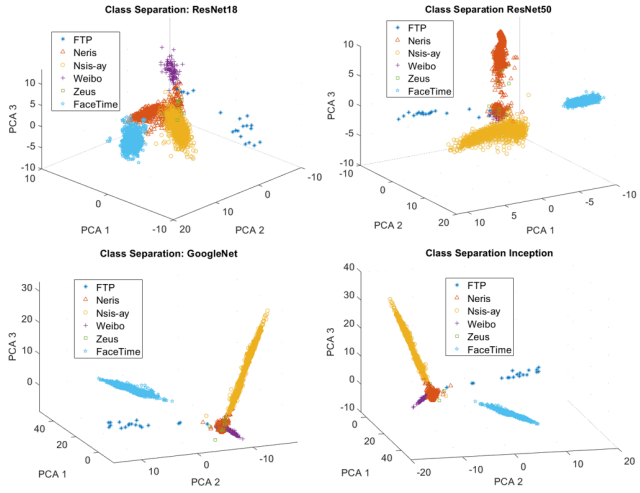

Fig. 4. Class Separations of Facetime, FTP, Weibo, Neris, Nsis-ay, and Zeus.

V. Class Separation Visualization and Performance Evaluations

A. Class Visualizations

Figure 4 illustrates the class separation of features extracted from malware packet payloads using Principal PCA for four different network architectures: ResNet18, ResNet50, GoogleNet, and Inception. Each subplot corresponds to a specific network architecture and displays the distribution of the six distinct malware classes: FTP, Neris, Nsis-ay, Weibo, Zeus, and FaceTime, represented by unique markers and colors. The ResNet18 and ResNet50 architectures demonstrate clear clustering, with ResNet50 showing a more defined separation, particularly between the Neris and FTP classes. The dense clustering of the Nsis-ay class in ResNet50 suggests enhanced discriminative capability of this deeper architecture. In addition, the isolated positioning of the FaceTime class in both ResNet architectures indicates effective feature extraction and separation.

The GoogleNet and Inception architectures also show distinct class separations, with GoogleNet displaying more compact clustering for the Zeus and Weibo classes. This suggests that GoogleNet effectively captures the distinctive characteristics of these classes. Inception, on the other hand, exhibits a broader spread of classes but maintains clear separations, especially for the Nsis-ay and FaceTime classes. The PCA plots indicate that while all four architectures are capable of distinguishing between different malware classes, ResNet50 and GoogleNet particularly excel in terms of class separability and clustering compactness. Given the observations in each network architecture's performance in class separation, it becomes evident that an ensemble learning approach could leverage the advantages of each model.

B. Performance vs Computation Times Study

In this section we examine the tradeoff between accuracy and computation time required to calculate M_l number of features in each iteration of Algorithm 1. Figure 5 shows plots of Accuracies versus number of features. The size of the markers indicates the times required to test all the testing data. Both Adaboost and Random Forest performances maintained stable around 99% ranges because they are ensemble learning methods. The inference times of Adaboost and Random Forest ranged 0.2-0.3 second and 1-2 seconds, respectively. Next, notice in the ResNet18-SVM plots, when the number of features are between 130 to 150 the accuracies peak at 99.712%. However, when more than 370 ResNet18-features are employed, we observed there is a sharp decrease in accuracy for both classifiers, SVM Poly and Gaussian Kernels. The inference times of SVM classifiers starts at low 0.2s with 50 features to 20 seconds classifying 512 fearures, where the inference times are represented by the marker's size.

Possible reasons for the decline in performance maybe the following. ResNet50-SVM and GoogleNet-SVM dropped in accuracy at lesser number of features maybe due to having more training parameters than ResNet18-SVM. This overall decrease in accuracy might be attributed to the curse of dimensionality, particularly evident as the number of features (dimensions) increases. Since SVMs rely on finding the optimal hyperplane that separates classes in the feature space. However, we do not need to rely on all features extracted by each CNN, but rather only employ the top M_l features, which reduce inference times.

C. CNN Ensemble Results

The outcomes of our CNN ensembles are presented in Table II and Figure 6. Figure 6 illustrates accuracies, precisions, sensitivities, specificities, and F1 scores sorted from highest $a(l,h)$ to the lowest, achieved by each CNN-classifier combination utilized. Our ensemble predictions, denoted as $D_{\beta,h}$, are integrated into the weighted majority voting decisions sequentially from the highest to the lowest accuracies. It's noteworthy to observe the performance metrics fluctuates as more $p\{(a(l,h))\}$ predictions are incorporated. This observation arises because the inclusion of more $p\{(a(l,h))\}$ predictions enables self-correction of misclassified data points by other predictions or versa. We found the highest accuracy achieved was 99.76% which is a 0.01% increased prior to ensemble fusion step. Additionally, each CNN-classifier combination utilizes only about 110 to 150 features, and their computations

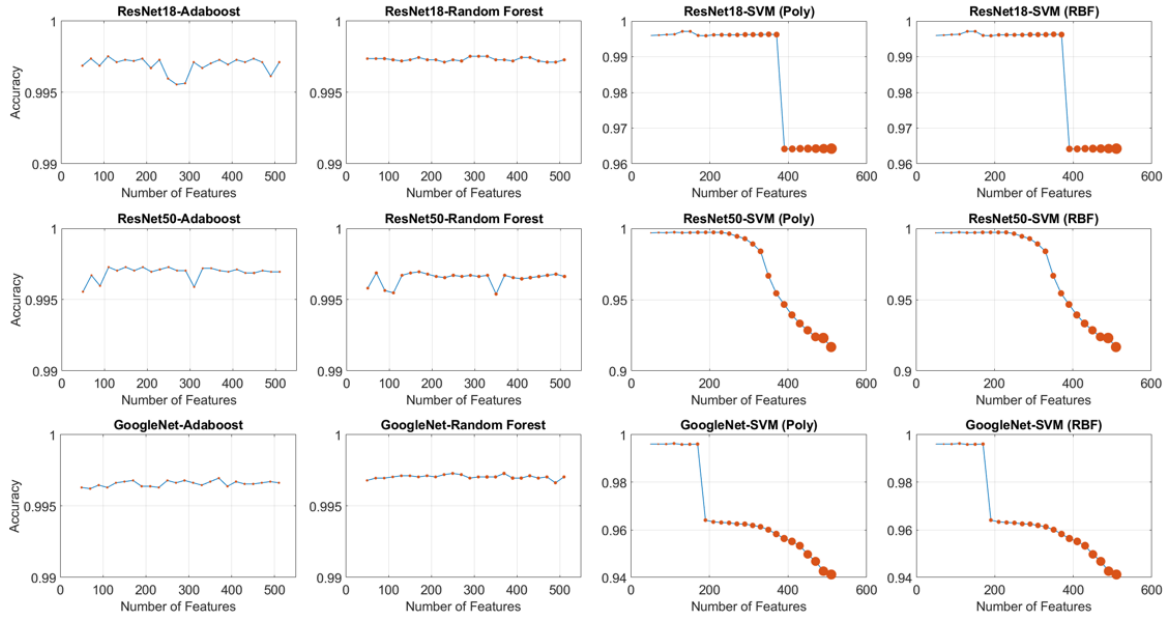

Fig. 5. Plots of Accuracies, $a_{\beta}(l_{\beta}, h)$, versus Number of Features using Algorithm 1

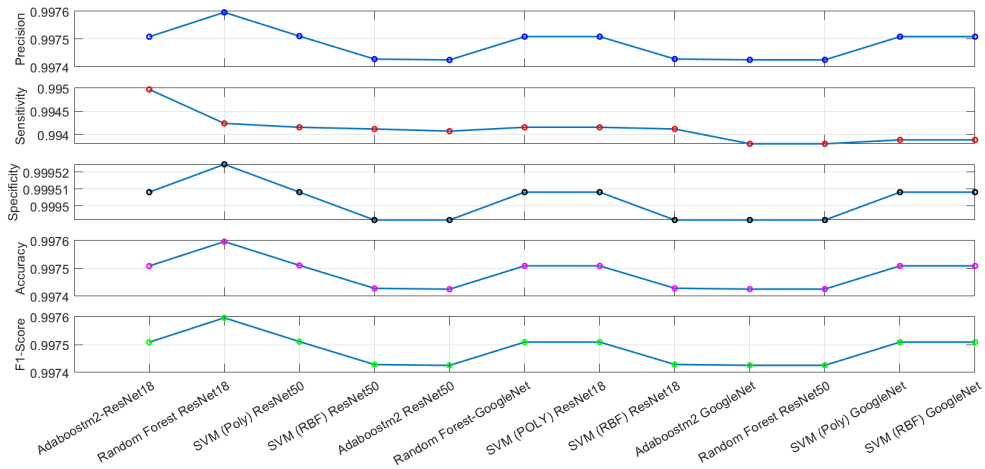

Fig. 6. Plots of Precisions, Sensitivities, Specifitices, Accuracies, and F1-Score vs CNN-Classifier combinations

are linearly added, enabling faster detection of malware(s). Table III presents a tabulated summary of the values depicted in Figure 6. It's shown we achieve improved performance levels by employing weighted majority voting.

The confusion matrix, result from highest accuracy (99.76%), presented in Figure 7 highlights the false positives and false negatives encountered by the classifier. Notably, the FTP class shows minimal false positives, with only two instances incorrectly classified as Nsis-ay and Neris. The Neris class has a slightly higher number of false positives, with seven instances misclassified as Weibo. False negatives are particularly evident in the Zeus class, where 13 instances are misclassified as FTP, contributing to a relatively lower accuracy for this class. The Nsis-ay and Weibo classes exhibit very few false positives and false negatives, with only two and three instances misclassified, respectively. The Facetime class has no false positives or false negatives, achieving perfect classification. Overall, the classifier demonstrates strong performance, but the confusion between the Zeus and FTP classes indicates a potential area for improvement.

VI. Conclusion

This work introduces a novel fusion of machine and deep learning approach for identifying malicious network packets. Our framework consists of three main steps: training compact CNN models with low parameter counts, identifying the most relevant features extracted by these CNNs using a combination of SVD and MRMR, and integrating individual prediction decisions via weighted majority voting. By meticulously selecting features, we achieve improved performance with fewer

TABLE II
BEST INFERENCE TIMES FOR EACH CNN-CLASSIFIER COMBINATION

	[1]AdaB	[1]RF	[1]SVM(P)	[1]SVM(R)	[2]AdaB	[2]RF	[2]SVM(P)	[2]SVM(R)	[3]AdaB	[3]RF	[3]SVM(P)	[3]SVM(R)
Accuracy	99.75	**99.76**	99.75	99.74	99.74	99.75	99.75	99.74	99.74	99.74	99.75	99.75
# of Feats.	110	170	110	110	110	290	130	130	370	250	110	110
Times (s)	0.256	1.437	2.162	1.888	0.285	2.023	1.640	1.536	0.248	1.334	1.272	1.197
PPS	47593	8464	5627	6441	42684	6014	7418	7922	48993	9116	9559	10164

[1] ResNet18, [2] ResNet50, [3] GoogleNet, Packet per Section (PPS)

TABLE III
PERFORMANCE METRICS FOR CNN-CLASSIFIERS

	1	2	3	4	5	6	7	8	9	10	11	12
Precision	99.75	99.76	99.75	99.74	99.74	99.75	99.75	99.74	99.74	99.74	99.75	99.75
Sensitivity	99.50	99.42	99.42	99.41	99.41	99.42	99.42	99.41	99.38	99.38	99.39	99.39
Specificity	99.95	99.95	99.95	99.95	99.95	99.95	99.95	99.95	99.95	99.95	99.95	99.95
Accuracy	99.75	99.76	99.75	99.75	99.75	99.75	99.75	99.75	99.75	99.75	99.75	99.75
F1-Score	99.62	99.59	99.58	99.57	99.57	99.58	99.58	99.57	99.56	99.56	99.57	99.57

[1] AdaB-ResNet18 [2] RF-ResNet18 [3] SVM(Poly)-ResNet50 [4] SVM(RBF)-ResNet50 [5] AdaB-ResNet50 [6] RF-GoogleNet
[7] SVM(POLY)-ResNet18 [8] SVM(RBF)-ResNet18 [9] AdaB-GoogleNet [10] RF-ResNet50 [11] SVM(Poly)-GoogleNet [12] SVM(RBF)-GoogleNet

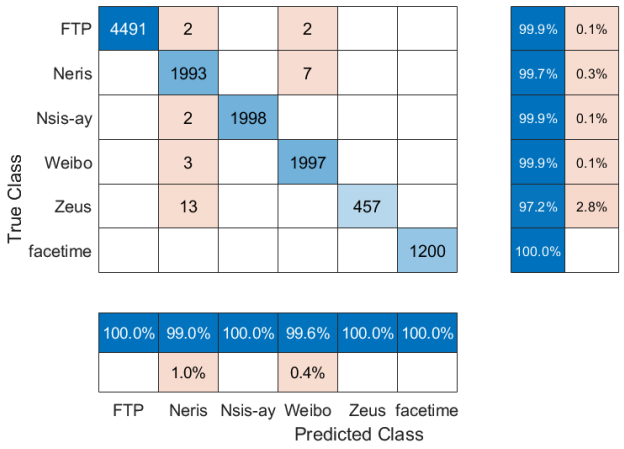

Fig. 7. Confusion Matrix of the highest accuracy achieved.

features, resulting in lower detection times. Leveraging the outcomes of our feature selection process, we enhance accuracy through CNN ensembles. Our pipeline shows promise as an effective malware detector, delivering high accuracy in shorter timeframes.

REFERENCES

[1] Y. Tanaka et al., Analysis of malware download sites by focusing on time series variation of malware, Journal of Computational Science, Volume 22, 2017, Pages 301-313, ISSN 1877-7503.

[2] M. A. Qbeitah and M. Aldwairi, "Dynamic malware analysis of phishing emails," 2018 9th International Conference on Information and Communication Systems (ICICS), Irbid, Jordan, 2018, pp. 18-24.

[3] Ö. A. Aslan and R. Samet, "A Comprehensive Review on Malware Detection Approaches," in IEEE Access, vol. 8, pp. 6249-6271, 2020.

[4] A. H. Lashkari et al., "Towards a Network-Based Framework for Android Malware Detection and Characterization," 2017 15th Annual Conference on Privacy, Security and Trust (PST), Calgary, AB, Canada, 2017, pp. 233-23309.

[5] Talal, M., Zaidan, A.A., Zaidan, B.B. et al. Comprehensive review and analysis of anti-malware apps for smartphones. Telecommun Syst 72, 285–337 (2019).

[6] M. Sewak et al., "Comparison of Deep Learning and the Classical Machine Learning Algorithm for the Malware Detection," 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Busan, Korea (South), 2018, pp. 293-296.

[7] A. Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, Advances in Neural Information Processing Systems, Curran Associates, Inc., 2012.

[8] Wang, Z. The applications of deep learning on traffic identification. BlackHat USA 2015, 24, 1–10

[9] J. Liu et al., Cyberattack detection model using deep learning in a network log system with data visualization. J Supercomput 77, 10984–11003 (2021)

[10] M. R. Smtith et al., "Malware Generation with Specific Behaviors to Improve Machine Learning-based Detection," 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 2021, pp. 2160-2169

[11] W. Wang et al., "Malware traffic classification using convolutional neural network for representation learning," 2017 International Conference on Information Networking (ICOIN), Da Nang, Vietnam, 2017, pp. 712-717

[12] S. Basumallik et al., Packet-data anomaly detection in PMU-based state estimator using convolutional neural network, International Journal of Electrical Power and Energy Systems, Volume 107, 2019, Pages 690-702

[13] F. Osamor and B. Wellman, "Comparative Analysis of LSTM and CNN for Efficient Malware Detection," 2022 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2022, pp. 965-970

[14] M. Rhode et al., Early-stage malware prediction using recurrent neural networks, Computers and Security, Volume 77, 2018, Pages 578-594

[15] Akhtar, M.S.; Feng, T. Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time. Symmetry 2022, 14, 2308

[16] K. He et al., Deep Residual Learning for Image Recognition, arXiv, 10 Dec 2015

[17] C. Szegedy et al., Going Deeper with Convolutions, arXiv, 17 Sep 2014

[18] Y. Li et al., "MS-RMAC: Multiscale Regional Maximum Activation of Convolutions for Image Retrieval," in IEEE Signal Processing Letters, vol. 24, no. 5, pp. 609-613, May 2017

[19] K. Kim et al., A design framework for hierarchical ensemble of multiple feature extractors and multiple classifiers, Pattern Recognition, Volume 52, 2016, Pages 1-16, ISSN 0031-3203.

[20] Ding, C., and H. Peng. "Minimum redundancy feature selection from microarray gene expression data." Journal of Bioinformatics and Computational Biology. Vol. 3, Number 2, 2005, pp. 185–205.

[21] https://github.com/yungshenglu/USTC-TFC2016