# Cloud Computing Malware Detection and Classification Using ANN Based Machine Learning Model

(IJCSNT) International Journal of Communication Systems and Network Technologies
ISSN Number: 2053-6283
Volume 14, Issue No. 3, 149–157

https://journals.mirlabs.in/index.php/ijcsnt
DOI-10.18486/ijcsnt/14.2.013
IJCSNT

**Anurag Sinha[1], Mini Kumari[2], Harsh Raj[3], Bhaskar Singh[4], Michael Wiryaseputra[5], Ahmed Alkhayyat[6]**

## Abstract

This summary outlines the investigation into detecting and categorizing cloud computing malware using Artificial Neural Network (ANN)-based Machine Learning models. The surge in cloud services has elevated the risk of malware attacks in cloud environments. This research explores the application of ANN-based ML techniques to bolster cloud system security, emphasizing robust malware detection through feature extraction, dataset preparation, and model training. Various ANN architectures are considered and evaluated against real-world cloud malware datasets, revealing the efficacy of ANN models in detecting and classifying cloud-based malware with promising accuracy and efficiency. The deployment of a neural network classifier for binary malware that classifies Windows Portable Executable (PE) files according to imported library function calls is the specific topic of this article. The applied model demonstrates its capacity to generalize against an independent set with an astounding 97.8% average accuracy, 97.6% precision, and 96.6% recall. These findings demonstrate the practicality of the suggested malware categorization technique for bolstering cloud security against changing cyberthreats.

## Keywords

ANN, Malware, Malware Categorization, Malware Identification, Neural Network

**Received:** 11 August 2025; **Revised:** 28 November 2025; **Accepted:** 23 December 2025; **Published:** 31 December 2025;

[1]School of Computing and Information Sciences, Department of Mathematics, IGNOU, New Delhi, India.
[2]Department of Criminology, University of Madras, Chennai, Tamil Nadu, India.
[3]Computer Science Department, Amity University Jharkhand, Ranchi, Jharkhand, India.
[4]Department of Computer Science Engineering, Dr. M.G.R Educational Research Institute University, Chennai, Tamil Nadu, India.
[5]Catholic University, Semarang, Indonesia.
[6]College of Technical Engineering, The Islamic University, Najaf, Iraq.

**Corresponding author:**
Email: anuragsinha257@gmail.com

## Introduction

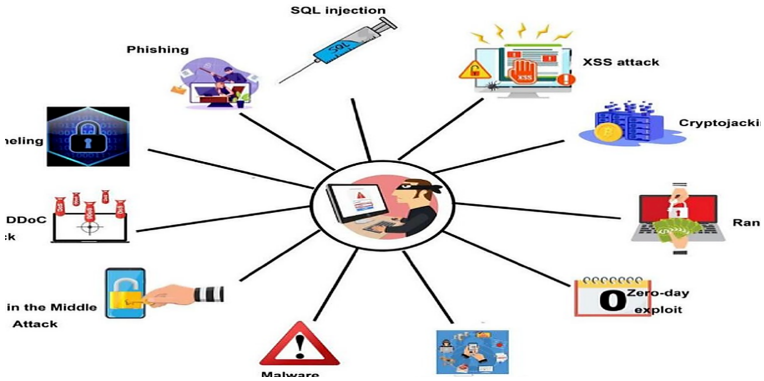
Based on the latest security analysis from AV-TEST[1], it is predicted that by the end of 2016, there will be more than 600 million distinct malware instances, primarily Windows-based PEs. That is an ongoing upward trajectory in the evolution of malware. By comparison, in 2006 there were only 2.6 million unique malware cases reported. A variety of malware identification techniques, such as signature, hash, and heuristic-based approaches, have been proposed and/or implemented by the academic community and the antivirus industry in response to this unprecedented and inventive surge in malware variations. Comparative tests by independent AV testers[2] show that despite efforts to apply the newest detection techniques, solutions incorporating these techniques offered by the AV industry are still unimpressive. This is attributed to the inherent limitations of known malware detection techniques used by AV systems, hindering every engine from achieving the highest possible detection rate with no false-positive results. As such, the ongoing challenge is to find the best malware detection techniques to combat the creative malware that writers create. Researchers are now examining the use of ML for of malware identification in an effort to overcome the drawbacks of the existing methods (heuristic, signature, and hash). Several notable works targeting the classification of PE have already been conducted for of malware identification and classification, showing impressive detection rates. Nevertheless, the limitations of these works result in a gap in the research. In order to classify heterogeneous data gathered from computer networks, Kruczkowski and Szynkiewicz[3] used (SVM) in a Classification of two-class patterns. They were able to achieve best generalisation with low-false positives and high sensitivity.

Nonetheless, their approach raised concerns due to the substantial computational requirements for handling heterogeneous data.Attaining robust generalization with a sensitivity is high and false-positive rate is low was successfully demonstrated in their approach, but the substantial computational requirements for handling heterogeneous data raised significant concerns. Consequently, in this research, our focus shifted towards employing a straightforward binary classifier based on neural networks and imported Windows library functions. To close the unstudied gap in the area of applying ML methods for identifying malwares, the main goal is to categorize each unseen PE file as malicious or benign. When compared to the work of earlier researchers, this method has the potential to produce results with greater precision and accuracy.The structure of this paper is as follows in the following sections: Neural networks, static examination and feature extraction, as well as recent related works are all covered in the literature review in Section 2. The model of a proposed neural network classifier is described in Section 3, along with the operational framework that includes procedures for labeling, sample collection, and feature extraction. Part 4 talks about the experiment' s parameters and validation methods. Comparative analysis is used in Section 5 to assess and talk about the experiment results. A route for future research is provided in Section 6, which also concludes with a discussion of implications.

## Literature Review

### *Malicious Software (Malwares)*

Malicious software, also called as system malware, includes any software intended to get data from a victim' s pc, prevent the victim' s computer from operating normally, grant unauthorised entry to the PC of the victim. Computer viruses were the original type of malwares , but over time, other popular types such as logic-bombs, worm's, root-kits, ransomware, backdoors, and Trojans have also become prevalent.




## *Characteristics of Static Analysis*

In contrast to dynamic analysis, which comprises running the program and observing its behavior, static analysis examines a program' s code and structure without actually running it[4,5]. To find malware, a variety of static techniques are applied to a huge number of features. In order to facilitate the investigation of every possible execution path, these procedures involve evaluating op-code n-grams, brought in Dynamic Link Libraries (DLLs), library calls, byte sequences, and printable strings that are hardcoded[6]. A number of features, including brought in DLLs, PE header data, Entropy, file size, and hard-coded strings, can be extracted with the help of PE metadata analysis. Because most malware appears as binary files, bytecoder decompiled/disassembled instructions are also inspected[5].For example, Gonzalez and Vazquez[7] presented a method based on the number of calls an application makes to the Application Programming Interface (API) from DLLs it imports.User-level programs use the IDA disassembler to disassemble binary samples because they depend on APIs to interact with the operating system and make use of hardware resources.They used a variety of learning methods for their neural network model' s training, and when they used the Leven- Marquardt algorithm with one hidden layer, their neural network was able to achieve a maximum accuracy rate of 97.95%. Moreover, a number of researchers have proposed malware detection methods that center on statically examining a sample' s API calls. Static analysis' s excellent performance and low resource consumption are major advantages, particularly when combined with ML.

The model itself can require a lot of time, resources, and performance during training (with a highest time complexity).

## *Artificial Neural Network (ANN)*

Artificial neural network (ANN) is a type of computer model simulates the architecture of neural networks found in the brain and provides a useful way to learn continuous, categorical, and vector-valued functions from examples[8,9]. The basic processing unit is the 1957-invented perceptron, which is the first supervised learning algorithm[10]. In order to define a hyperplane that separates negative and positive spaces, it receives input signals $(x_1, x_2, \ldots, x_j)$ in number form or vectors from its environment or output from other perceptrons. Each of these signals has a weight $(w_1, w_2, \ldots, w_j)$, which determines whether the instance is positive or negative 0 or 1 based on the weighted sum, $\sum_j w_j * x_j + j$. Each

input is given a weight based on how significant and influential it is on the final product relative to other inputs.[10,11]. This simplest form is effective for solving linearly separable problems, making a perceptron analogous to an Artificial Neural Network with a single neuron.

### *Previous Research*

Kolosnjaji et al.[12] developed & carried out a neural network-based method to classify unknown malware samples into pre-established malware family groups. They looked at the inclusion of a sizable number of covered layers to mimic complex functions, making malware detection easier, thanks to advancements in GPU-enabled parallel processing.

They used two different types of neural networks to build their layers: convolutional and recurrent. Convolutional networks counted only the n-grams' presence and relationship in the function call trace, using sets of n-grams of system calls instead of sequential modeling. Sequence modeling was made simpler as a result, but information fidelity might have been compromised. On the other hand, recurrent networks adopted full sequential modeling, adding a layer of complexity by relying on particular system call appearances from prior sequences. The advantage of their method is that it leverages modern hardware, such as GPUs, for effective large-scale processing, even though they only achieved an accuracy rate of 89.4%. The sandbox environment' s dynamic features, which malware can detect and avoid, could be the reason for the low accuracy rate. This could result in unnecessary series of system calls.A malware classification technique based on a neural network presented by Saxe and Berlin[13] and had two secret levels. Integrated were unchanging characteristics comprising Entropy, metadata, PE import, and a string 2D histogram. The two covered levels were made up of 1024 Parametric Rectified Linear Units (PReLu), while the output layer used a sigmoid neuron to classify events as benign or malevolent. Using a dataset of 400,000 samples, their study yielded a 95% identification rate with a 0.1% false positive rate. Notably, the greatest true positive rate was obtained for them when all static characteristics were used instead of individual features. There is no comparison between their study and deep neural networks with many hidden layers, or shallow neural networks, which may yield better results rates, even though their results demonstrate the possibility of quickly building a highly accurate, resource-efficient machine learning classification.

## Proposed Solution

### *Formation of Samples*

It has two main techniques used to obtain harmful samples. One involves setting up malware honeypots on a vulnerable machine to capture malicious samples over time, albeit this method can be time-consuming. The other approach is to collect samples from malware ' zoos' or repositories that researchers or organizations share. In order to improve model generalization, samples regarding this work are drawn from an unsafe repository that was distributed via a colleague researcher[14]. These samples include a variety of malware types, including viruses, Trojan horses, and rootkits.Clean PE samples are obtained by traversing through a clean Windows system directory. While there is no fixed rules for determining the required sample size for neural network training and testing, a data collection of size 4,000 (1,000 benign and 3,000 harmful samples) is utilized, based on previous research works and the complexity of their neural networks. Initially, as many samples as possible are collected, with numbers gradually reducing through the labeling and cleaning process.

### Labeling

The first step of the process is classifying the samples in respective folders, such "malicious_samples" for harmful samples and "benign_samples" for benign samples. The manual tagging of each file is no longer necessary thanks to this organizing strategy. However, some standards are set in order to guarantee sample usability and precise classification as benign or harmful:

- Valid PE file: Verified by looking for the "MZ" magic number in a binary file' s first two bytes.

- Removal of recognized packaged malware: determined by comparing the signature to those of well-known packers, such as the UPX packer.

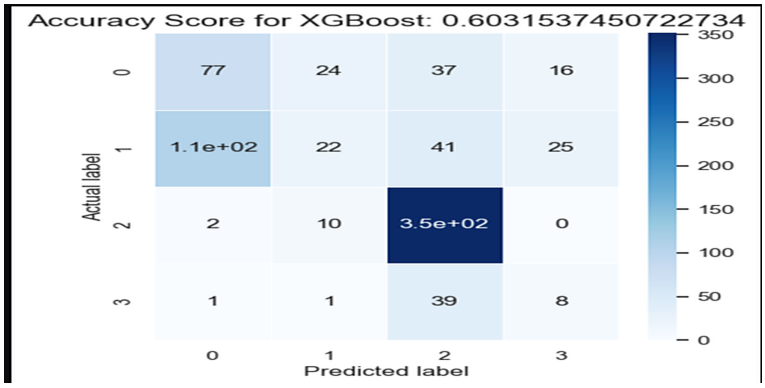
- Verification using the VirusTotal API in opposite to well-known antivirus programs: Benign samples must remain undiscovered by any AV (0%

detection rate), whereas malicious samples with a detection rate less than 75% are eliminated.

The purpose of these stages is to remove any samples that could impact the generalisation of the model..

### Extractions of Features

It is essential that Windows library function calls be extracted for every sample. These calls are represented by a binary feature vector matrix. This can only be achieved by extracting and uniquely listing the function calls for each sample. After that, every sample' s function calls are compare to this list, with the matching index in the binary vector space being assigned a value of 1. Thus, every sample is characterised by a binary vector space of same size & the total of the unique function calls performed in all sample files.




### Neural Network Classification

Malware classification recommended in this study is a basic neural network binary classifier. For every class label, a one-hot vector is utilized to describe whether the sample is benign (1,0) or malevolent (0,1).

The total number of different function calls is represented by the number of inputs (n) in the input layer. Every input node is a feature that was extracted from the binary vector space, which is the list of

| | malware | SY1 | SY2 | SY3 | SY4 | SY5 | SY6 | SY7 | SY8 | SY9 | SY10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | vundo | 656 | 228 | 36 | 48 | 112 | 6572 | 176 | 0 | 192 | 18 |
| 1 | vundo | 708 | 230 | 0 | 84 | 44 | 2802 | 54 | 0 | 82 | 24 |
| 2 | vundo | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | vundo | 860 | 88 | 0 | 164 | 20 | 3596 | 32 | 0 | 28 | 14 |
| 4 | vundo | 2 | 0 | 12 | 4 | 0 | 124 | 0 | 0 | 0 | 0 |

function calls. The optimizer function use the Gradient Descent method with a declining learning percent to update weights and reduce the model' s associated cost prior to the next epoch. Each forward pass entails back-propagation. The output neuron uses the softmax function in combine with a cross-Entropy cost-function ,which is frequently used in the output layer[15].

Fig. 1 depicts the overall design and structure of the neural network Shown in this Paper.

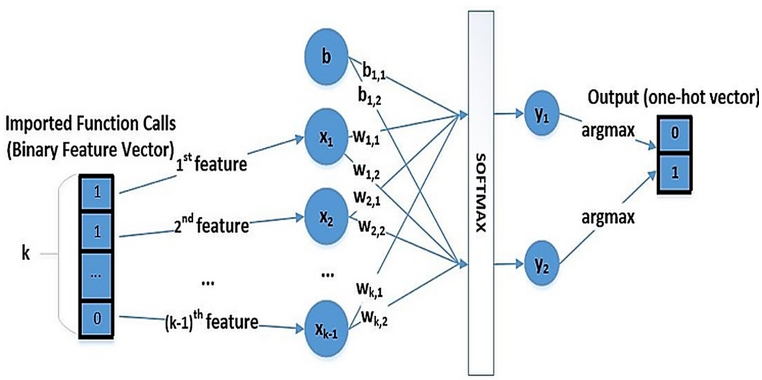



**Figure 1.** A neural network classifier with a bias unit and a softmax output function is proposed.

Table 1.Performance evaluation of the model based on XGBoost.

## Experiments

### *Trial Setup*

The recommended neural network classifier was implemented using TensorFlow-GPU, a Python package that takes advantage of GPU parallel processing capabilities. The execution environment contained an NVIDIA GeForce GTX-970 4GB GDDR5 with 5.2 Compute Capability, enabled for CUDA. Furthermore, scikit-learn was used to conduct the experiment using cross-validation. Based on a dataset

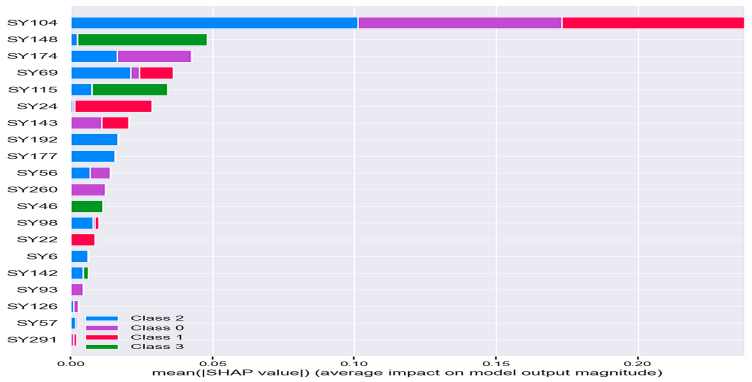



**Figure 2.** Comparative Analysis with Previous Works

of 4,000 samples, of which 1,000 were benign and 3,000 malicious, a variety of parameters were selected for the tests. 34,087 distinct Windows library function calls made up the input size following dataset preparation. They were organized into batches of 128 samples and utilized for training, validation, and testing. The training used a decay rate of 0.975 and a learning rate of 0.01 across 10 epoch runs.

### *Cross-validation Experiment*

To assess the predictive model performance against an unknown dataset in practical scenarios, validation was conducted using cross-validation techniques, specifically 10-fold cross-validation. This approach involves folding the dataset 10 times, yielding 10 unique sets of training and testing for the model to undergo training and testing cycles. The project employed metrics such as accuracy, recall, and precision for evaluation, with macro-averaging applied to the results. The data collection was divided into 29 training batches and 2 testing batches in each iteration by implementing an 80/20 split.

## Result and Analysis

Following the completion of cross-validation, the model is now ready for evaluation under the assumption that its performance in practical scenarios aligns with the achieved average results. Metrics were recorded for each iteration using the defined formulas. Table 1 presents a comparative analysis of the Accuracy score and the Predicted label results.

The chart in Fig. 2 provide a Table and visual compare of the outcomes obtained in this experiment and those from various works reviewed in the literature. The comparison is based on macro-averaged results.

From the results, it is evident that the proposed model surpasses the performance of previous neural network-based works, such as the study by Kolosnjaji et al.[12]. This implies that improved model quality isn' t always a result of using deep learning with dynamic features. Performance overhead rises as superfluous complexity is added to the model, and dynamic characteristics enhance the model' s susceptibility to generalisation deterioration. Despite using static features and deep learning as well, Saxe and Berlin' s model did not do as well as the validated model in this experiment.

This might be because their dataset was smaller and had less features than the validated model.Beyond neural networks, other studies have utilised various ML algo. and technique , and the macro-averaged model metrics should be considered in conjunction with these alternate approaches.

## Conclusion

This research presents a neural network model that uses loaded library function calls to categorize unknown Portable Executable (PE) files as harmful or benign. With mean accuracy of 97.8%, precision of 97.6%, and recall of 96.6% over a sample of 4,000 PE files—3,000 classified as malicious and 1,000 as benign—the deployed model proved to be remarkably effective. These very encouraging findings highlight the usefulness of the suggested malware categorization method, which makes it suitable for use in both academic and industrial settings.

The focus of this paper was on unpacked samples, avoiding those packed with known packers, as the utilization of static features could be compromised in such cases. Hybrid analysis, incorporating results from both static and dynamic analysis, may yield more meaningful features for classification. This research also investigates the processing of unseen files using a simple neural network binary classifier. Future research may examine the merging of deep/wide learning to create increasingly complex models for malware classification while reducing performance overhead, as hardware technology develops and GPU-enabled parallel processing becomes more complex.

## References

[1] Pulsipher, D. W., Scott, A., & Reeb, F. An Argument for a Holistic Approach to Critical Infrastructure Security.

[2] Chaudhary, S., Gkioulos, V., & Katsikas, S. (2023). A quest for research and knowledge gaps in cybersecurity awareness for small and medium-sized enterprises. Computer Science Review, 50, 100592.

[3] Schmitz-Berndt, S., & Cole, M. D. (2022). Towards an Efficient and Coherent Regulatory Framework on Cybersecurity in the EU: The Proposals for a NIS 2.0 Directive and a Cyber Resilience Act. Applied Cybersecurity & Internet Governance, 1.

[4] Tian, D., Zhao, R., Ma, R., Jia, X., Shen, Q., Hu, C., & Liu, W. (2022). MDCD: A malware detection approach in cloud using deep learning. Transactions on Emerging Telecommunications Technologies, 33(11), e4584.

[5] Zheng, L., & Zhang, J. (2022). A new malware detection method based on vmcadr in cloud environments. Security and Communication Networks, 2022.

[6] Ramesh, G., Logeshwaran, J., & Aravindarajan, V. (2022). The Performance Evolution of Antivirus Security Systems in Ultra dense Cloud Server Using Intelligent Deep Learning. BOHR International Journal of Computational Intelligence and Communication Network, 1(1), 15-19.

[7] Mishra, P., Gupta, A., Aggarwal, P., & Pilli, E. S. (2022). vServiceInspector: Introspection-assisted evolutionary bag-of-ngram approach to detect malware in cloud servers. Ad Hoc Networks, 131, 102836.

[8] Nguyen, P. S., Cuong, N. N., & Long, H. V. (2023). Cloud-Based Malware Detection Using Machine Learning Methods. In Risk Detection and Cyber Security for the Success of Contemporary Computing (pp. 171-197). IGI Global.

[9] Singh, A., Mushtaq, Z., Abosaq, H. A., Mursal, S. N. F., Irfan, M., & Nowakowski, G. (2023). Enhancing ransomware attack detection using transfer learning and deep learning ensemble models on cloud-encrypted data. Electronics, 12(18), 3899.

[10] Arunkumar, M., & Kumar, K. A. (2023). GOSVM: Gannet optimization based support vector machine for malicious attack detection in cloud environment. International Journal of Information Technology, 15(3), 1653-1660.

[11] Al-Saedi, K. H. (2023). Enhancing cloud security through the integration of deep learning and data mining techniques: A comprehensive review. Periodicals of Engineering and Natural Sciences, 11(3), 176-192.

[12] Hesabi, M., & Deypir, M. (2023). An Improved Method for Malware Attack Detection in Cloud Computing Using Collective Learning. Scientific Journal of Electronical & Cyber Defence, 10(4).

[13] Ramish, M., Sinha, A., Desai, J., Raj, A., Rajawat, Y. S., & Punia, P. (2022, April). IT Attack Detection and Classification using Users Event Log Feature And Behavior Analytics through Fourier EEG Signal. In 2022 IEEE 11th International Conference on Communication Systems and Network Technologies (CSNT) (pp. 577-582). IEEE.

[14] Miranda, T. J. C. (2022). Profiling and Visualizing Android Malware Datasets (Doctoral disertation, CentraleSupélec)

[15] Aljuhani, A., Kumar, P., Kumar, R., Jolfaei, A., & Islam, A. N. (2022). Fog intelligence for secure smart villages: Architecture, and future challenges. IEEE Consumer Electronics Magazine.