Hindawi
International Journal of Intelligent Systems
Volume 2023, Article ID 9544481, 20 pages
https://doi.org/10.1155/2023/9544481

Research Article

Feed-Forward Deep Neural Network (FFDNN)-Based Deep Features for Static Malware Detection

Priyanka Singh,[1] Samir Kumar Borgohain,[1] Achintya Kumar Sarkar,[2] Jayendra Kumar ,[3] and Lakhan Dev Sharma[3]

[1]*Department of Computer Science Engineering, National Institute of Technology Silchar, Silchar, Assam 788010, India*
[2]*Department of Electronics and Communication Engineering (ECE Group), Indian Institute of Information Technology Sri City, Sri City, Andhra Pradesh 517646, India*
[3]*School of Electronics Engineering, VIT-AP University, Amaravati, Andhra Pradesh 522 237, India*

Correspondence should be addressed to Jayendra Kumar; jayendra854330@gmail.com

Received 18 October 2022; Revised 28 December 2022; Accepted 17 January 2023; Published 20 February 2023

Academic Editor: Said El Kafhali

The portable executable header (PEH) information is commonly used as a feature for malware detection systems to train and validate machine learning (ML) or deep learning (DL) classifiers. We propose to extract the deep features from the PEH information through hidden layers of a feed-forward deep neural network (FFDNN). The extraction of deep features of hidden layers represents the dataset with a better generalization for malware detection. While feeding the deep feature of one hidden layer to the succeeding layer, the Gaussian error linear unit (GeLU) activation function is applied. The FFDNN is trained with the GeLU activation function using the deep features of individual layers as well as concatenated deep features of all hidden layers. Similarly, the ML classifiers are also trained and validated in with individual layer deep features and concatenated features. Three highly effective ML classifiers, random forest (RF), support vector machine (SVM), and *k*-nearest neighbour (k-NN) have been investigated. The performance of the proposed model is demonstrated using a statically significant large dataset. The obtained results are interesting and encouraging in terms of classification accuracy. The classification accuracy reaches 99.15% with the internal discriminative deep feature for the proposed FFDNN-ML classifier with the GeLU activation function.

1. Introduction

The statistical data till April 2022 show that the Windows operating system is being used by around 75% of computer users globally [1]. Being the widely used operating system for desktops and laptops in organizations and for personal computing, these systems are more prone to malware infections. Healthcare organizations were highly affected during the peak of the COVID-19 pandemic [2]. The computers of employees working from home (WFH) and educational institutions were attacked severely [3]. Cyber infection is disseminated through PE in various forms stored or installed in the system to perform a variety of malicious actions such as gaining access to its resources, stealing confidential data, and redirecting to malware host websites. Malware attacks have spread terror among the users of the computer and society. With the increase in the use of computer systems, malware has been replicating and proliferating exponentially since the beginning of the computer era. Despite the security concerns raised by the industry to safeguard computers and networks from cyber-attack, privacy and security are still of high concern. To halt this nuisance created by the malware creators, continuous observation and prevention are needed. Also, the systems need to be updated in real time. Nevertheless, as malware has evolved into the form of viruses, worms, Trojans, and its variants, different malware detection techniques have been developed by researchers.

Traditional malware detection such as the signature-based approach has become outdated as it can only detect malware if the pattern or the signature is matched against the signature stored in the repositories [4]. These signatures are

the unique patterns generated during the static execution of the malware executables [5]. Also, code obfuscation and shuffling of the code snippet are no more backbreaking jobs for the creator of the malware that results in the signature change of the same malware variant [6]. The behaviour-based approach evaluates malware dynamically executing the thief and analyzing it in an isolated environment. The behavioural log such as the execution sequence of the malware is recorded for further prosecution [7]. However, the direct execution of malware on a system for monitoring is always dangerous as it corrupts system files. On the other hand, the newly evolved malware had become smart enough to bypass the isolated environments [5]. The heuristic-based approach performed better than the signature and behaviour-based ones for detecting unknown malware and zero-day attack [8]. This approach has sufficient rules and patterns for the investigation of malware behaviour and controls the flow of the code. However, a higher false-positive rate is identified as a major limitation of the heuristic-based malware detection systems [9]. With the increasing number of computer users and the availability of advanced malware creation tools, malware detection became trivial. This led to the proliferation and speedy propagation of malware all over the globe leading to a massive amount of malware data. Researchers and industry have now adopted the data mining and machine learning (ML) approach to detect and classify malware [10, 11]. Rapid research contribution has been recognized in the last few decades in the field of malware detection using the ML approach.

In ML techniques, data in terms of features are fed to train a model to learn and estimate a possible pattern of the data. Therefore, feature extraction is an important step in ML while preparing a dataset [12]. Definitely, a better representation of features makes the training and detection highly efficient. Nowadays, a drastic increase in malware has made the training phase time-consuming and tedious and ML classifiers inefficient [13]. The effectiveness of an ML classifier highly depends on the feature representation and the nature of the dataset used for training. The ML approach depends on feature extraction which requires proficient knowledge of the realm. It makes the false-positive rate (FPR) low which is an important consideration during the implementation of ML algorithms [14].

Research in the realm of malware detection has embraced the diverse computational approach. In recent years, the nature-inspired computational approach has been geared up for malware detection, feature optimization, and classification [15–17]. That nature has been always very successful in solving complex objectives which inspired researchers to adopt bioinspired algorithms. The bioinspired algorithms have been successfully implemented for various applications such as image processing, robotics, fraud detection, and intrusion detection. Neurocomputing or artificial neural network is a subfield of nature or bioinspired computation that has been applied for malware detection and classification on windows, android systems, and IoT devices [18, 19]. However, the study and discussion about malware detection for android and IoT devices are out of the scope of this work. As with the availability of a massive amount of malware data to experiment with, deep neural networks or deep learning mimicking the human brain solves the problem of processing information from a huge volume of data. This deep bioinspired computation provides more and more generalization as an effect of the processing of neurons. As bioinspired evolutionary computing has gained momentum in the field of optimization, the deep learning techniques inspired by human brain functioning are the best learners in the context of generalization [20, 21]. Deep learning has been used in the field of cybersecurity in recent times for the prediction of cyberattacks. As the human brain learns to perceive, represent, and process the data, a deep neural network learns data representations to generate the model with the representations and utilizes this learning efficiently when new data come. This improves the generalization capability of the model as compared to other bioinspired methods. Also, scalability is no more an issue to performance as the huge volume of data can be handled without being more careful about feature engineering steps.

Different malware detection models based on DNN or ML classifiers such as random forest (RF) [22], support vector machine (SVM) [23], and *k*-nearest neighbour (k-NN) [24] classifiers use PEH, signature, behavioural log such as the execution sequence as features for malware detection. Credit card fraud detection with DNN [25] and optimization of model hyperparameters using genetics and swarm algorithms can be found in [26–28]. In recent years, deep feature extraction of DNN has gained attention in many domains of pattern recognition [29–32]. In deep feature extraction, the DNN is trained using input data such as PEH with an activation function to minimize the classification error. Now, the output of the subsequent layer without the activation functions is considered deep features. Now the deep features of a hidden layer are used to train ML classifiers. As per our survey, the deep features have not been exploited much for malware detection. Therefore, we propose to extract deep features of an FFDNN with a multitask objective function that simultaneously classifies benign and malware. Thereafter, the deep features of hidden layers have been used to train the ML classifiers, RF, SVM, and k-NN. Besides, we also consider the concatenation of different deep features for malware detection. The network parameters are optimized using error backpropagation, and the output of DNN neurons and gradient contribution is controlled by the activation function. The nonlinear modelling capability of deep neural networks possesses the significant discriminative capability and is a potent back-end classifier.

In this work, we propose a deep feature-based malware detection system where an FFDNN [33] is trained with multitask objective functions. During the network backpropagation for parameter optimization, the GeLU activation function is used to control the DNN neuron's output and gradient contribution. The sigmoid [34, 35] and rectified linear unit (ReLU) [36] activation functions are commonly used for different applications such as speech recognition [37, 38] and image processing [39, 40] in the literature. The sigmoid function squashes the input space from 0 to 1. This results in a small derivative and gradient vanishing. The gradient vanishing poorly updates the initial layers yielding

ineffective training; thus, the model lack generalization [41, 42]. The ReLU has a dynamic range of input and can be used for effective training, but it needs stochastic regularization as it lacks probabilistic interpretation [43]. The GeLU is introduced in [43] as a deterministic activation function that combines stochastic regularization. It is shown in [43] that the GeLU performs better than the rectified linear unit (ReLU) and exponential linear unit (eLU) for speech and language processing and computer vision applications. We have investigated the performance of different activation functions and found GeLU best performing for the proposed malware detection system too.

2. Contributions

The contributions of this work are in many folds as follows:

(i) We propose a DNN-based deep feature for a malware detection system, where PEH information are fed as input to the FFDNN with multitask objective functions at the output layer. It simultaneously classifies the malware and nonmalware sample and minimizes the prediction for the input at the output. Afterward, the output from a particular hidden layer is used as deep features. This deep feature representation is used to train the FFDNN and ML classifiers, RF, SVM, and KNN.

(ii) The deep features of one hidden layer are extensively analyzed and also concatenated with deep features of other hidden layers.

(iii) To the best of our knowledge, the GeLU activation function has been used for the first time in combination with DNN and ML classifiers for static malware detection. The performance is evaluated against the commonly used ReLU, scaled exponential linear units (SeLU), and eLU on a large dataset for statistical significance.

(iv) The proposed deep feature-based FFDNN-ML malware detection system shows better performance over the state of the art

3. Related Work

A 2-hidden layer DNN-based malware detection system is reported in [44], and results are presented against a binary dataset of 431,926 instances. Entropy histogram, PE Import, 2D strings, and PE metadata features have been used as extracted features. A better convergence was achieved with the parametric ReLU (PReLU) activation function and Adam optimizer. This system achieves a 95% detection rate (DR) with a 0.1% false-positive rate (FPR) with 1024 neurons in the input layer. An interesting time split validation performed over a traditional cross-validation approach is appreciable. However, the performance of time split validation has been depicted as degrading on test data. In [45], a staked autoencoder (SAE)-based DNN system for malware detection is reported. The system performance is demonstrated using the Windows API calls for a dataset consisting of 50,000 malware and benign executables. They achieved an accuracy of 96.85% with 3 hidden layers of 100 neurons per layer. However, the implication of the activation function may further improve the performance of this system. Yuxin and Siyi detected malware using a malware detection system based on the deep belief network (DBN) model with 2 hidden layers and a maximum of 200 neurons [46]. Performance evaluation is carried out by partitioning an OpCode-*n*-gram dataset into 4 sets of 850 malware and 850 benign samples. An accuracy of 96.5% is achieved using 200 distinct features. DBN is stacked up with restricted Boltzmann machine (RBM) layers that are related to global weight and tuning of the weights. However, DBN suffers a vanishing gradient problem [47]. Thus, the nonlinear activation function in DBN is critical in reducing the network complexity and improving the performance of pattern recognition.

DeepMalNet architecture for malware detection has been reported in the literature [48]. This malware detection system is evaluated on the static representation extracted from portable executables. The Ember dataset [49] used in this work is a labelled benchmark dataset. The proposed work is trained using 10 hidden layers, an input layer with 2350 neurons and 1 output neuron. Experiments were performed on activation functions such as sigmoid, tanh, ReLU, eLU, and SeLU. The learning rate between 0.01 and 05 has been applied to the experimentation during 1000 iterations. An accuracy of 98.9% is achieved with the ReLU activation function with 10 hidden layers of DNN. However, the elaborated experimentation results with various parameter tuning are not mentioned in the work. Ye et al. proposed a heterogeneous deep learning framework for intelligent malware detection known as DeepAM [50]. This framework is composed of SAE multilayer restricted Boltzmann machines and a layer of associative memory. To train the model, a balanced dataset consists of 4500 benign files and 4500 malware files that have been used and tested using 1000 samples. WindowAPI call sequence was extracted for feature representation. For fine-tuning, a wake-sleep algorithm and a gradient descent algorithm are applied to modify the downward generation of weights between the layers. An accuracy of 98.82% is achieved with 3 hidden layers of 100 neurons in each layer. However, weight updating is not as smooth as backpropagation in FFDNN [51]. Rathore et al. proposed a malware detection system with an ML classifier and deep learning that used autoencoders for dimensionality reduction [52]. The dataset consists of 14,507 samples (2819 benign and 11308 malware executables). OpCode frequency was used as a discriminatory feature for evaluation. The mean square error (MSE) is used as a loss function, and the network is trained for 120 epochs. An accuracy of 98.9% is achieved using a 7 hidden layer DNN model.

The malware visualization approach using a convolution neural network (CNN) for malware detection and classification has been trending in malware research as state of the art. Zhong and Gu proposed a multilevel deep learning system (MLDLS) where multiple deep learning models are organized in a tree structure [53]. Both static and dynamic features are extracted from malware and binary executables

to develop a dataset with 2242,234 malware samples and 3425,176 benign samples. The major contribution is parallel computation by data distribution for each group of malware. MLDLS outperforms the other single deep learning models. A leaky ReLU activation function is used in the overall experiment. The model construction time is significantly less due to the parallel distribution of data and organization of the tree structure. As the number of samples is increased, the true positive rate (TPR) is significantly increased above 96% and the false positive rate (FPR) is decreased below 0.1%. However, the computational time is considerably high. Also, the conversion of tabular data to images using CNN has certain limitations [54].

In [55], a DNN malware detection system based on visualization is presented. The malware detection system developed using deep learning architectures has been implemented on static, dynamic analysis, and image processing in the work. The authors have implemented a hybrid deep learning architecture incorporating CNN, recurrent neural network (RNN), and fully connected DNN for malware detection. In this work, various models both traditional and advanced ML models are trained for performance evaluation. The FCNN (DNN) for static analysis outperforms other models with an accuracy of 98.9%. However, the robustness of the proposed system framework is not discussed in this work. Authors in [56] proposed a malware detection system using convolution neural network-based VGG16 network functions for visualization of static and dynamic features of executables.

The performance evaluation of the system is carried out for two models based on static and hybrid visualization. The hybrid model outperforms the static model with an accuracy of 94.70% on the test dataset. However, the experimentation is performed on a small dataset that is statistically insignificant. In [57], a similar visualization approach has been used. The PE executable is converted into a grayscale image, and then the image patterns are learned for classification. The authors proposed a deep forest method applying sliding windows and cascading layers wherein the sliding window approach is adapted from CNN. The highest accuracy obtained is 98.65% on the Malimg dataset. The malware visualization technology provides a vivid image of different malware types that aids in visualizing the difference between variants of malware. Although, the visualization method can tackle the code obfuscation problem but requires more time for the extraction of complex image patterns. Also, the transformation of tabular data to images suffers certain limitations.

4. Bioinspired Neural Network

Human brains possess superiority in executing tasks such as pattern recognition, flexible inference, control, perception, intuition, prediction, and decision-making. An artificial neural network is loosely inspired by the human nervous system and focuses on the learning and problem-solving skills of the system at hand. The neurons are the fundamental computation unit of the human brain connected to each other with a minute junction called the synapse. These junctions are responsible for cognitive abilities, such as perception, inference, and thought processing. Neurocomputation is brain-like computing and also referenced with artificial neural network (ANN) which is a subfield of bioinspired computation [20].

Deep artificial neural networks or deep learning architectures imitate the pursuit in layers of neurons in the neocortex. It intelligently learns the stratified layers, representation levels, and abstractions. It comprehends the patterned data that it perceives from various sources such as numerical data, images, sound, signals, and text. Abstraction at a higher layer is the combination of abstractions of lower layers. These deep abstractions are generated from more than a single layer of nonlinear feature transformation. Deep learning automatically learns the feature representation at multiple layers of abstraction that aids in the automated mapping of the functions from the input to the output layer. This automated mapping does not require expert knowledge of feature engineering.

4.1. Neural Perspective. The study of the human nervous system is a vast domain. In this work, the organization and processing of the human nervous system are discussed in the context of a computational artificial neural network. Different levels of the nervous system can be organized as molecules, synapses, neurons, layers, maps, and systems. Out of the complex organization, the neurons are more evident and potential units for signal generation and capable of transmitting information to connected cells. Each neuronal unit performs its own specific function. One of such eminent units for comprehension of signal processing is synapses in the nervous system [58].

Neurons consist of special extensions called neurites that can be divided into dendrites and axons. The function of the dendrites is to receive signals from other neurons. The axon performs propagation of the output signals to the connected neurons. The neurons are specialized in sending and receiving signals to and from other neurons. The neurons that send the signal are known as *presynaptic* neurons, and the neurons that receive the signal are known to be *postsynaptic*. Thus, *synapses* are the junctions between the presynaptic and postsynaptic neurons. An equivalent nervous system representation and its components with directional signal processing are depicted in Figures 1(a) and 1(b), respectively. The propagation of the signals up to the cell body from presynaptic neurons is integrated after signal generation. The potential membrane is then responsible for making a decision whether the neurons are firing or sending the output signal to postsynaptic neurons depending on the neuron *threshold*. The connection between the neurons can be either forward or backward in the nervous system. These interconnected neurons are called neural networks. In a network of neurons, these neurons are organized into input, hidden, and output layers.

ANN maps the architecture, features, and performance criteria similar to the human nervous system. The fundamental information processing occurs in neurons or nodes wherein neurons can exchange signals among other

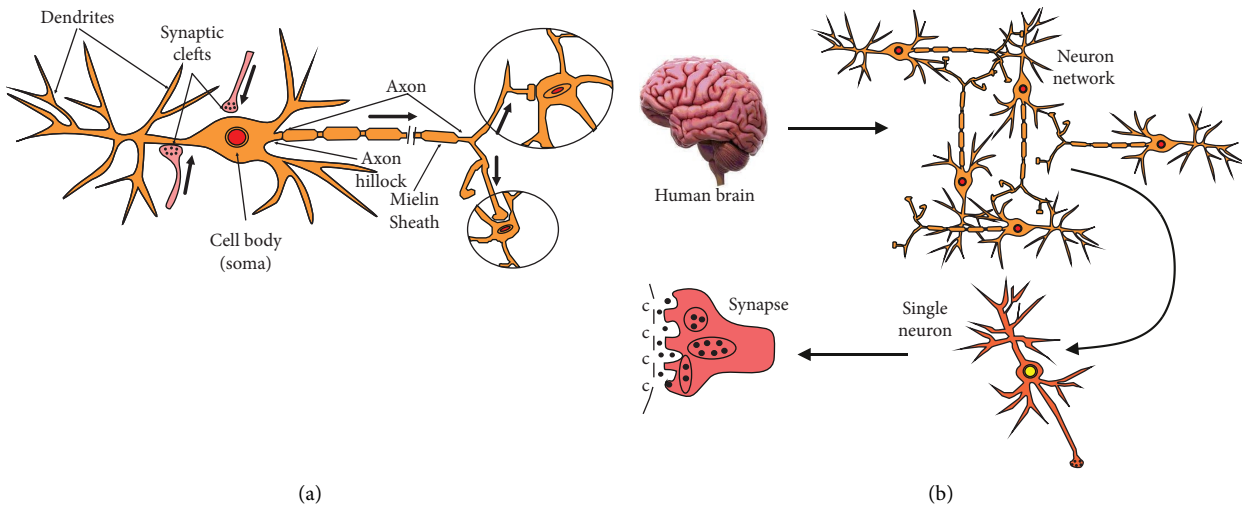

FIGURE 1: Illustration of neurons and neural network equivalent representation in the nervous system. (a) Structural representation and signal flow. (b) Level-wise structural components in the nervous system.

neurons. The neurons are connected to each other forming a neural network. The strength of the synapses corresponds to the weight value in the neurons. The learning process from the given data for adaptive knowledge is similar to the learning of the nervous system from the environment. However, the computational neural network does not fully resemble the complete structure and working of the nervous system.

4.2. Representation Perspective. The interconnected neurons demonstrate complex behaviours and information processing capabilities even in a small cluster. Single neurons cannot store and process complete or whole knowledge. The representation of information is distributed in a hierarchical manner, and the parallel processing of this information over various neurons is an important feature of the neural network [59]. In order to perform the more specific complex tasks with respect to information processing, specialized architectures are incorporated into the larger structure of networks. For most bioinspired artificial intelligence (AI) and ML algorithms, the performance heavily depends on the representation of data. So, feature representation plays a crucial role in the performance of the system designed [60, 61]. Traditionally, the handcrafted predefined feature as sectional header information shown in Figure 2 is fed to DNN/ML algorithms for classification.

Recently, DNN has gained the attention of researchers in different applications including speech and image processing. It is demonstrated by researchers that a system based on the deep representation feature yields better or provides complementary information. In other words, deep learning is found very useful for data learning, i.e., the mapping between the input and the output. In this concept, traditional (say, handcrafted) features are fed to train a DNN with an objective function to classify the target at the output layer (need label information, i.e., supervised) and predict the input at the output (unsupervised, i.e., no need label information) or combination of them. It is believed that different layers of the DNN capture different attributes for a particular task in high dimensional space. So, the output from the particular hidden layer of the DNN for a given input is extracted as a *deep feature*. The deep features are then used for classification using ML techniques. Another advantage of DNN is that it can handle a large amount of data by manipulating its model parameters. However, human-crafted data lacks flexibility and cannot be applied to diverse scenarios. Also, systems using such data cannot adapt to new data leading to poor generalization ability. This has been a powerful motivation for crafting flexible and automated feature representation methods.

The evolution of deep learning can be considered as representation learning, wherein the process of feature extraction is automated when the deep architecture processes the data, learns, and understands the mapping between the input and the output. Deep learning is a data-driven process, as the model learns from the data presented. This aids in a significant boost in performance as human-developed feature extraction is deficient in accurate detection and generalization. Besides automation in feature extraction and feature learning, the learned feature representations occur in a distributed and hierarchical manner. The feature learning task is distributed over the layers each one computing its allotted task. In a distributed dense feature representation, multiple neurons concurrently represent a learned parameter.

5. Methodology

In this section and the subsequent subsections, we elaborately describe the proposed system for malware detection. It consists of FFDNN, deep feature extraction, classification using DNN, and different ML techniques. The overall system is illustrated in Figure 3. The components of the proposed DNN architecture are briefly discussed as follows. The experimentation of the proposed work starts with a traditional approach of training the ML classifiers and performing

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers…	Characteristics
Byte [8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	0000122C	00001000	00001400	00000400	00000000	00000000	0000	0000	60000020
.data	0000034C	00003000	00000200	00001800	00000000	00000000	0000	0000	C0000040
.idata	00000664	00004000	00000800	00001A00	00000000	00000000	0000	0000	40000040
.rsrc	00000420	00005000	00000600	00002200	00000000	00000000	0000	0000	40000040
.reloc	00000148	00006000	00000200	00002800	00000000	00000000	0000	0000	42000040

Figure 2: Illustration of some header/attribute information of the files in the dataset.

classification with the k-fold method. As our objective is to deal with a large volume of current and future generations of data, these ML classifiers alone are not sufficient. Thus, the DNN and ML classifiers are effectively integrated and discussed in the following sections.

Initially, exclusive RF, SVM, and k-NN ML classifiers have been trained and tested for classification accuracy. The performance of these classifiers has been found to be poor due to a large number of instances, and this motivated authors to investigate the bioinspired DNN for efficient classification. The deep features of each layer are exploited in independent capabilities and concatenation. The proposed FFDNN-ML malware prediction and classification model is implemented in two-phase with the concept of feature concatenation and deep feature extraction. In both phases, feature concatenation is applied in the context of feature representation learning for classification. In the first phase, the features generated from each hidden deep layer are finally concatenated to train the FFDNN for prediction and classification. In the second phase, deep features of hidden layers are applied separately or in concatenation to the ML classifier, RF, SVM, and k-NN ML with parametric tuning for classification. This extracted deep feature provides better generalization for unrelated and untrained data. Feature concatenation effectively sums up the features with different criteria that aid in an efficient classification process.

5.1. FFDNN. The FFNNs have been used for training the model for multitasking objectives [62–64]. In a fully connected layer also called a dense layer, all the neurons in a layer are connected or mapped to each and every neuron in the previous layer. The input vector, $x = [x_1, x_2, \cdots x_m]$, is multiplied with a $m \times n$ matrix of weights, and the output result from the layer is submitted to a nonlinear activation function. The weight matrix W is given by matrix 1.

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & \vdots & \dots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix}. \quad (1)$$

The neuron functions as a combinatory circuit perform the scalar product of the input vector x and the weight vector $w = [w_1, w_2, \cdots w_m]$. The result generated is applied with the activation function f. The whole neural network generates the output vector $y = [y_1, y_2, \cdots y_n]$. The activation function remains constant for each neuron. Thus, each NN is characterized by the weight matrix.

In this method, handcrafted PEH information is fed to train an FFDNN with GeLU activation using multitask objective functions: we simultaneously (i) discriminate the malware and nonmalware classes and (ii) predict the input at the output layer of the DNN using the cross-entropy (L_{ce}) and L_1 loss functions, respectively. At the last hidden layer of the DNN, the output from the previous different hidden layers is first concatenated and then projected to the output layer. The overall loss (L) function of the proposed FFDNN can be expressed as follows:

$$\begin{aligned} L &= \gamma L_{ce} + (1-\gamma) L_1 \\ &= -\gamma \frac{1}{N} \sum_{i=1}^{N} z_i \log p(X_i) \\ &\quad + (1-\gamma) \frac{1}{N} \sum_{i=1}^{N} \left\| X_i - \widehat{X}_i \right\|, \end{aligned} \quad (2)$$

where z_i and $\widehat{X}_i$ indicate the label and predicted value of X_i sample, respectively. γ decides the weight between the loss function. In our experiment, we provide equal weight to both loss functions. For analysis, we perform the experiment for different numbers of hidden layers (varies from 3 to 20), activation functions (ReLU, eLU, and SeLU), and the number of neurons per layer (varies from 32 to 256) in the proposed method.

5.1.1. Perceptron. Perceptron is the simplest and basic ANN architecture. Rosenblatt [65] proposed the perceptron training algorithm that was inspired by Hebb's rule [66]. According to Donald Hebb, when a human neuron activates another neuron, bonding between the neurons becomes more and more strong. This ANN accepts one data point at once to make a prediction and uses the perceptron learning rule (PLR) to minimize error for any incorrect prediction during training. The PLR is presented in equation (3). One training instance is fed at a time to perceptrons to make predictions. For a wrong prediction of an output neuron, the perceptron learning rule strengthens the connections for the reduction of the error generated. This reinforcement rule contributes toward the correct prediction. The PLR is presented in the following equation:

$$w_{i,j}^{(\text{next})} = w_{i,j} + \eta \left(y_j - \widehat{y}_j \right) x_i, \quad (3)$$

where $w_{i,j}$ is weight connecting the i^{th} neuron and j^{th} neuron, x_i is the i^{th} instance for training, y_j is the output of the j^{th} neuron, $\widehat{y}_j$ is the predicted output value of the j^{th} neuron to generate output, and η is the learning rate used during PLR.

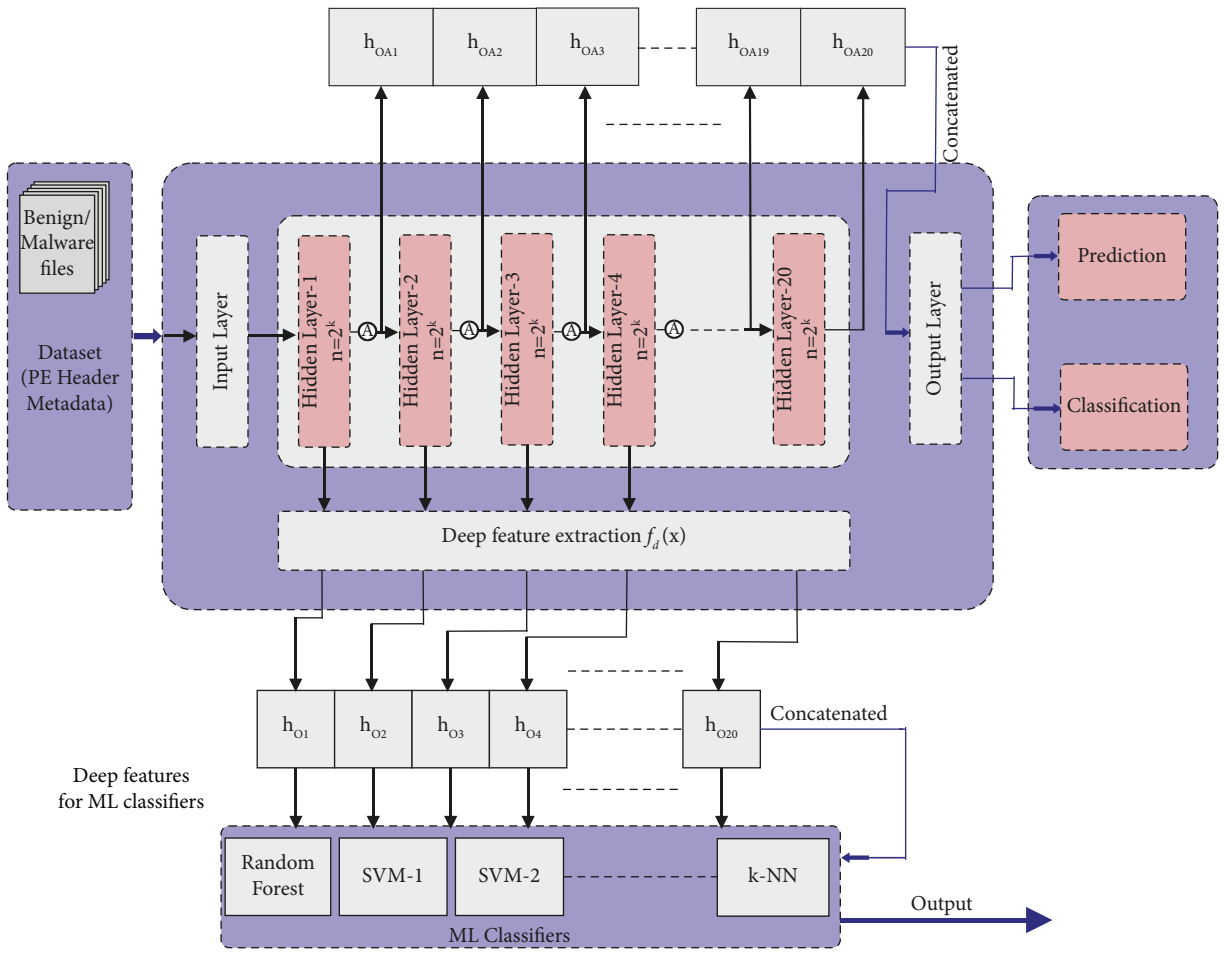

Figure 3: Depiction of the proposed bioinspired deep neural network static malware detection system.

This basic ANN model is unable to learn complex or nonlinear patterns for decision-making as the plane separating each class is linear. The limitation of the perceptron can be covered by using multilayer perceptron (MLP) ANN architecture. The MLP with various modifications is being used in deep learning research. Fundamentally, it is organised with an input layer, an output layer, and one or more hidden layers existing between the input and output layers. An MLP with all-to-one mapping between the neurons of subsequent layers is known as a fully connected neural network (FCNN) in deep learning as shown in Figure 4. An MLP with a deeper stack than one hidden layer is a deep neural network. The decision boundary for each output neuron or class is linear, so this basic perceptron model is unable to learn complex or nonlinear patterns for decision-making. The limitation of perceptrons can be covered by using multilayer perceptron (MLP) ANN architecture. The MLP with various modifications is being used in deep learning research. An MLP is organized with one input layer, one or more layers of hidden layers, and one final layer called the output layer. The MLP with all-to-one mapping between the neurons of subsequent layers is known as a fully connected FFNN in deep learning and is depicted in Figure 4. All layers excluding the output layer consisting of a bias neuron are fully connected to the next layer. An MLP with a deeper stack than one hidden layer is a deep neural network.

5.1.2. Backpropagation. During the network training, the output either for a single or multiobjective task is generated in the forward pass. This generated output value may be different from the original value. So, the backpropagation of an error signal is evaluated to minimize the cost function. For this proposed work, cross-entropy function (L_{ce}) and L_1 loss function are used to evaluate the classification probability and minimize the reconstruction error, respectively. The reason behind applying L_1 loss for this work is that the data have been used without preprocessing which reduces feature preprocessing overhead in deep learning. So, the data may contain outliers, and the L_1 loss function is not affected by the outliers present in the dataset. For multitask objectives, both the cost function for classification and prediction needs to be backpropagated to strengthen the connection. Cross-entropy function (L_{ce}) and L_1 loss function are expressed in the following equations:

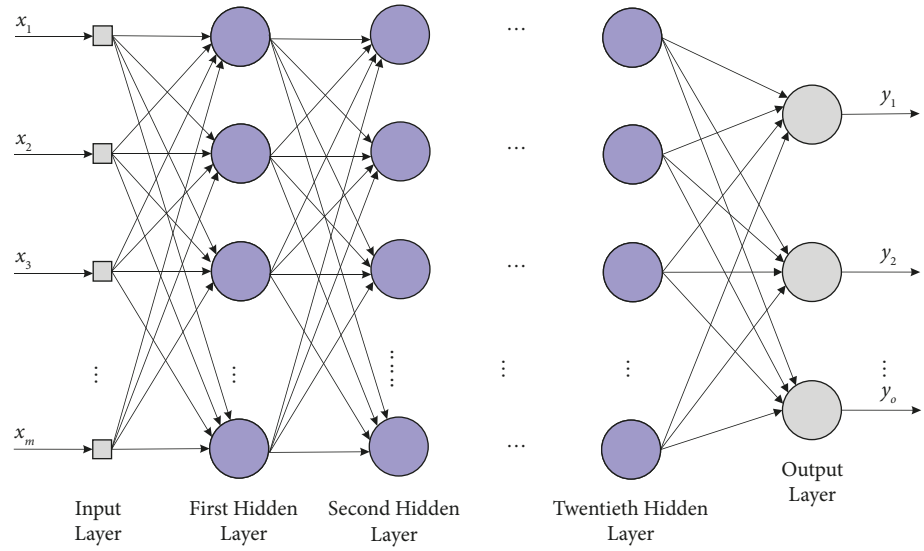

Figure 4: Illustration of an FFDNN network architecture with input, hidden layer, neurons, and output.

$$L_{ce} = -\gamma \frac{1}{N} \sum_{i=1}^{N} z_i \log p(X_i), \tag{4}$$

$$L_1 = (1-\gamma) \frac{1}{N} \sum_{i=1}^{N} \left\| X_i - \hat{X}_i \right\|. \tag{5}$$

A schematic diagram of the backpropagation technique is depicted in Figure 5. The working of the backpropagation algorithm is discussed henceforth. The full training is completed in batches in multiple iterations wherein each iteration is known as an epoch. The training instances are fed to the input layer that integrates the required parameters with each instance and fires it to the initial hidden layer as per activation. This forward passing of the feature vectors with required bias, weight, and activation function from one input layer to the next or a series of hidden layers and eventually to the output layer to generate the target output value is forward propagation. This phenomenon is also known as forwarding pass. The loss function is used to calculate the error between the predicted output value and the labelled output. For optimizing the value for better performance and generalization of the model, backward propagation or backward pass is applied during the training. The algorithm analyses and computes how much each connection weight and each bias parameter can be altered for error reduction by applying the chain rule which helps the backpropagation step fasten accurately. Backpropagation minimizes the cost function by fine-tuning the network's weights and biases. For adjusting this parameter, the gradients of the cost function with respect to weights and bias need to be computed using the chain rule given in the following equation [61], where n_i is the i_{th} node in the network.

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial n_i} + \frac{\partial n_i}{\partial x}. \tag{6}$$

5.1.3. GeLU Activation Function. The initial perceptron used binary threshold units [67]. The binary decision was further flattened using sigmoid activation functions and trained with backpropagation [68]. As the network grew deeper and deeper, the sigmoid function became less effective due to nonlinearity [61]. ReLU succeeded over the sigmoid which is preferred mostly due to its fast and better convergence property. Then, the eLU activation function which is a modification of ReLU was developed which improved the training speed on negative values [69]. The prediction of the deterministic decision during the evaluation of the network leads to new nonlinearity. The stochastic regularizer yields nonlinearity for an input x that randomly applies the identity or zero maps to a neuron's input. The Gaussian error linear unit (GeLU) is a high-performing neural network activation function [70]. Unlike signs in ReLU, the inputs are weighted in magnitude by the GeLU nonlinearity. The GeLU activation function is $x\Phi(x)$, where $\Phi(x)$ is the standard Gaussian cumulative distribution function. The GeLU function expressed using equation (7) is estimated using equation (8).

$$\text{GeLU}(X) = xP(X \le x) = x\Phi(x), \tag{7}$$

$$\text{GeLU}(X) = 0.5x\left(1 + \tanh\sqrt{\frac{2}{\pi\left(x + 0.044715x^3\right)}}\right). \tag{8}$$

5.1.4. Deep Feature. In this step, the handcrafted feature vector for a given input $X = \{x_1, x_2, \ldots, x_T\}$ is fed to the DNN obtained in Section 5.1. Then, the output from the particular hidden layer (without applying the activation function) is extracted as a deep feature (for the particular input). It can be expressed as

$$X_{DF-h} = f_{FF-\text{DNN}}(X), \tag{9}$$

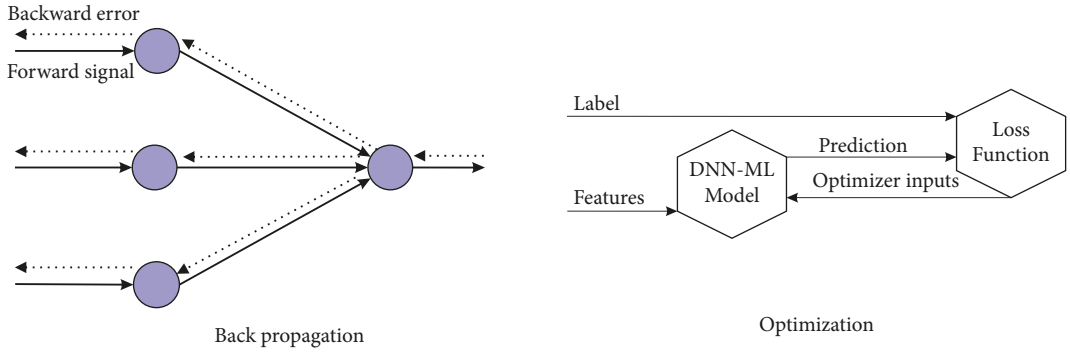

Figure 5: Illustration of backpropagation and optimization processes in a neural network.

where X_{DF-h} denotes the deep feature from the h^{th} hidden layer of the DNN for input X. This basically represents the mapping of the input data from one space to another. In this manner, deep features are extracted for all input data in the experiments. In the case of 512 neurons, per hidden layer in DNN yields the 512 dimension deep feature vector. Besides, we also consider the study of the concatenation of deep features extracted from the different hidden layers as a presentation. For example, $DF(1-5)$ indicates the concatenated deep feature of the hidden layers 1 to 5. If each hidden layer has 512 neurons, then $DF(1-5)$ gives 5×512 dimensional vector.

The deep feature extracted from each hidden layer of the FFDNN has better representation and generalization. In the final step, deep features extracted from each hidden layer are then used layer-wise and in concatenation for the classification of malware using RF, SVM, and KNN classifiers. The deep feature extracted from each hidden layer of the FFDNN constitutes a malware classification system.

6. Experimental Setup

Experiments are conducted on the dataset [71], and details are presented in Section 6.1. The datasets are randomly split into $n = 10$ folds. At a time, one set is considered for evaluation, and the rest are used for evaluation, i.e., one split is leave out at a time. The process is continued until all the data split evaluation is completed. For the FFDNN, the number of hidden layers (3, 10, 15, and 20), the number of neurons (32, 64, 128, 256, and 512) per layer, and different activation functions (GeLU, ReLU, eLU, and SeLU) are varied to study for the analysis of the system performance. The value of the drop-out rate, regularization parameter, number of epochs, learning rate, and batch size are considered, respectively, 0.01, 0.0001, 30, 0.001, and 1024. The framework of the proposed DNN and ML classifier-integrated static malware detection model is shown in Figure 3. In phase-I, the multitask objective is implemented using a feed-forward neural network (FFNN) with backpropagation. The objectives to be obtained are the prediction and classification of malware and benign sample. The different training network is designed with 3 to 20 hidden layers excluding input and output layers for evaluation. The number of neurons or units in each hidden layer is set from $n = 2^5$ to 2^9 to verify the performance enhancement and robustness of the training network. During the implementation of each network training session, the number of neurons is increased to evaluate the classification error and reconstruction error. The output of the network is generated by the softmax layer implementing the softmax function. Feature dimension vectors generated from each hidden layer are concatenated after the last hidden layer output generation and used to train the DNN and ML classifiers. These feature vectors after being generated from the respective hidden layer are passed on for activation by applying the activation function. Popular DNN activation functions such as ReLU, SeLU, eLU, and GeLU have been investigated in combination with the traditional DNN approach with neurons $n = 2^5$ to 2^9 to choose the best performing one. GeLU is identified as a suitable and high-performance activation function for static malware detection. The concatenated features are then fed as input to the softmax layer for evaluation. The loss function, L_1, is applied for the calculation of prediction loss and the difference between the predicted output value and the actual output value or label value. The reason for applying the L_1 loss function is that no previous data preprocessing is required for the data. For classification, the cross-entropy function is applied. After obtaining the error, the network is trained using the backpropagation algorithm for optimizing the weight and bias to improve the reconstruction loss. The Adam optimizer is used as an optimizer for faster convergence.

6.1. Dataset. The dataset used in this work consists of the metadata from the portable executable header. The sources of the dataset are discussed in [71]. Each feature value has its own function regarding the type of file. The malware dataset consisting of 138048 instances and 54 features can be represented in a 138048×54 matrix, as $X \in \mathbb{R}^{138048\times 54}$ presented in matrix 11.

$$X = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_{54}^1 \\ x_1^2 & x_2^2 & \cdots & x_{54}^2 \\ \vdots & \vdots & \cdots & \vdots \\ x_1^{138048} & x_2^{138048} & \cdots & x_{54}^{138048} \end{bmatrix}. \tag{10}$$

The dataset used in this work consists of 138048 data points and 54 dimensions. These features are extracted from the header of the PEH of malware and benign software for the Windows operating system. 41324 benign executables have been extracted from .exe and .dll files of the Windows operating system. 96,724 malware executables have been extracted from data available on Virus Share. The malware and benign samples are labelled as "0" and "1," respectively, after extraction. The features can be extracted statically from the PEH using the pefile module in python. The values in the header can be viewed using tools such as CFF-Explorer.

6.2. DNN Classification Task. Different classification components such as the softmax function, regression classifier, and cost function have been expressed in [61]. For a given instance x, the softmax classification model initially computes the score $s_k(x)$ for each class k given in equation (11). Then, softmax function is applied to the initial scores to estimate the probability for each class, where $\theta^{(k)}$ is the parameter vector of each class.

$$s_k(x) = X^T \theta^{(k)}. \tag{11}$$

The probability of class k and $\widehat{p}_k$ is computed by applying the scores to the softmax function given in equation (12). The exponential of each score is computed, and then normalization is applied to the function. The calculated scores are termed as logits.

$$\begin{aligned} \widehat{p}_k &= \sigma(s(x))_k, \\ &= \frac{\exp\left(s_k(x)\right)}{\sum_{j=1}^{K} \exp\left(s_j(x)\right)}, \end{aligned} \tag{12}$$

where k is the total number of classes, $s(x)$ is a vector with the scores of each class for the instance x, and $\sigma(s(x)_k$ is the estimated probability that x belongs to class k. The class with the high estimated probability that is the class with the highest score is predicted using the softmax regression classifier expressed in the following equation:

$$\widehat{y} = \operatorname{argmax}\left(\sigma(s(x))_k\right). \tag{13}$$

To generate a deep learning model that can calculate both high and low probability for each class, minimized cost function is required. Cross-entropy as a cost function given in equation (14) can be used to match the estimated class probabilities to the target class. Here, the objective is to generate a deep model that estimates a high and low probability for each target class, respectively. In order to achieve this objective, minimizing the cost function is required. Cross-entropy as a cost function given in equation (14) can be used to match the estimated class probabilities to the target class.

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log\left(\widehat{p}_k^{(i)}\right), \tag{14}$$

where $y_k{}^i$ is the target probability and i^{th} instance belongs to class k. The gradient vector of the cross-entropy cost function with respect to $\theta^{(k)}$ is expressed in the following equation:

$$\nabla_{\theta^{(k)}} J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\widehat{p}_k^{(i)} - \widehat{y}_k^{(i)}\right) x^{(i)}. \tag{15}$$

Once the gradient vector is computed for each class, the Adam optimizer is used to find the parameter Θ to minimize the required cost function.

6.3. Performance Enhancement. The training of the DNN with a large number of hidden layers can be an exhaustive task. To speed up the training process that aids in performance enhancement, certain techniques have been used during the implementation of the proposed training method.

6.3.1. Adam Optimizer. Optimization is a crucial part of deep network training and learning. Optimization in deep learning is basically applied for minimization of the cost function by modifying the model parameters such as weight and bias for better convergence. Initially, the model parameters are initialized randomly. To boost the speed of training the network, a faster optimizer such as Adam [72] which stands for adaptive moment estimation has been applied for the optimization in the proposed work. Adam is a stochastic, robust optimizer and adaptive learning rate algorithm with less memory requirement. It requires less tuning of the learning rate hyperparameter. Adam is often performing well in adaptive learning and outperforms other adaptive techniques such as the stochastic gradient descent (SGD) optimizer.

6.3.2. Learning Rate (η). The learning rate (η) is an important hyperparameter of optimization used for fast convergence. The learning rate determines how much each epoch model parameter should be updated according to the gradient of the loss function. The decision to choose a proper learning rate can be tedious. An important parameter in optimization is the step size, determined by the learning rate hyperparameter. If the learning rate is too small, then the algorithm requires many epochs for converging, thus taking a long time. For a higher learning rate, the steps will be bouncing across the curve with larger values making the algorithm diverge, thus failing to find an optimal solution, as presented in Figure 6.

6.4. Preventing Overfitting. Overfitting is a common problem, where a model performs perfectly on training data but does not generalize well to untrained data or test data. To prevent overfitting during evaluation, methods such as batch normalization, $l2$ regularization, and dropout have been applied during DNN training.

6.4.1. Batch Normalization. When the deep model is trained, the distribution of hidden layers gets altered as there is an update in the parameters of the previous layers. There is a regular adjustment in the distribution in the respective layers. This adjustment in distribution is known as an

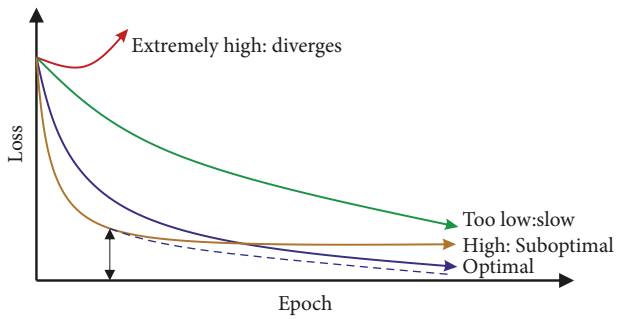

Figure 6: Illustration of the effect of the learning rate on the loss and number of epochs [64].

internal covariate shift. This problem is solved by a technique called batch normalization. In this technique, the input is normalized for every mini-batch and then the normalized weights are trained. The advantage of this technique is that even higher learning rates can be applied and weight initialization becomes less significant. During the training of DNN, the distribution of each input layer gets modified as the parameters of the previous layers change. This results in slowing down the training that demands lower learning rates and careful parameter initialization. Batch normalization solves this problem by normalizing the input for every mini-batch. Batch normalization is a popular method to prevent overfitting that normalizes the layers and trains the normalized weights. This allows us to use much higher learning rates and be less careful about initialization. It can be applied to any layer in the network.

6.4.2. Regularization. A model that suffers from overfitting has a high variance, due to having many parameters, which results in the generation of a complex model. Similarly, the model can also suffer from underfitting or high bias, which results in generating a highly simple model that is incapable of learning the patterns well in the training data and performs poorly on unseen data. Regularization can be a better approach to tune to the model complexity for a good bias and variance [61]. Regularization is a very useful method that tackles high correlation among features, filters out noise from data, and finally prevents overfitting. The network is regularized by adding constraints to the parameters such as imposing a squared penalty on the weights such that the higher the weight so is the penalty called $l2$ regularization. It is discussed in [73] and expressed in the following equation, where λ is the regularization parameter.

$$\frac{\lambda}{2}\|w\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2. \tag{16}$$

6.5. Deep Feature Classification. The features extracted from the training of hidden layers of the deep model are called deep features. These deep features are then projected for classification to the classifiers such as random forest, support vector machine, and K-nearest neighbour with various parameters for performance evaluation. The features derived from each hidden layer or in the concatenation of features extracted from hidden layers can be fed to the classifiers. This feature can be more robust as it is generated during potent DNN training [74]. Also, it can generalize to large and untrained data.

7. Experimentation and Result Analysis

In this work, we have used the publicly available database for the system evaluation. The link to the dataset is as follows [71]. For unbiased experimentation, we consider the N-fold leave-one-out criterion for the system evaluation. It is well known that leave-one-out is an unbiased approach for system evaluation. In addition, it consists of ≈ 139 k trials which are statistically significant. Therefore, the study of the technique on the other databases is kept for the future direction. The primary investigation of classification starts with the training of three popular ML classifiers, RF, SVM, and k-NN. Also, different packages of SVM and k-NN named in this paper as SVM-1, SVM-2, and SVM-3 and k-NN1 with different numbers of nearest neighbours have been trained and validated.

SVM-1 uses the linear kernel, along with regularization parameter (RP) 1, a maximum number of iterations as 10000, and automatic selection of gamma. SVM-2 is similar to SVM-1 but has higher tolerance. SVM-3 uses radial basis function (RBF), with a similar RP and a maximum number of iterations of 10000 but with gamma fixed to 0.7. The accuracy of these classifiers extracted exclusively is depicted in Figure 7. It is clear in Figure 7 that the ML classifiers alone are not sufficient to deal with a large volume of data. The highest accuracy of 91.16% is exhibited by the RF classifier. The performance of SVM-1 and SVM-2 is very poor as they use a linear kernel. The dataset is large and definitely nonlinear in nature. SVM-3, which uses the RBF kernel, has a better accuracy of 85.88%. The k-NN has better performance for a lower number of nearest neighbours, but it is still not up to the mark. Also, a lower number of nearest neighbours make the classifier biased and may have poor variance.

In the later stage, a 20-layer FFDNN with different activation functions is trained and its deep features are extracted to train the FFDNN as well as ML classifiers. The investigation is performed using different activation functions, SeLU, eLU, ReLU, and GeLU, of the DNN. The objective of this investigation is to identify the best performing activation function for the malware detection application. The obtained accuracy of the DNN with different activation functions is presented in Table 1. Here, the DNN is used in a traditional way and functions as a black box. The hidden layer features are unexplored and have no direct intervention in the final results. As discussed in the earlier sections, GeLU is a more high-performance activation function than the other one, thus achieving the best accuracy of 98.17% among all. It is understood that GeLU performs well with a small number of neurons. The performance of other activation functions is reasonably good for a larger number of neurons. ReLU exhibits its best accuracy for 128 neurons and eLU and SeLU for 64 neurons. In the next step,

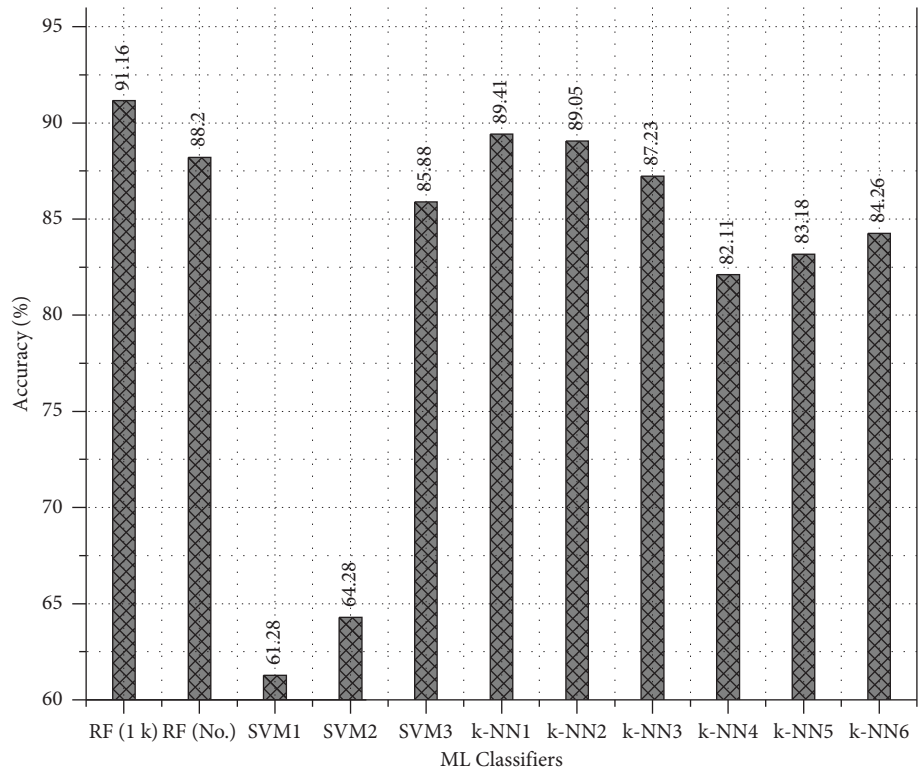

Figure 7: Comparison of the performance of malware detection in terms of accuracy with RF, SVM, and k-NN ML classifiers for different parameters.

Table 1: Accuracy of DNN with 3 hidden layers HL for different numbers of neurons and activation functions.

No. of neurons	GeLU	ReLU	eLU	SeLU
32	**98.17**	97.85	97.83	97.2
64	98.15	97.81	**97.88**	**97.22**
128	98.13	**97.89**	97.78	97.03
256	97.95	97.72	97.65	96.78
512	97.42	97.51	97.37	97.26

Bold values indicate the peak accuracy achieved using the corresponding activation function.

Table 2: Accuracy of DNN with GeLU activation function for different numbers of HL and neurons.

HL	$N = 32$	$N = 64$	$N = 128$	$N = 256$	$N = 512$
3	98.17	98.15	98.13	97.95	97.42
10	98.15	98.156	98.17	**98.23**	**98.21**
15	98.19	98.16	**98.18**	98.21	98.14
20	**98.20**	**98.22**	98.15	98.20	98.14

Bold values indicate the peak accuracy achieved for the corresponding numbers of neurons.

the effectiveness of GeLU is examined against the increasing number of the DNN hidden layers and the number of neurons, as presented in Table 2.

It is observed that GeLU exhibits better accuracy with a less number of neurons but a larger number of hidden layers or a small number of hidden layers but a large number of neurons in an FFDNN. In both cases, the obtained accuracy has a marginal difference. For the 32 neurons and 20 hidden layers, the accuracy is 98.20%, whereas for 256 neurons and 10 hidden layers, the accuracy is 98.21%. So, either case may be used for a high-performance classification. From the previous observations, the GeLU activation function has been used and investigated for either case of the proposed model.

The training and validation loss of the DNN for 256 neurons, 10 deep hidden layers, and 11 splits is shown in Figure 8. The training loss and validation loss in the graph represent how perfectly the model fits into the trained and untrained data, respectively. As the dataset is larger, the data are separated into the train, validate, and test sets. From the graph, it can be observed that there is a minimal gap between training loss and validation loss in each split. Also, the loss is getting smaller with the number of epochs and converging fast.

Proceeding to the experimentation, the proposed model with a different number of hidden layers, a number of neurons, and a number of splits has been created and investigated. At one time, each hidden layer in the network is trained using neurons $n = 2^5$ to 2^9 and increasing the number of deep layers from 3 to 20 subsequently. In the first deep model training, 3 hidden layers with 11 splits and 32 neurons in each layer have been examined. Due to the huge volume of data and with an objective to improve performance, the training data are randomly distributed and split into batches of 1024 data points. Each batch is executed in 30 epochs during the training to generate scores later used for the calculation of the reconstruction error and classification

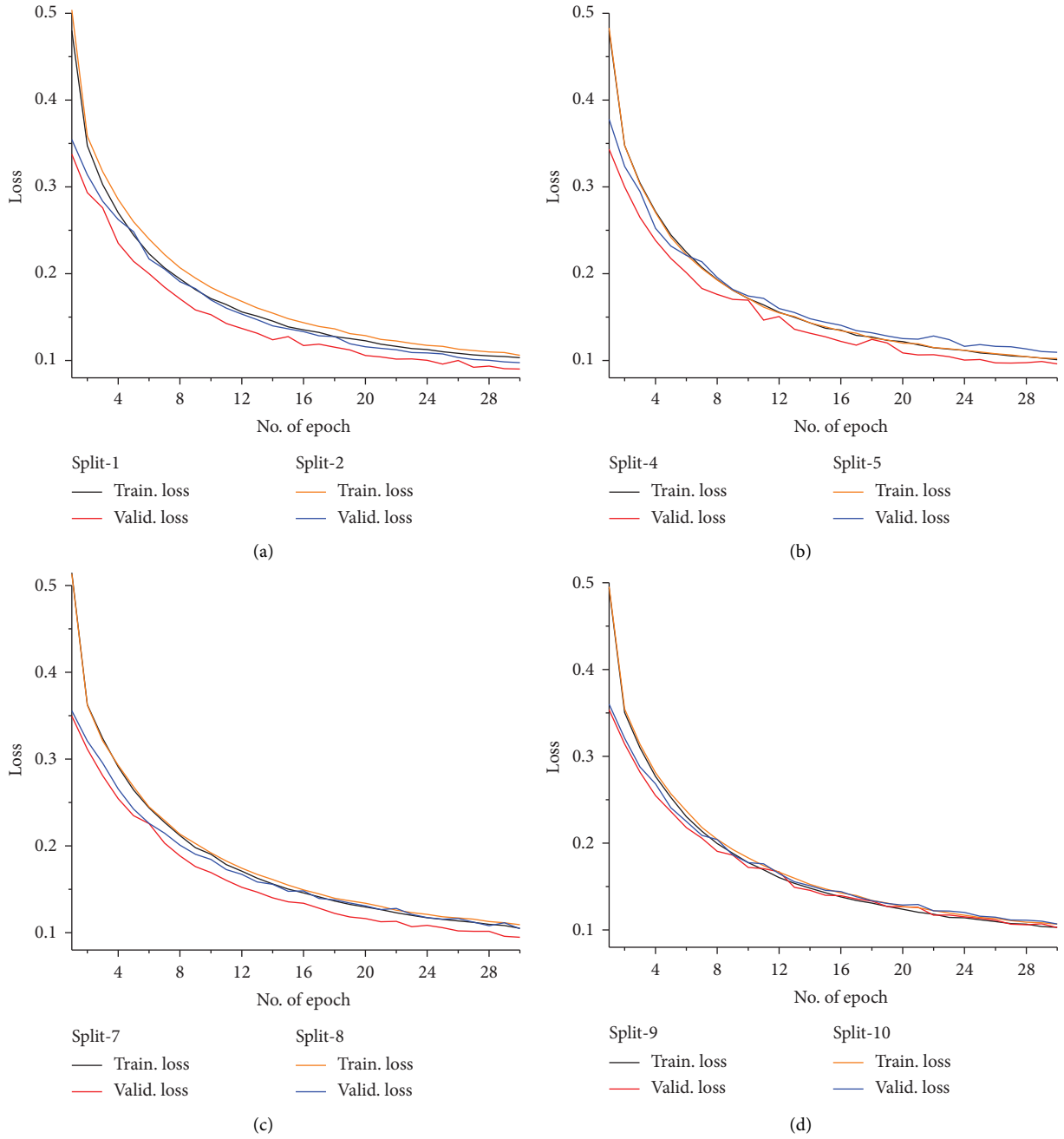

FIGURE 8: Illustration of the training and validation losses over a number of epochs on different folds of data for an FFDNN consisting of 10 hidden layers and 256 neurons/layers: (a) splits 1 and 2; (b) splits 4 and 5; (c) splits 7 and 8; (d) splits 9 and 10.

probability. 10-fold cross-validation is applied to the data points to validate the training. $L1$ loss and cross-entropy loss functions are used to calculate reconstruction loss and classification loss. The Adam optimizer is used for weight optimization during backpropagation for faster convergence with a learning rate $\eta = 0.001$. To prevent overfitting, the encoding parameters such as the $L2$ regularizer and he-initializer have been used for the network training. However, a few conclusive results are presented in Tables 3–6. In Table 3, the results are presented for the 3-layer 256 neurons with 11 splits. The accuracy of this model is reasonable with the highest accuracy of 98.78% for the k-NN with 3 nearest neighbours. As discussed earlier, k-NN with a lower neighbour count may be biased, and a further investigation is carried out. Although, it is observed that the RF and SVM classifiers performed well with a higher number of deep hidden layer features or concatenated features of all layers. The k-NN exhibits better performance for a lower number of hidden layer features. The maximum achieved accuracy of all models is appreciable and above 98%. The results of the second model presented in Table 4 have the same number of neurons and splits but 10 hidden layers. Among all models, the best accuracy of 99.15%

Table 3: Accuracy of ML classifiers against individual and concatenated 3-HLs deep features for 256 neurons and 11 splits.

Features	RF	SVM			k-NN					
		SVM1	SVM2	SVM3	K-NN1	K-NN2	K-NN3	K-NN4	K-NN5	K-NN6
DF1	96.38	56.34	96.77	98.34	**98.77**	**98.54**	**98.78**	**98.72**	98.75	98.68
DF2	96.35	96.99	97.3	98.3	98.74	98.52	98.76	98.71	**98.76**	**98.69**
DF3	97.35	**98.09**	98.02	98.22	98.61	98.41	98.68	98.61	98.67	98.62
DF (1–3)	**98.16**	96.72	**98.09**	**98.39**	98.73	98.52	98.77	98.71	98.75	98.67

The bold values indicate the peak accuracy achieved using the corresponding Machine Learning classifier.

Table 4: Accuracy of ML classifiers against individual and concatenated 10-HLs deep features for 256 neurons and 11 splits.

Features	RF	SVM			k-NN					
		SVM1	SVM2	SVM3	K-NN1	K-NN2	K-NN3	K-NN4	K-NN5	K-NN6
DF1	96.32	56.29	96.77	98.33	**98.78**	98.64	98.24	98.61	98.61	98.53
DF2	96.36	97.03	97.31	98.19	98.29	**98.83**	98.82	**98.66**	98.67	98.47
DF3	97.63	97.17	97.02	98.1	98.38	98.50	**98.74**	98.72	**98.81**	**98.98**
DF4	97.06	97.55	97.09	98.38	98.17	98.13	98.69	98.51	98.87	98.24
DF5	97.60	97.67	97.23	98.15	98.41	98.17	98.44	98.87	98.55	98.65
DF6	97.71	97.69	97.30	98.92	97.94	98.26	98.28	98.18	98.15	98.24
DF7	97.61	97.81	97.42	98.15	97.62	98.07	98.52	98.21	98.25	98.16
DF8	97.50	**98.87**	98.31	98.77	97.34	97.36	98.03	98.35	98.03	98.11
DF9	97.72	98.18	97.83	98.94	97.25	97.48	98.43	98.09	98.14	98.06
DF10	97.64	98.32	97.25	98.88	97.51	97.06	98.41	98.19	97.91	98.23
DF (1–10)	**97.95**	98.56	**98.37**	**99.15**	97.83	97.50	98.10	98.05	98.08	98.20

The bold values indicate the peak accuracy achieved using the corresponding machine learning classifier.

Table 5: Accuracy of ML classifiers against individual and concatenated 15-HLs deep features for 256 neurons and 11 splits.

Features	RF	SVM			k-NN					
		SVM1	SVM2	SVM3	K-NN1	K-NN2	K-NN3	K-NN4	K-NN5	K-NN6
DF1	97.63	48.62	96.11	96.14	98.12	97.96	98.16	98.17	98.01	97.95
DF2	96.26	96.17	95.46	96.49	98.45	97.21	98.03	98.10	98.37	98.34
DF3	96.12	96.26	96.33	96.76	**98.62**	97.90	98.21	**98.57**	98.47	98.71
DF4	96.21	96.15	97.43	96.73	98.55	98.08	98.21	98.02	98.45	98.37
DF5	97.08	97.01	96.35	96.29	98.04	**98.41**	**98.33**	98.33	**98.57**	98.18
DF6	97.45	96.53	96.54	97.01	97.50	97.03	97.96	98.25	98.02	98.25
DF7	95.24	96.39	96.97	96.13	97.49	97.44	98.17	98.24	98.53	97.67
DF8	97.26	96.94	96.67	97.38	97.93	97.09	98.08	98.19	98.15	**98.80**
DF9	96.98	96.80	96.35	96.88	97.59	97.19	97.50	97.57	97.57	97.38
DF10	96.95	97.17	96.97	97.18	97.65	97.38	97.35	97.99	97.54	98.46
DF11	**97.86**	97.82	96.40	96.68	97.90	97.16	96.44	97.38	97.50	98.36
DF12	96.80	**97.99**	96.34	97.33	97.20	97.14	97.91	97.51	98.38	98.34
DF13	96.29	96.70	96.92	97.36	97.53	97.97	96.70	98.13	97.72	97.38
DF14	97.68	97.20	97.52	97.60	96.94	97.77	97.01	98.58	96.29	97.68
DF15	97.73	97.67	97.83	98.18	96.88	97.04	96.94	97.68	96.48	97.99
DF (1–15)	97.41	97.55	**98.13**	**98.40**	96.57	96.24	96.87	96.21	97.65	96.95

The bold values indicate the peak accuracy achieved using the corresponding machine learning classifier.

is achieved by SVM-3 in concatenation form with 256 neurons and 11 splits. The obtained result is definitely remarkable for such a large dataset. The performance of k-NN is also considerable for a lower number of deep layers. The k-NN achieves the highest accuracy of 98.98% for the independent third hidden layer. Similarly, the results of the proposed model with 15 and 20 layers have similar observations. The performance of some models is inconclusive which is natural for artificial inelegance algorithms. For example, SVM-1 exhibits an accuracy of 98.16% using the independent 14^{th} layer feature of the 20-layer model. From the previously mentioned observations, it is sure that the RF and SVM ML classifiers with bioinspired DNN GeLU activation functions are a potential solution to the current regime for malware detection and prevention.

The proposed work is compared with some existing work and presented in Table 7. While designing the system model, the core network, the type of feature used, activation functions, and most importantly, the feature representation have been considered, as presented in Table 7. In the

Table 6: Accuracy of ML classifiers against individual and concatenated 20-HLs deep features for 256 neurons and 11 splits.

Features	RF	SVM			k-NN					
		SVM1	SVM2	SVM3	K-NN1	K-NN2	K-NN3	K-NN4	K-NN5	K-NN6
DF1	96.14	48.63	96.11	96.14	98.12	97.92	98.14	98.16	98.00	97.96
DF2	96.27	96.15	95.48	96.52	98.45	98.21	**98.43**	98.13	98.35	98.32
DF3	96.12	96.24	96.31	96.76	98.54	97.93	98.19	97.97	98.47	98.73
DF4	96.23	96.17	97.46	96.69	**98.75**	98.08	98.24	97.97	98.45	98.37
DF5	97.04	97.01	96.32	96.25	98.03	**98.39**	98.32	**98.29**	**98.58**	98.18
DF6	97.49	96.51	96.57	97.03	97.52	97.01	97.95	98.23	98.03	98.22
DF7	95.21	96.43	96.96	96.16	97.49	97.47	98.17	98.20	98.54	97.65
DF8	97.28	96.98	96.62	97.40	97.92	97.11	98.07	98.16	98.16	**98.83**
DF9	97.00	96.79	96.35	96.89	97.63	97.19	97.50	97.59	97.61	97.35
DF10	96.93	97.19	96.99	97.17	97.61	97.37	97.34	98.03	97.56	98.44
DF11	97.86	97.81	96.42	96.67	97.86	97.16	96.43	98.11	97.53	98.38
DF12	96.80	96.99	96.37	97.31	97.23	98.11	97.92	97.52	98.39	98.32
DF13	96.31	96.71	96.94	97.34	97.51	97.98	96.72	98.11	97.71	97.40
DF14	97.67	**98.16**	97.51	97.58	96.94	97.77	97.01	98.57	96.30	97.67
DF15	**97.72**	97.68	97.53	98.15	96.91	97.01	96.90	97.66	96.46	98.03
DF16	97.44	97.55	97.14	98.41	96.57	96.24	96.86	96.23	97.66	96.96
DF17	97.24	97.83	97.08	**98.83**	97.41	96.60	96.83	96.22	97.35	97.51
DF18	96.83	97.62	97.31	98.02	97.11	96.68	97.02	96.83	97.40	97.79
DF19	97.65	97.98	97.83	98.23	96.38	97.02	96.86	97.01	97.06	97.17
DF20	97.60	97.77	97.88	98.61	97.07	96.15	96.63	97.21	97.38	97.36
DF (1–20)	97.43	97.64	**97.95**	98.47	97.20	97.69	96.52	97.68	97.18	97.57

The bold values indicate the peak accuracy achieved using the corresponding machine learning classifier.

literature, numerous works have been reported for malware detection using neural networks. However, different works use different types of core networks such as FFDNN [44], SAE-DBN [45, 50], DBN [46], and CNN [55–57]. Similarly, different works used different types of features, activation functions, and feature representations. In most of the work, the core learning model is being trained by a traditional feature representation such as PEH, API call, and image. Different evaluation parameters such as accuracy (Ac), detection rate (Dr), and true positive rate (TPR) have been presented. In [46], a DBN is trained using the deep feature extracted using an unsupervised dataset and a limited investigation is performed using deep features. It is clear in Table 7 that an extensive investigation is carried out in this work, and the performance achieved using deep feature representation is superior.

7.1. Network Model Construction Time. This proposed work is implemented on Linux on dual boot with Windows operating System configured with Intel(R) Core(TM) i5-8250U CPU 1.60 GHz and 8.00 GB of RAM. Training of deep learning models is computationally expensive. The construction time or the training time of the FFDNN network for different layers (in hours) is depicted in Figure 9. The time consumed in each scenario is observed and noted in approximation. The DNN training for classification and prediction was conducted in comparatively lesser time than the training of ML-integrated DNN classifiers. The SVM model for the deep features was trained for more than 32 hours for 3 hidden layers with neuron size 32. It can be observed that the time of model construction increases as the number of hidden layers is increased with the increasing size of neurons. However, the training of the proposed model is a one-time process and the training time can be significantly reduced due to the availability of online cloud-based graphical processing units (GPUs). A consumer may get access to these GPUs for a short period of time to train the proposed model at a minimal cost.

Table 7: Comparison of the proposed work with some existing literature.

Ref. no.	Core network	Feature type	Activation function	Feature representation	Performance (%)
[44]	FFDNN	PE import, 2D string, and entropy histogram	ReLU and PReLU	Traditional	95 Dr
[45]	SAE-DNN	Windows API calls	NA	Traditional	96.85 Ac
[46]	DBN	OpCode	NA	Deep features	Approx 98 Ac
[48]	DNN	PEH	ReLU	Traditional	98.9 Ac
[50]	SAE-DBN	System call	NA	Traditional	98.8 Ac
[53]	Multilevel-DNN	API system call	Leaky ReLU	Traditional	96 TPR
[55]	CNN, RNN, LSTM	Image and API call	ReLU	Traditional	96.3 Ac
[56]	CNN-VGG16	Image	NA	Traditional	94.7 Ac
[57]	CNN, RF	Image	NA		98.65 Dr
This work	FFDNN-ML	PEH	eLU, ReLU, SeLU, and GeLU	Concatenated deep features	99.15 Ac

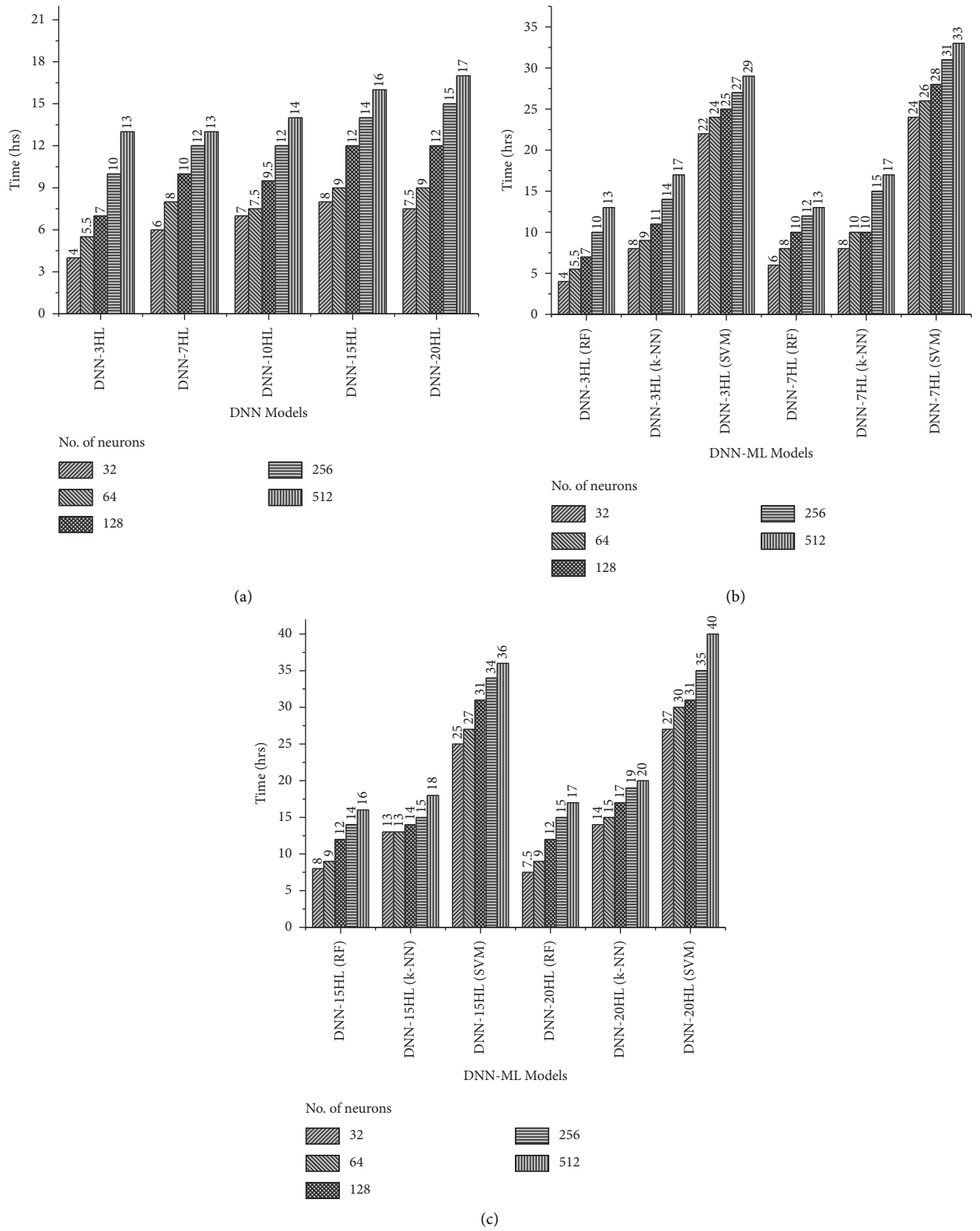

Figure 9: Comparison of the computation (in terms of times) required for the construction of the proposed model for different system configurations (neurons and no. of hidden layers). (a) Exclusive DNN. (b) DNN-ML with 3 and 7 layers. (c) DNN-ML with 15 and 20 layers.

8. Conclusion

We propose deep feature extraction of PEH for an FFDNN-ML malware detection system. The examinations have been carried out from scratch including training and validation of exclusive ML classifiers and FFNN. The observations of the primary investigations are highly informative, and they helped in developing the proposed bioinspired DNN-based malware detection system. The conclusive observations of the proposed work are as follows:

(i) The classification accuracy obtained using an individual hidden layer indicates that the extracted deep features of hidden layers have better representation than a raw dataset. Also, the deep features of different layers have different presentations, and by making the network deeper, a dataset can be represented in n-dimensions with better generalization.

(ii) The RF, SVM, and k-NN ML classifiers are individually insufficient to efficiently detect malware when trained using a large dataset. The RF, k-NN, and SVM-RBF have some degree of accuracy, but SVM-linear is not at all recommended.

(iii) The GeLU activation function is highly recommended even when using the DNN in a traditional way. That is, the DNN will just act as a black box and it will produce a classified value from the output layer.

(iv) The FFDNN with the GeLU activation function performs well for a large number of deep hidden layers with a less number of neurons per layer or a small number of hidden layers with a large number of neurons per layer. Either of these two cases may be used for effective classification.

(v) The proposed bioinspired DNN-based malware detection system exploits the deep hidden features and makes use of it for effective classification when trained using the dataset used in this work. In addition, the concatenation of these deep features in the proposed way has shown highly interesting results in combination with RF and SVM ML classifiers.

(vi) Deep feature extraction with complex DNN/architecture such as transformer and ResNet could be interesting to capture more attributes on the data.

Data Availability

The dataset that supports the findings of this study is openly available on web.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

All authors have contributed equally to this work.

References

[1] http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/.

[2] http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/.

[3] B. Pranggono and A. Arabo, "COVID-19 pandemic cybersecurity issues," in *Internet Technology Letters*vol. 2, p. 247, 2021.

[4] J. Scott, "Signature based malware detection is dead," in *Cybersecurity Think Tank*Institute for Critical Infrastructure Technology, Washington, D.C, USA, 2017.

[5] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*No starch press, San Francisco, CA, USA, 2012.

[6] D. Lin and M. Stamp, "Hunting for undetectable metamorphic viruses," *Journal in Computer Virology*, vol. 7, no. 3, pp. 201–214, 2011.

[7] M. Alaeiyan, S. Parsa, and M. Conti, "Analysis and classification of context-based malware behavior," *Computer Communications*, vol. 136, pp. 76–90, 2019.

[8] A. V. Kozachok and V. I. Kozachok, "Construction and evaluation of the new heuristic malware detection mechanism based on executable files static analysis," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 3, pp. 225–231, 2018.

[9] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[10] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: research developments, trends and challenges," *Journal of Network and Computer Applications*, vol. 153, Article ID 102526, 2020.

[11] A. K. Ma and J. Cd, "Automated multi-level malware detection system based on reconstructed semantic view of executables using machine learning techniques at VMM," *Future Generation Computer Systems*, vol. 79, no. 1, pp. 431–446, 2018.

[12] P. Singh, S. K. Borgohain, L. D. Sharma, and J. Kumar, "Minimized feature overhead malware detection machine learning model employing MRMR-based ranking," *Concurrency and Computation: Practice and Experience*, vol. 34, Article ID e6992, 2022.

[13] F. Xiao, Z. Lin, Y. Sun, and Y. Ma, "Malware detection based on deep learning of behavior graphs," *Mathematical Problems in Engineering*, vol. 2019, Article ID 8195395, 10 pages, 2019.

[14] Kaspersky Research, "Machine learning methods for malware detection," *Symmetry*, vol. 14, no. 11, p. 2304, 2022.

[15] A. Firdaus, N. B. Anuar, M. FA. Razak, and A. K. Sangaiah, "Bio-inspired computational paradigm for feature investigation and malware detection: interactive analytics," *Multimedia Tools and Applications*, vol. 77, no. 14, pp. 17519–17555, 2018.

[16] B. Ji, X. Lu, G. Sun, W. Zhang, J. Li, and Y. Xiao, "Bio-inspired feature selection: an improved binary particle swarm optimization approach," *IEEE Access*, vol. 8, pp. 85989–86002, 2020.

[17] J. Jiang and F. Zhang, "Detecting portable executable malware by binary code using an artificial evolutionary fuzzy LSTM immune system," *Security and Communication Networks*, vol. 2021, Article ID 3578695, 12 pages, 2021.

[18] F. Mercaldo and A. Santone, "Deep learning for image-based mobile malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 16, no. 2, pp. 157–171, 2020.

[19] R. Feng, S. Chen, X. Xie, G. Meng, S.-W. Lin, and Y. Liu, "A performance-sensitive malware detection system using deep learning on mobile devices," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1563–1578, 2021.

[20] L. N. de Castro, *Fundamentals of natural computing*CRC Press, Boca Raton, FL, USA, 2006.

[21] X. Fan, W. Sayers, S. Zhang, Z. Han, L. Ren, and H. Chizari, "Review and classification of bio-inspired algorithms and their applications," *Journal of Bionics Engineering*, vol. 17, no. 3, pp. 611–631, 2020.

[22] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[23] J. Han, M. Kamber, and P. Jian, *Data Mining: Concepts and Techniques*, Elsevier, Morgan Koufmann Publisher, Amsterdam, Netherlands, 3rd edition, 2011.

[24] N. J. Nilsson, "Introduction to machine learning," 2020, https://ai.stanford.edu/%20nilsson/MLBOOK.pdf.

[25] M. Tayebi and S. E. Kafhali, "Deep neural networks hyperparameter optimization using particle swarm optimization for detecting frauds transactions," in *Advances on Smart and Soft Computing*, pp. 507–516, Springer, Singapore, 2022.

[26] M. Tayebi and S. E. Kafhali, *Hyperparameter Optimization Using Genetic Algorithms to Detect Frauds Transactions*, AICV, in *Proceedings of the International Conference on Artificial Intelligence and Computer Vision*, April, 2021.

[27] M. Tayebi and S. E. Kafhali, "Performance analysis of metaheuristics based hyperparameters optimization for fraud transactions detection," *Evolutionary Intelligence*, 2022.

[28] M. Tayebi and S. E. Kafhali, "Credit card fraud detection based on hyperparameters optimization using the differential evolution," *International Journal of Information Security and Privacy*, vol. 16, no. 1, 2022.

[29] J. Ma, X. Jiang, A. Fan, J. Jiang, and J. Yan, "Image Matching from Handcrafted to Deep Features: A Survey," *International Journal of Computer Vision*, vol. 129, no. 1, pp. 23–79, 2021.

[30] M. Rashid, M. A. Khan, M. Alhaisoni et al., "A sustainable deep learning framework for object recognition using multilayers deep features fusion and selection," *Sustainability*, vol. 12, no. 12, p. 5037, 2020.

[31] W. Saad, W. A. Shalaby, M. Shokair, F. A. El-Samie, and E. Abdellatef, "COVID-19 classification using deep feature concatenation technique," *Journal of Ambient Intelligence and Humanized Computing*, vol. 13, no. 4, pp. 2025–2043, 2022.

[32] N. Noreen, S. Palaniappan, A. Qayyum, I. Ahmad, M. Imran, and M. Shoaib, "A deep learning model based on concatenation approach for the diagnosis of brain tumor," *IEEE Access*, vol. 8, pp. 55135–55144, 2020.

[33] G. Hinton, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, pp. 82–97, 2012.

[34] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning,," in natural to artificial neural," in *Computation. IWANN 1995*, pp. 195–201, Springer, Berlin, Germany, 1995.

[35] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning," 2018, http://arxiv.org/abs/1811.03378.

[36] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the International Conference on International Conference on Machine Learning*, pp. 807–814, Pittsburgh Pennsylvania USA, June, 2010.

[37] Z. Yue, H. Christensen, and J. Barker, "Autoencoder bottleneck features with multi-task optimisation for improved continuous dysarthric speech recognition," in *Proceedings of the Interspeech*, pp. 4581–4585, Shanghai, China, July, 2020.

[38] D. B. Ramsay, K. Kilgour, D. Roblek, and M. Sharifi, "Low-dimensional bottleneck features for on-device continuous speech recognition," in *Proceedings of the Interspeech*, pp. 3456–3459, Incheon, Korea, April, 2019.

[39] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "Arcface: additive angular margin loss for deep face recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4685–4694, Nashville, TN, USA, June, 2019.

[40] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the International Conference on Machine Learning*, pp. 1597–1607, Atlanta GA USA, July, 2020.

[41] M. Zeiler, M. Ranzato, R. Monga et al., "On rectified linear units for speech processing," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3517–3521, Rhodes Island, June, 2013.

[42] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural net-works for lvcsr using rectified linear units and dropout," in *Proceedings of the Of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8609–8613, Greece, June, 2013.

[43] D. Hendrycks and K. Gimpel, "Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units," 2018, https://arxiv.org/abs/1606.08415.

[44] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proceedings of the 2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 11–20, Puerto Rico, October, 2015.

[45] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "DL4MD: a deep learning framework for intelligent malware detection," in *Proceedings of the International Conference on Data Mining (DMIN'16)*, pp. 61–67, Las Vegas, NV, USA, July, 2016.

[46] D. Yuxin and Z. Siyi, "Malware detection based on deep learning algorithm," *Neural Computing & Applications*, vol. 31, no. 2, pp. 461–472, 2017.

[47] M. M. Lau and K. H. Lim, "Investigation of activation functions in deep belief network," in *Proceedings of the 2017 2nd International Conference on Control and Robotics Engineering (ICCRE)*, pp. 201–206, Bangkok, Thailand, April, 2017.

[48] R. Vinayakumar and K. P. Soman, "DeepMalNet: evaluating shallow and deep networks for static PE malware detection," *ICT Express*, vol. 4, no. 4, pp. 255–258, 2018.

[49] H. S. Anderson and P. Rot, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," 2018, https://arxiv.org/abs/1804.04637.

[50] Y. Ye, L. Chen, S. Hou, W. Hardy, and X. Li, "DeepAM: a heterogeneous deep learning framework for intelligent malware detection," *Knowledge and Information Systems*, vol. 54, no. 2, pp. 265–285, 2018.

[51] G. Ciaburro, V. K. Ayyadevara, and A. Perrier, "Get Hands-On Machine Learning on Google Cloud Platform now with the O'Reilly learning platform," 2021, https://www.oreilly.com/library/view/hands-on-machine-learning/9781788393485/f73852fe-4f59-44e5-b154-1bc7b2de1375.xhtml.

[52] H. Rathore, S. Agarwal, S. K. Sahay, and M. Sewak, "Malware detection using machine learning and deep learning," *Lecture*

Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 11297, pp. 402–411, 2018.

[53] W. Zhong and F. Gu, "A multi-level deep learning system for malware detection," *Expert Systems with Applications*, vol. 133, pp. 151–162, 2019.

[54] Y. Zhu, T. Brettin, and F. Xia, "Converting tabular data into images for deep learning with convolutional neural networks," *Scientific Reports*, vol. 11,11325 pages, 2021.

[55] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46717–46738, 2019.

[56] X. Huang, L. Ma, W. Yang, and Y. Zhong, "A method for windows malware detection based on deep learning," *Journal of Signal Processing Systems*, vol. 93, no. 2-3, pp. 265–273, 2021.

[57] S. A. Roseline, S. Geetha, S. Kadry, and Y. Nam, "Intelligent vision-based malware detection and classification using deep random forest paradigm," *IEEE Access*, vol. 8, pp. 206303–206324, 2020.

[58] P. Ulinski, "Fundamentals of Computational Neuroscience," vol. 40, no. 5, Oxford, UK, OUP Oxford, 2003.

[59] R. Hecht-Nielsen, "Neurocomputing: picking the human brain," *IEEE Spectrum*, vol. 25, no. 3, pp. 36–41, March 1988.

[60] N. B. Paperback and N. Locascio, *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*, O'Reilly Media, Massachusetts, MA, USA, 1st edition, 2017.

[61] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media, Massachusetts, MA, USA, 1st edition, 2019.

[62] D. Svozil, V. Kvasnička, and J. Pospíchal, "Introduction to multi-layerfeed-forward neural network," *Chemometrics and Intelligent Laboratory Systems*, vol. 39, no. 1, pp. 43–62, 1997.

[63] J. Yang and J. Ma, "Feed-forward neural network training using sparse representation," *Expert Systems with Applications*, vol. 116, pp. 255–264, 2019.

[64] A. Bhardwaj, W. Di, and J. Wei, *Deep Learning Essentials: Your Hands-On Guide to the Fundamentals of Deep Learning and Neural Network Modeling*, Packt Publishing Ltd, Birmingham, UK, 1st edition, 2018.

[65] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[66] D. O. Hebb, "The organization of behavior: a neuropsychological theory," *Science Education*, vol. 34, no. 5, 1950.

[67] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–2558, 1982.

[68] S. Narayan, "The generalized sigmoid activation function: competitive supervised learning," *Information Sciences*, vol. 99, no. 1-2, pp. 69–82, 1997.

[69] D. A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (eLUs)," in *Proceedings of the 4th International Conference on Learning Representations*, Puerto Rico, May, 2016.

[70] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units," 2016, https://arxiv.org/abs/1606.08415.

[71] Tinyurl.com, "Mastering-Machine-Learning-for-Penetration-Testing," 2009, https://tinyurl.com/wcbuchdt.

[72] D. P. Kingma and J. Baar, "Adam: a method for stochastic optimization," in *Proceedings of the 3rd International Conference for Learning Representations (ICLR)*, San Diego, CA, USA, May, 2015.

[73] Y. Qian, N. Chen, and K. Yu, "Deep features for automatic spoofing detectio," *Speech Communication*, vol. 85, pp. 43–52, 2016.

[74] S. Raschka and V. Mirjalili, *Python Machine Learning: Macine Learning and Deep Learning with Python, Scikit-Learn, and TensorFlow 2*, Packt Publishing Ltd, Brmingham, UK, 3rd edition, 2018.