PriMera Scientific Engineering
Volume 6 Issue 5 May 2025
DOI: 10.56831/PSEN-06-197
ISSN: 2834-2550

PriMera Scientific Publications

# Static Malware Classification: A Comparative Analysis of Machine Learning Techniques on Byte and Opcode Features

**Type:** Research Article
**Received:** March 29, 2025
**Published:** April 28, 2025

**Citation:**
Chijioke Erasmus Ogbonna., et al. "Static Malware Classification: A Comparative Analysis of Machine Learning Techniques on Byte and Opcode Features". PriMera Scientific Engineering 6.5 (2025): 19-29.



**Chijioke Erasmus Ogbonna[1]*, Michael Nsor[2], Praise Onyehanere[3], Husein Harun[4], Aghoghomena Emadoye[5], Ifunanya Obinwanne[6], Gloria Nwachukwu Ogochukwu[7] and JOHN Joshua Junior[8]**

[1]*Northumbria University, 18 Towpath Walk E9 5HX, London, United Kingdom*

[2]*The School of Computer Science, Western Illinois University 1 Westford pl, apt 2, Allston, Massachusetts, USA*

[3]*University of Brighton, England, United Kingdom*

[4]*University of Nebraska-Lincoln, Lincoln, Nebraska, USA*

[5]*10Alytics Inc, 18602 Alderwood Mall Pkwy, Lynnwood, WA 98037 USA*

[6]*Computer Information and Decision Management, West Texas A&M University, USA*

[7]*Rensselaer Polytechnic Institute, Troy, New York (110 8th Street, Troy, NY 12180-3590), Information Technology, USA*

[8]*Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy*

***Corresponding Author:** Chijioke Erasmus Ogbonna, Northumbria University, 18 Towpath Walk E9 5HX, London, United Kingdom.

## Abstract

Malware classification is essential for cybersecurity, enabling early detection and mitigation of threats. This study investigates the efficacy of static analysis combined with various machine learning algorithms for identifying malware families. We extract three distinct feature sets from program binaries: byte histograms, opcode frequencies, and section distributions. A comprehensive comparative analysis is conducted using a suite of traditional machine learning classifiers, including Decision Tree, Support Vector Classifier, K-Nearest Neighbors, Naive Bayes, Stochastic Gradient Descent, Logistic Regression, and Random Forest, alongside a foundational Deep Neural Network. Evaluation on the completed byte histogram and opcode frequency feature sets reveals significant performance variations across models and feature representations. Opcode frequency features generally yield superior classification accuracy compared to byte histograms. Notably, Logistic Regression consistently demonstrates high accuracy and low false positive rates on both feature sets, achieving up to 92.9% accuracy with opcode features. While traditional models like Logistic Regression and SVC perform strongly, the initial evaluation of the basic DNN architecture shows promising improvement on opcode data but requires further

exploration. These findings underscore the continued relevance of static feature engineering and comparative model analysis in malware detection, highlighting opcode patterns as particularly discriminative. Analysis incorporating section distribution features is ongoing to provide a more complete understanding.

***Keywords:*** Malware Classification; Static Analysis; Machine Learning; Cybersecurity; Digital Forensics; Byte Histograms; Opcode Frequencies

## Introduction

Malware, a term derived from "malicious software," encompasses a variety of software programs designed to infiltrate, damage, or disrupt computer systems. As cyber threats continue to evolve in complexity, malware remains a significant concern for individuals, businesses, and governments worldwide. Cybercriminals leverage sophisticated techniques such as polymorphism, obfuscation, and packing to evade traditional security measures, making malware detection an ongoing challenge. The rapid proliferation of malware variants, coupled with the increasing sophistication of attack methodologies, necessitates the development of advanced detection mechanisms.

Traditional malware detection techniques primarily rely on *signature-based* and *heuristic-based* approaches [1]. Signature-based detection involves maintaining a database of known malware signatures and scanning files for matches. While effective against previously identified threats, this method fails against *zero-day attacks* and novel malware strains [2].

Heuristic-based approaches aim to detect unknown threats by analyzing behavioral patterns and code structures. However, these methods often suffer from high false positive rates and are computationally expensive [3]. Given these limitations, the research community has turned toward *machine learning (ML) and deep learning (DL) techniques* to enhance malware classification and detection capabilities.

### *Machine Learning for Malware Classification*

The ever-increasing volume and sophistication of malicious software pose a significant threat to digital security. Manual analysis of malware is infeasible given the sheer scale and rate of new sample generation. Machine learning offers a powerful and scalable alternative, enabling automated analysis and classification by learning patterns inherent in malware characteristics. Applied to cybersecurity, machine learning algorithms can process large datasets of executable files, identifying signatures, behaviors, or structural properties indicative of malicious intent.

Machine learning approaches to malware classification broadly involve extracting features from samples, either through static analysis (examining the code without execution) or dynamic analysis (observing runtime behavior), and training models to distinguish between benign and malicious instances or classify samples into specific malware families. A wide array of machine learning techniques are applicable to this domain, ranging from classical algorithms like Support Vector Machines, Decision Trees, and Naive Bayes, which excel at pattern recognition on structured feature sets, to more advanced methods like deep neural networks, capable of automatically learning complex, hierarchical representations directly from raw or minimally processed data. The effectiveness of any given machine learning method hinges on the quality and relevance of the input features and the model's capacity to generalize from the training data to unseen samples. This study explores the comparative performance of diverse machine learning techniques, utilizing static features derived from program binaries, to identify robust methods for effective malware classification.

### *Research Objectives and Contributions*

This study is motivated by the critical need for effective and scalable automated malware classification techniques. To address this, our research pursues the following primary objectives:

- To conduct a rigorous comparative analysis of diverse machine learning models, encompassing both traditional algorithms and a foundational deep neural network, for the task of static malware classification.
- To evaluate and contrast the discriminative capabilities of distinct static feature representations—specifically byte histograms, opcode frequencies, and section distributions—in facilitating accurate malware family identification.
- To provide a detailed and reproducible methodology for the extraction of these static features and the subsequent training and evaluation of machine learning classifiers.
- To quantify and report the performance of each evaluated model on each feature set using standard classification metrics, thereby offering empirical evidence of their respective strengths and weaknesses in this domain.

In pursuing these objectives, this work offers several key contributions to the field of cybersecurity and machine learning for malware analysis:

- A systematic comparative performance analysis of a broad spectrum of machine learning models applied to static malware features.
- An empirical assessment of the effectiveness of byte, opcode, and section-based static features for malware classification.
- A transparent and replicable experimental framework detailing feature engineering and model evaluation procedures.
- Quantitative results providing insights into the most promising machine learning techniques and static feature types for further development in static malware detection systems.

***Paper Organization***

The remainder of this paper is organized as follows:

- ***Section II*** reviews related work in machine learning-based malware detection.
- ***Section III*** describes the methodological approach, encompassing dataset collection, feature extraction, and preprocessing techniques.
- ***Section IV*** presents the experimental results and comparative analysis.
- ***Section V*** discusses the insights gained from the research as well as limitations and future research directions.
- ***Section VI*** concludes the paper with key findings.

## Related Work

The persistent and evolving threat of malware has driven extensive research into automated detection and classification techniques. Traditional signature-based methods, while effective against known threats, struggle to identify novel or obfuscated malware variants due to their reliance on predefined patterns [4]. This limitation has spurred significant interest in machine learning based approaches, which can learn generalizable patterns from large datasets of both benign and malicious samples [5, 6]. Machine learning offers the potential to detect previously unseen malware and adapt to the constantly changing threat landscape [7].

Static analysis, which examines executable files without execution, is a fundamental technique in malware analysis [8]. It offers advantages in terms of speed and scalability compared to dynamic analysis, and can detect dormant code that might not be triggered during runtime observation [9]. Various static features have been explored in the literature to characterize malware. Byte-level features, such as byte n-grams and frequency distributions (histograms), capture the raw binary content and have been widely used due to their simplicity and independence from file structure knowledge [10, 11]. Studies have shown the effectiveness of byte n-grams of varying lengths for malware classification [10, 11].

Another crucial source of static features is the disassembled code, particularly opcode sequences [12]. Analysis of opcode frequencies or sequences can reveal characteristic instruction patterns indicative of malicious functionality [13]. Research has explored using opcode sequences as features for machine learning models, including approaches that transform sequences into graphical representations or use n-grams of opcodes [12, 14].

Structural information within executable file formats, such as the Portable Executable (PE) format for Windows binaries, also provides valuable static features. PE header information and section characteristics (e.g., section names, sizes, entropy) can offer insights into the file's structure and potential anomalies introduced by malware, such as packing or obfuscation [15, 16]. Studies have utilized PE header fields and section properties as features for classification [17].

Machine learning models have been extensively applied to static malware analysis using these feature sets. Comparative studies have evaluated the performance of various traditional classifiers. For instance, research has compared models like Support Vector Machines (SVM), Decision Trees, K-Nearest Neighbors (KNN), Naive Bayes, Random Forest, and Logistic Regression for malware detection and classification based on static features, demonstrating varying levels of effectiveness depending on the dataset and feature representation [5, 7, 18]. Some studies highlight models like Random Forest or SVM as strong performers on certain static feature sets [5, 18].

More recently, deep learning models have gained traction in static malware analysis, leveraging their ability to automatically learn complex features. Approaches include treating raw binary files as images and applying Convolutional Neural Networks (CNNs) to capture spatial patterns [19], or using Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks to process sequential data like opcode streams or API call sequences [13, 20]. While advanced deep learning architectures show promise, exploring the performance of foundational deep neural networks in comparison with established traditional methods on well-defined static features remains a relevant area of investigation to understand their relative efficacy and the potential benefits of increased model complexity [6].

This study builds upon this existing body of work by conducting a focused comparative analysis of a range of machine learning techniques, including a foundational deep neural network, on three distinct and commonly used static feature sets: byte histograms, opcode frequencies, and section distributions. By providing a detailed, reproducible methodology and empirical results, this research contributes to a clearer understanding of the relative strengths of these features and models for static malware classification.

## Methodology

This methodology enhances the investigation of the efficacy of various machine learning models for malware classification based on static features extracted from program binaries. It encompasses data preprocessing, feature engineering from distinct static representations, model training, and comprehensive evaluation, detailed below to ensure reproducibility. Figure 1 provides a high-level overview of our methodological approach.

### *Dataset and Initial Processing*

The research uses a dataset comprising malware samples provided in two formats: hexadecimal byte representations (.bytes files) and disassembled code (.asm files). Corresponding ground truth labels, indicating the malware family (Class), are sourced from an accompanying trainLabels.csv file which was manually collected and annotated. This file maps each sample's unique identifier, to one of nine distinct malware families. These families are denoted by simple numerical labels ranging from 1 to 9. Our initial processing step involves loading this label mapping into a Pandas DataFrame, creating an efficient lookup structure essential for associating features with their correct class during subsequent stages.

### *Feature Engineering*

We extracted three distinct sets of features from the raw sample files to evaluate their respective discriminative power: byte histograms, opcode frequencies, and section distributions

1. ***Byte Histogram Features***: For each sample's .bytes file, we performed frequency analysis on its hexadecimal content. Each file was read line by line. Lines were tokenized, discarding the initial offset address. The remaining tokens, representing hexadecimal byte values (excluding placeholder '??' values), were counted. This process generated a dictionary mapping each unique byte value (e.g., '00' through 'FF') to its frequency within the file. This frequency dictionary, along with the sample's corresponding class

label, formed a single feature vector instance. All instances were compiled into a Pandas DataFrame. Any resulting Not-a-Number (NaN) values, potentially arising from files lacking certain byte values present in others, were explicitly converted to zero using *numpy.nan_to_num* to ensure numerical stability for model training.

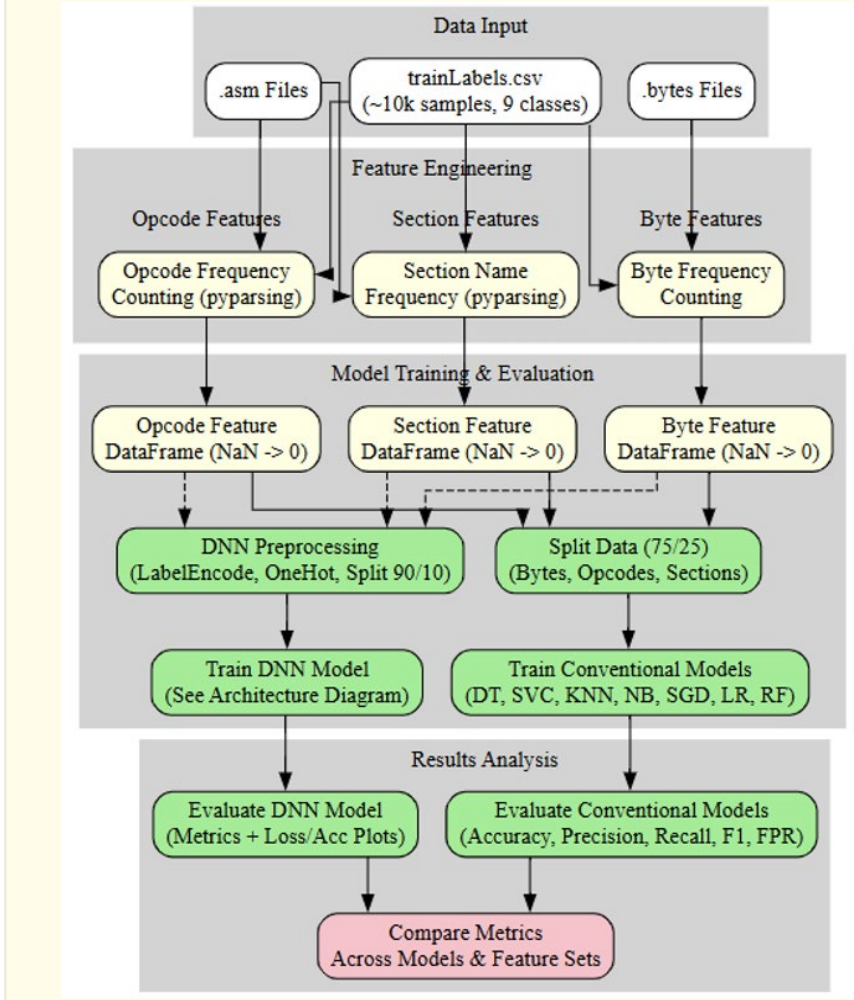


***Figure 1:*** Methodology Workflow Diagram.

2. ***Opcode Frequency Features***: Features were derived from the *.asm* files by analyzing the frequency of assembly language operational codes (opcodes). We utilized the *pyparsing* library to define a parsing grammar targeting lines within the *.text* segment of the assembly code. This grammar specifically extracted instruction opcodes (e.g., *mov*, *jmp*, *push*). For each *.asm* file, we iterated through its lines, applying the parser. Successfully parsed opcodes were aggregated, counting the occurrences of each unique opcode. Similar to the byte features, this frequency dictionary, augmented with the class label, constituted the feature vector for the sample. These vectors were assembled into a DataFrame, and NaN values were imputed with zero via *numpy.nan_to_num*.

3. ***Section Distribution Features***: We analyzed the structural composition of the binaries by extracting information about their sections from the *.asm* files. Again using *pyparsing*, a simpler grammar was defined to identify lines declaring section names (e.g., *.text*, *.data*, *.rdata*). The frequency of each unique section name declaration within a file was counted. This count dictionary, along with the class label, formed the feature vector. A DataFrame was constructed from these vectors, with NaN imputation performed as described previously.

***Model Training and Evaluation Setup***

For each of the three feature sets (Bytes, Opcodes, Sections), we partitioned the data into training and testing subsets. For conventional machine learning models, we employed *sklearn.model_selection.train_test_split* using the default 75% training and 25% testing split, setting *random_state* = 0 for deterministic partitioning crucial for reproducibility.

The following classification algorithms from the scikit-learn library were trained and evaluated:

- ***Decision Tree (DT):*** Configured with *criterion="gini"* and *max_depth=3*.
- ***Support Vector Classifier (SVC):*** Employed a linear kernel (*kernel='linear'*) with regularization parameter C=1.
- ***K-Nearest Neighbors (KNN):*** Implemented with k=7 neighbors (*n_neighbors=7*).
- ***Gaussian Naive Bayes (GNB):*** Utilized the default *GaussianNB* implementation.
- ***Stochastic Gradient Descent (SGD):*** Trained as a linear classifier with default hinge loss, *alpha=0.001* for regularization, and *max_iter=100*.
- ***Logistic Regression (LR):*** Applied with default settings (*random_state=0*).
- ***Random Forest (RF):*** Configured with *max_depth=2* and *random_state=0*.

Additionally, a Deep Neural Network (DNN) was implemented using Keras. Figure 2 illustrates the model architecture for this neural network.

For the DNN, a separate preprocessing step was required:

- ***DNN Preprocessing***: Class labels were transformed into numerical format using *sklearn.preprocessing.LabelEncoder* and then converted into a one hot encoding using *keras.utils.np_utils.to_categorical*. The entire feature set and encoded labels were split into 90% training and 10% testing sets using *train_test_split with test_size=0.1* and *random_state=0*.
- ***DNN Architecture***: A sequential model was constructed with an input layer matching the feature dimension, followed by three hidden layers containing 10 neurons each (using 'relu' activation), and an output layer with neurons equal to the number of classes using 'softmax' activation.
- ***DNN Training***: The model was compiled using the 'adam' optimizer and 'categorical_crossentropy' loss function. Training was performed for 10 epochs (*epochs=10*) with a *batch_size=2*. Validation was performed on the test set during training.

***Performance Metrics***

Model performance was assessed using standard classification metrics provided by *sklearn.metrics*:

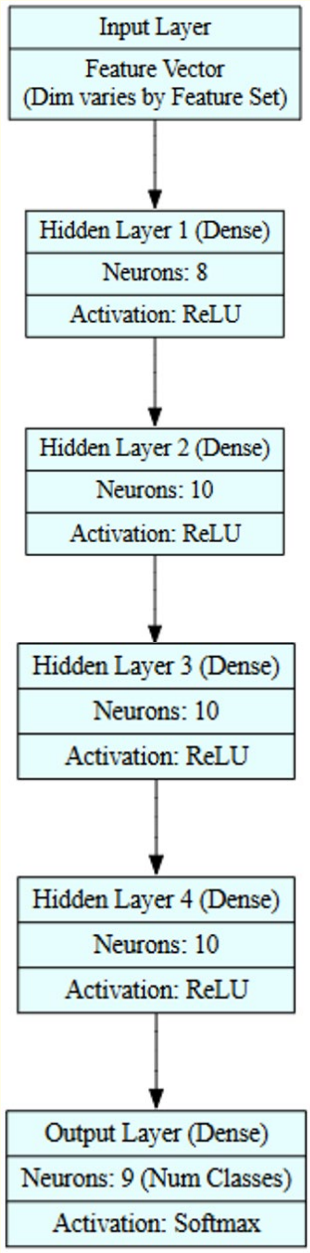


***Figure 2:*** DNN Model Architecture Diagram.

- ***Accuracy***: Overall proportion of correctly classified instances.
- ***Precision, Recall (TPR), F1 Score***: Calculated on a per class basis and then averaged using average='weighted' to account for potential class imbalance. Recall is equivalent to the True Positive Rate (TPR).
- ***False Positive Rate (FPR)***: Computed explicitly from the confusion matrix. For each class i, $FPR_i = FP_i / (FP_i + TN_i)$, where FP is the count of false positives and TN is the count of true negatives. The reported FPR is the average FPR across all classes.

## Results

This section presents the experimental outcomes detailing the performance of various machine learning classifiers applied to malware classification tasks using distinct static feature sets derived from program binaries. We report key performance indicators including Accuracy, Precision, Recall (True Positive Rate), F1 Score, and False Positive Rate for models trained on Byte Histogram features and Opcode Frequency features.

### *Performance with Byte Histogram Features*

Evaluation on the Byte Histogram feature set revealed varying degrees of model effectiveness, with results summarized in Table 1.

| *Classifier* | *Accuracy* | *Precision* | *Recall (TPR)* | *F1- Score* | *FPR* |
|---|---|---|---|---|---|
| Decision Tree | 0.7714 | 0.7911 | 0.7714 | 0.7777 | 0.0309 |
| SVC | 0.8571 | 0.8672 | 0.8571 | 0.8506 | 0.0221 |
| K-Nearest Neighbors | 0.7429 | 0.7287 | 0.7429 | 0.7311 | 0.0369 |
| Naive Bayes | 0.7286 | 0.7813 | 0.7286 | 0.7390 | 0.0331 |
| SGD | 0.6857 | 0.5971 | 0.6857 | 0.6286 | 0.0475 |
| Logistic Regression | **0.9000** | **0.8972** | **0.9000** | **0.8951** | **0.0138** |
| Random Forest | 0.7286 | 0.5962 | 0.7286 | 0.6441 | 0.0432 |
| DNN | 0.5357 | 0.4446 | 0.5357 | 0.4786 | 0.0701 |

***Table 1:*** Performance Metrics for Byte-based Classifiers.

Logistic Regression demonstrated superior performance, achieving the highest accuracy at 90.0% and the lowest False Positive Rate (FPR) at approximately 0.014. This indicates a strong ability to correctly classify instances while minimizing misclassifications of benign samples as malicious (or vice versa, depending on class definition and averaging). The model also secured high Precision (0.897) and F1 Score (0.895), underscoring its balanced performance across classes.

The Support Vector Classifier (SVC) also performed commendably, obtaining an accuracy of 85.7% and an FPR of approximately 0.022. Decision Tree and Naive Bayes models yielded moderate results, with accuracies around 77.1% and 72.9% respectively. Their FPRs were correspondingly higher than Logistic Regression and SVC, at approximately 0.031 and 0.033.

Simpler models like K-Nearest Neighbors (KNN) and Random Forest showed lower accuracy, 74.3% and 72.9% respectively, coupled with higher FPRs (approximately 0.037 for KNN and 0.043 for Random Forest). The Stochastic Gradient Descent (SGD) classifier exhibited the lowest accuracy (68.6%) and the highest FPR (approximately 0.048) among the conventional models.

The Deep Neural Network (DNN), despite its complexity, registered the lowest performance on the Byte Histogram features, with an accuracy of 53.6% and a notably high FPR of approximately 0.070. This suggests that for this specific feature representation and dataset scale, the defined DNN architecture did not effectively capture the discriminative patterns compared to traditional methods.

### *Performance with Opcode Frequency Features*

Transitioning to the Opcode Frequency features, a general improvement in classification performance was observed across several models compared to the Byte Histogram set. Table 2 provides an overview of the performance results. Logistic Regression again emerged as the leading model, achieving an accuracy of 92.9% and an exceptionally low FPR of approximately 0.010. Its Precision (0.932) and F1 Score (0.924) further solidify its robust performance on this feature set.

The Decision Tree model showed significant improvement, reaching an accuracy of 84.3% with an FPR of approximately 0.021. Both SVC and K-Nearest Neighbors also performed well, with accuracies of 82.9% and 85.7% respectively, and similarly low FPRs (approx-

imately 0.024 for SVC and 0.022 for KNN). Naive Bayes maintained a strong performance with an accuracy of 84.3% and an FPR of approximately 0.030.

SGD performance improved on Opcode features, achieving 78.6% accuracy and an FPR of approximately 0.031. Random Forest, however, continued to exhibit lower relative performance, with an accuracy of 75.7% and a higher FPR of approximately 0.046.

Notably, the DNN model also demonstrated improved performance on the Opcode Frequency features compared to the Byte features, reaching an accuracy of 75.0%. While its FPR of approximately 0.049 was still higher than most conventional models, this result indicates that opcode-based patterns were more effectively learned by the network architecture than byte frequencies.

Overall, Opcode Frequency features appear to yield generally better classification results than Byte Histogram features with the models evaluated, with Logistic Regression consistently delivering the most accurate and precise classifications while maintaining low false positive rates across both completed feature sets. The analysis of Section Distribution features will provide further insight into the most effective static representation for this malware classification task.

| *Classifier* | *Accuracy* | *Precision* | *Recall (TPR)* | *F1- Score* | *FPR* |
|---|---|---|---|---|---|
| Decision Tree | 0.8429 | 0.8898 | 0.8429 | 0.8510 | 0.0215 |
| SVC | 0.8286 | 0.8362 | 0.8286 | 0.8257 | 0.0244 |
| K-Nearest Neighbors | 0.8571 | 0.8672 | 0.8571 | 0.8485 | 0.0216 |
| Naive Bayes | 0.8429 | 0.8149 | 0.8429 | 0.8175 | 0.0298 |
| SGD | 0.7857 | 0.8110 | 0.7857 | 0.7745 | 0.0305 |
| Logistic Regression | **0.9286** | **0.9323** | **0.9286** | **0.9238** | **0.0098** |
| Random Forest | 0.7571 | 0.7233 | 0.7571 | 0.7018 | 0.0457 |
| DNN | 0.7500 | 0.6327 | 0.7500 | 0.6694 | 0.0488 |

***Table 2:*** Performance Metrics for Opcode-based Classifiers.

## Discussion

This research undertook a comparative analysis of machine learning models and static feature sets for malware classification, providing empirical insights into their effectiveness. The experimental results from the byte histogram and opcode frequency features reveal several key observations regarding the discriminative power of these static representations and the performance characteristics of the evaluated classifiers.

A principal finding is the generally superior performance achieved when using opcode frequency features compared to byte histogram features across most evaluated models. This suggests that features derived from the operational codes within the disassembled binary capture more semantically relevant information about the program's intended actions than simple byte value distributions. Opcodes represent the fundamental instructions executed by the processor, and their patterns can reflect underlying code structures and functionalities, which are often distinct between benign and malicious software or across different malware families. Byte histograms, while providing a foundational view of the binary composition, may be more susceptible to noise or less directly indicative of control flow and logic.

Among the evaluated models, Logistic Regression consistently demonstrated exceptional performance on both feature sets, achieving the highest accuracy and lowest false positive rates. This highlights the robustness and effectiveness of this linear model for static malware classification when coupled with well-engineered frequency-based features. The strong performance of Logistic Regression, alongside other traditional models like SVC and K-Nearest Neighbors, suggests that for these specific static feature representations and dataset characteristics, linear separability or relatively simple decision boundaries are sufficient to achieve high classification accuracy.

The initial performance of the foundational Deep Neural Network was comparatively lower than several traditional models, particularly on the byte histogram features. While its performance improved with opcode features, it did not surpass the leading traditional classifiers. This outcome may be attributed to several factors. The defined DNN architecture, comprising a limited number of layers and neurons, might be too simplistic to fully leverage the potential of deep learning for complex pattern recognition in this domain, especially when compared to architectures specifically designed for sequence or image data (like LSTMs or CNNs).

Furthermore, the size and specific characteristics of the dataset used in this study may not be sufficient to allow a basic DNN to learn the highly complex, hierarchical representations that are the hallmark of deep learning's success in other fields. Traditional models, with fewer parameters, may be less prone to overfitting or may converge more effectively on datasets of this scale and feature complexity.

Despite the valuable insights gained, this study has certain limitations. The evaluation was conducted on a specific dataset, and the generalizability of these findings to other malware collections or evolving threats requires further validation. The focus solely on static analysis, while providing efficiency, inherently misses behavioral indicators that dynamic analysis could reveal, particularly for heavily obfuscated or polymorphic malware. The deep learning component was limited to a basic feedforward network; exploring more sophisticated deep learning architectures tailored for binary or sequential data is essential to fully assess their potential in this context.

These limitations suggest several promising avenues for future research. Expanding the evaluation to larger and more diverse datasets is critical for confirming the generalizability of model and feature performance. Integrating dynamic analysis features with static features in a hybrid approach could yield more robust classification systems capable of handling a wider range of malware evasion techniques. Investigating advanced deep learning architectures, such as CNNs for binary image representations, LSTMs for opcode or API call sequences, or Transformer networks, is necessary to determine if increased model complexity can lead to significant performance gains on static features. Furthermore, exploring feature selection or dimensionality reduction techniques could potentially improve model efficiency and performance, particularly for high-dimensional feature spaces like byte n-grams. Finally, future work could delve into explainable AI techniques to provide insights into why certain models make specific classification decisions, enhancing trust and enabling security analysts to better understand the indicators of maliciousness learned by the models.

## Conclusion

This research systematically investigated the application of machine learning techniques for static malware classification, focusing on the comparative efficacy of various models and distinct static feature representations. We explored features derived from byte histograms, opcode frequencies, and section distributions, evaluating a suite of traditional machine learning classifiers alongside a foundational deep neural network.

The experimental results, particularly from the byte and opcode feature sets, demonstrate that the choice of both feature representation and classification model significantly impacts performance. Opcode frequency features consistently provided a stronger basis for classification than byte histogram features, indicating their richer semantic content regarding program functionality. Among the evaluated models, Logistic Regression proved exceptionally effective, achieving high accuracy and low false positive rates on both completed feature sets, highlighting its suitability for this task when utilizing frequency-based static features. While the initial deep neural network architecture did not outperform the leading traditional models, its improved performance on opcode features suggests potential for more sophisticated deep learning approaches.

This study contributes to the field by providing a detailed comparative analysis of widely used static features and machine learning models, offering empirical evidence to guide future research and development in static malware analysis. The reproducible methodology presented facilitates further investigation and validation. Although limitations exist, including dataset scope and the foundational nature of the deep learning model explored, the findings underscore the continued value of static analysis and machine learning for automated malware classification. Future work will incorporate the analysis of section distribution features and explore advanced deep learning architectures and hybrid analysis approaches to enhance detection capabilities against evolving malware threats.

## References


1. OA Aslan and R Samet. "A Comprehensive Review on Malware Detection Approaches". IEEE Access 8.1 (2020): 6249-6271.
2. AK Chakravarty., et al. "A study of signature-based and behaviour-based malware detection approaches". Int. J. Adv. Res. Ideas Innov. Technol 5.3 (2019): 1509-1511.
3. Z Bazrafshan., et al. "A survey on heuristic malware detection techniques". in Proc. 5th Conf. Inf. Knowl. Technol. (IKT) (2013): 113-120.
4. S Altaha and K Riad. "Machine Learning in Malware Analysis: Current Trends and Future Directions". International Journal of Advanced Computer Science and Applications 15.1 (2024): 124-131.
5. PK Anajani., et al. "Static Malware Analysis Using Optimal Machine Learning Algorithm for Malware Detection". NeuroQuantology 20.10 (2022): 4128.
6. M Miraoui and MB Belgacem. "Binary and multiclass malware classification of Windows portable executable using classic machine learning and deep learning". Frontiers in Computer Science 7 (2025).
7. B Abduraimova., et al. "Comparative study of machine learning applications in malware forensics". [Online]. CPITS-II 2024: Workshop on Cybersecurity Providing in Information and Telecommunication Systems II (2024): 139-152.
8. A Shabtai., et al. "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey". Information Security Technical Report 14.1 (2009): 16-29.
9. M Hassen, MM Carvalho and PK Chan. "Malware classification using static analysis based features". IEEE Xplore (2017).
10. E Raff., et al. "An investigation of byte n-gram features for malware classification". Journal of Computer Virology and Hacking Techniques 14.1 (2016): 1-20.
11. S Jain and YK Meena. "Byte Level n-Gram Analysis for Malware Detection". Communications in Computer and Information Science (2011): 51-59.
12. A Mello., et al. "Malware identification on Portable Executable files using Opcodes Sequence". Congresso Brasileiro de Inteligência Computacional (2023): 1-8.
13. A Lakshmanarao and M Shashi. "Android Malware Detection with Deep Learning using RNN from Opcode Sequences". International Journal of Interactive Mobile Technologies (iJIM) 16.01 (2022): 145-157.
14. I Santos., et al. "Using opcode sequences in single-class learning to detect unknown malware". IET Information Security 5.4 (2011): 220.
15. Offwhite Security. "Section Headers". [Online].
16. ASTRA Labs. "PE Header Fundamentals: The First Step in Malware Analysis". (2024) [Online].
17. Y Liao. "PE-Header-Based Malware Study and Detection". UGA School of Computing [Online].
18. B Khan, M Arshad and Sajid Ullah Khan. "Comparative Analysis of Machine Learning Models for PDF Malware Detection: Evaluating Different Training and Testing Criteria". Journal of cyber security 5.0 (2023): 1-11.
19. L Chen. "Deep Transfer Learning for Static Malware Classification". arXiv.org (2018).
20. S Zhang., et al. "A Malware-Detection Method Using Deep Learning to Fully Extract API Sequence Features". Electronics 14.1 (2025): 167-167.