The ISC Int'l Journal of Information Security
ISeCure

October 2025, Volume 17, Number 3 (pp. 1–11)
http://www.isecure-journal.org

Presented at the ISCISC'2025 in Tehran, Iran.

Static Malware Detection in Windows Executables Using Deep Neural Networks and Custom Binary Features **

Sajjad Rezaei [1,*], Ali Fanian [1]
[1] *Department of Computer Engineering, Isfahan University of Technology, Isfahan, Iran.*

ARTICLE INFO.

Keywords:
Binary Feature Extraction, Deep Neural Network, Malware Classification, Portable Executable Files, Static Analysis

Type:

doi:

ABSTRACT

The extensive use of malware targeting Windows systems, particularly through Portable Executable (PE) files, has prompted significant research into malware detection. Although many approaches have been proposed, the increasing complexity and evasiveness of modern malware continue to present substantial challenges, underscoring the need for further advancements in detection strategies. This paper introduces a static malware detection framework based on deep learning and a set of carefully engineered binary features extracted directly from raw PE files. In contrast to conventional methods that rely on metadata or dynamic analysis, our approach performs detailed parsing of file headers, section layouts, entropy levels, import/export tables, and embedded resources to form a comprehensive feature set. A deep neural network is trained on these features, with its architecture and hyperparameters fine-tuned using Bayesian optimisation. The model is evaluated on a balanced dataset of benign and malicious PE files, achieving high accuracy (98.83%) and an F1-score of 98.95%. Fully automated and independent of dynamic execution or commercial tools, the proposed solution is well-suited for deployment in real-world applications such as antivirus systems and intrusion detection platforms.

1 Introduction

Malware remains a major threat to cybersecurity, particularly on Windows systems where attackers frequently use Portable Executable (PE) files to deploy malicious payloads [1]. Traditional detection techniques, especially those relying on signature-based antivirus engines or sandboxing, have struggled to keep pace with the increasing volume and sophistication of modern malware [1]. These methods are often limited by their dependence on known patterns, susceptibility to obfuscation, and high computational costs in dynamic analysis environments [2]. Furthermore, while signature-based antivirus engines remain fundamental tools in Windows malware detection, they face increasing challenges when dealing with novel and evolving threats that do not match predefined signatures [3].

To overcome these limitations, static malware detection has emerged as a complementary approach [4].

* Corresponding author.
**The ISCISC'2025 program committee effort is highly acknowledged for reviewing this paper.
Email addresses: s.rezaei@ec.iut.ac.ir, a.fanian@iut.ac.ir
ISSN: 2008-2045

Static methods offer faster and more scalable solutions since they do not require code execution [4]. However, many existing static approaches rely either on high-level metadata or demand expert-driven feature engineering. Additionally, they often depend on datasets that have been cleaned or enhanced by commercial tools, making them less applicable in real-world settings where raw and noisy binaries must be analysed under constrained resources [4]. Recent studies such as [5] show that hybrid static models using PE structure and deep learning can offer significant gains, but challenges remain, especially when dealing with obfuscated or adversarially manipulated files.

Moreover, despite advances in machine learning-based malware detection, prior work offers limited critical reflection on the generalisability of proposed models [2]. Many approaches perform well under ideal conditions but falter when faced with damaged or irregular PE structures, raising concerns about their robustness in deployment. This highlights the need for more resilient, interpretable, and fully automated solutions that work directly on raw executable content without reliance on auxiliary tools or manually labelled metadata.

In this paper, we propose a deep learning-based malware detection framework that parses and analyses raw PE files to extract a diverse set of carefully engineered binary features. These include header fields, section characteristics, entropy measurements, import/export table statistics, and resource-based attributes. Our system trains a deep neural network (DNN) on these features and employs Bayesian Optimisation to fine-tune the model's architecture and hyperparameters automatically, ensuring both high performance and generalisation.

We evaluate our approach on a balanced dataset of malicious and benign PE files and demonstrate its effectiveness through quantitative metrics such as accuracy, F1-score, and ROC-AUC. These results highlight the feasibility of using static binary analysis coupled with optimised deep learning models for scalable malware classification.

The remainder of this paper is organised as follows: Section 2 reviews recent work in static malware detection and feature-based methods. Section 3 presents our proposed methodology, including feature extraction, preprocessing, and deep neural network architecture. Section 4 describes the experimental setup and presents quantitative results. Section 5 discusses the implications and limitations of our approach, followed by conclusions in Section 6.

2 Related Work

Static malware detection using machine learning and deep learning has gained considerable attention due to its scalability and efficiency compared to dynamic analysis [2]. This section reviews several recent studies that employ static analysis techniques, particularly those based on Portable Executable (PE) file structure, and evaluates their strengths and limitations.

Recent comprehensive surveys highlight the growing importance of deep learning in malware detection. Redhu *et al.* [6] conducted an extensive review following PRISMA guidelines, analysing deep learning models from 2015 to 2023. Their study evaluated various architectures, including Recurrent Neural Networks, Deep Autoencoders, LSTM, Deep Neural Networks, and Deep Convolutional Neural Networks, demonstrating consistently high accuracy and low false-positive rates in real-world scenarios. Gibert *et al.* [7] examined the evolution of machine learning approaches for malware detection and classification, noting significant research developments and ongoing challenges in the field.

In the domain of PE file analysis, Yousuf *et al.* [3] designed a comprehensive static detection pipeline that extracts multiple feature sets, including imported DLLs, API functions, PE header attributes, and section characteristics. Their approach achieved high detection rates by combining 52 PE header attributes with 100 section attributes. However, their methodology relies heavily on accurate PE parsing, which may be compromised by malformed or adversarial binaries.

Kumar *et al.* [8] proposed a machine learning model for detecting malicious PE files aiming for high accuracy with low computational overhead. They developed an integrated feature set combining raw PE header field values with derived features. These derived features were computed by validating raw header fields against standard guidelines, including properties like file entropy, compilation year, packer information, and counts of suspicious sections. Using algorithms such as Decision Tree and Random Forest, their model achieved **98.4%** classification accuracy in 10-fold cross-validation with the integrated feature set. This result demonstrated a 15% improvement over using raw features alone when tested on a novel dataset. They emphasised that PE header features are most effective when used as complementary features.

More recently, Rezaei and Hamze [9] introduced an efficient approach for malware detection focusing on PE header specifications, driven by the need for fast and real-time detection systems. Their method

is distinct in that it extracts only nine static key features from the PE file header and its structure. These features were specifically chosen for their significant differences between malware and benign files, intentionally avoiding raw header field values that might not generalise well. Examples of these features include the number of non-standard sections, empty section names (Boolean), a decimal representation of data directories, file and section entropies, checksum status (Boolean for zero value), and the number of `FileInfo` fields. Employing Random Forest, SVM, and kNN classifiers, their approach achieved **95.5%** accuracy using Random Forest. Notably, Rezaei and Hamze claimed their method outperformed Kumar et al.'s approach (95.5% vs. 94.9% as reported in their comparative analysis), achieving higher accuracy with significantly fewer features (9 compared to Kumar et al.'s 68). This finding underscored the efficiency and effectiveness of their selected feature set for rapid malware detection.

Maniriho *et al.* [5] proposed EarlyMalDetect, a novel approach for early Windows malware detection based on sequences of API calls. Their method demonstrates the effectiveness of API sequence analysis for malware identification. Similarly, Kunwar *et al.* [2] provided a systematic overview of leveraging Transformers for malware analysis, highlighting the potential of attention-based mechanisms in capturing long-range dependencies in malicious patterns.

Alshomrani *et al.* [1] conducted a comprehensive survey of Transformer-based malicious software detection systems, analysing 13 different architectures across various malware types. Their study revealed that Transformer models achieve superior performance in capturing sequential patterns and contextual relationships in malware samples, particularly when processing API call sequences and bytecode patterns.

Miao *et al.* [4] introduced a lightweight malware detection model based on knowledge distillation, achieving a balance between computational efficiency and detection accuracy. Their approach demonstrated how model compression techniques can maintain high performance while reducing resource requirements, making deep learning models more suitable for resource-constrained environments.

Several studies have explored novel architectural approaches. Singh *et al.* [10] implemented a feedforward deep neural network (FFDNN) for static malware detection, focusing on feature optimisation and network depth. Damaševičius *et al.* [11] proposed an ensemble-based classification approach using neural networks and machine learning models specifically for Windows PE malware detection, demonstrating the benefits of combining multiple models.

Recent work by Gibert *et al.* [12] provides a comprehensive analysis of machine learning approaches for malware detection, examining research developments, trends, and challenges from 2010 to 2020. Their survey identifies key limitations in existing approaches, including dependency on feature engineering, vulnerability to adversarial attacks, and limited generalisation to new malware families.

In the context of deep learning architectures, Redhu *et al.* [13] evaluated eight popular deep learning approaches across various malware datasets, highlighting the need for standardised benchmarks and the challenges of model interpretability. They emphasised the importance of Explainable AI (XAI) and Interpretable Machine Learning (IML) in malware detection systems.

Wei *et al.* [14] addressed the critical issue of malware evolution's impact on API sequence-based Windows malware detectors. Their work demonstrated techniques for mitigating performance degradation caused by evolving malware patterns, providing valuable insights for maintaining detection effectiveness over time.

Almazroi and Ayub [15] introduced a BERT-based feedforward neural network framework (BEFNet) for IoT malware detection, achieving 97.99% accuracy. Their approach demonstrates the effectiveness of transformer-based embeddings in capturing malware characteristics across different platforms.

Contreras *et al.* [16] evaluated memory-optimised machine learning solutions for static malware analysis, comparing artificial neural networks, support vector machines, and gradient-boosting machines. Their study provides insights into the effectiveness of lightweight models under extreme memory constraints, achieving 93.44% accuracy with optimised neural networks.

Fellicious *et al.* [17] focused specifically on API call-based malware detection, providing both theoretical foundations and practical implementation considerations. Their work includes comprehensive datasets and evaluation methodologies that advance the state-of-the-art in API-based malware analysis.

Kumar *et al.* [18] conducted a systematic literature review of machine learning algorithms for malware detection, providing a taxonomy that considers performance accuracy, analysis types, and detection approaches. Their work highlights ongoing challenges in detecting unexpected malware attacks and the need for adaptive detection mechanisms.

Recent industrial perspectives, such as those from

Tsao [19], demonstrate real-world applications of machine learning in cybersecurity solutions. The TrendX Hybrid Model combines static and behavioural features using two-phase training, achieving faster and more accurate malware detection while reducing false positives in production environments.

Although these studies represent significant advances in static malware detection, several limitations persist. Many approaches depend on preprocessed or clean datasets, remain vulnerable to adversarial attacks, and rely on narrowly scoped features that may not generalise to evolving threats. As malware authors increasingly adopt sophisticated obfuscation, packing, and mimicry techniques, there is a critical need for more robust and adaptive detection frameworks.

Our work aims to address these gaps by proposing a static malware detection framework that combines comprehensive feature extraction from raw PE files with an optimised deep neural network. Unlike previous methods [2, 3], we emphasise full automation, minimal reliance on pre-existing metadata, and robust training using balanced, raw datasets. This positions our approach as a scalable and realistic solution for modern malware detection challenges.

3 Methodology

In this section, we present the proposed static malware detection approach in a detailed and structured manner. The methodology comprises several key phases that together form a complete and automated detection pipeline. First, raw Windows PE files are parsed to extract a set of handcrafted features using a custom-designed feature extraction tool built on top of the `pefile` Python library [3]. These features capture structural, semantic, and statistical properties of each binary. Next, the extracted data undergoes preprocessing, including normalisation and transformation, to prepare it for training. A deep neural network (DNN) is then constructed and trained on the processed feature vectors. To ensure optimal performance and generalisation, we use Bayesian optimisation to fine-tune the model's architecture and hyperparameters automatically.

3.1 Feature Extraction and Preprocessing

To build a robust and fully static malware classifier, we designed an end-to-end feature extraction pipeline that operates directly on raw Windows Portable Executable (PE) files. Using the `pefile` Python library [3], we extracted low-level structural and metadata features without relying on external logs, execution traces, or commercial toolkits. This approach enables the pipeline to be both scalable and deployment-ready in restricted environments.

3.1.0.1 Conventional feature groups.

Based on domain knowledge and prior work [3, 4], features were organised into several groups:

- **Header Information:** number of sections, machine type, file timestamp, checksum flag, and binary-level flags such as `ExecutableImage`, `DLL`, and `RelocsStripped`.
- **Section Characteristics:** section name patterns, entropy, raw and virtual sizes, and the presence of suspicious or unnamed sections.
- **Import/Export Tables:** number of dynamically linked libraries (DLLs), imported API functions, and exported symbols.
- **Resources:** entropy and size statistics of embedded resources (e.g., icons, strings).
- **Version Information:** number of metadata entries found in version and signature blocks.

Each executable sample yields approximately 77 structured features, including both categorical and numerical types. Table 1 shows an example subset extracted from a single PE file.

Table 1. Sample extracted features from a PE file

Feature	Value
NumberOfSections	5
FileEntropy	6.37
ImportsNbDLL	11
ResourcesNb	3
Characteristic_ExecutableImage	1
SuspiciousSections	1
SectionsMeanEntropy	4.26
SectionsMaxEntropy	7.91

3.1.0.2 Novel and extended features.

In addition to conventional PE attributes, we designed several new or extended descriptors that enrich the representation space and address gaps identified in prior studies. Subtle structural irregularities and behavioural indicators that are often exploited by malware authors are captured by these features:

- **DataDirectoryBits.** While most studies analyse data directories individually, we encode their presence as a compact binary representation. Every PE file contains a data directory table pointing to optional but semantically important structures. We jointly analyse six key directories: `DEBUG`, `EXPORT`, `LOAD_CONFIG`, `RESOURCE`, `BASERELOC`, and `TLS`. Each entry is encoded as a binary indicator, and the com-

bination is compactly represented as a decimal bitmask. For instance, a binary pattern of 011101 (absence of `DEBUG`, presence of `EXPORT`, `LOAD_CONFIG`, `RESOURCE`, absence of `BASERELOC`, and presence of `TLS`) corresponds to the value 29. This method compresses multiple security-critical cues into a single numerical feature while preserving interpretability.

- **Enhanced entropy calculation.** While previous implementations used either external tools or internal calculations independently, we combine both approaches to achieve greater accuracy. We integrate three entropy values: the Shannon entropy of the full file (measured with the `ent` utility for consistency with system-level forensic tools), and the entropy of the `.text` and `.data` sections computed via `pefile`. This dual-method approach ensures both precision and compatibility with different analysis workflows, providing a global view of randomness and a localised perspective on critical code and data regions.
- **Comprehensive characteristics flags.** The `FILE_HEADER.Characteristics` field encodes 16 bitwise flags describing the binary (e.g., `ExecutableImage`, `DLL`, `RelocsStripped`, `DebugStripped`, etc.). Previous work often collapses these into a single binary variable (e.g., "is DLL") or ignores them entirely. We instead expand all 16 flags into independent binary features, enabling the model to capture fine-grained usage patterns. For example, benign system DLLs typically set `DLL` and `LargeAddressAware`, whereas malware often strips debugging information (`DebugStripped`=1) or turns off relocations (`RelocsStripped`=1) to complicate analysis.
- **Header-to-code size ratio.** We propose a new feature defined as the ratio of header size to code size:

$$\text{HeaderToCodeSizeRatio} = \frac{\text{SizeOfHeaders}}{\max(1,\ \text{SizeOfCode})}.$$

  Packed or malformed binaries often exhibit disproportionately large headers relative to the code section, whereas legitimate software tends to follow more consistent ratios. This measure is lightweight to compute yet strongly indicative of tampering or obfuscation techniques commonly employed by malware authors.
- **Enriched resource descriptors.** Resources (icons, strings, dialogs) are often underutilised in static analyses. Beyond counting the number of resources, we compute statistical summaries of both their size and entropy, including mean, minimum, and maximum values. Large or unusually high-entropy resources may correspond to embedded payloads or encrypted blobs. This richer characterisation allows the model to detect anomalies that would be invisible if only the count of resources were considered, providing deeper insight into potential data hiding techniques.

Together, these features extend the discriminative scope beyond conventional attributes such as imports and section counts. As shown later in our ablation study (Section 4.2), their removal results in measurable performance degradation, demonstrating their complementary contribution to the final model.

3.1.0.3 Cleaning and preprocessing.

To ensure reliability, corrupted or incomplete PE files were removed. Missing values were handled by mean imputation or domain-informed defaults (e.g. zero for missing imports or entropies). All numerical features were normalised using `StandardScaler` [17], while skewed features (e.g. file size, imports) were log-transformed. Binary categorical attributes (e.g. `Characteristic_DLL`) were kept as integers; one-hot encoding was not required since all features fit into a fixed-length dense vector. Labels were encoded as 0 for benign and 1 for malware.

In summary, the pipeline produces a clean, normalised, and feature-rich representation of PE files that captures both well-established and novel discriminative signals, enabling deployment in deep neural network classifiers.

3.2 Fully Connected DNN for PE File Classification

We implemented a fully connected feedforward neural network using TensorFlow and Keras, designed specifically to classify PE files based on extracted static features. The input layer dimension matches the feature vector size (77 handcrafted features). The model includes a variable number of hidden layers, each followed by Batch Normalisation and Dropout to enhance generalisation and stabilise training.

Each hidden layer uses activation functions selected from ReLU, SELU, and GELU [4], based on hyperparameter tuning with the Keras Tuner framework. We used the sigmoid function in the output layer to return the probability of a file being malicious, and binary cross-entropy as the loss function.

The model is optimised using adaptive optimisers such as Adam, Nadam, or RMSprop [4], with learning rates also selected via tuning. Additionally, we applied L2 regularisation to mitigate overfitting, alongside

Table 2. Final tuned architecture (from input to output).

Layer	Units	Activation	BatchNorm	Dropout	L2
Hidden 1	384	ReLU	No	0.1	1×10^{-5}
Hidden 2	640	ReLU	Yes	0.3	1×10^{-5}
Hidden 3	768	GELU	Yes	0.3	1.29×10^{-4}
Hidden 4	640	SELU	No	0.4	5.38×10^{-5}
Hidden 5	384	SELU	Yes	0.4	1.80×10^{-4}
Hidden 6	384	SELU	Yes	0.3	2.28×10^{-4}
Output	1	Sigmoid	–	–	–

early stopping and learning rate scheduling during training.

The architecture is not fixed; instead, we leveraged Bayesian optimisation with Keras Tuner to search over hyperparameters such as the number of layers (3–6), the number of neurons per layer (128–1024), the type of activation functions, dropout rates, and the L2 penalty. This approach allowed us to identify the best-performing configuration based on validation accuracy. The optimal architecture comprised **6** hidden layers, detailed in Table 2, followed by a sigmoid output layer (1 unit). The tuned optimiser was Adam with an initial learning rate of 1×10^{-4}.

3.3 Bayesian Hyperparameter Optimisation

We employed Bayesian optimisation via Keras Tuner to systematically search the hyperparameter space and identify the model configuration with the best validation accuracy. The use of Bayesian optimisation was motivated by the need to efficiently explore a large hyperparameter space while minimising computational cost compared to grid or random search methods. Bayesian methods construct a probabilistic model of the objective function and use it to choose promising hyperparameter values based on past evaluations [20].

Our search space included the following dimensions:

- Number of dense layers: **3 to 6**
- Units per layer: **128 to 1024**
- Dropout rate: **0.1 to 0.5**
- Learning rate: **log-uniform range from** 10^{-5} **to** 10^{-2}
- Activation function: **ReLU, SELU, or GELU**

Each trial involved building and training a unique model configuration for up to 50 epochs with early stopping based on validation loss. Over 100 trials were executed, and the optimal model achieved a validation accuracy of **98.83%**. The best hyperparameters corresponded to the **six-hidden-layer** architecture reported in Table 2. The final test performance was: Accuracy = **0.9883**, Precision = **0.9907**, Recall = **0.9888**, and F1-score = **0.9898**.

3.4 Training Setup

The final dataset consisted of a balanced set of benign and malicious PE files from the PE Malware Machine Learning Dataset [21], split into training (80%) and test (20%) subsets using stratified sampling to preserve class distribution. Training was conducted on Google Colab with GPU acceleration (NVIDIA Tesla T4) to reduce runtime and support large-batch processing.

The model was trained with a batch size of 512 and a maximum of 100 epochs. Early stopping with a patience of 15 epochs and learning rate reduction on plateau were applied to prevent overfitting. To ensure reproducibility, random seeds were fixed for NumPy, TensorFlow, and Python's random module.

4 Experiments and Results

In this section, we present the experimental design, dataset details, model training configuration, evaluation metrics, and quantitative results that demonstrate the effectiveness of the proposed static malware detection system.

4.1 Dataset Description

We use the *PE Malware Machine Learning Dataset* by Lester [21], which provides raw Windows Portable Executable (PE) files together with metadata and labels. The corpus contains **201,549** PE samples, of which **114,737** are labelled as malicious (blacklist) and **86,812** as legitimate (whitelist).

Table 3. Summary statistics of the dataset (Lester [21]).

Total PE files	201,549
Malicious (blacklist)	114,737
Legitimate (whitelist)	86,812
Compressed size	43.8 GB
Uncompressed size	117 GB
File types	PE executables (`.exe`, `.dll`, `.scr`)

Malicious samples are primarily aggregated from public repositories such as VirusShare, MalShare, and TheZoo. At the same time, legitimate files are mostly sourced from Windows installations and common user applications, as documented by the dataset author. All files were handled in a sandboxed environment following the dataset's terms of use. Sample integrity was verified via cryptographic hashes and metadata consistency checks, and files that could not be parsed were excluded.

For each PE file, we extracted a **77-dimensional** feature vector directly from the binary, covering header fields, section layout statistics, entropy-based attributes, import/export table signals, and resource-related features (see Section 3). Exploratory analysis highlighted that attributes such as file entropy, number of imports, and section sizes provide strong discriminative power between benign and malicious files.

4.2 Ablation Study on Feature Groups

To quantify the contribution of both conventional and novel features, we conducted a single-group ablation study. In accordance with this setup, the classifier is trained and evaluated using only one feature group at a time, and the results are compared to the full feature set. This comparison allows us to assess the standalone predictive power of each group and verify the added value of our proposed descriptors.

Table 4. Single-group ablation results on the test set.

Feature group	Accuracy	F1-score
All Features	0.9786	0.9813
Entropy (File/Text/Data)	0.9233	0.9322
Imports/Exports	0.8006	0.8286
Sections (Sizes/Counts/Names)	0.9040	0.9145
Resources Features	0.8184	0.8479
Characteristics Flags	0.8750	0.8961
DataDirectoryBits	0.7815	0.8190
Header-to-Code Size Ratio	0.6021	0.7392

The results show that entropy-based features remain the most informative conventional group, followed by section-level statistics. However, our novel features also exhibit measurable standalone discriminative power: *DataDirectoryBits* alone achieves an F1-score of 0.8190, while *Characteristics Flags* reach 0.8961. The *Header-to-Code Size Ratio*, though individually weaker, captures a highly specific anomaly signal that complements other groups in the full model.

Overall, removing these novel features leads to a noticeable drop in performance, confirming that they are not redundant with existing attributes. This result validates the design choices introduced in Section 3.1 and highlights the benefit of combining structural, statistical, and semantic descriptors for malware classification.

4.3 Runtime and Efficiency

We measured inference latency of the final DNN on Google Colab Pro (NVIDIA Tesla T4, batch inference). The average per-sample inference time was **2.16 ms**, yielding a throughput of about **462 samples/s**. [1]

These results indicate that the classifier itself meets real-time constraints typical of endpoint scanning and network-gateway prefiltering. In practice, the end-to-end throughput is dominated by feature extraction; we therefore expose a streaming extractor and batched inference API to overlap I/O and compute.

4.4 Experimental Setup

The dataset was shuffled and divided using an 80/20 stratified split to maintain equal class proportions across training and testing subsets. Data loaders were implemented to efficiently batch input features for the neural network. The model was trained for a maximum of 100 epochs using a batch size of 512 and monitored using validation accuracy and loss.

Early stopping with a patience of 15 epochs and learning rate reduction on plateau were applied to prevent overfitting. Training was conducted on Google Colab Pro with GPU acceleration (NVIDIA Tesla T4), enabling efficient large-batch processing and fast experimentation.

To ensure consistent results across different executions and environments, we initialised all major random number generators with fixed seed values. Specifically, random seeds were set for NumPy, TensorFlow, and Python's built-in random module. This practice guarantees the reproducibility of model training, hyperparameter optimisation, and evaluation outcomes.

4.5 Evaluation Metrics

To comprehensively assess performance, we employed multiple classification metrics:

- **Accuracy:** Proportion of correctly predicted samples out of all samples.
- **Precision:** Ratio of true positives to total predicted positives.
- **Recall:** Ratio of true positives to actual positives.
- **F1-score:** Harmonic mean of precision and recall.
- **ROC-AUC:** Area under the receiver operating characteristic curve.
- **Confusion Matrix:** Quantitative breakdown of true vs. predicted classifications.

[1] Reported numbers are for model inference only; static feature extraction is linear-time and can be parallelised. Our pipeline is amenable to batching, quantisation, and TensorRT/ONNX deployment to further reduce latency.

4.6 Quantitative Results

The proposed DNN model achieved superior performance across all metrics on the test set. Table 5 summarises the evaluation metrics.

Table 5. Performance metrics on the test set

Metric	Value
Accuracy	98.83%
Precision	99.07%
Recall	98.88%
F1-score	98.98%
ROC-AUC	0.995

The confusion matrix confirmed minimal false positives and false negatives, as shown in Figure 1. This result demonstrates high reliability, which is essential for security-critical applications such as antivirus engines and malware screening gateways.

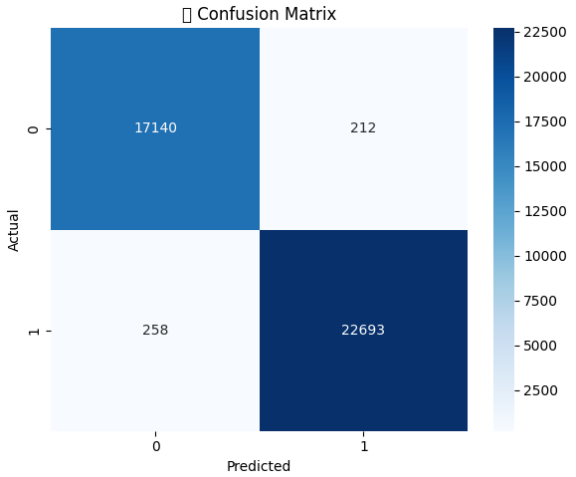

Figure 1. Confusion matrix of the proposed model on the test set.

4.7 Visual Analysis

The ROC curve in Figure 2 illustrates the strong discriminative capacity of our classifier. The curve rises sharply towards the top-left corner, indicating a high true positive rate even at low false positive thresholds.

Feature importance analysis was conducted using permutation testing and ablation studies. When individual features were removed, entropy-related features, number of imports, and section size data were found to cause the largest drop in model performance, confirming their critical role in classification. In addition, Shapley (SHapley Additive exPlanations) values were plotted to provide a more detailed interpretation of feature contributions. Figure 3 shows the mean absolute SHAP values (global importance of each feature), while Figure 4 illustrates the distribution and direction of feature impacts across individual test samples.

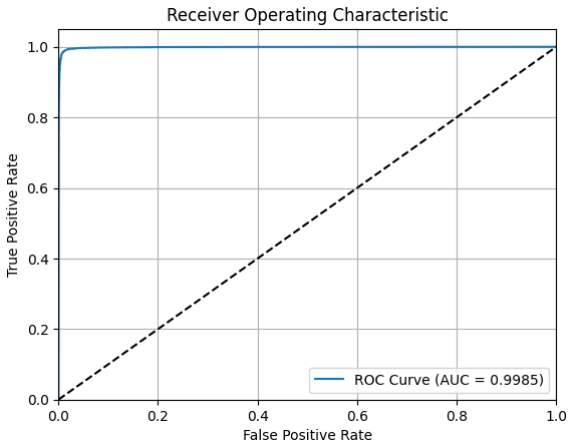

Figure 2. ROC curve of the final DNN classifier.

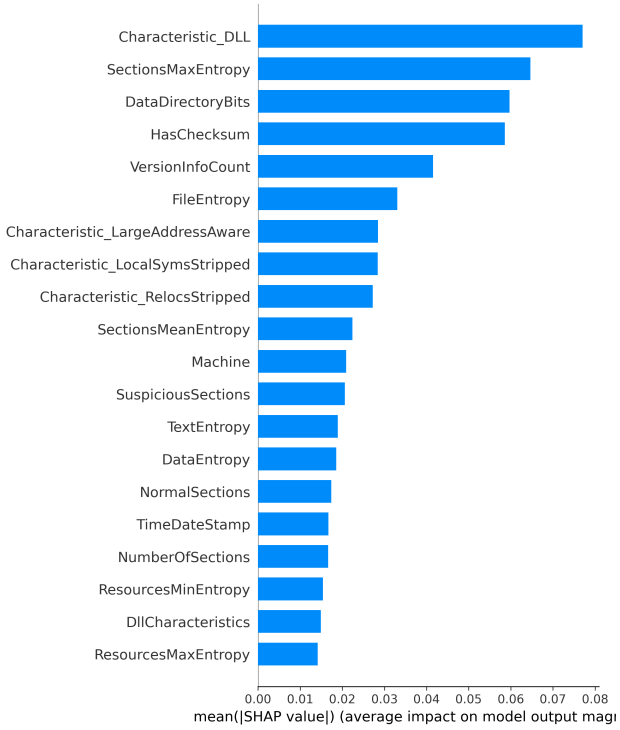

Figure 3. Global feature importance based on mean absolute SHAP values.

These results highlight the potential of integrating domain-specific binary analysis with modern deep learning to deliver scalable, accurate, and fully static malware detection tools that can operate in real-world security environments without dynamic execution or sandboxing.

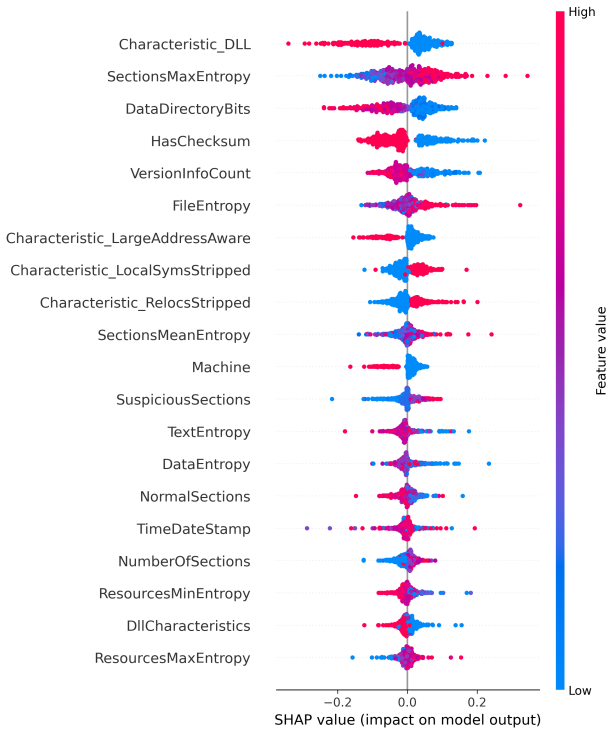

Figure 4. Beeswarm plot of SHAP values showing feature impact distribution and direction.

4.8 Comparative Analysis with Prior Work

To better position our work within the existing literature, we compare it against several representative studies on static malware detection using Portable Executable (PE) features. Table 6 summarises key aspects including the number of features, dataset size, algorithms, and reported performance.

Kumar *et al.* [8] proposed a machine learning model with an *integrated feature set* combining raw PE header fields and derived attributes such as entropy, compilation year, packer information, and suspicious section counts. Using Random Forest, they achieved 98.4% accuracy under 10-fold cross-validation on a dataset of 5,210 samples, but accuracy dropped to 89.2% on a novel test set, highlighting limited generalisability.

Rezaei and Hamze [9] designed a lightweight approach focusing on only nine header-centric features, such as empty section names, a decimal representation of data directories, and entropy values. Their Random Forest classifier achieved 95.6% accuracy with minimal computational cost, demonstrating efficiency for near real-time scenarios, but at the expense of broader generalisation.

Contreras *et al.* [16] explored low-parameter learners for static analysis of program metadata, motivated by IoT and constrained environments. Using ANN, SVM, and Gradient Boosting on 138,047 samples, they achieved up to 94.7% accuracy while optimising for memory efficiency. This trade-off favours deployment on resource-limited devices, though accuracy remains lower compared to DNN-based methods.

By contrast, our method leverages a deep neural network with Bayesian hyperparameter optimisation, trained on the large-scale Lester PE dataset (201,549 samples). Through a richer feature set (∼77 descriptors), including novel contributions such as `DataDirectoryBits`, enhanced entropy calculations, and enriched resource descriptors, our system achieves 98.83% accuracy and 98.98% F1-score. As confirmed by ablation studies (Section 4.2), the removal of these novel features results in measurable performance degradation, underlining their importance.

Table 6. Compact comparison with representative PE–based static detectors.

Study	#Feats	Dataset	Model(s)	Acc.
This work	∼77	Lester PE (201k)	DNN (Bayes-Opt)	98.83%
Kumar et al. [8]	68	5,210 (+132 test)	RF, DT, kNN, LR, LDA, NB	98.4% (CV); 89.2% (new)
Rezaei & Hamze [9]	9	2,460	RF, SVM, kNN	95.6% (CV)
Contreras et al. [16]	57	138k (metadata)	ANN, SVM, GBM	94.7% (max)

Overall, prior approaches trade off between accuracy, feature set size, and computational efficiency. Our contribution lies in combining a large-scale dataset with carefully engineered and novel binary features, integrated into an optimised DNN framework. This result enables both high accuracy and robustness, while maintaining scalability for real-world deployment.

5 Discussion

The proposed DNN model demonstrates significant performance improvements over traditional classifiers. Although some traditional models, such as Random Forest, achieved high F1-scores (e.g. 99.2%), their simplicity involves trade-offs in flexibility, interpretability, and scalability compared to deep learning approaches.

A primary advantage of the DNN is its capacity to capture complex interactions among features, as confirmed by feature importance analysis and ablation studies. Entropy-related attributes, import counts, and suspicious section flags were among the most critical contributors to classification accuracy.

Furthermore, the static nature of the system, which does not rely on dynamic execution or commercial preprocessing, makes it suitable for large-scale deployment in environments where speed and resource efficiency are critical. However, there are some limitations to consider: the handcrafted feature set, although effective, may fail to capture deep semantic patterns present in packed or obfuscated malware.

Future work could explore integrating byte-level n-grams, raw binary embeddings, or Transformer-based models to further improve generalisation. The model could also be evaluated against adversarial attacks or real-time zero-day samples to assess its resilience. Lightweight deployment on embedded systems such as IoT devices is another promising direction.

6 Conclusion

This paper proposed a fully automated framework for static malware detection using handcrafted binary features and a deep neural network. By extracting structural indicators directly from PE files—such as entropy, import counts, and section metadata—the model achieved high accuracy (98.83%) and an F1-score (98.98%) on a balanced dataset. The approach eliminates the need for dynamic analysis or third-party tools, making it practical for real-world deployment.

The model's simplicity, combined with effective Bayesian tuning and lightweight inference, supports its integration into security solutions such as antivirus and intrusion detection systems. Our results demonstrate that static analysis, when combined with deep learning and well-engineered features, can provide robust malware detection with high generalisation capacity.

References

[1] Mohammed Alshomrani, Aiiad Albeshri, Badraddin Alturki, Fouad Alallah, and Abdulaziz Alsulami. Survey of transformer-based malicious software detection systems. *Electronics*, 13(23):4677, 2024. .

[2] Pradip Kunwar, Kshitiz Aryal, Maanak Gupta, Mahmoud Abdelsalam, and Elisa Bertino. Sok: Leveraging transformers for malware analysis. *IEEE Transactions on Dependable and Secure Computing*, 2024. .

[3] Malik Ijaz Yousuf, Iftikhar Anwer, Muhammad Riasat, K.T. Zia, and Sung Kim. Windows malware detection based on static analysis with multiple features. *PeerJ Computer Science*, 9:e1319, 2023. .

[4] Chunyu Miao, Liang Kou, Jilin Zhang, and Guozhong Dong. A lightweight malware detection model based on knowledge distillation. *Mathematics*, 12(24):4009, 2024. .

[5] Pascal Maniriho, Abdun Naser Mahmood, and Mohammad Jabed Morshed Chowdhury. Earlymaldetect: A novel approach for early windows malware detection based on sequences of api calls. *arXiv preprint*, 2024.

[6] Ananya Redhu, Prince Choudhary, Kathiravan Srinivasan, and Tapan Kumar Das. Deep learning-powered malware detection in cyberspace: a contemporary review. *Frontiers in Physics*, 12:1349463, 2024. .

[7] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020. .

[8] Ajit Kumar, K.S. Kuppusamy, and G. Aghila. A learning model to detect maliciousness of portable executable using integrated feature set. *Journal of King Saud University - Computer and Information Sciences*, 2017. . URL http://dx.doi.org/10.1016/j.jksuci.2017.01.003.

[9] Tina Rezaei and Ali Hamze. An efficient approach for malware detection using pe header specifications. page 234, 2020.

[10] Prabhjot Singh et al. Feed-forward deep neural network (ffdnn)-based deep features for static malware detection. *International Journal of Intelligent Systems*, pages 1–20, 2023. .

[11] Robertas Damaševičius, Algirdas Venčkauskas, Jevgenijus Toldinas, and Šarūnas Grigaliūnas. Ensemble-based classification using neural networks and machine learning models for windows pe malware detection. *Electronics*, 10(4):485, 2021. .

[12] Daniel Gibert, Carles Mateu, and Jordi Planes. A survey of malware detection using deep learning. *Machine Learning and Cybersecurity*, 5:100227, 2024. .

[13] Ananya Redhu et al. A comprehensive survey on deep learning approaches for malware detection: Taxonomy, current challenges, and future directions. *IEEE Access*, 12:45123–45145, 2024. .

[14] Xingyuan Wei, Ce Li, Qiujian Lv, Ning Li, Degang Sun, and Yan Wang. Mitigating the impact of malware evolution on api sequence-based windows malware detector. *arXiv preprint*, 2024.

[15] Abdulwahab Ali Almazroi and Naveed Ayub. Deep learning hybridization for improved malware detection in smart internet of things. *Scientific Reports*, 14:7838, 2024. .

[16] Carlos Contreras, Robert Baker, Arturo Gutiérrez, and Jose Cerda. Static malware analysis using low-parameter machine learning models. *Information*, 13(3):59, 2024. .

[17] Christofer Fellicious, Manuel Bischof, Kevin Mayer, Dorian Eikenberg, Stefan Hausotte, Hans P. Reiser, and Michael Granitzer. Malware detection based on api calls. *arXiv preprint*, 2025.

[18] Raj Kumar et al. Machine learning algorithm for malware detection: Taxonomy, current challenges, and future directions. *IEEE Access*, 11: 23456–23478, 2024. .

[19] Spark Tsao. Faster and more accurate malware detection through predictive machine learning: Correlating static and behavioral features. *Trend Micro Security Research*, 2019. URL https://www.trendmicro.com/vinfo/us/security/news/security-technology/faster-and-more-accurate-malware-detection-through-predictive-machine-learning-correlating-static-and-behavioral-features. Available at: https://www.trendmicro.com/vinfo/us/security/news/security-technology/.

[20] Hyunghun Cho, Yongjin Kim, Eunjung Lee, Daeyoung Choi, Yongjae Lee, and Wonjong Rhee. Basic enhancement strategies when using bayesian optimization for hyperparameter tuning of deep neural networks. *IEEE Access*, 8: 52588–52608, 2020. . URL https://doi.org/10.1109/ACCESS.2020.2981072.

[21] Michael Lester. Pe malware machine learning dataset. Online dataset, 2021. URL https://practicalsecurityanalytics.com/pe-malware-machine-learning-dataset/. Available at: https://practicalsecurityanalytics.com/pe-malware-machine-learning-dataset/.

Sajjad Rezaei is a PhD candidate in Computer Engineering at Isfahan University of Technology (IUT). His research focuses on the intersection of deep learning and system security, with interests in cybersecurity, static and dynamic malware analysis, and digital forensics. He is currently developing resilient frameworks for malware detection using deep neural networks and optimized binary feature extraction to address evolving malware challenges.

Ali Fanian is an assistant professor in the Department of Electrical and Computer Engineering at Isfahan University of Technology (IUT), Isfahan, Iran. He has extensive research experience in the fields of computer networks and information security. His research interests encompass hardware-accelerated security, the design of high-performance intrusion detection systems, and secure communication protocols. Dr. Fanian's work specifically involves the efficient implementation of cryptographic algorithms and the development of adaptive security mechanisms for modern network infrastructures.