Advanced Android Malware Detection: Merging Deep Learning and XGBoost Techniques

Araştırma Makalesi/Research Article

Esra KAVALCI YILMAZ [1]*, Rezan BAKIR [1]

[1]Department of Computer Engineering, Sivas University of Science and Technology, Sivas, Türkiye.

esra.kavalci@sivas.edu.tr, rezan.bakir@sivas.edu.tr

(Geliş/Received:20.09.2024; Kabul/Accepted:15.11.2024)
DOI: 10.17671/gazibtd.1553548

Abstract**—** The increasing importance of Android devices in our lives brings with it the need to secure personal information stored on these devices, such as contact details, documents, location data, and browser data. These devices are often targeted by attacks and malware designed to steal this data. In response, this work takes a novel approach to Android malware detection by integrating deep learning with traditional machine learning algorithms. An extensive experimental study was conducted using the DroidCollector network traffic analysis dataset. Eight different deep learning methods are analysed for malware classification. In the first phase, experiments were conducted on both original and stabilised datasets and the most effective methods were identified. In the second phase, the best performing deep learning methods were combined with XGBoost for classification. This hybrid approach increased classification success by 3-4%. The highest F1 and accuracy values obtained after 150 epochs of training with BiLSTM+XGBoost were 95.12% and 99.33% respectively. These results highlight the superiority of combining deep learning and traditional machine learning techniques over individual models and significantly improve classification accuracy. This integrated method provides a very important strategy for developing high-performance models for various applications.

Keywords**—** malware detection, machine learning, deep learning, XGBoost

Gelişmiş Android Kötü Amaçlı Yazılım Tespiti: Derin Öğrenme ve XGBoost Tekniklerinin Birleştirilmesi

Özet**—** Android cihazların hayatımızdaki artan önemi, bu cihazlarda depolanan kişisel bilgileri (iletişim bilgileri, belgeler, konum verileri ve tarayıcı verileri gibi) güvence altına alma ihtiyacını beraberinde getirir. Bu cihazlar genellikle bu verileri çalmak için tasarlanmış saldırılar ve kötü amaçlı yazılımların hedefi olur. Bu duruma önlem olarak, bu çalışma derin öğrenmeyi geleneksel makine öğrenimi algoritmalarıyla entegre ederek Android kötü amaçlı yazılım tespitine yeni bir yaklaşım sunmaktadır. DroidCollector ağ trafiği analizi veri kümesi kullanılarak kapsamlı bir deneysel çalışma yürütülmüştür. Kötü amaçlı yazılım sınıflandırması için sekiz farklı derin öğrenme yöntemi analiz edilmiştir. İlk aşamada, hem orijinal hem de önişlemden geçirilmiş (SMOTE, SMOTETomek, ClusterCentroids) veri kümeleri üzerinde deneyler yürütülmüş ve en etkili yöntemler belirlenmiştir. İkinci aşamada, en iyi performans gösteren derin öğrenme yöntemleri sınıflandırma için XGBoost ile birleştirilmiştir. Bu hibrit yaklaşım, sınıflandırma başarısını %3-4 oranında artırmıştır. BiLSTM + XGBoost modelinin 150 epoch ile eğitilmesiyle elde edilen en yüksek F1-score ve doğruluk değerleri sırasıyla %95,12 ve %99,33 olmuştur. Bu sonuçlar, derin öğrenme ve geleneksel makine öğrenimi tekniklerinin bireysel modellere göre birleştirilmesinin üstünlüğünü vurgular ve sınıflandırma doğruluğunu önemli ölçüde iyileştirir. Bu hibrit yöntem, çeşitli uygulamalar için yüksek performanslı modeller geliştirmek amacıyla önemli bir strateji sunmaktadır.

Anahtar Kelimeler**—** kötü amaçlı yazılım tespiti, makine öğrenmesi, derin öğrenme, XGBoost

1. INTRODUCTION

In today's digital landscape, mobile devices, especially smartphones and tablets based on the Android operating system, have become seamlessly integrated into our daily lives. They serve as versatile tools for communication, entertainment, business, and managing daily activities. However, this ubiquitous reliance on mobile devices also presents a significant challenge: the ever-present risk of security vulnerabilities and malicious software threats [1]. Indeed, the central role of Android devices in our daily lives is underlined by the vast trove of data they hold. From cherished personal photos to vital contact details, from sensitive financial records to indispensable calendar entries, these devices store a wealth of intimate and confidential information. Through sophisticated application and service integration, Android devices have become central tools for organizing and customizing users' lives. However, this wealth of data also presents significant security risks. The sensitive information stored on Android devices is a prime target for cybercriminals and malware. Unauthorized access to this data can have dire consequences, including financial loss, identity theft, and other serious disruptions. As a result, ensuring the security of Android devices is not only important but essential for user protection and peace of mind [2].

The Android ecosystem offers a wide range of applications and customization possibilities, but it also presents significant security vulnerabilities that malicious software can exploit to infiltrate devices. These vulnerabilities can pose serious threats to individual users, as well as companies and organizations [3]. To detect such threats, various analysis methods are employed in malware detection, which can be broadly classified as static, dynamic, and hybrid analysis approaches. Static analysis involves examining the file structure or code patterns of malware without executing it, making it fast and efficient but often ineffective against malware with advanced encryption and compression techniques. On the other hand, dynamic analysis examines the behavior of malware by running it in a controlled environment, which is more effective for detecting advanced threats but requires more resources. Hybrid analysis aims to overcome the limitations of both static and dynamic methods by combining the strengths of each, providing a more comprehensive and powerful solution for detecting complex malware. This hybrid approach ensures a more accurate assessment of potential threats, which is crucial in securing the Android ecosystem from malicious attacks.

The use of advanced technologies such as deep learning offers a promising solution to detect malware on Android devices. This paper scrutinizes novel methodologies aimed at enhancing the security of Android devices, with a particular emphasis on research integrating deep learning and machine learning techniques. To the best of our knowledge, the proposed approach represents an unprecedented endeavor within the literature, promising innovative strides toward fortifying Android device security.

1.1.Motivation

The open source architecture of Android devices makes it easier for attackers to analyze and target these devices. Moreover, the diversity of devices in the Android ecosystem and the irregularity in update processes make it difficult to patch vulnerabilities quickly. For these reasons, effective and rapid detection of malware on Android devices has become a critical requirement for user security. In the face of increasingly sophisticated malware attacks, traditional security methods are insufficient. In the literature, machine learning and deep learning methods have been successfully applied for Android malware detection, but the hybrid combination of these two techniques and the integrated utilization of their advantages is very limited. This study aims to investigate how the hybrid use of machine learning and deep learning methods can improve classification success by examining the effects of unbalanced data distribution on malware detection on Android devices.

1.2.Novelties and Contributions

In this work, the impact of balanced data distribution and the hybrid use of deep learning and machine learning (XGBoost) algorithms is studied to enhance accurate classification performance within the critical security domain of malware detection on Android systems. Android devices, due to their large user base and open-source nature, have become prime targets for malware attacks, making the accurate detection of these threats essential. Although various machine learning and deep learning techniques have been proposed in the literature for Android malware classification, the combined strengths of both methods remain underexplored. Deep learning models demonstrate superior performance in extracting discriminative features from high-dimensional and complex data structures, while the XGBoost algorithm achieves high accuracy in tree-based ensemble methods (boosting) due to its advanced optimization, regularization, and parallel processing capabilities. Through this hybrid approach, the study seeks to combine the adaptability of deep learning with the robust classification ability of XGBoost to reveal its impact on classification performance.The contributions of the study are listed below:

1-Elimination of Imbalanced Data Distribution: In malware classification processes, the number of data mislabeled as harmless (False Positive) is critical, as this can allow malware to infiltrate the system. In order to avoid such security risks, the data set should have a balanced distribution. In this study, to investigate the impact of balanced data distribution on classification performance, "imbalanced data sampling" methods are applied, enabling a comparative analysis of performance differences between balanced and imbalanced data distributions.

2-Hybrid Use of Deep Learning and Machine Learning Methods: The hybrid use of deep learning and machine learning (XGBoost) methods is crucial for complex problems such as malware classification. While deep learning models exhibit strong performance in learning complex patterns and extracting discriminative features in high-dimensional data, the XGBoost algorithm provides more accurate classification results thanks to its regularization, error rate optimization and parallel processing capabilities. In this study, the potential for improving classification performance is analyzed by combining the strengths of both methods, demonstrating the hybrid approach's ability to deliver both flexibility and accuracy.

2. RELATED WORKS

Research in the field of Android malware detection has witnessed notable progress, as scholars have delved into a range of machine learning and deep learning methodologies to enhance detection accuracy. In this section, notable studies are reviewed, contributing to the comprehension and advancement of efficient malware detection systems.

2.1. Deep Learning Studies

Given the rise in malware targeting Android systems, recent research has increasingly focused on deep learning approaches to enhance detection accuracy. This section reviews notable studies that leverage deep learning architectures for malware detection, highlighting their methodologies and performance outcomes in comparison to traditional techniques.

Elayan and Mustafa conducted a study introducing a Gated Recurrent Unit (GRU)-based Recurrent Neural Network (RNN) as an innovative approach for detecting malware on the Android operating system. Trained using static features extracted from Android applications, such as API calls and permissions, their model demonstrated significantly higher performance than traditional methods, achieving an accuracy rate of 98.2% on the CICAndMal2017 dataset. [4]. In the work of Bakour and Ünver [5], a hybrid model called DeepVisDroid was proposed for Android malware detection, combining deep learning techniques with image-based features. This model transforms the source code of Android applications into four different grayscale image datasets, from which local and global features are extracted and analyzed. The proposed DeepVisDroid model achieved high success, reaching an accuracy of 98.96%. Yadav et al. presented an approach that utilizes images derived from bytecode files for malware detection and proposed an EfficientNet-B4 CNN based model. The EfficientNet-B4 architecture was chosen as the feature extractor for this process and worked with 226x226 images. In the study, 5986 samples were collected, converted to color images, and mapped to binary files. These images derived from Android bytecode representations were evaluated with their proposed model. They showed that their method was effective by achieving 95.7% accuracy [6]. Yumlembam et al. investigated the effectiveness of graph neural networks in detecting attacks. In the study, unique global descriptors were created using local and global graphs obtained from API features. The importance weight of each feature was calculated using linear regression, and graph embedding and model training were performed using graph neural networks. The experiments were conducted on two datasets, one with 15,848 and the other with 56,461 samples. In this study, the proposed model was evaluated in terms of accuracy, precision, recall and F-score, and successful results up to 99.18% were obtained. In addition, a hostile malware generation model called VGAE-MalGAN was developed and 98.43% accuracy was achieved with this model [7]. Furthermore, In their study, Bakır and Bakır [3] emphasized the importance of feature extraction methods for malware detection in Android systems. Therefore, they proposed a new feature extraction method, autoencoder-based DroidEncoder. In the study, an image-based dataset was created from Drebin and Malgenome datasets and studies were carried out using this dataset. The authors proposed three different autoencoders based on ANN, CNN and VGG19. At the end of the study, it was observed that the proposed method gave successful results in terms of different metrics. Mohammed et al. investigated deep learning techniques for Android application categorization. In this context, they proposed a deep belief neural network (DBN)-based application categorization method. Using the CIC-AAGM2017 dataset of 1900 instances, the proposed model was compared with four traditional feed-forward neural networks and seven machine learning models. The results show that the DBN-based model is effective in classifying Android apps as benign or malicious with 98.7% accuracy [8]. Moreover, Tang et al. in their [9] research study propose a new classification method for detecting Android malware by addressing the weaknesses of traditional static analysis methods. The proposed method utilized a deep neural network that combines hashed bytecode image and attention mechanism. The method (ResNet-CBAM) processes the bytecode sequence of executable files into grayscale and Markov images and fuses these features to generate a feature space that can characterize Android malware. Experiments showed that the proposed ResNet-CBAM method can effectively represent bytecode sequence files, extract, and classify features. Based on the mixed image features, the malware detection accuracy reaches 98.67% and outperforms other similar methods [9]. Fu et al. stated that traditional methods cannot detect malware accurately and effectively due to their limitations. Therefore, they proposed a hybrid approach that combines multi-scale convolutional neural network (MSCNN) and ResNet networks. The approach was able to detect Android malware with high accuracy and precision by creating an advanced feature extraction network with MSCNN and a detection network with ResNet. At the end of the study, the authors confirmed that the results of the experiments show that the use of MSCNN as a multilevel feature extraction network significantly improves the performance of the hybrid models [10]. Liu et al. pointed out that semantic behavior feature extraction is critical for training a robust malware detection model. Therefore, they proposed SeGDroid, a novel Android malware detection method that focuses on learning semantic information from sensitive function call graphs (FCGs). SeGDroid preserves sensitive API call context and removes irrelevant FCG nodes using graph pruning methods. Attributes of graph nodes are extracted by proposing a node representation method based on word2vec and social network-based centrality. This representation aimed to extract semantic information and graph structure of function calls. Experimental results showed that SeGDroid achieved 98.37% accuracy in the case of malware detection on the CICMal2020 dataset [11].

2.2. Machine Learning Studies

In recent years, machine learning (ML) techniques have played a crucial role in Android malware detection, offering effective solutions through algorithms capable of identifying patterns and anomalies in application behavior. This section explores key studies that apply machine learning models to classify malware, highlighting approaches such as decision trees, support vector machines, and ensemble methods. These studies underscore the adaptability and efficiency of machine learning in tackling the challenges posed by Android malware, setting a foundation for further advancements and hybrid approaches in the field.

For instance, Raman et al., in their work, proposed an ML-based method for detecting Android malware. The proposed method is optimized to detect Android malware with a KNN classification system using data stream-based API calls. Based on 1,050 malicious materials and 1,160 benign samples, the study [12] showed that the dataflow-based API-level features are successful (97.66%) in effectively detecting Android malware [12]. Similarily, Alani and Awad presented an ML-based system called AdStop for detecting Android adware by analyzing features in network traffic flow. While developing AdStop, they targeted design features such as high accuracy, speed, and generalizability. To improve the accuracy of adware detection and reduce the time burden, a feature reduction phase was applied, thus reducing the number of features used from 79 to 13. In the experiments, AdStop was found to be successful with 98.02% accuracy, 2% false positive rate, and 1.9% false negative rate [13]. In their study, Duran and Bakir [14] used machine learning algorithms for static analysis-based malicious application detection for the Android operating system. The imbalance of the class distribution in the dataset was eliminated by generating artificial data with the SMOTE algorithm. They also performed hyperparameter optimization to increase the accuracy of machine learning algorithms. This optimization determined the most appropriate hyperparameters with the Grid Search method. With the increasing threat of Android malware, it has become important to develop effective detection techniques. In the [15] study, the performance of various machine learning algorithms was evaluated. The study reveals that the LightGBM algorithm has the highest accuracy (91%), precision (89%), and F1 score (89%) for Android malware detection. Evaluations on a 5-class dataset containing both benign and malicious applications suggest that these findings can contribute to the development of effective Android malware detection systems. In another study [16], AlOmari et al. addressed the challenges faced by cybersecurity researchers focusing on developing new detection systems with the rapid increase in Android mobile malware threats. They examined the performance of various machine learning algorithms and then focused on achieving maximum accuracy by normalizing numerical features with the Synthetic Minority Oversampling Technique (SMOTE). 11,598 APKs were used on a large dataset and the highest accuracy value was 95.49% with the light gradient boosting model. Furthermore, In order to secure Android mobile applications used in industrial platforms and smart cities, the authors of [17] present a machine learning-based approach called as the Hybrid Multimodal Machine Learning-Driven Android Malware Recognition and Classification (HM3-AMRC) model. HM3-AMRC accurately identifies and classifies Android malware using a new technique for feature selection and analysis that is according to authors more efficient than previous methods. A comprehensive benchmark analysis highlights that the HM3-AMRC method outperforms existing techniques with an accuracy of 99.01 [17].Furthermore, Jundi and Alyasiri in their study developed a hybrid system for malware detection on Android smartphones. They used Extreme Gradient Boosting (XGBoost) and Grammatical Evaluation (GE) to determine the optimal parameters for this detection model. The experimental results of the study showed that the proposed model outperforms conventional parameter tuning. As a result of the study, the proposed model achieved 98% accuracy for CICMalDroid-2020, 99.02% accuracy for Drebin, and 99.28% accuracy for Malgenome [18]. On the other hand, Seyfari and Meimandi conducted a study in order to take precautions against malicious software that has increased with the widespread use of smartphones with Android operating system. In their study, they developed a method using simulated annealing algorithm and fuzzy logic to detect Android malware with machine learning algorithms. The study concluded that the proposed method achieved optimal results with a 99.02% accuracy rate using the KNN classifier in combination with a permission-based feature set. [19].

Table 1 includes some of the recently published studies in the domain of Malware detection.

Table 1. Some of the related studies

Paper	#Data	#Class	Method	Accuracy (%)
Elayan & Mustafa, 2021	712	2	GRU	98.20
Bakour & Unver, 2021	9700	2	DeepVisDroid	98.96
Raman et al., 2022	2210	2	KNN	97.66
Alani & Awad, 2022	86228	2	AdStop with RF	98.14
Yadav et al., 2022	5986	2	EfficientNet-B4 CNN	95.70
Yumlembam et al., 2023	15848	2	VGAE-MalGAN	98.33
	56461	2		98.68
Baghirov, 2023	11598	5	LightGBM	91
AlOmari vd., 2023	11598	2	LightGBM	95.49
A vd., 2023	2000	2	HM3-AMRC	99.01
Jundi & Alyasiri, 2023	3799	2	GE-XGBoost	99.2
	15036	2		99.0
	11598	5		97.9
Bakır & Bakır, 2023	6000	2	DroidEncoder	98.56
Mohammed et al., 2023	1900	2	DBN-Based Model	98.70
Tang vd., 2024	22901	2	ResNet-CBAM	98.67
Seyfari & Meimandi, 2024	15036	2	Proposed Method with KNN	99.02
Fu vd, 2024	11598	5	MSCNN+ResNet18	99.20
Liu vd, 2024	11598	5	SeGDroid	98.37

Examining Table 1 reveals that previous studies on malware detection primarily employ either deep learning methods or traditional machine learning algorithms. Our proposed model, however, integrates deep learning models with the XGBoost algorithm in a hybrid approach, leveraging the strengths of both. Deep learning models are adept at extracting distinctive features from high-dimensional and complex data, capturing intricate patterns that are essential for effective malware detection. These extracted features are then fed into XGBoost, a robust classifier known for its high accuracy and generalization capabilities. By combining the feature extraction power of deep learning with the strong classification performance of XGBoost, this hybrid approach achieves a more accurate and resilient malware detection system.

3. MATERIALS AND METHODS

This section provides a comprehensive overview of the dataset used in the study, describing its key characteristics and relevance for malware detection. To address the imbalance in the dataset, three data distribution techniques—SMOTE, SMOTETomek, and ClusterCentroids—are presented, each explained in detail to demonstrate their roles in rebalancing the data. Finally, this section describes the eight different deep learning models and the XGBoost algorithm used in the study, highlighting their specific functionalities and how they contribute to the hybrid approach for improved malware detection accuracy.

3.1. Used Dataset

This study analyzes the DroidCollector network traffic analysis dataset [17], [18]. Comprising 7844 data samples and spanning 17 attributes, this dataset serves as a foundational resource for our investigation. It is specifically designed for detecting malicious activity in Android applications based on network traffic analysis. The dataset was obtained through dynamic analysis, allowing for the capture of real-time network traffic behavior during the execution of Android applications in a controlled environment. This dynamic approach helps in identifying subtle differences in the network patterns of malicious and benign applications. The details of the dataset, including its attributes and sample sizes, are presented in Table 2.

3.2. Preparing Dataset

Preparing a dataset is crucial for the success of machine learning tasks like Android malware detection. It directly impacts the quality, generalization, and fairness of the model. A well-prepared dataset reduces bias, enhances interpretability, and ensures compliance with ethical considerations. It also facilitates reproducibility, saves computational resources, and increases the real-world applicability of the model. Overall, proper dataset preparation is essential for building reliable, accurate, and ethical machine learning models that contribute to the security of mobile devices and the digital ecosystem.

A meticulous preliminary analysis brought to light the presence of missing values (NaN) within certain attributes. Recognizing the potential impact of these missing values on the accuracy of our analysis, a strategic approach was formulated. Specifically, the attributes 'duracion,' 'avg_local_pkt_rate,' and 'avg_remote_pkt_rate' were identified as containing NaN values and subsequently removed from the dataset. This meticulous curation of the dataset serves a dual purpose: it not only ensures precision and coherence in our analytical processes but also elevates the dataset's reliability by adeptly addressing the challenge posed by missing values. Consequently, this methodical handling contributes to the robustness of our findings and enhances the overall quality of the dataset employed in our study.

The final attributes and descriptions of the dataset are presented in Table 2.

Following these procedures, it was observed that the 'name' and 'type' attributes of the dataset consisted of object expressions. To facilitate further analysis, the Label Encoder method was implemented, converting these attributes into numeric values. Subsequently, adjustments were made to the 'type' column, rendering it suitable for classification purposes. The resultant 'Benign' and 'Malicious' class distributions of the dataset are detailed in Table 3.

Table 2. Features and description of dataset

Feature	**Description**
name	‘AntiVirus' 'Browser' 'chess' 'Communication' 'DailyLife' 'Education' 'Finance' 'HealthAndFitness' 'Input' 'MediaAndVideo' 'NewsAndMagazines' 'Personalization' 'Photography' 'Productivity' 'Reading' 'Shopping' 'Social' 'Sport' 'Tools' 'TravelAndLocal' 'Ackposts' 'Acnetdoor' 'Adrd' 'Adsms' 'Aks' 'Antares' 'Anudow' 'BaseBridge' 'Boxer' 'DroidDream' 'DroidKungFu' 'DroidRooter' 'DroidSheep' 'EICAR-Test-File' 'EWalls' 'ExploitLinuxLotoor' 'FaceNiff' 'FakeDoc' 'FakeFlash' 'FakeInstaller' 'Fakelogo' 'Fakengry' 'FakeRun' 'FakeTimer' 'FinSpy' 'Fjcon' 'FoCobers' 'Fujacks' 'Gamex' 'Gapev' 'Gappusin' 'GGtrack' 'GinMaster' 'Glodream' 'Gmuse' 'Gonca' 'Hamob' 'Hispo' 'Iconosys' 'Imlog' 'JSExploit-DynSrc' 'JSmsHider' 'Kmin' 'Ksapp' 'Loozfon' 'Luckycat' 'Maxit' 'MMarketPay' 'Mobilespy' 'Mobsquz' 'Moghava' 'Nandrobox' 'Nickspy' 'NickyRCP' 'Nyleaker' 'Opfake' 'Pirater' 'Pirates' 'PJApps' 'Placms' 'Plankton' 'Raden' 'RootSmart' 'SafeKidZone' 'Saiva' 'Sakezon' 'Sdisp' 'SeaWeth' 'SendPay' 'SerBG' 'Smspacem' 'SMSreg' 'Spy.GoneSixty' 'Spy.ImLog' 'SpyHasb' 'SpyMob' 'SpyPhone' 'Spyset' 'Stealer' 'Stealthcell' 'Steek' 'Tesbo' 'TheftAware' 'Trackplus' 'TrojanSMS.Denofow' 'TrojanSMS.Hippo' 'Updtbot' 'Vdloader' 'Vidro' 'Xsider' 'YcChar' 'Yzhc' 'Zitmo' 'Zsone'
tcp_packets	it has the number of packets TCP sent and got during communication.
dist_port_tcp	it is the total number of packets different from TCP
external_ips	represents the number the external addresses (IPs) where the application tried to communicated
vulume_bytes	it is the number of bytes that was sent from the application to the external sites
udp_packets	the total number of packets UDP transmitted in a communication
tcp_urg_packet	represents a special type of TCP packet expressing an emergency situation, where the "URG" flag in the TCP header is used

source_app_packets	it is the number of packets that were sent from the application to a remote server
remote_app_packets	number of packages received from external sources
source_app_bytes	this is the volume (in Bytes) of the communication between the application and server
remote_app_bytes	this is the volume (in Bytes) of the data from the server to the emulator

Table 3. Dataset distributions

Type	Number of Data
Benign	4704
Malicious	3141

3.3. Addressing the Imbalanced Data Sampling Challenge

When dealing with imbalanced class distributions in a dataset, conventional classification algorithms may exhibit a bias towards the majority class, diminishing the effectiveness of detecting minority class instances. This imbalance poses a significant challenge in achieving optimal performance with deep learning algorithms. Therefore, it becomes imperative to rectify this issue by employing techniques that balance the dataset, enhancing reliability and efficiency. Two commonly used methods are oversampling (introducing additional data) and undersampling (removing data), as highlighted by [19].

3.3.1. SMOTE (Synthetic Minority Over-Sampling Technique)

SMOTE is a powerful technique designed to fortify the minority class in datasets exhibiting class imbalance, thereby promoting a more balanced learning model. This method generates synthetic examples by interpolating instances from the minority class, enabling the learning model to better discern minority class examples and improve overall performance. SMOTE effectively mitigates overfitting issues associated with random oversampling and addresses information loss resulting from random undersampling. This ensures that the model possesses a more robust and generalizable structure [20].

3.3.2. SMOTETomek

SMOTETomek is a rebalancing strategy that creates a balanced dataset by over-sampling the minority class while simultaneously under-sampling the majority class. This strategy aims to obtain the examples used to achieve balance between classes in a more balanced and effective manner, as well as improve the model's capacity to obtain information from both categories more effectively. In this way, the learning model gains a more generalizing structure and reduces the possibility of misclassification, allowing more reliable results to be obtained. SMOTETomek improves the performance of learning algorithms by providing an effective solution to balance between classes with few and many examples [21].

3.3.3. ClusterCentroids

ClusterCentroids is a technique that generates synthetic samples by clustering minority class instances within the dataset using clustering algorithms and leveraging the centers of these clusters. This method aims to alleviate the challenges posed by class imbalance by improving the representation of minority class instances. By creating synthetic samples based on clustered representations, ClusterCentroids contributes to a more balanced and representative dataset, thereby enhancing the performance of learning models, especially in scenarios with imbalanced class distributions [22]. The results obtained from this phase of the study are presented in Table 4.

Table 4. Dataset distribution after preprocessing

Data Type	#Benigndata	#Maliciousdata
Original	4704	3141
SMOTE	4704	4704
SMOTETomek	4485	4485
ClusterCentroids	3141	3141

3.4. Deep Learning Methods

In recent years, deep learning has rapidly solved complex problems in various scientific fields and gained importance as a subfield of artificial intelligence. This development has revealed deep learning methods that are used effectively in applications such as pattern recognition and data analysis. Deep learning involves deep neural networks consisting of hierarchical layers that are capable of automatic learning, often on large and complex datasets. These methodologies have demonstrated remarkable success, notably in fields such as image and voice recognition, natural language processing, malware detection, and other cognitively demanding tasks [23], [24], [25], [26]. Deep learning contributes to the acceleration of scientific and technological developments with its ability to reveal complex relationships within data [27].

3.4.1. CNN

Convolutional Neural Networks (CNN) are one of the deep learning models that are effective in tasks such as computer vision [28] recognition [29] and classification [30]. CNN provides the ability to learn and generalize patterns and features more effectively, especially by being used in areas such as image and video analysis. Thanks to their filtering and pattern recognition capabilities, CNN models are used in many application areas to achieve high performance on complex visual data [31].

3.4.2. RNN

Traditional Neural Networks typically do not retain their final results for subsequent phases, whereas Recurrent Neural Networks (RNNs) are specifically engineered to address this constraint. RNNs offer a unique capability for data persistence through internal feedback loops, enabling them to retain memory of previous information using interconnected components. Thanks to these features, they can successfully process sequential datasets such as language modeling, text generation, and time series forecasting. In order to learn long-term dependencies more effectively, models such as GRU and LSTM, which are advanced variants of RNNs, are also used [32].

3.4.3. GRU

Gated Recurrent Unit (GRU) is an RNN variant that aims to learn long-term dependencies more effectively. GRU is a deep learning model that is particularly successful when applied to sequential data processing tasks such as language modeling, text generation, and time series analysis. GRUs are specifically designed to solve the vanishing gradient problem in traditional RNNs, providing an effective solution to prevent gradients from shrinking excessively over time and to prevent long-term dependencies. Thanks to their lightweight structure, GRUs offer faster training processes and less computational complexity, providing effective performance, especially on large datasets [33].

3.4.4. BiGRU

Different from unidirectional GRU models, the Bidirectional Gated Recurrent Unit (BiGRU) model includes information in both forward and reverse time directions. Forward GRU captures prior information and reverse GRU captures subsequent information, obtaining a wide range of context information in the network intrusion traffic prediction task and effectively extracting deep features of the traffic. These two GRUs with opposite directions jointly determine the output of the current location, thus providing a more comprehensive prediction/classification capability [34].

3.4.5. LSTM

Long Short-Term Memory (LSTM) networks consist of three main gates: input, output, and forget gates. These gates include a sigmoid neural network layer and a point multiplication process, which processes the input vector to determine the rate at which each component is allowed to pass. LSTM is a type of RNN and is particularly successful in time series analysis, language modeling, and natural language processing tasks. Its ability to effectively learn long-term dependencies and its capacity to store information make LSTM an effective tool in complex intra-temporal relationship and pattern recognition tasks [35], [36].

3.4.6. BiLSTM

Bidirectional Long Short-Term Memory (Bi-LSTM) is a type of RNN that combines memory cells and a gate mechanism, enabling efficient modeling of sequential data. Bi-LSTM has a bi-directional structure, consisting of two LSTM layers, with the input sequence being processed in the forward direction and used in the backward direction. The outputs of the two layers are combined to produce the final output, and the output of the hidden layers is passed through a linear layer that calculates probability scores. Bi-LSTM, with its ability to capture both prior and subsequent contextual information in the input sequence, provides a more comprehensive contextual understanding by simultaneously evaluating information before and after the current time step using forward and backward LSTM layers [37].

3.4.7. CNN+BiGRU

Compared to traditional neural networks, CNN offers advantages in weight sharing between the receiver field view and the hidden layer, especially given the non-linearity and randomness of network traffic data. Thanks to the weight-sharing mechanism, CNN can reduce network complexity and facilitate feature extraction with the same convolution kernel. The CNN-BiGRU model combines CNN and Bidirectional Gated Recurrent Unit (BiGRU) architectures, which are effective in image and sequential data analysis, capturing spatial and temporal context and offering a wide range of applications. This model can be successfully used in areas such as visual data and time series analysis. [38].

3.4.8. CNN+BiLSTM

Compared to traditional techniques, CNN-based feature learning enables an end-to-end information processing process from input to output, bypassing the feature extraction phase. However, considering that a single model may not provide ideal results in predicting time series data, more effective results can be achieved by successfully combining the local feature extraction capabilities of CNN with the nonlinear temporal processing capabilities of BiLSTM. In this context, the CNN-BiLSTM model is a deep learning model that can be effectively used in visual data analysis and sequential data processing. The CNN part is used for feature extraction and the BiLSTM part is used for sequential data analysis, capturing spatial and temporal context simultaneously, providing a wide range of applications [39].

3.4. Machine Learning Methods

Machine learning is a branch of artificial intelligence that can make accurate predictions by providing applications with the capacity to learn from experience and data rather than predetermined rules. Machine learning algorithms work by using input data as features to predict new output values. This discipline focuses on pattern recognition and learning, improving the ability of computer systems to learn from experience and data [40].

3.5.1. XGBoost

XGBoost, or eXtreme Gradient Boosting, is a standout Gradient Boosting algorithm renowned for its exceptional scalability. Boasting high speed and performance, XGBoost is known to be ten times faster than alternative methods. Its superiority lies in swift model tuning, facilitated by a distinctive regularization technique that mitigates overfitting. This algorithm is a formidable asset for tackling regression and classification challenges, demonstrating efficacy across a myriad of applications. Leveraging optimization techniques such as parallel processing, tree regularization, and feature selection, XGBoost emerges as a powerful and versatile tool in data analysis.[41], [42]. In this study, XGBoost was employed due to its remarkable scalability, high speed, and performance, making it well-suited for handling large datasets efficiently. Its unique regularization technique helps prevent overfitting, ensuring the robustness of the models developed in the study. Additionally, XGBoost's effectiveness in regression and classification tasks, coupled with its proven track record in various applications, made it a compelling choice for enhancing the accuracy of the malware detection system being investigated.

4. EXPERİMENTAL RESULTS AND DISCUSSION

4.1. Experimental setup and evaluation metrics

The computer used in the experiments is equipped with features that offer high performance and processing capacity. The main component of the system is a powerful 48-core Xeon processor, which is capable of handling intensive processing loads. The system is also equipped with 256 GB of RAM for efficient data processing and memory management. This combination of hardware allows us to perform our experiments efficiently and effectively. Python programming language was employed for the execution of the experiments.

Standard evaluation metrics were employed in this study to assess the results obtained from the experiments such as accuracy, precision, recall, and F1-score.

The model used in the study is presented in Figure 1.As seen in the Figure 1. the study consists of two phases. In the first phase of the study, classification tasks were conducted using eight different deep learning methods, including CNN, RNN, GRU, Bi-GRU, LSTM, Bi-LSTM, CNN+BiGRU, and CNN+BiLSTM, on both the original dataset and the dataset enhanced with imbalanced data sampling techniques. Furthermore, to achieve optimal results in the classification process, hyperparameter optimization was performed using Optuna in conjunction with a genetic algorithm. The hyperparameter ranges used for optimization are provided in Table 5. Throughout this phase, the epoch value was kept constant at 50. Additionally, experiments were conducted on three different distributions of 70/30, 80/20, and 90/10 as train and test. The same training and test sets were used throughout both phases of the experiment. Specifically, after the deep learning model was trained, features were extracted and passed as input to the XGBoost classifier. Consistency was maintained across both stages by keeping the same validation and test sets during feature extraction and classification, thereby preventing data leakage or bias in model evaluation. This approach ensured a fair comparison of performance between the standalone deep learning models and the hybrid model with XGBoost. The accuracy, precision, recall, and F1-score results of these studies are presented in detail in Tables 6, 7, 8, and 9. Table 6 contains the results of the study conducted with the original dataset, while Tables 7, 8, and 9 present the results of the studies conducted with the datasets obtained after the Smote, SmoteTomek, and ClusterCentroid processes, respectively.The methods that yielded the best results were determined during this phase. In the second phase, the classification process was carried out using the deep learning structures identified as the most effective in the first phase, combined with the XGBoost ML classifier. This approach involved training the deep learning models and subsequently feeding their outputs into the XGBoost ML model. The f1-score was used as the evaluation metric, as it provides a balanced assessment of model performance by considering both precision and recall. The f1-score results for this phase are presented in Table 10.

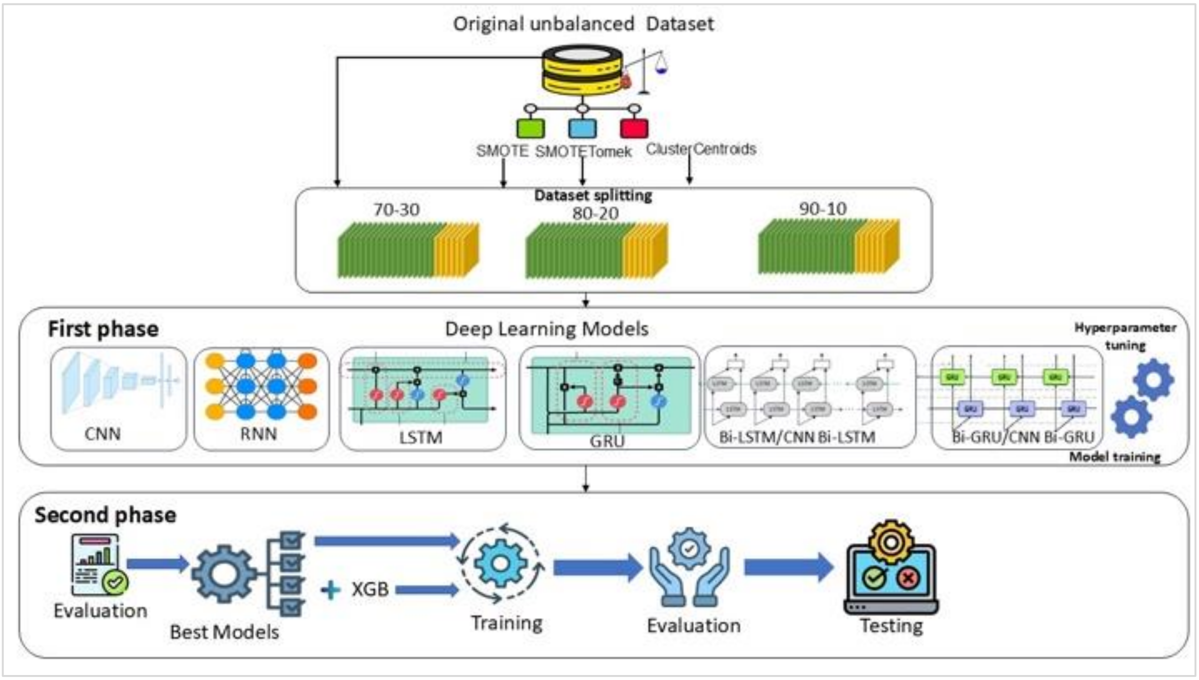

Figure 1. The proposed method

Table 5. Hyperparameter ranges

According by	Range
Optimization alg.	Mini-batchGD, MomentumGD, Adam, Adadelta, Adagrad, Adamax, Nadam
Conv_layer_number	1, 2, 3
Filter_size	32, 64, 96, 128
Kernel_size	3,5
Activation	relu, tanh, gelu, swish, selu, LeakyReLU
Kernel_initializer	uniform, lecun_uniform, normal, zero, glorot_normal, he_normal, he_uniform
Dense_layers_number	1, 2, 3, 4, 5, 6
Dense_neuron_number	32, 64, 96, 128
Dense_activation	relu, tanh, gelu, swish, selu, LeakyReLU
Dense_kernel_initializer	uniform, lecun_uniform, normal, zero, glorot_normal, he_normal, he_uniform

Table 6. Results of the Original Dataset

Model	Train (%)	Accuracy	Precision	Recall	F1-Score
CNN	70	75.83	89.11	75.06	81.48
	80	85.53	82.99	91.94	87.24
	90	79.11	87.30	80.68	83.86
RNN	70	88.91	93.88	88.29	91.00
	80	81.26	89.63	80.97	85.08
	90	74.39	83.40	77.23	80.20
GRU	70	85.00	88.61	86.58	87.58
	80	80.05	84.60	82.40	83.48
	90	83.95	88.73	85.91	87.30

Bi-GRU	70	91.08	89.25	95.51	92.27
	80	**91.84**	**92.94**	**93.34**	**93.14**
	90	87.90	90.57	90.02	90.30
LSTM	70	78.08	95.23	74.87	83.83
	80	87.25	91.76	87.46	89.56
	90	86.88	90.37	88.73	89.54
Bi-LSTM	70	82.33	88.26	83.17	85.64
	80	88.59	90.48	90.38	90.43
	90	**91.85**	**93.65**	**93.27**	**93.46**
CNN+BiGRU	70	86.79	87.62	89.99	88.78
	80	72.72	87.70	72.37	79.30
	90	88.15	92.62	88.80	90.67
CNN+BiLSTM	70	87.51	88.26	90.58	89.40
	80	88.34	89.73	90.60	90.17
	90	78.98	89.55	79.31	84.12

Table 7. Results of the Smote Dataset

Model	Train (%)	Accuracy	Precision	Recall	F1-Score
CNN	70	83.21	86.71	81.49	84.02
	80	88.95	85.94	91.23	88.51
	90	80.13	84.00	78.24	81.02
RNN	70	87.42	88.24	87.21	87.72
	80	86.82	83.48	89.22	86.25
	90	76.51	80.63	74.80	77.61
GRU	70	88.66	88.31	89.30	88.80
	80	87.83	85.09	89.81	87.38
	90	87.35	84.00	90.27	87.02
Bi-GRU	70	61.60	98.05	57.16	72.22
	80	77.52	67.38	84.07	74.81
	90	90.33	88.63	91.92	90.25
LSTM	70	81.44	86.78	78.87	82.64
	80	83.85	84.55	83.12	83.83
	90	83.74	79.16	87.44	83.09
Bi-LSTM	70	83.60	90.26	80.06	84.85
	80	**90.38**	**90.67**	**89.99**	**90.33**
	90	76.83	81.05	75.05	77.94
CNN+BiGRU	70	81.72	85.87	79.77	82.71
	80	90.06	87.88	91.71	89.75
	90	90.33	91.16	89.83	90.49
CNN+BiLSTM	70	88.81	87.06	90.59	88.79
	80	88.10	89.27	87.03	88.14
	90	87.04	85.89	88.12	86.99

Table 8. Results of the SmoteTomek Dataset

Model	Train (%)	Accuracy	Precision	Recall	F1-Score
CNN	70	87.70	82.59	91.41	86.78
	80	87.79	83.99	91.94	87.79
	90	85.68	84.97	86.67	85.81
RNN	70	86.59	85.55	86.81	86.17
	80	90.13	90.72	90.43	90.57
	90	75.03	92.81	68.93	79.11
GRU	70	81.05	82.21	79.66	80.91
	80	75.64	85.91	72.52	78.65
	90	86.68	92.37	83.30	87.60
Bi-GRU	70	87.11	86.77	86.83	86.80
	80	84.34	91.89	80.77	85.97
	90	74.81	97.60	67.47	79.79
LSTM	70	83.62	80.30	85.30	82.73
	80	85.95	85.17	87.60	86.36
	90	74.03	77.56	73.10	75.26
Bi-LSTM	70	91.05	93.16	89.03	91.04
	80	**94.87**	**96.16**	**94.15**	**95.14**
	90	88.35	89.54	87.82	88.67
CNN+BiGRU	70	88.78	86.31	90.29	88.26
	80	88.91	89.43	89.34	89.39
	90	76.47	86.71	72.50	78.97
CNN+BiLSTM	70	89.64	87.83	90.66	89.22
	80	86.23	88.15	85.86	86.99
	90	88.46	86.06	90.80	88.37

Table 9. Results of the ClusterCentroid Dataset

Model	Train (%)	Accuracy	Precision	Recall	F1-Score
CNN	70	82.02	84.89	81.14	82.97
	80	77.65	77.16	77.78	77.47
	90	89.67	91.02	89.68	90.34
RNN	70	70.93	96.51	64.62	77.41
	80	85.12	89.94	81.95	85.76
	90	76.47	78.14	77.68	77.91
GRU	70	81.17	71.53	89.92	79.68
	80	78.84	79.87	78.13	78.99
	90	87.12	84.73	90.42	87.48
Bi-GRU	70	68.22	93.73	62.90	75.28
	80	80.51	68.05	90.45	77.67
	90	71.22	56.89	83.70	67.74
LSTM	70	84.14	84.28	84.89	84.58
	80	83.93	84.19	83.65	83.92

	90	77.74	78.44	79.39	78.92
Bi-LSTM	70	88.70	94.35	85.32	89.60
	80	80.51	83.87	78.48	81.08
	90	**89.98**	**91.92**	**89.50**	**90.69**
CNN+BiGRU	70	85.36	82.63	88.25	85.35
	80	88.46	88.02	88.73	88.37
	90	88.55	87.13	90.94	88.99
CNN+BiLSTM	70	87.48	87.67	88.03	87.85
	80	78.04	73.96	80.38	77.04
	90	88.39	88.32	89.67	88.99

Tables 6, 7, 8, and 9 show that the optimal training/testing distribution, which yields the best results for each data type, occurs when 80% of the dataset is allocated to training and 20% to testing. The analysis of deep learning methods reveals that the RNN model consistently produces the lowest performance results. Furthermore, bidirectional models appear to achieve higher success rates compared to other deep learning methods. Notably, when the dataset obtained after the SmoteTomek process is used, the most successful results are achieved.

Considering these findings, an optimal f1-score of 95.14% was achieved using the Bi-LSTM deep learning method with an 80/20 training/test distribution on the dataset obtained after the SmoteTomek process. Based on the results from the first phase, it was decided to use the SmoteTomek dataset in the second phase, with 80% allocated for training and 20% for testing. The first-phase results also revealed that the lowest performance was observed in studies conducted with the dataset obtained after applying the ClusterCentroid method. This may be attributed to the data loss resulting from the data reduction process of the ClusterCentroid method, which likely negatively impacted the results.

In the second stage of the study, different epoch values, such as 50, 100, and 150, were evaluated to assess their impact on model performance. The models were created using the hyperparameter values shared in Table 11, and the results were then compared. The variation in epoch values was chosen to investigate how training duration affects the performance of the deep learning models. The models were re-run both with and without the XGBoost algorithm, allowing for the evaluation of the impact of different epochs on the classification results and enabling a comparative analysis to identify the optimal configuration for each scenario. The results of the Bi-LSTM, Bi-GRU, and CNN models, which achieved the three highest success rates in the second phase of the study, are presented in Table 10.

Table 10. F1-Score and Accuracy (Acc) results for the top 3 results in the second phase of the study

Epoch	Score	BiLSTM	BiLSTM +XGB	BiGRU	BiGRU+XGB	CNN	CNN +XGB
50	**F1- Score**	95.04	95.79	85.97	90.42	87.79	92.03
	Acc	94.77	97.44	84.34	92.00	87.79	93.72
100	**F1- Score**	92.88	95.72	94.18	94.42	91.60	92.21
	Acc	92.36	97.97	93.98	98.24	91.36	96.41
150	**F1- Score**	95.12	**95.12**	93.50	94.25	91.68	92.75
	Acc	94.93	**99.33**	93.26	98.85	92.92	99.13

Table 11. Hyperparameter Values for Each Model

Model	Hyperparameter	Vlaue
CNN	Optimization alg	Adamax
	Conv_layer_number	3
	Conv_Filters	(96, 96, 32)
	Conv_kernel_size	(3, 3, 5)
	Conv_activation	('tanh', 'tanh', 'tanh')
	Conv_kernel_initializer	('glorot_normal', 'lecun_uniform', 'he_uniform')
	Dense_layers_number	3
	Neuron_number in Dense layers	(96, 96, 64)
	Dense_activation	('tanh', 'tanh', 'relu')

	Dense_kernel_initializer	('lecun_uniform', 'lecun_uniform', 'uniform')
RNN	Optimization alg	Adamax
	Rnn_layer_number	3
	Rnn_units	(32, 96, 32)
	Rnn_activation	('relu', 'tanh', 'selu')
	Rnn_kernel_initializer	('normal', 'glorot_normal', 'uniform')
	Dense_layers_number	2
	Dense_neuron_number	(128, 128)
	Dense_activation	('tanh', 'tanh')
	Dense_kernel_initializer	('glorot_normal', 'uniform')
LSTM	Optimization alg	Adamax
	Lstm_layer_number	2
	Lstm_units	(32, 128)
	Lstm_activation	('swish', 'tanh')
	Lstm_kernel_initializer	('lecun_uniform', 'glorot_normal')
	Dense_layers_number	4
	Dense_neuron_number	(64, 32, 96, 64)
	Dense_activation	('swish', 'gelu', 'swish', 'swish')
	Dense_kernel_initializer	('glorot_normal', 'glorot_normal', 'normal', 'normal')
BiLSTM	Optimization alg	Nadam
	Lstm_layer_number	2
	Lstm_units	(128, 64)
	Lstm_activation	('tanh', 'LeakyReLU')
	Lstm_kernel_initializer	('normal', 'he_uniform')
	Dense_layers_number	5
	Dense_neuron_number	(32, 128, 64, 128, 96)
	Dense_activation	('relu', 'swish', 'relu', 'tanh', 'gelu')
	Dense_kernel_initializer	('lecun_uniform', 'glorot_normal', 'lecun_uniform', 'lecun_uniform', 'glorot_normal')
GRU	Optimization alg	Nadam
	Gru_layer_number	2
	Gru_units	(128, 128)
	Gru_activation	('swish', 'tanh')
	Gru_kernel_initializer	('he_normal', 'he_uniform')
	Dense_layers_number	1
	Dense_neuron_number	(64)
	Dense_activation	('relu')
	Dense_kernel_initializer	('glorot_normal')
BiGRU	Optimization alg	Nadam
	BiGru_layer_number	1
	BiGru_units	(64)
	BiGru_activation	('tanh')
	BiGru_kernel_initializer	('glorot_normal')
	Dense_layers_number	3
	Dense_neuron_number	(64, 32, 32)
	Dense_activation	('tanh', 'relu', 'selu')
	Dense_kernel_initializer	('he_uniform', 'lecun_uniform', 'normal')
CNN+BiGRU	Optimization alg	Adam
	Conv_layer_number	1
	Conv_Filters	(32)
	Conv_kernel_size	(3)
	Conv_activation	('tanh')
	Conv_kernel_initializer	('he_normal')
	BiGru_layer_number	3
	BiGru_units	(128, 128, 96)
	BiGru_activation	('gelu', 'LeakyReLU')
	BiGru_kernel_initializer	('normal', 'he_normal', 'uniform')
	Dense_layers_number	4
	Dense_neuron_number	(64, 96, 96,96)
	Dense_activation	('relu', 'relu', 'tanh', 'tanh')
	Dense_kernel_initializer	('lecun_uniform', 'he_uniform', 'normal', 'he_uniform')
CNN+BiLSTM	Optimization alg	Adamax
	Conv_layer_number	3
	Conv_Filters	(32, 96, 128)
	Conv_kernel_size	(3, 5, 5)
	Conv_activation	('tanh', 'swish', 'swish')
	Conv_kernel_initializer	('he_normal', 'uniform', 'he_normal')
	BiLstm_layer_number	3
	BiLstm_units	(32, 96, 64)
	BiLstm_kernel_initializer	('he_uniform', 'he_normal', 'he_uniform')
	Dense_layers_number	3
	Dense_neuron_number	(32, 32, 64)
	Dense_activation	('relu', 'tanh', 'tanh')

	Dense_kernel_initializer	('lecun_uniform', 'normal', 'he_uniform')

According to the results of Table 10, it can be concluded that the results obtained by the hyperdization of deep learning with XGBoost outperform the results of deep learning models alone. This finding underscores the significance of adopting a combined approach, demonstrating that machine learning models achieve more effective results when leveraged together.

Furthermore, it was observed that model performance improved as the number of epochs increased. This indicates that additional learning phases allowed the model to better capture patterns within the dataset, resulting in a more generalizable representation. The increase in epochs positively impacted classification performance by improving the model's ability to capture complex patterns in the data. However, to prevent overfitting—where the model memorizes rather than generalizes—the number of epochs was capped at 150. The BiLSTM+XGBoost method achieved a remarkable 99.33% accuracy and 97.30% F1-score after 150 epochs, demonstrating the hybrid model's strong performance. Additionally, comparisons in Table 1 show that these results surpass the benchmarks of other state-of-the-art studies. The integration of deep learning and machine learning algorithms effectively complements each method's limitations, resulting in enhanced classification accuracy. These results suggest that combining the strengths of deep learning and traditional machine learning approaches can yield more robust and generalizable models for complex datasets. This hybrid methodology demonstrates considerable promise for advanced classification tasks, offering valuable applications in both academic research and industry.

5. CONCLUSION

This paper presents a novel approach to Android malware detection by integrating machine learning and deep learning methods, validated through an extensive experimental study. In malicious application detection, a critical risk lies in misclassifying malicious applications as benign, potentially allowing harmful software to infiltrate the system. This integrated approach aims to mitigate such risks by enhancing detection accuracy and robustness. One of the most important steps to solve this problem is to ensure a balanced distribution of the dataset. For this reason, this study first uses unbalanced data sampling techniques to balance the dataset. Then, eight different deep learning methods were used to classify the original dataset, and the data organized using unbalanced data sampling techniques. At this stage, a rigorous examination of the different training and test set distributions was performed while maintaining a constant epoch value to identify the methods that gave the most favourable results. These initial findings demonstrated the effectiveness of deep learning models on different datasets, especially when supported by unbalanced data sampling techniques.

In the next phase, the deep learning methods that showed the most promising results from the first phase were selected and the hybrid approach combining deep learning with XGBoost was applied. The analysis of the results showed that this hybrid approach improved the classification performance by 3-4%, with a significant increase especially as the epoch value increased.

The proposed hybrid model achieved an impressive accuracy of 99.33%. When compared to results from other benchmark studies, our approach consistently outperforms existing methods, showcasing its superior effectiveness in detecting Android malware. This highlights the potential of the hybrid model in delivering more accurate and reliable outcomes in malware detection.

A limitation of the proposed method is the potential increase in computational complexity resulting from the integration of deep learning with traditional machine learning algorithms. This integration may require significant computational resources and time, especially in the training phase. Furthermore, the performance of the combined model may be sensitive to hyperparameter settings and may require extensive tuning to achieve optimal results.

In addition, it is essential that such systems prevent privacy violations when processing and storing users' personal data. Therefore, the development of transparent and accountable AI systems should not only enhance security but also protect users' rights and privacy.

To address these limitations, future work will include the evaluation of different machine learning algorithms withdifferent deep learning constructs. Furthermore, an ablation study will be conducted to investigate the impact of hyperparameter tuning on both machine learning and deep learning models to improve overall performance. In addition, the use of automated hyperparameter tuning techniques such as grid search, random search or Bayesian optimisation will be investigated to efficiently search the hyperparameter space and identify optimal configurations, thus minimising the computational overhead.

Acknowledgements

In this study, we thank you for using the servers in "LÜTFİ ABAY ARTIFICIAL INTELLIGENCE AND ROBOTICS LABORATORY" in the application of machine learning and deep learning methods.

REFERENCES

[1] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, “A Survey of Android Malware Detection with Deep Neural Models”, ACM Comput. Surv., c. 53, sy 6, s. 126:1-126:36, 2020.

[2] H. Zhu, Y. Li, L. Wang, and V. S. Sheng, “A multi-model ensemble learning framework for imbalanced android malware detection”, Expert Systems with Applications, c. 234, s. 120952, 2023.

[3] H. Bakır and R. Bakır, “DroidEncoder: Malware detection using auto-encoder based feature extractor and machine learning algorithms”, Computers and Electrical Engineering, c. 110, s. 108804, 2023.

[4] O. N. Elayan and A. M. Mustafa, “Android Malware Detection Using Deep Learning”, Procedia Computer Science, c. 184, ss. 847-852, 2021.

[5] K. Bakour and H. M. Ünver, “DeepVisDroid: android malware detection by hybridizing image-based features with deep learning techniques”, Neural Comput & Applic, c. 33, sy 18, ss. 11499-11516, 2021.

[6] H. AlOmari, Q. M. Yaseen, and M. A. Al-Betar, “A Comparative Analysis of Machine Learning Algorithms for Android Malware Detection”, Procedia Computer Science, c. 220, ss. 763-768, 2023

[7] A. Arthi., K. Aggarwal, R. Karthikeyan, S. Kayalvili, S. S, and A. Srivastava, “Hybrid Multimodal Machine Learning Driven Android Malware Recognition and Classification Model”, 2023 7th International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India: IEEE, ss. 1555-1560, 2023.

[8] P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, and T. D. Pham, “EfficientNet convolutional neural networks-based Android malware detection”, Computers & Security, c. 115, s. 102622, 2022.

[9] R. Yumlembam, B. Issac, S. M. Jacob, and L. Yang, “IoT-Based Android Malware Detection Using Graph Neural Network With Adversarial Defense”, *IEEE Internet of Things Journal*, c. 10, sy 10, ss. 8432-8444, 2023.

[10] Z. Z. Jundi and H. Alyasiri, “Android Malware Detection Based on Grammatical Evaluation Algorithm and XGBoost”, 2023 Al-Sadiq International Conference on Communication and Information Technology (AICCIT), Al-Muthana, Iraq: IEEE, ss. 70-75, 2023.

[11] M. A. Mohammed, M. Asante, S. Alornyo, and B. O. Essah, “Android applications classification with deep neural networks”, Iran J Comput Sci, c. 6, sy 3, ss. 221-232, 2023.

[12] J. Tang et al., “Android malware detection based on a novel mixed bytecode image combined with attention mechanism”, *Journal of Information Security and Applications*, c. 82, s. 103721, 2024

[13] Y. Seyfari and A. Meimandi, “A new approach to android malware detection using fuzzy logic-based simulated annealing and feature selection”, Multimed Tools Appl, c. 83, sy 4, ss. 10525-10549, 2024

[14] X. Fu, C. Jiang, C. Li, J. Li, X. Zhu, and F. Li, “A hybrid approach for Android malware detection using improved multi- scale convolutional neural networks and residual networks”, Expert Systems with Applications, c. 249, s. 123675, 2024.

[15] Z. Liu, R. Wang, N. Japkowicz, H. M. Gomes, B. Peng, and W. Zhang, “SeGDroid: An Android malware detection method based on sensitive function call graph learning”, Expert Systems with Applications, c. 235, s. 121125, 2024.

[16] R. Raman, K. R. Nirmal, A. Gehlot, S. Trivedi, D. Sain, and R. Ponnusamy, “Detecting Android Malware and Sensitive Data Flows Using Machine Learning Techniques”, 2022 5th International Conference on Contemporary Computing and Informatics (IC3I), Uttar Pradesh, India: IEEE, ss. 1694-1698, 2022.

[17] M. M. Alani and A. I. Awad, “AdStop: Efficient flow-based mobile adware detection using machine learning”, Computers & Security, c. 117, s. 102718, 2022.

[18] A. Duran and H. Bakır, “Hiperparametreleri Ayarlanmış Makine Öğrenimi Algoritmalarını Kullanarak Android Sistemlerde Kötü Amaçlı Yazılım Tespiti”, *Uluslararası Sivas Bilim ve Teknoloji Üniversitesi Dergisi*, c. 2, sy 1, Art. sy 1, 2023.

[19] E. Baghirov, “Evaluating the Performance of Different Machine Learning Algorithms for Android Malware Detection”, 2023 5th International Conference on Problems of Cybernetics and Informatics (PCI), Baku, Azerbaijan: IEEE, ss. 1-4, 2023.

[20] A. Zhang, H. Yu, S. Zhou, Z. Huan, and X. Yang, “Instance weighted SMOTE by indirectly exploring the data distribution”, Knowledge-Based Systems, c. 249, s. 108919, 2022.

[21] M. G. Lanjewar, K. G. Panchbhai, and L. B. Patle, “Fusion of transfer learning models with LSTM for detection of breast cancer using ultrasound images”, Computers in Biology and Medicine, c. 169, s. 107914, 2024.

[22] W.-C. Lin, C.-F. Tsai, Y.-H. Hu, and J.-S. Jhang, “Clustering-based undersampling in class-imbalanced data”, Information Sciences, c. 409-410, ss. 17-26, 2017.

[23] R. Ghanem and H. Erbay, “Spam detection on social networks using deep contextualized word representation”, Multimed Tools Appl, c. 82, sy 3, ss. 3697-3712, 2023.

[24] H. Bakir and R. Bakir, “Evaluating The Robustness of Yolo Object Detection Algorithm in Terms Of Detecting Objects in Noisy Environment”, Journal of Scientific Reports-A, sy 054, ss. 1-25, 2023.

[25] J. B. Lee and H. G. Lee, “Quantitative analysis of automatic voice disorder detection studies for hybrid feature and classifier selection”, Biomedical Signal Processing and Control, c. 91, s. 106014, 2024.

[26] Y. Alaca and Y. Çelik, “Cyber attack detection with QR code images using lightweight deep learning models”, Computers & Security, c. 126, s. 103065, 2023.

[27] J. Zhang, W. Gong, L. Ye, F. Wang, Z. Shangguan, and Y. Cheng, “A Review of deep learning methods for denoising of medical low-dose CT images”, Computers in Biology and Medicine, s. 108112, 2024.

[28] S. Kaushal, D. K. Tammineni, P. Rana, M. Sharma, K. Sridhar, and H.-H. Chen, “Computer vision and deep learning-based approaches for detection of food nutrients/nutrition: New insights and advances”, Trends in Food Science & Technology, c. 146, s. 104408, 2024.

[29] S. Raziani and M. Azimbagirad, “Deep CNN hyperparameter optimization algorithms for sensor-based human activity recognition”, Neuroscience Informatics, c. 2, sy 3, s. 100078, 2022.

[30] S. Bhardwaj and M. Dave, “Enhanced neural network-based attack investigation framework for network forensics: Identification, detection, and analysis of the attack”, Computers& Security, c. 135, s. 103521, 2023.

[31] E. K. Yılmaz, K. Adem, S. Kılıçarslan, and H. A. Aydın, “Classification of lemon quality using hybrid model based on Stacked AutoEncoder and convolutional neural network”, Eur Food Res Technol, c. 249, sy 6, ss. 1655-1667, 2023.

[32] N. Raj, “Prediction of Stock Market Using LSTM-RNN Model”, içinde 2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS), Erode, India: IEEE, ss. 623-628, 2023.

[33] Y. Yang, Chaoluomeng, and N. Razmjooy, “Early detection of brain tumors: Harnessing the power of GRU networks and hybrid dwarf mongoose optimization algorithm”, Biomedical Signal Processing and Control, c. 91, s. 106093, 2024.

[34] W. Zheng, P. Cheng, Z. Cai, and Y. Xiao, “Research on Network Attack Detection Model Based on BiGRU-Attention”, içinde 2022 4th International Conference on Frontiers Technology of Information and Computer (ICFTIC), Qingdao, China: IEEE, ss. 979-982, 2022.

[35] E. K. Yılmaz and M. A. Akcayol, “SUST-DDD: A Real-Drive Dataset for Driver Drowsiness Detection”, Proceeding of the 31st Conference of Fruct Associatıon, 2022.

[36] R. Ghanem, H. Erbay, and K. Bakour, “Contents-Based Spam Detection on Social Networks Using RoBERTa Embedding and Stacked BLSTM”, SN COMPUT. SCI., c. 4, sy 4, s. 380, 2023.

[37] R. Wang, X. Ji, S. Xu, Y. Tian, S. Jiang, and R. Huang, “An empirical assessment of different word embedding and deep learning models for bug assignment”, *Journal of Systems and Software*, c. 210, s. 111961, 2024.

[38] T. Wang, L. Fu, Y. Zhou, and S. Gao, “Service price forecasting of urban charging infrastructure by using deep stacked CNN- BiGRU network”, Engineering Applications of Artificial Intelligence, c. 116, s. 105445, Kas. 2022.

[39] B. Song, Y. Liu, J. Fang, W. Liu, M. Zhong, and X. Liu, “An optimized CNN-BiLSTM network for bearing fault diagnosis under multiple working conditions with limited training samples”, Neurocomputing, c. 574, s. 127284, 2024.

[40] B. Samia, Z. Soraya, and M. Malika, “Fashion Images Classification using Machine Learning, Deep Learning and Transfer Learning Models”, içinde 2022 7th International Conference on Image and Signal Processing and their Applications (ISPA, ss. 1-5), 2022.

[41] W. Cao, Y. Liu, H. Mei, H. Shang, and Y. Yu, “Short-term district power load self-prediction based on improved XGBoost model”, Engineering Applications of Artificial Intelligence, c. 126, s. 106826, 2023.

[42] A. Maleki, M. Raahemi, and H. Nasiri, “Breast cancer diagnosis from histopathology images using deep neural network and XGBoost”, Biomedical Signal Processing and Control, c. 86, s. 105152, 2023.

.