Performance Enhancement and Analysis of Deep Learning Algorithms using Voting Classifier for Malware Detection

A Major Project Report

Submitted in partial fulfillment of the

Requirements of VIII-Semester for the degree

Of

Bachelor of Technology

in

COMPUTER SCIENCE & ENGINEERING

by

Aditya Singh (Roll No.: 17115004)

Irshad Ali (Roll No.: 17115031)

Milind Kumar Verma (Roll No.: 17115042)

Under the guidance of

Dr. Naresh Kumar Nagwani

Associate Professor

NIT-RAIPUR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY RAIPUR
RAIPUR, CG (INDIA)

APRIL 2021

DECLARATION

We hereby declare that the work described in this report, entitled **"Performance Enhancement and Analysis of Deep Learning Algorithms using Voting Classifier for Malware Detection"** which is being submitted by us in partial fulfillment for the VIII-Semester of the degree of Bachelor of Technology in the Department of **Computer Science and Engineering** to the National Institute of Technology, Raipur is the result of investigations carried out by us under the guidance of **Dr. Naresh Kumar Nagwani (Associate Professor).**

The work is original and has not been submitted for any Degree/Diploma of this or any other Institute/university.

Signature:

Name of the Candidates and Roll No.:

Aditya Singh (Roll No.: 17115004)
Irshad Ali (Roll No.: 17115031)
Milind Kumar Verma (Roll No.: 17115042)

Place: RAIPUR

Date: 15/04/2021

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY RAIPUR

CERTIFICATE

This is to certify that the project entitled **"Performance Enhancement and Analysis of Deep Learning Algorithms using Voting Classifier for Malware Detection"** that is being submitted by Aditya Singh (Roll No.: 17115004), Irshad Ali (Roll No.: 17115031), Milind Kumar Verma (Roll No.: 17115042) in partial fulfillment for the requirements of VIII-Semester of the degree of **Bachelor of Technology in Computer Science & Engineering** to National Institute of Technology, Raipur is a record of bonafide work carried out by them under my guidance and supervision.

The matter presented in this project document has not been submitted by them for the award of any other degree/diploma elsewhere.

Signature of Supervisor

Dr. Naresh Kumar Nagwani

Associate Professor

Department of Computer Science & Engineering

National Institute of Technology, Raipur (C.G)

Project Coordinator

Dr. K. Jairam Naik

Department of C.S.E

NIT Raipur (CG.)

H.O.D.

Dr. Pradeep Singh

Department of C.S.E

NIT Raipur (CG.)

ACKNOWLEDGEMENT

We would like to acknowledge our college **National Institute of Technology, Raipur** for providing a holistic environment that nurtures creativity and research-based activities.

We express our sincere thanks to **Dr. Naresh Kumar Nagwani, Associate Professor CSE Department**, **NIT Raipur,** the supervisor of the project for guiding and correcting throughout the process with attention and care. He has frequently suggested creative ideas and guided us through the major hurdles that occurred during the duration of the project.

We would also thank **Dr. Pradeep Singh, Head of the department,** and all the faculty members without whom this project would be a distant reality. We also extend our heartfelt thanks to my family and friends who supported us.

Thank You!!

Aditya Singh (Roll No.: 17115004)

Irshad Ali (Roll No.: 17115031)

Milind Kumar Verma (Roll No.: 17115042)

ABSTRACT

As the usage of IoT-based android devices has increased exponentially, security and protection from malware have become an inevitable issue. In order to design a malware-protection system, there is a need for techniques to detect malware. The traditional detection methods, based on the signature and permissions have become ineffective with the increased complexity of malware due to techniques like code obfuscation and encryption. Deep learning methods of machine learning are better equipped to deal with such complexities. In this project, an implementation as well as analysis of various deep learning methods has been carried out, including Convolution Neural Networks (CNN), Recurrent Neural Networks (RNN), and Deep Belief Networks (DBN). Consequently, a voting classifier ensemble model has been proposed in order to enhance the performance in terms of malware detection. A comparative analysis of the accuracy of the method is done for two scenarios: the first being the analysis of performance of the model with and without the voting classifier; and the latter being the analysis of performance of these models by making use of the complete feature set versus employing the selected features set (which are selected using the ANOVA feature selection method). As a result, an inference has been drawn that the usage of the voting classifier with selected features dataset, has led to the improvement in the accuracy of the system.

CONTENT

DESCRIPTION	PAGE NO.
DECLARATION	ii
CERTIFICATE	iii
ACKNOWLEDGEMENT	iv
ABSTRACT	v
CONTENTS	vi
LIST OF FIGURES	vii
LIST OF TABLES	ix
1. INTRODUCTION	1
1.1 Overview	1
1.2 Deep Learning in Android Malware Detection	2
1.3 Significance & Objectives	5
1.4 Scope & Motivation	6
1.5 Organiztion of the Project	7
2 LITERATURE REVIEW	8
2.1 Existing Work	8
2.2 Malware-Detection Approaches	9
2.3 Summary of Literature	15
3. PROPOSED METHODOLOGY	19
3.1 Overview	19
3.2 Convolutional Neural Network Model	20
3.3 Recurrent Neural Network Model	21
3.4 Deep Belief Network Model	23
3.5 Voting classifier	24
4. EXPERIMENT	27
4.1 Evaluation Metrics and Datasets	27
4.2 Code Snippets	30
4.3 Result	32
4.4 Performance Evaluation	35
5. CONCLUSION	41
6. FUTURE SCOPE	41
REFERENCES	42

LIST OF FIGURES

Figure. No.	Title	Page No.
1	System model for Android Malware Detection	3
2	CNN methodology employing APK permissions	4
3	Taxonomy of malware detection approaches	9
4	Structure of the malware detection process	10
5	Signature based Malware identification	12
6	Behavior based Malware identification	14
7	Summary of Analysis Approaches	15
8	CNN Model	21
9	RNN Model	22
10	DBN Model	23
11	Voting Classifier Model	25
12	Pseudo-code for the implementation of the proposed system	26
13	Code Snippet representing feature selection	30
14	Code Snippet representing implementation of CNN model	30
15	Code Snippet representing implementation of RNN model	30

16	Code Snippet representing implementation of DBN model	31
17	Code Snippet representing implementation of Max Voting classifier model	31
18	Confusion Matrix for the proposed system	32
19	Performance statistics of CNN model implementation	32
20	Performance statistics of RNN model implementation	33
21	Performance statistics of DBN model implementation	33
22	Performance statistics of Max Voting classifier model implementation	34
23	F-value of the features	35
24	Comparative analysis of Training Accuracy achieved by different models	37
25	Comparative analysis of Validation Accuracy achieved by different models	37
26	Comparative analysis of Precision scores achieved by different models	39
27	Comparative analysis of Recall scores achieved by different models	39
28	Comparative analysis of F-scores achieved by different models	40

LIST OF TABLES

Table. No.	Title	Page No.
1	Table summarizing the existing works	16
2	Accuracy of the models using all features	36
3	Accuracy of the models using selected features	36
4	Evaluation metrics for all models with all features	38
5	Evaluation metrics for all models with selected features	38

1. INTRODUCTION

1.1 OVERVIEW

With the rapid development of communication technology, miniaturized control, and the internet, the field of artificial intelligence has flourished and its use in various aspects of the industry has increased. Internet of Things or IoT [1] has recently become very popular due to its affordable and portable characteristics. With its widespread use, the security and privacy offered by them are also a big concern. Android-based devices have become vulnerable targets for malware attacks due to the versatility of the platform. Statistically, it has been found that more than 20 billion devices that serve in the domain of the Internet of Things such as sensors, etc. have been reported to be connected to the Internet [2]. Besides, the expected amount will rise to 50.1 billion in 2020. With an increase in such devices, the security of these devices has become a major issue that needs to be addressed since they have various security issues such as backdoors, weak passwords, etc.

Malware attacks have been used as a very common way to commit cyber attacks over the decade, and with development in technology, an increase in the population of smartphones and smart-devices users has been recorded, who store private and confidential information in these devices, which has made them an important target for developers of malware attacks. Various software dedicated to virus and malware protection such as Kaspersky antivirus, McAfee, antivirus, etc. have developed methods for malware detection but the performance they exhibit is not satisfactory. Malware detection and malware analysis tools are essential in the prevention of the spread of malware.

For malware analysis, two general approaches have been employed: dynamic malware analysis and static analysis of malware. To find suspicious patterns in the APKs, static analysis is focused on the study and evaluation of source code and binaries. In dynamic analysis, it needs the execution of the APK in an isolated environment of the analyzed program while tracking and tracing its behavior to detect any malicious behavior.

Another approach called hybrid detection is the amalgamation of both detection techniques. The aggregation of signature libraries and human interaction by malware researchers are primarily focused on conventional malware detection techniques, and hence it has been difficult for the conventional techniques, to match with the exponential growth over the decade in the Android malware.

Malicious software, also known as "malware," such as worms and viruses, has become a long-standing and growing concern. This influence can only grow as society grows more reliant on computing technology. Single injuries cost businesses billions of dollars in settlements on a daily basis. As shown by the explosive growth of modern malware, malware has proved to be successful for its developers. The requirement for better resources to stop ransomware and assist researchers and protection experts is only going to intensify when malware becomes more prevalent.

1.2 DEEP LEARNING IN ANDROID MALWARE DETECTION

Applications with the apk extension can be found on Android smartphones. Aside from certain developer-specified parameters including programme versions and package names, some information like camera access, SMS sending, microphone access, picture album access, vibration, internet access, and so on is controlled by the user's consent. This information is prepared by the developer and saved in AndroidManifest.xml in each apk package. Permissions are registered by installing apps from the market, but consumers typically download the app they choose to use by granting consent whether they don't think about the authorization data or don't know what they are. As a result of this scenario, ransomware was installed on the computer, and sensitive information was accessed with the user's permission.

It is impossible to avoid malware from infecting a computer by determining if apk files are harmful when it is running. As a result, malware identification is done automatically in this study using a static analysis approach rather than running the programme.

Fig. 1 depicts the system model. The apk files in the AndroTracker dataset are transferred into an ApkReader open source repository in the framework model, and the authorization data from these files is retrieved.

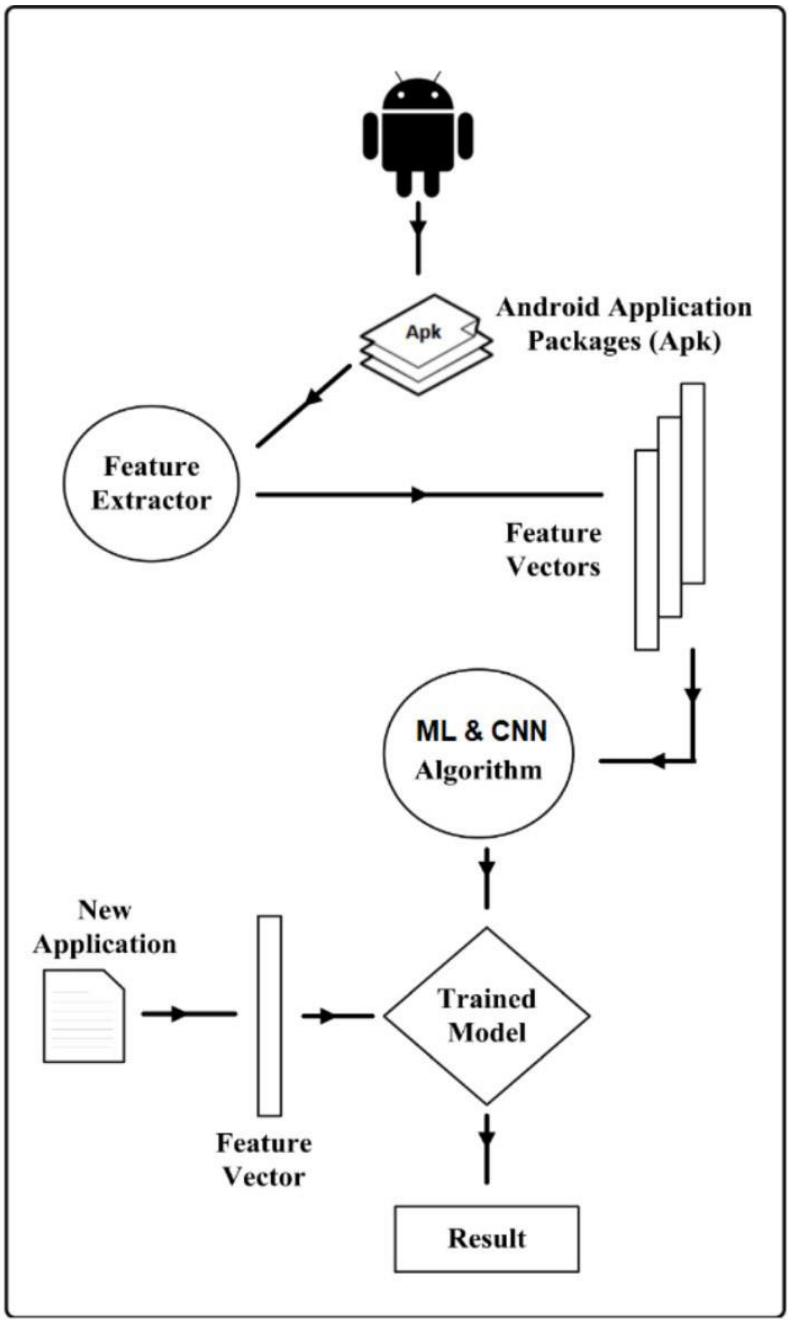


Fig. 1: System model for Android Malware Detection [3]

The AndroidManifest.xml file in each file is accessed by this library, which lists the necessary permissions for these files. To build a new data collection, this consent data is incorporated into the functionality of harmful or benevolent applications. The authorization data requested by the application is allocated to 1 when the function value is passed to the data collection, while those that have been not requested are assigned to 0. The latest data collection, which consists of feature vectors, is then subjected to machine learning models and CNN.

Convolutional Neural Network (CNN): A convolutional neural network (CNN) is a deep learning architecture focused on artificial neural networks. Artificial neural networks and CNN vary in that CNN can accommodate far more hidden layers than ANN.

When artificial neural networks have more than five levels, the method gets complex and slows down, and the problem becomes unsolvable. Many architectures, such as Concentration Boltzmann Machines, automated coders, and Convolutional Neural Networks, are used in the Deep learning process, which has many nodes and parameters. Since the image's Convolutional Neural Networks are one of the most common deep learning techniques, this approach is often tested on two-dimensional signal data. Each layer in convolutional neural networks with several layers that can be trained has three layers: the feature pooling layer, the filter bank layer, and the non-linear layer [4]. Fig. 2 shows the convolutional neural network architecture developed for malware detection.

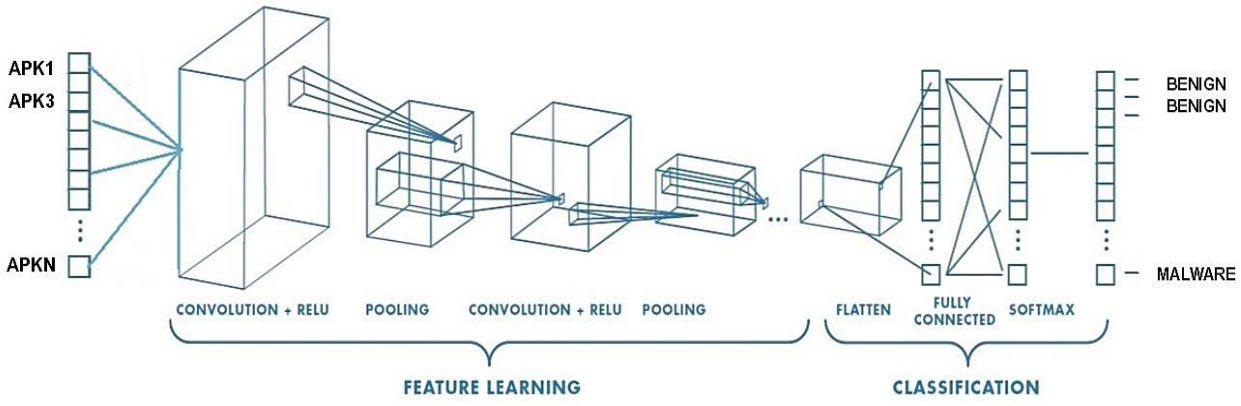


Fig. 2: CNN methodology employing APK permissions [5]

1.3 SIGNIFICANCE & OBJECTIVES

Malicious software recognition is a challenging task to solve, and machine learning techniques have been used to solve it for decades. However, progress has been sluggish, in part due to a range of specific challenges with the mission that arise during each step of the machine learning system development process: data collection, naming, function design and development, model identification, and assessment. This project will go through data collection, function extraction, model creation, and assessment in this survey, as well as some of the latest approaches and difficulties of malware classification.

The discussion made in this report will provide considerations for the limitations that must be understood for machine learning-based solutions in this domain, as well as problems that have yet to be addressed for which artificial intelligence could provide a solution.

This survey is intended to be helpful to both cybersecurity professionals who want to learn something about how deep learning could be implemented to the malware issue and data scientists who want to get a better understanding of the problems in this particular field.

The main contributions of the project work are as follows:

i. Implementing and comparing deep learning techniques such as CNN, RNN, and DBN for malware detection and then combine them using a voting classifier ensemble.

ii. The performance of these models using the complete feature set and selected features set, which are selected using the ANOVA feature selection method have also been compared.

1.4 SCOPE & MOTIVATION

- **Scope**

Similar to other classification problems, malware detection using Voting classifier model intends to divide objects into various sets on the basis of their characteristic similarities and structure. Nevertheless, In unconstrained methods, there is a vast range of real-world problems and the scenarios might be very distinct such that commonly used classification algorithms are not sufficient for handling those situations. Primarily, certain distribution assumptions of data form the basis of a lot of clustering methodologies. For instance, the k-means algorithm presumes that the data points of various classes are near to the centroids of the classes.

The main scope of this project is to enact the concept of Voting classifier on the basis of the score of performance metrices between the objects present in the feature space. For obtaining optimized results, each instance of a pivot object can be directly connected with its corresponding k-nearest neighbors having the same identity.

- **Motivation**

The motivation behind this project entirely rounds about the fact that in order to generate decent deep learning model classification results, the computation of the Voting classifier amongst an instance as well as its k-nearest neighbors is both necessary and sufficient, similar to a few important primary features in the case of clustering, which are commonly used in various desktop/online photo albums for photo management are grouping, tagging faces in these albums, organizing a large volume of face image/video collections for fast retrieval in strict time-sensitive scenarios such as forensic investigations [6], and automatically data cleaning/labeling for generating large-scale datasets. Moreover, in the proposed work, an attempt has been made in order to show the improvement in the performance of the model after removing the noisy data from the data-set. Results have shown great improvement after mitigating the noises in the sample space.

1.5 ORGANIZATION OF THE PROJECT

Malware recognition and classification is one field where there is need for change. Malware labeling is a long-studied process that entails one of two tasks: 1) identifying new malware (i.e., differentiating between innocuous and harmful applications) or 2) identifying between two or more recognized malware forms or families. The first of these is known as malware identification, and it is extremely helpful in preventing malware from spreading. Anti-virus (AV) software actually uses a signature-based technique to execute this purpose. Signatures are inherently unique to the threats they track, and creating them may be time-consuming for an expert. With an increase in data, the complexity of computations increases, and hence it is required to switch to solutions that can handle these complexities, for this the concept of Machine learning comes in handy. Machine learning which is being widely popular due to development in the computing power of devices, these methods offer another viewpoint for effective and automated Android malware detection [7].

Various methods based on machine learning and deep learning primarily consist of a series of predefined steps: Firstly, various types of Android applications are collected as in their APK's, and then the features from these to classify the Android apps are extracted. Then, models of machine learning are trained and tested to detect malware. Fourth, by predicting test samples, the performance of the qualified models is assessed, for example, In a machine learning-based static detection method, it is required to have a a specific method for de-compilation for the parsing of binary files, and additionally, various rules are defined as well to distinguish useful information from garbage. However, in dynamic detection, the behavioral features such as system function calls, and network communications, etc, the target application needs to be performed in complete isolation.

To make the cost of feature engineering and implement automatic feature learning, the researchers are now focusing on developing such methods to remove expert interference. The project report is further organized into different chapters: Chapter 2 gives the literature survey, Chapter 3, gives the background of the related topics, Chapter 4, gives the methodology for the paper, Chapter 5 provides the results and finally, a conclude has been drawn regarding the project in Chapter 6.

2. LITERATURE REVIEW

2.1 EXISTING WORK

Malware detection was formerly done using the traditional methods, which are no longer used due to the increase in strength of the malware. Conventional machine learning techniques and algorithms typically have fewer than three layers of computing units, with reduced computation power to process raw data. These methods, on the other hand, use a set of machine learning models to construct a prediction model based on derived features from an Android application package (APK). As a result, the success of machine learning techniques is highly reliant on the extracted features, but these pre-defined function listings are expected of becoming outdated as the Android operating system evolves and software innovation advances. Furthermore, malware authors strive to improve their deception methods in order to exploit consumers and businesses by circumventing security tools and well-trained machine learning models. Due to the growing difficulty of detecting Android malware using conventional machine learning methods, it is challenging to develop a reliable and straightforward detection model or framework. The three categories into which the traditional methods of malware detection can be classified are static and dynamic detection.

Static detection methods such as permission-based malware detection, bytecode-based malware detection methods, signature-based detection methods, and hybrid-based method for the detection, check the static APK so that they can identify the malicious features before without actually executing the application.

Assessing malware remotely, by articulating corresponding guidelines and reviewing the habits and source code of suspect Android Apps, is indeed a time-consuming procedure that cannot be generalized to a huge number of Android apps. Furthermore, manual malware detection did not keep up with the evolving attack tactics due to the constantly changing malware techniques. In recent years, a significant amount of research work on automated Android malware identification has been proposed, with data mining and deep learning approaches yielding saliently promising detection outcomes and results.

2.2 MALWARE-DETECTION APPROACHES

The knowledge of obscure malware security is a fundamental topic in malware identification according to machine learning techniques [8] as a consequence of the emerging malware in innovation. There are two types of machine learning strategies: guided and unsupervised. Malware detection strategies are classified into two categories: behavior-based and signature-based methods [9]. In addition, there are two types of malware detection: static [10] and dynamic [11] malware analysis that are often used to detect malicious apps.

A malware identification taxonomy focused on machine learning methods has been displayed in Fig. 3. The API calls features, assembly features, and binary features are examples of current malware detection methods, as seen in this diagram. Machine learning algorithms are used to simulate and identify harmful files in these features.

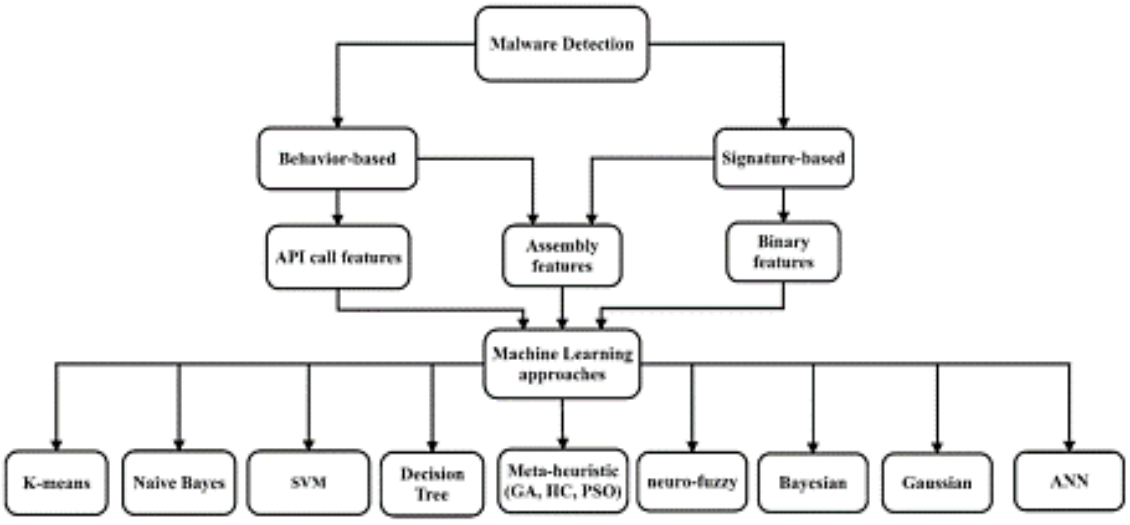


Fig. 3: Taxonomy of malware detection approaches [12]

A series of instructions specific to a malware is used to produce a malware signature, that is obtained by experts in a laboratory setting of signature-based techniques (Goertzel, 2009).

Any malware that exhibits the malicious behaviour defined by the signature must be able to be identified by the signature. The majority of antivirus scanners depend on signatures. The analysis of the actions of proven and alleged malicious code is the subject of behavior-based identification techniques. The malware's source and destination addresses, the attachment forms through which they are embedded, and statistical irregularities in malware-infected networks are all examples of such activities (Goertzel, 2009). Symantec's proprietary histogram-based malicious code identification system is a good example of such a behavior-based detection strategy.

Malware Detection is divided into two methods: Signature-Based and Behavior-Based techniques and each technique can be applied using static analysis or dynamic analysis or hybrid analysis (Idika and Mathur, 2007), Fig. 4 shows the organization of malware detection.

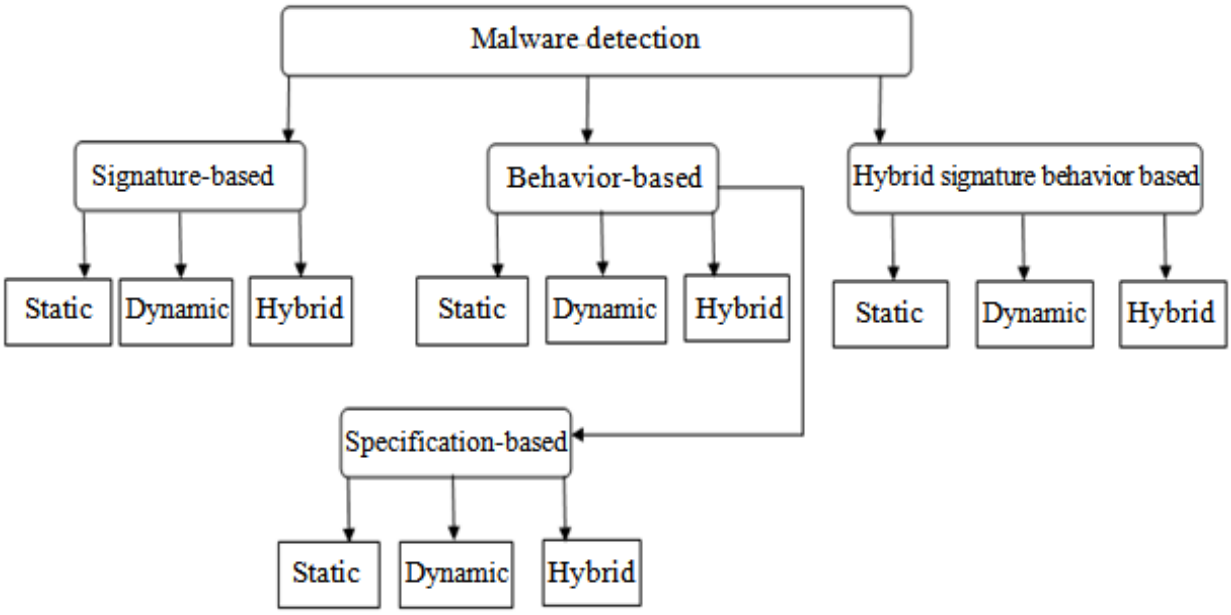


Fig. 4: Structure of the malware detection process [13]

Robust accurate signatures are generated based on semantic patterns or the distinct series of characters in the source file of the APK through signature-based methods. Using a predefined set of documented attacks, the signature-based method finds interruptions [12].

Even though the discussed method has the potential to recognize malware in a versatile program, the predefined signature database needs to be continuously overhauled. Besides, because of the constantly evolving existence of portable malware, it turns out to be not very effective for the detection of virulent activities. On account of their efficiency, commercial anti-malware products have widely used such methods.

- **Signature based malware identification**

  Signature-based identification, which emphasises exact similarity, has recently been the most widely used technique in antivirus programming. Instead of using element behavioural approaches, malware detection has mostly relied on static investigations that examine the code-structure label of infections [14]. Using a predefined array of documented attacks, the signature-based device detects interruptions. Regardless of the fact that such a scheme will detect malware in mobile applications, it necessitates regular updates to the pre-specified signature database. Furthermore, because of the rapidly evolving existence of portable malware, signature-based techniques are less successful in detecting noxious exercises [15, 16]. Signature-based techniques depend on outstanding raw byte instances or standard articulations, referred to as signs, made to arrange the noxious text. Static characteristics of a record, for example, are used to determine if it is malware. The biggest benefit of signature-based approaches is their thoroughness, since they cover all possible document execution scenarios.

  Current malicious artefacts inside the malware structure have features that can be used to create a special digital signature. The anti-malware supplier employs meta-heuristic algorithms to search the malicious entity effectively and monitor its signature [17]. The observed signature is applied to the current archive as known malware after the malicious entity is identified. A large range of signatures that identify harmful items are used in the database sources. There are many characteristics of signature-based malware detection, including fast recognition, ease of use, and general accessibility [18].

Since the digital signature proposals are derived from well-known malware, they are therefore well-known. As a result, programmers can easily avoid them by using simple chaos procedures. As a result, malicious code may be altered, and signature-based detection can be avoided. Anti-malware vendors can't discern unknown malware or even versions of recognised malware because they're designed on the basis of known malware. As a result, they are unable to differentiate polymorphic malware without an identical digital signature. In this vein, signature-based recognition does not have zero-day security. Furthermore, since a signature-based predictor uses an isolate signature for each malware variant, the signature database grows exponentially [19]. Assembly features and binary features are the two major mechanisms for implementing malware identification approaches in deep learning methods in signature-based malware detection. A typical signature-based malware identification architecture utilising data mining techniques as seen in Fig. 5.

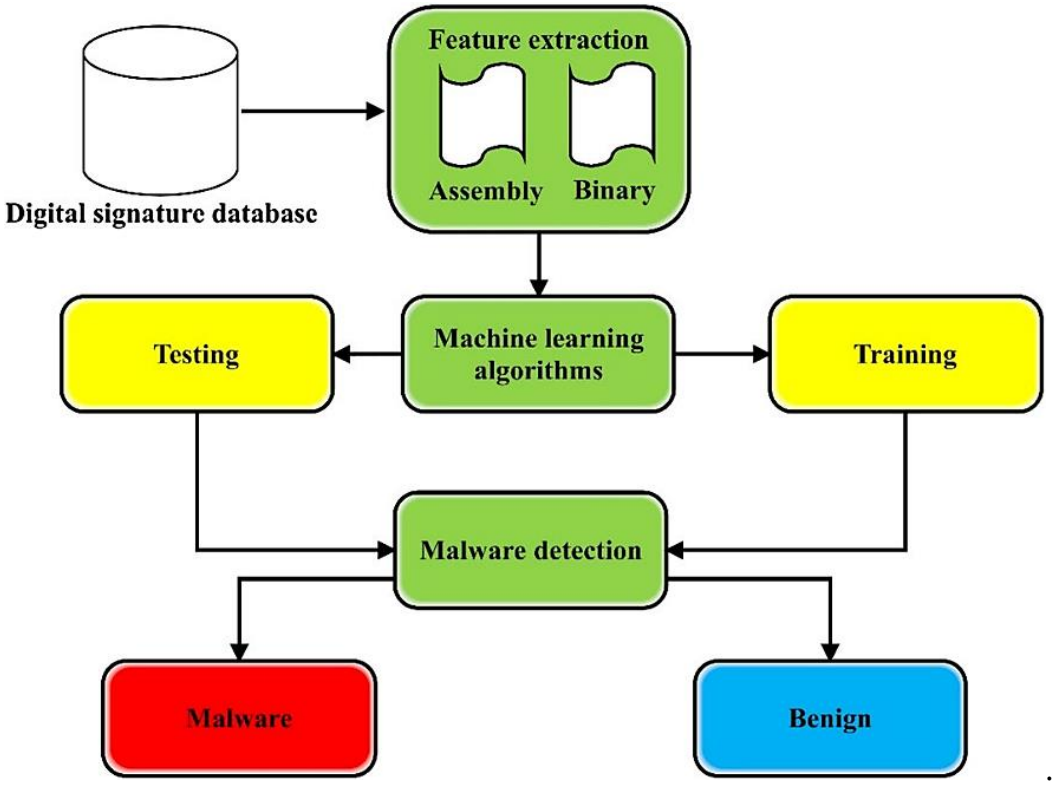


Fig. 5: Signature based Malware identification [12]

- **Behavior based malware identification**

In a behavior-based malware detection approach, The applications which are suspected to be malware are analyzed based on their behavior in an isolated environment. Efforts to accomplish distinctly suspicious or unofficial actions will specify that the alleged object is malicious, or at least apprehensive. By the usage of some complex analysis the harmful properties, are identified [13]. The first security on the Android platform against malware is permissions. Software author must declare the permissions in the AndroidManifest.xml file. Hence, the permission-based approach mainly analyses suspicious permissions to identify potential malware [20]. Extraction of the bytecode file directly from the Android APK file and transforming the bytecode file into a two-dimensional bytecode matrix can also be used to identify malware in this method, CNN will automatically learn bytecode file characteristics that can be used to detect malware [21]. A hybrid static analyzer extracts different kinds of features like FlowDroid to further improve the accuracy of detection [22]. Static analysis methods are relatively fast and less complex but are not effective when dealing with malware that uses encryption and code complications and better detection efficiency can be accomplished by using the dynamic features in conjugation with static features for the detection of malware as it will result in better efficiency as well as speed.

Behavior-based methodologies demand that a specified example be run in a sandboxed environment, with run-time exercises being tested and registered. To conduct malware and erase its practises, dynamic investigative programmes use virtualization and imitating environments. The primary benefit of the behavior-based methodology is that it has a better understanding of how malware is created and deployed [23, 24].

The suspicious artefacts are judged based on their actions that they cannot perform in the machine in the behavior-based malware strategy. Attempts to achieve explicitly irregular or unofficial actions would indicate that the suspected object is deceptive, or at the very least suspicious. A dynamic analysis that tests malicious purpose from the object's code and layout is used to identify malicious actions.

API calls and assembly functions are two major approaches for implementing machine learning algorithms in behavior-based detection. Fig. 6 shows how data mining algorithms are used to identify malware based on its actions.

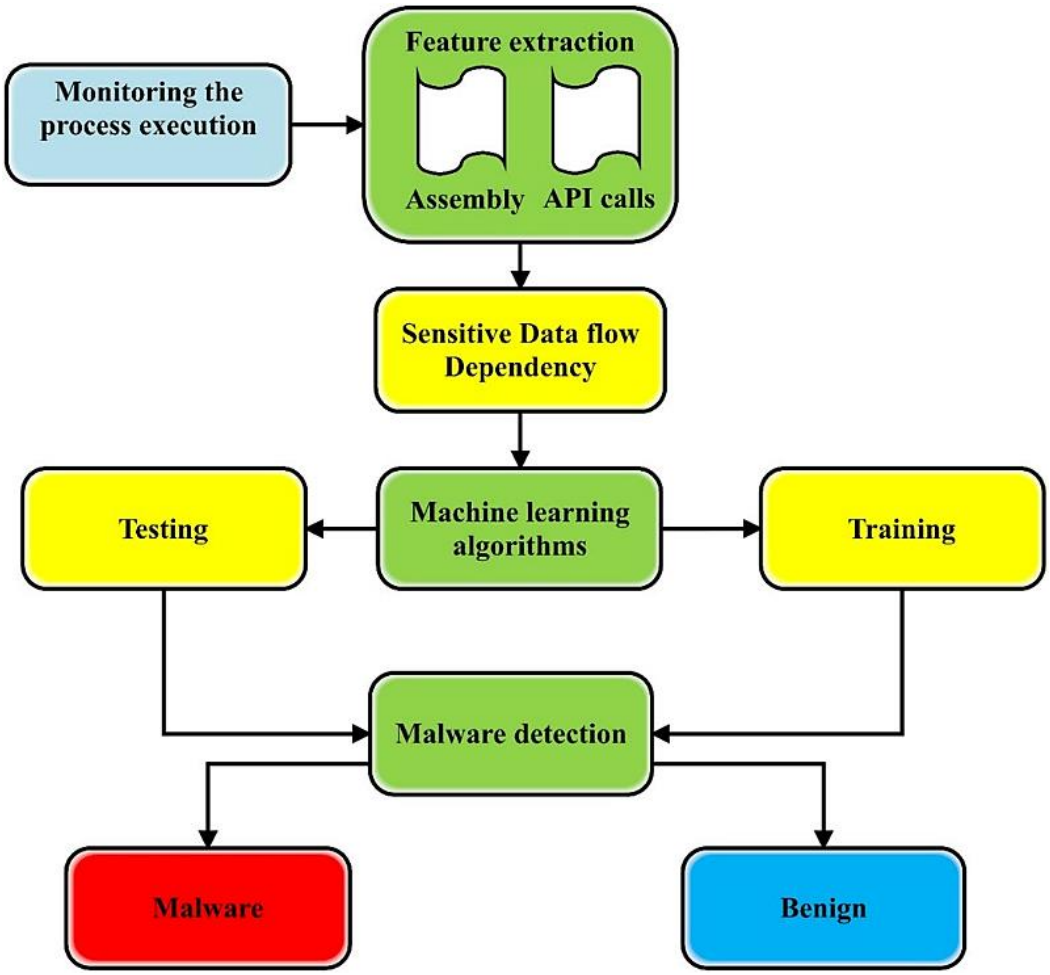
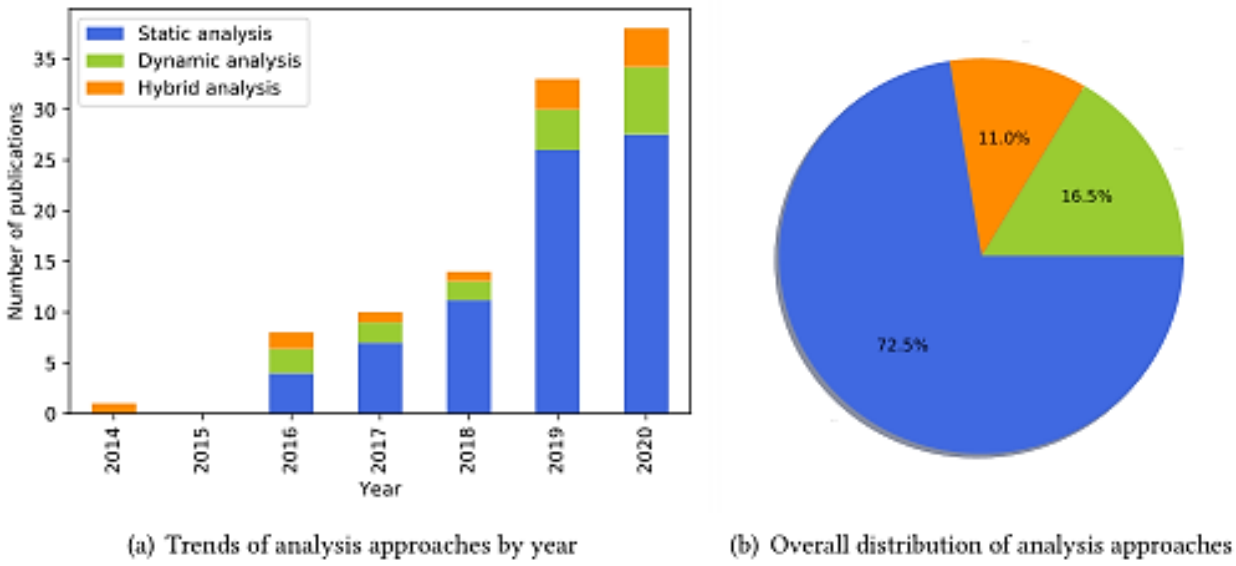


Fig. 6: Behavior based Malware identification [12]

The implementation of different machine learning algorithms in behavior-based detection is by using two main features, The System calls and assembly features [9]. Standard machine learning models such as random forest [25] and multiple deep learning models such as Convolutional Neural network and its variants [26], Recurrent Neural Networks [27] are widely used in the analysis of Android malware in terms of model selection.

2.3 SUMMARY OF LITERATURE

Since the Android APK file is a binary executable file, it must be transformed into a formalised representation consistent with Deep Learning models, normally in vectorized form, before being fed into DL models. Typically, analysis studies process and interpret APKs first, then perform additional operations on the data derived from applications. The analysis can be classified into static analysis, dynamic analysis, and hybrid analysis depending on the path to software analysis. After analysis, attribute encoding methods are used to translate the processed file or collected data information into formalised formats, which are then fed into Deep Learning models. The APK characterization approaches in the collected studies are summarised in Fig. 7, and this section provides an outline of the research methods and attribute encoding approaches used in the examined work.



(a) Trends of analysis approaches by year (b) Overall distribution of analysis approaches

Fig. 7. Summary of Analysis Approaches [28]

As to summarize the literatgure survey, the above-mentioned methods are studied and some of them are implemented. Their advantages and disadvantages have also been discussed and analyzed. The properties and performance of various marlware detection frameworks are also studied and discussed. The study can be summarized as shown in Table 1.

Table 1: Table summarizing the existing works

S. no.	Paper Title	Authors	Year	Method used	Limitation
1	Recurrent Neural Network for Detecting Malware [28]	Hoang, V. L., Taniar, D., Prashar, D. and S. Jha	2020	Recurrent Neural Network	Inappropriate for samples of external events, Absence of analysis of accuracy conditions and feature selection, High complexity, Failed to analyze unmatched files.
2	Android Malware Detection based upon Bytecode Image [22]	Ding, Y., Zhang, X., and Hu, J.	2020	Bytecode file of an application, imaging it using CNN	Fails to work for higher number of malware variants, high latency is invoved in execution.

3	A State-of-the-Art Survey of Malware Detection approaches using Data Mining Techniques [12]	Souri, A. and R. Hosseini	2018	Data mining techniques Signature-based and Behaviour-based techniques	Run-time overhead, Unable to contemplate dynamic analysis of Android apps, Dataset over-fitting
4	An Inside Look at IoT Malware [2]	Wang, A, Liang, R., Liu, X., Zhang, Y., Chen, K. and Li, J.	2017	IoT Malware Analysis based on Exploiting Vulnerabilities	Due to the amount of specific difficulties and expenses involved with processing binary data, the data collection and processing phases are critical to all other stages of the machine learning phase.
5	CNN-Based Android Malware Detection [27]	Ganesh, M, Pednekar, P., Prabhuswamy, P., Nair, D. S., Park, Y. and Jeon, H.	2017	Convolutional Neural Network	Conflict amongst classifiers due to large number of datasets, Lower accuracy & efficiency than malware-behavior heuristic monitoring methods, Inapt malicious payload.

6	HybriDroid: Static Analysis Framework for Android Hybrid Applications [23]	Lee, S., J. Dolby, and S. Ryu	2016	Static analysis for Android Malware Detection	Datasets suffer shifts in data distribution across different domains resulting in performance degradation.
7	Permission-based Android Malware Detection [21]	Zarni Aung, Win Zaw	2013	K-Mean clustering model, Classification	Problem of class supremacy, the problem of forced assignment, and the problem of is no class (It has been discovered that no individual method can have greater precision and recall).
8	Malware Detection based on Hybrid Signature behavior Application Programming Interface call graph [13]	Elhadi, A.A., M.A. Maarof, and A.H. Osman	2012	Static and Dynamic analysis techniques, Signature-Based and Behaviour-Based techniques.	Complex construction, management and miniminzing/filtering of graphs, Unable to filter out certain threats and malicious software and hence, less accurate.

3. PROPOSED METHODOLOGY

3.1 OVERVIEW

In this project, the dataset is used and executed through three different deep learning algorithms and with the use of a voting classifier determine the most suitable algorithm for malware detection and compare the results, first with a dataset including all features and again with the top selected features. Fig. 13 represents the implementation of feature-selection process.

In the project, the Android Malware dataset has been employed, namely, CICMalDroid 2020, which consists of mainly 5 types of labels, Adware, SMS malware, Riskware, Banking Malware, and Benign. The .csv system log features have been used as an input for the models. The dataset is split into a training set constituting 70% of the dataset and a test set constituting 30% of the dataset.

As the dataset contains a wide range of values it is important to convert them into a common scale without distortion of differences. To achieve this, preprocessing on the data is done to train the model more efficiently. The z-score normalization method has been used for preprocessing, in this method mean (μ) and standard deviation (σ) of the imported data are calculated, and using the following equation 1, the normalized dataset is obtained.

$$x_{norm} = \frac{x-\mu}{\sigma} \quad (1)$$

F- value is the statistical term used to compare the features with the target value and determine which is fits the model the best. It is given by the equation 2:

$$F\ value = \frac{\text{variance of the group means}}{\text{mean of the within group variances}} \quad (2)$$

This obtained normalized data is then used as an input for the training and testing of the proposed models, and using the ANOVA method which uses the f-score of the features., with *selectkbest(),* and *f1_classif()* functions the top features are selected, which have a positive f value.

3.2 CONVOLUTIONAL NEURAL NETWORK MODEL

A convolutional neural network or CNN is a branch of the deep learning methods. These are multi-layer neural networks and are also classified as space invariant artificial neural networks or shift-invariant, referring to their shared-weight design and translation invariance of the network. A CNN's design is inspired by that of a Neuron in the Human Brain's connectivity pattern.

An input layer, hidden layers, and an output layer form a fully convolutional neural network. For the construction of a feed-forward neural network, the activation function and final convolution mask their inputs and outputs, and the middle layers, if they exist, are considered secret. The hidden layers in a CNN contain the layer that usually executes convolutions in the neural network and ReLU is generally the activation feature used in them. In addition to this various other layers of convolution, such as pooling layers, completely associated layers, and layers of normalization are also included in the hidden layers. The training of these layers is done by dividing the dataset into training and testing sets. The breakthrough of the convolution operation of a neural network is that weights are the values of the filter to be acquired through network preparation. The network can learn from the data what kinds of characteristics to extract. In particular, the network is required to learn to remove features from the picture during stochastic gradient descent training to reduce losses for the unique task equipped to solve the network. The structure of a typical CNN Network can be seen in Fig. 8.

The designed CNN Model consists of different layers, consisting of Conv1D(), Dense() and Flatten() layer, classifies the outputs into five categories. The normalized data is given as an input to the model defined as in Fig. 9 with activation function as 'relu'. The implementation of the CNN model can be referred from the code snippet shown in Fig. 14.

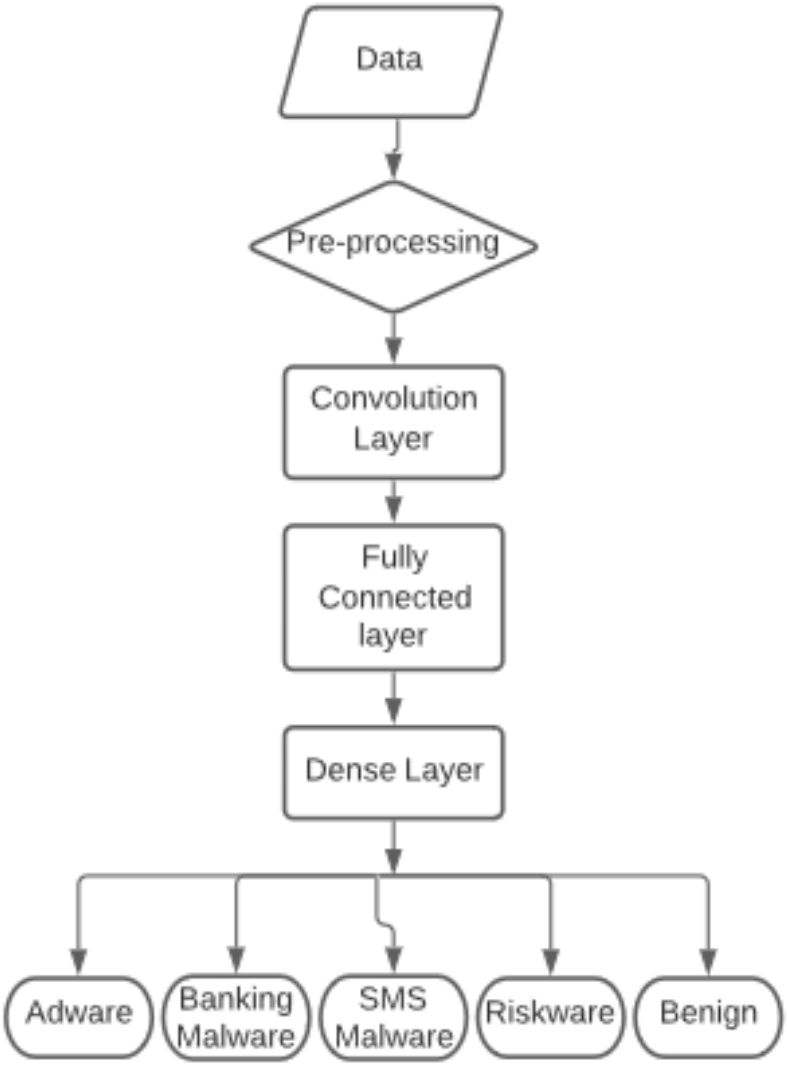


Fig. 8: CNN Model

3.3 RECURRENT NEURAL NETWORK MODEL

A recurrent neural network or RNN is a multilayer perceptron network which is a subtype of deep learning neural network where a guided graph along a temporal sequence forms ties between nodes. This enables temporal complex actions to be displayed. The network is presented with a single time step of the input, then measure its current state using the current input set and the previous states After repetition of the steps according to the requirement of the problem information accumulated by the previous states is joined and once all the time steps are completed, the final current state is utilized in order to compute the output. The output obtained is then

compared to the target/required output and the generated error value is backpropagated to adjust the weights of the RNN layers for training and better accuracy.

One RNN architecture that is used is obtained by using the layers of long short-term memory or LSTM. LSTM has feedback links, unlike normal feedforward neural networks. It is not only responsible for the processing of the individual data values but also the processing of complete data sequences. The structure of a typical RNN Network can be seen in Fig. 9. The designed RNN Model consists of different layers, consisting of LSTM(), Dense() and Flatten() layer, classifies the outputs into five categories. The normalized data is given as an input to the model defined as in the Fig. 10 with activation function as 'relu'. Fig. 15 shows the implementation of RNN model.

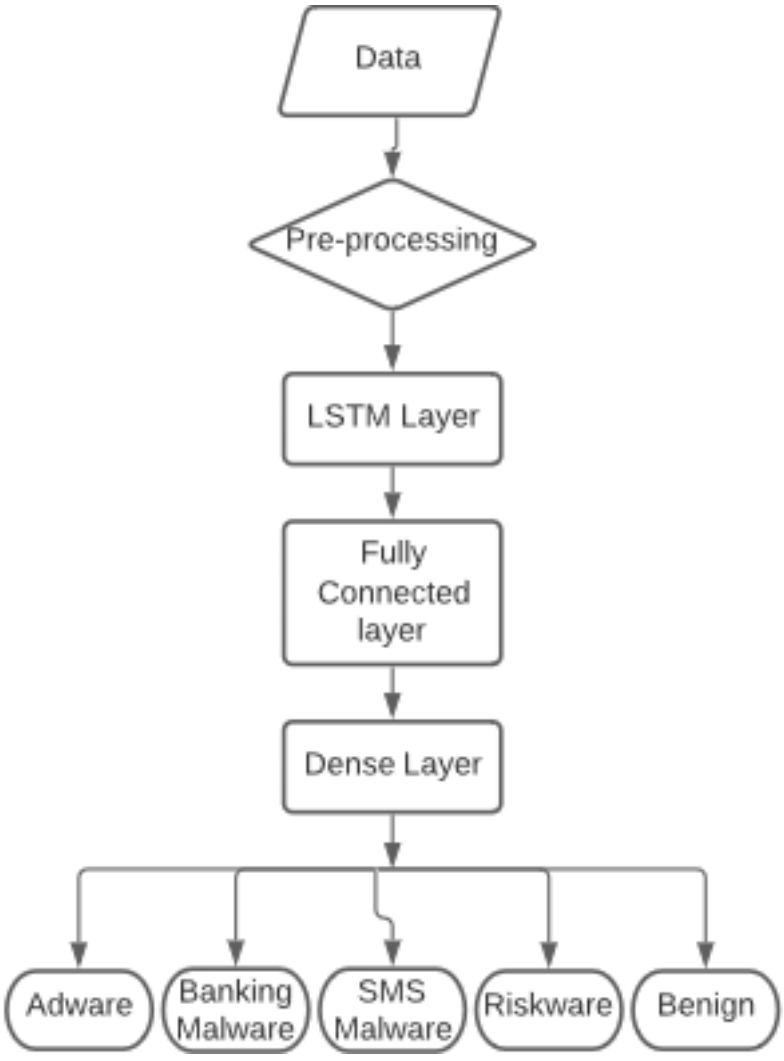


Fig. 9: RNN Model

3.4 DEEP BELIEF NETWORK MODEL

One of the deep neural network approaches, a deep belief network or DBN is a compositional graphical model, consisting of several layers of latent variables, with relations between all the layers isn't between units across each layer. Deep belief networks are algorithms that generate outputs using probabilities and unsupervised learning. They are composed of binary latent variables and both undirected and guided layers are contained.

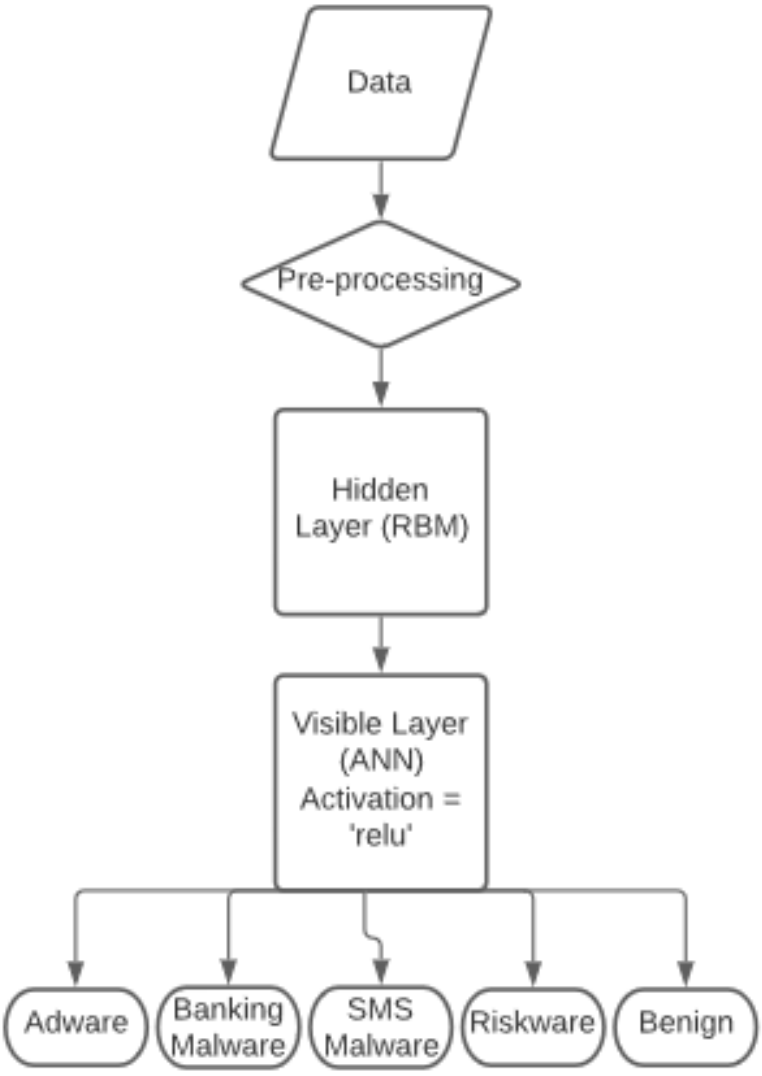


Fig. 10: DBN Model

A DBN is capable to learn to reconstruct its inputs based on probabilities when it is trained on a dataset without supervision. These are the composition of various simple, unsupervised networks such as the autoencoders or the limited Boltzmann machines (RBMs). An RBM with a "visible" input layer and hidden layers with connections between but not inside layers is an undirected, generative energy-based model. This design of layers results in a fast and unmonitored process of training. It can be found that DBNs can be greedily trained, layer-by-layer, and this results in one of the first successful algorithms for deep learning. To pre-train deep belief networks, greedy learning algorithms are used. A typical DBN network can be shown in Fig. 10.

In the designed Deep Belief Network the Restricted Boltzmann Machines (RBMs) have been employed in which the input normalized data is given and the hidden layer serves as the visible layer for the next layer. The normalized data is given as an input to the model defined as in the Fig. 11 with activation function as 'relu' and the output is classified into five categories. The implementation of the DBN model has been shown in Fig. 16.

3.5 VOTING CLASSIFIER

A Voting Classifier is a model in which an ensemble of multiple models are trained and the output of it is based on the highest likelihood of the selected class as the output. It collects the outcomes of each classifier model passed into it and forecasts the output class based on the highest majority of votes. It focuses on a single model is built that trains an ensemble of the models rather than individual dedicated models and then present the predicted output based on the majority results in the votes.

$$Final\ predication = mode(Pred_1, Pred_2 \ldots, Pred_i) \quad (3)$$

where, $Pred_i$ = ith model prediction.

For the voting classifier, all the models have been ensembled together, the CNN model, RNN model, and DBN model in a series, and training the model.

The final prediction for every instance of the testing dataset is done by finding the mode of the three predicted outputs and the output recurring the most is selected by the voting classifier. The architecture can be seen in Fig. 11.

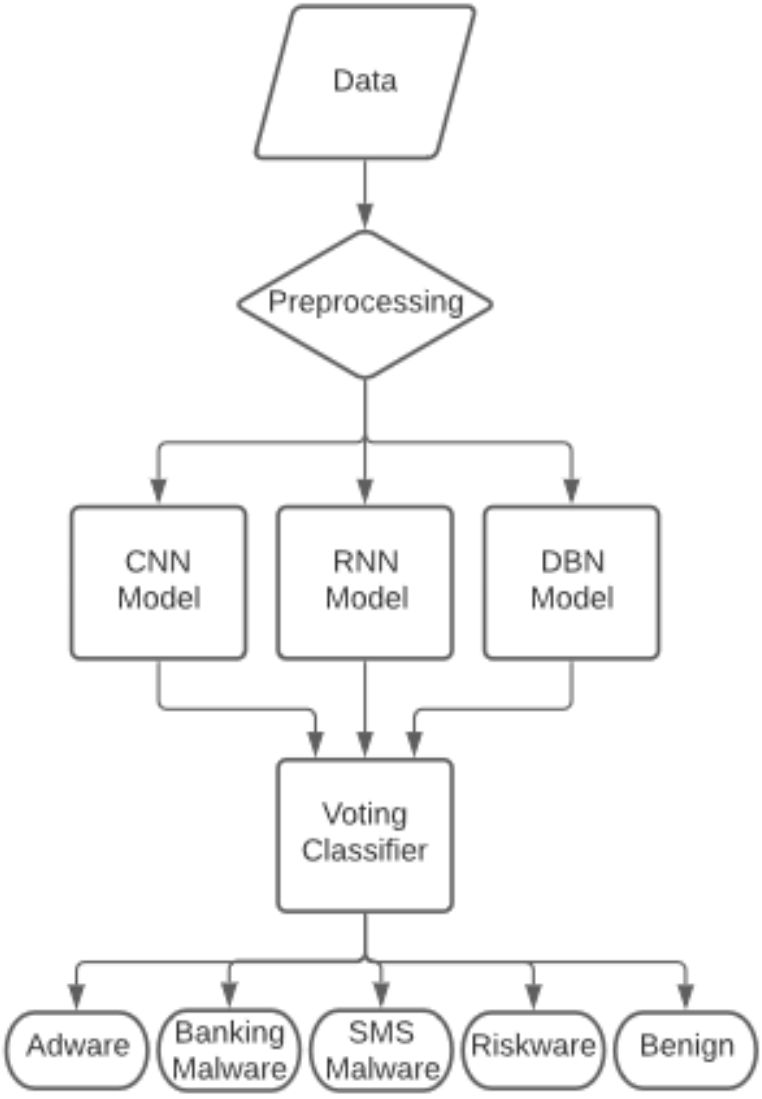


Fig. 11: Voting Classifier Model

The proposed system takes normalized data along with the standard variance and mean as input, which is followed by feature selection on the input data. Thereafter, the dataset is fed to all the three models, namely CNN, RNN and DBN model. The final prediction is the mode of the the predictions made by the above three models. Fig. 12 represents the pseudo-code for the implememntation of the proposed system.

```
Input data

data1 = normalize data, standard variance, mean

feature selection on data = SelectKBest(score_func =
f_classif, k = 98)

Select and split dataset, x_train, x_test, y_train, y_test

Define CNN model1

Add Conv1D 200 layers, activation = 'relu', Dropout of 0.3,
100 Dense layer, activation = 'relu', Flatten(),6 Dense
layer,activation = 'relu'.

Compile with loss =
keras.losses.SparseCategoricalCrossentropy, optimizer =
'adam', accuracy metrics

Fit model, x_train, x_test, y_train, y_test, epochs = 120,
batch_size=128

Define RNN model2

Add LSTM 200 layers, activation = 'relu', Dropout of 0.35,
LSTM 80 layers, Dropout of 0.45, Dense layer of 6,
activation='relu'

Compile with loss =
keras.losses.SparseCategoricalCrossentropy, optimizer =
'adam',"accuracy" metrics

Fit model, x_train, x_test, y_train, y_test, epochs = 120,
batch_size=128

Define DBN model3

Add hidden_layers_structure = [256, 256], learning_rate_rbm
= 0.00000015, learning_rate = 0.3, n_epochs_rbm = 3,
n_iter_backprop = 120, batch_size = 128, activation_function
= 'relu')

Fit model, x_train, x_test, y_train, y_test

Predict Pred1, Pred2, Pred3
model1.predict(test_X),
model2.predict(test_X),
model3.predict(X_test)

Loop for length(x_train)
    final_prediction = mode of [pred1, pred2, pred3]
end
```

Fig. 12: Pseudo-code for the implememntation of the proposed system

4. EXPERIMENT

4.1 EVALUATION METRICS AND DATASETS

In this project, the Android Malware dataset has been used, namely, CICMalDroid 2020, which consists of mainly 5 types of labels, Adware, SMS malware, Riskware, Banking Malware, and Benign. As an input for the models, the .csv System log features have been used. The dataset is split into a training set constituting 70% of the dataset and a test set constituting 30% of the dataset. By making use of the ANOVA method which uses the f-score of the features along with the *selectkbest()* function as well as the *f1_classif()* function, the top features are selected, thereby providing a positive f-value.

The parameters for the CNN model are given as, the filter size for the convolution layer is 200, and the batch size is 128, and the dense layer of size 6. The parameters of RNN are as follows: *LSTM* filter size is 128 and the output size of dense layer is 6. The batch size is set to 128. The DBN has a hidden structure of 256x256 and a batch size of 128 as well.

Keras library in Python is used for the implementation of the models. The z-score preprocessing method is used for normalization. 'Accuracy' is used as the metric for the model training and 'adam' optimizer is used for the models. The runtime of the CNN, RNN, DBN, and voting classifier models are 11 secs, 56 secs, 3.7 minutes, and 4.9 minutes respectively.

System Specifications:

The experiment was performed on Spyder by Aaaconda on a system with Windows 10, i5 10^{th} Gen intel prcessor with 8GB RAM, 1GHz processor.

Perfromance Scores:

The scores employed as means of measuring the performance of the proposed system can be described as follows:

- **Accuracy:** The accuracy which is given by equation 4, where TP = true-positive, TN = true-negative, FP = false-positive = false-negative, of the models can be given by Table 1:

$$Accuracy = \frac{TP+TN}{FP+TP+FN+TN} \quad (4)$$

- **Precision:** Precision is the ratio of correct resuts with the total results in the output of a model,given by equation 5.

$$Precision = \frac{TP}{FP+TP} \quad (5)$$

- **Recall:** Recall is the ratio of correct results with the results that should have returned and is given by equation 6.

$$Recall = \frac{TP}{FN+TP} \quad (6)$$

- **F-score:** The F-score is the calculation of the test's accuracy of a model. It is given by the equation 7.

$$F - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (7)$$

Simplifying the equation 7 by substituting the values of Precision and Recall from equation 5 and equation 6 respectively; equation 8 is obtained, through which, the value of F-score can be evaluated directly from the elements of the confusion matrix.

$$F - Score = \frac{TP}{TP+(FP+FN)/2} \quad (8)$$

Macro-Average Scores:

Macro-average score takes the multiclass predictions and breaks them down into various sets of binary predictions, evaluates the measure for all of the binary cases, and afterwards combines the outcomes. Hence, for each class $i \in G$ of the five classes denoted by $G = \{1,\ldots,5\}$:

- **Macro-Average Precision:**

$$P_{macro} = \frac{1}{|G|}\sum_{i=1}^{|G|}\frac{TP_i}{FP_i+TP_i} = \frac{1}{|G|}\sum_{i=1}^{|G|}P_i \quad (9)$$

- **Macro-Average Recall:**

$$R_{macro} = \frac{1}{|G|}\sum_{i=1}^{|G|}\frac{TP_i}{FN_i+TP_i} = \frac{1}{|G|}\sum_{i=1}^{|G|}R_i \quad (10)$$

- **Macro-Average F-score:**

$$(F-Score)_{macro} = 2 * \frac{P_{macro} * R_{macro}}{P_{macro} + R_{macro}} \quad (11)$$

Weighted-Average Scores:

The weighted-average score can be described as a measure that considers the relative value of the data values in a data set. The most common use of a weighted average is to normalize the frequency of data values within the data set. Hence, for each class $i \in G$ having weight W_i:

- **Weighted-Average Precision:**

$$P_{weighted} = \frac{\sum_{i=1}^{|G|}W_iP_i}{\sum_{i=1}^{|G|}W_i} \quad (12)$$

- **Weighted-Average Recall:**

$$R_{weighted} = \frac{\sum_{i=1}^{|G|}W_iR_i}{\sum_{i=1}^{|G|}W_i} \quad (13)$$

- **Weighted-Average F-score:**

$$(F-Score)_{weighted} = 2 * \frac{P_{weighted} * R_{weighted}}{P_{weighted} + R_{weighted}} \quad (14)$$

4.2 CODE SNIPPETS

```
# feature selection
def select_features(X_train, y_train):
    # configure to select all features
    fs = SelectKBest(score_func=f_classif, k=98)
    # learn relationship from training data
    fs.fit(X_train, y_train)
    # transform train input data
    X_train_fs = fs.transform(X_train)
    return X_train_fs, fs

# feature selection
X_train_fs, fs = select_features(data_norm, y1)
X_train, X_test, y_train, y_test = train_test_split(X_train_fs, y1, test_size=0.3, shuffle=
train_X = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
test_X = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

Fig. 13: Code Snippet representing feature selection

```
#CNN
model1 = Sequential()
model1.add(Conv1D(200,1,input_shape = (1,1,98),padding = 'Valid',activation= 'relu'))
model1.add(Dropout(0.2))
model1.add(Dense(100, activation='relu'))
model1.add(Flatten())
model1.add(Dense(6,activation = 'relu' ))
model1.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer= 'adam',
    metrics=["accuracy"],
)
```

Fig. 14: Code Snippet representing implementation of CNN model

```
)
#RNN
model2 = Sequential()
model2.add(LSTM(200,return_sequences=True,input_shape=(X_train.shape[0], X_train.shape[1]),
model2.add(Dropout(0.35))
model2.add(LSTM(80))
model2.add(Dropout(0.45))
model2.add(Dense(6,activation='relu'))
model2.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer='adam',
    metrics=["accuracy"],
)
```

Fig. 15: Code Snippet representing implementation of RNN model

```
#DBN
model3 = SupervisedDBNClassification(hidden_layers_structure=[256, 256],
                                     learning_rate_rbm=0.00000015,
                                     learning_rate=0.3,
                                     n_epochs_rbm=3,
                                     n_iter_backprop=120,
                                     batch_size=128,
                                     activation_function='relu')
```

Fig. 16: Code Snippet representing implementation of DBN model

```
pred1=model1.predict(test_X)
pred2=model2.predict(test_X)
pred3=model3.predict(X_test)
op = np.argmax(pred1,axis=1)
op2 = np.argmax(pred2,axis=1)
final_pred = np.array([])
for i in range(0,len(X_test)):
    final_pred = np.append(final_pred, statistics.mode([op[i], op2[i], pred3[i]]))

print("Validation accuracy %f" %accuracy_score(y_test,final_pred))

pred11=model1.predict(train_X)
pred12=model2.predict(train_X)
pred13=model3.predict(X_train)
op1 = np.argmax(pred11,axis=1)
op12 = np.argmax(pred12,axis=1)
final_pred1 = np.array([])
for i in range(0,len(X_train)):
    final_pred1 = np.append(final_pred1, statistics.mode([op1[i], op12[i], pred13[i]]))

print("training accuracy %f" % accuracy_score(y_train,final_pred1))
y_pred = final_pred
print(classification_report(y_test, y_pred))
```

Fig. 17: Code Snippet representing implementation of Max Voting classifier model

4.3 RESULTS

- **Confusion Matrix:** Fig. 18 is the matrix representation of the confusion matrix obtained from the aforementioned algorithm for the proposed asystem. The diagonal elements of the matrix are the values that were correctly predicted.

Label	1	2	3	4	5
1	[[283,	39,	10,	18,	12],
2	[8,	521,	27,	32,	10],
3	[2,	7,	1202,	13,	1],
4	[13,	26,	10,	690,	30],
5	[15,	24,	17,	24,	446]],

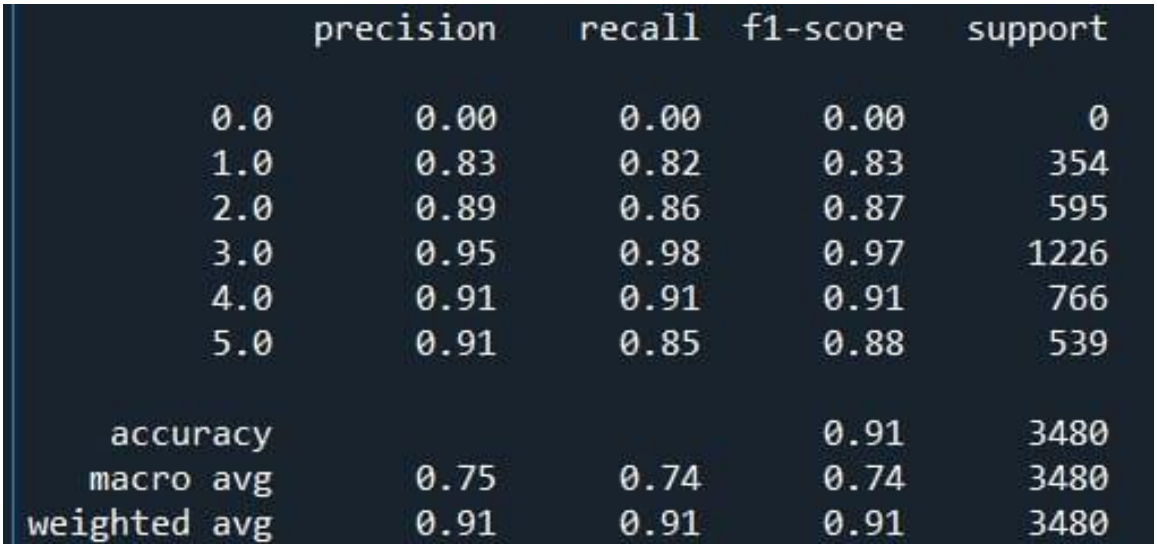
Fig. 18: Confusion Matrix for the proposed system

- **CNN Model:** The CNN model takes the preprocessed input data and processes it via three sets of layers, namely, the Convolution layer, Fully-connected layer, and the Dense layer to produce the output. The performance scores of the CNN model are depicted by Fig. 19.

	precision	recall	f1-score	support
1.0	0.85	0.78	0.81	376
2.0	0.86	0.89	0.87	643
3.0	0.96	0.98	0.97	1162
4.0	0.92	0.91	0.91	759
5.0	0.87	0.85	0.86	540
accuracy			0.91	3480
macro avg	0.89	0.88	0.89	3480
weighted avg	0.91	0.91	0.91	3480

Fig. 19: Performance statistics of CNN model implementation

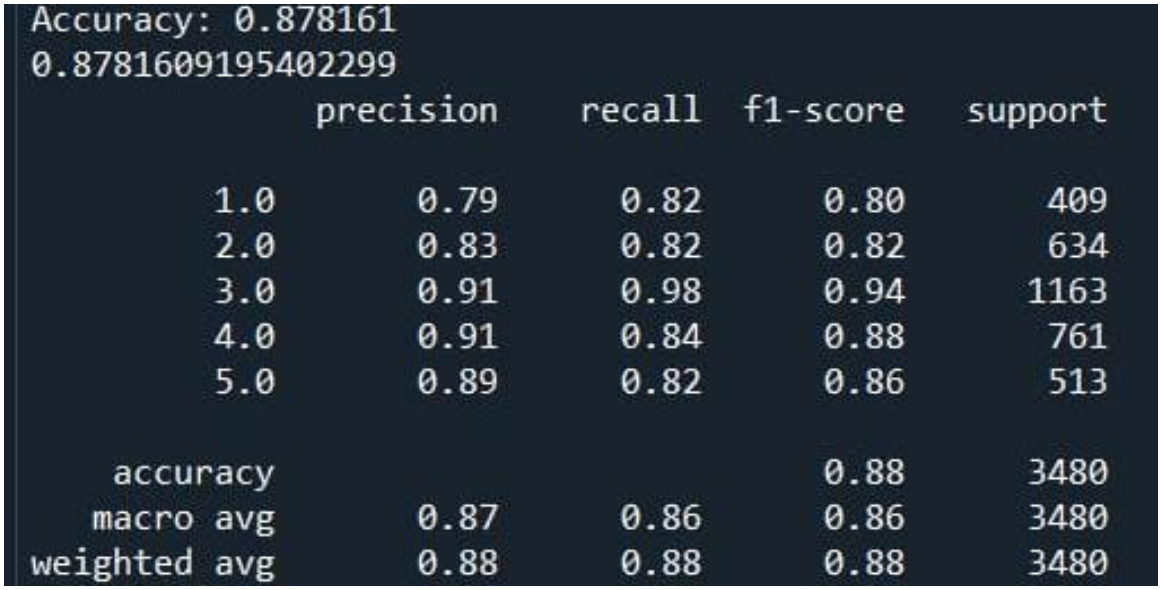
- **RNN Model:** In the RNN model, the preprocessed data passes through the LSTM layer, Fully-connected layer, and the Dense layer respectively, before providing the output. The performance scores obtained from the RNN model can be referred from Fig. 20.



	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	0
1.0	0.83	0.82	0.83	354
2.0	0.89	0.86	0.87	595
3.0	0.95	0.98	0.97	1226
4.0	0.91	0.91	0.91	766
5.0	0.91	0.85	0.88	539
accuracy			0.91	3480
macro avg	0.75	0.74	0.74	3480
weighted avg	0.91	0.91	0.91	3480

Fig. 20: Performance statistics of RNN model implementation

- **DBN Model:** The DBN model has only two layers, namely, the RBM (hidden) layer and the ANN (visible) layer with 'relu' function for activation. Fig. 21 demonstrates the performance stats of the DBN model.



Accuracy: 0.878161
0.8781609195402299

	precision	recall	f1-score	support
1.0	0.79	0.82	0.80	409
2.0	0.83	0.82	0.82	634
3.0	0.91	0.98	0.94	1163
4.0	0.91	0.84	0.88	761
5.0	0.89	0.82	0.86	513
accuracy			0.88	3480
macro avg	0.87	0.86	0.86	3480
weighted avg	0.88	0.88	0.88	3480

Fig. 21: Performance statistics of DBN model implementation

- **Max Voting Classifier Model:**

In the case of Max Voting Classifier model, the system takes normalized data along with the standard variance and mean of the data as its input, which is followed by feature selection on the input data. Thereafter, the dataset is fed to all the three models, namely CNN, RNN and DBN model and the processing of the input data is carried out in a parallel fashion. The implementation of the Max Voting Classifier model can be demonstrated by the code snippet shown in Fig. 17.

The results of the aforementioned models are then accumulated altogether and provided to the Max Voting Classifier model in orderto provide the outcome. The final prediction is the mode of the the predictions made by the above three models. The performance scores accomplished by the Max Voting Classifier model are represented by Fig. 20.

```
Validation accuracy 0.905460
training accuracy 0.953314
              precision    recall  f1-score   support

         1.0       0.86      0.80      0.83       362
         2.0       0.86      0.87      0.86       598
         3.0       0.95      0.98      0.97      1225
         4.0       0.90      0.90      0.90       769
         5.0       0.89      0.85      0.87       526

    accuracy                           0.91      3480
   macro avg       0.89      0.88      0.89      3480
weighted avg       0.90      0.91      0.90      3480
```

Fig. 22: Performance statistics of Max Voting Classifier model implementation

4.4 PERFORMANCE EVALUATION

With equal epochs and a batch size of 120 and 128 respectively. The total dataset consists of 139 features and 11598 instances, out of which the top 98 features have been used, selected using the Analysis of variance (ANOVA) method of feature selection given by Fig. 14. The x-axis indicates the complete set of features and the y-axis represednts the F-Score of the respective features.

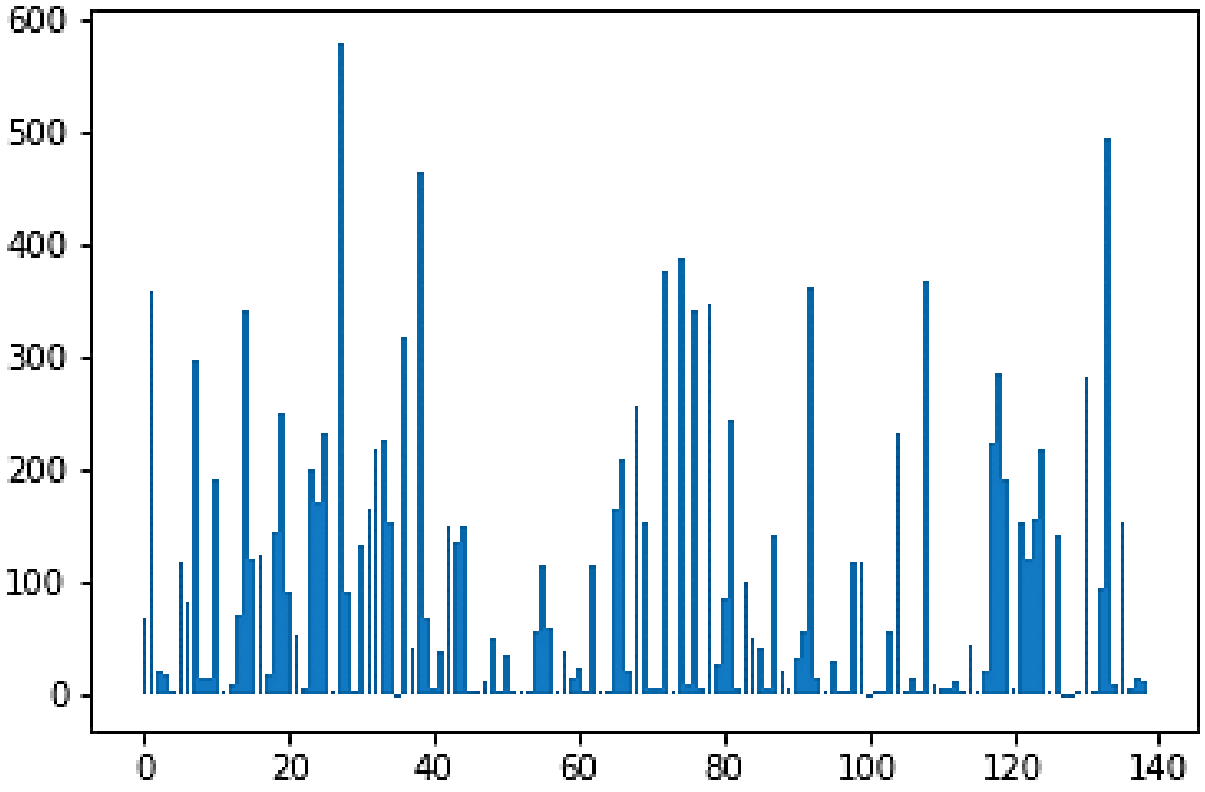


Fig. 23: F-value of the features

The accuracy of the models and the ensemble voting classifier using all features has been represented by Table 2 and the accuracy of the models and the ensemble voting classifier using selected features can be referred from Table 3.

Table 2: Accuracy of the models using all features

S No.	Models (All features, 11598 instances)			
	Name	*Epoch*	*Training Accuracy*	*Validation Accuracy*
1	CNN Model	120	95.17%	88.65%
2	RNN Model	120	94.38%	88.30%
3	DBN Model	120	95.90%	85.84%
4	Voting Classifier Ensemble (Proposed Model)	120	94.87%	90.63%

Table 3: Accuracy of the models using selected features

S No.	Models (98 selected features, 11598 instances)			
	Name	*Epoch*	*Training Accuracy*	*Validation Accuracy*
1	CNN Model	120	94.91%	90.87%
2	RNN Model	120	94.65%	90.01%
3	DBN Model	120	94.32%	90.20%
4	Voting Classifier Ensemble (Proposed Model)	120	95.07%	92.03%

Fig. 24 and Fig. 25 provide the comparison between the Training accuracy as well as the Validation accuracy repectively, achieved by various models and the proposed model using all features in first case and selected features in the second case.

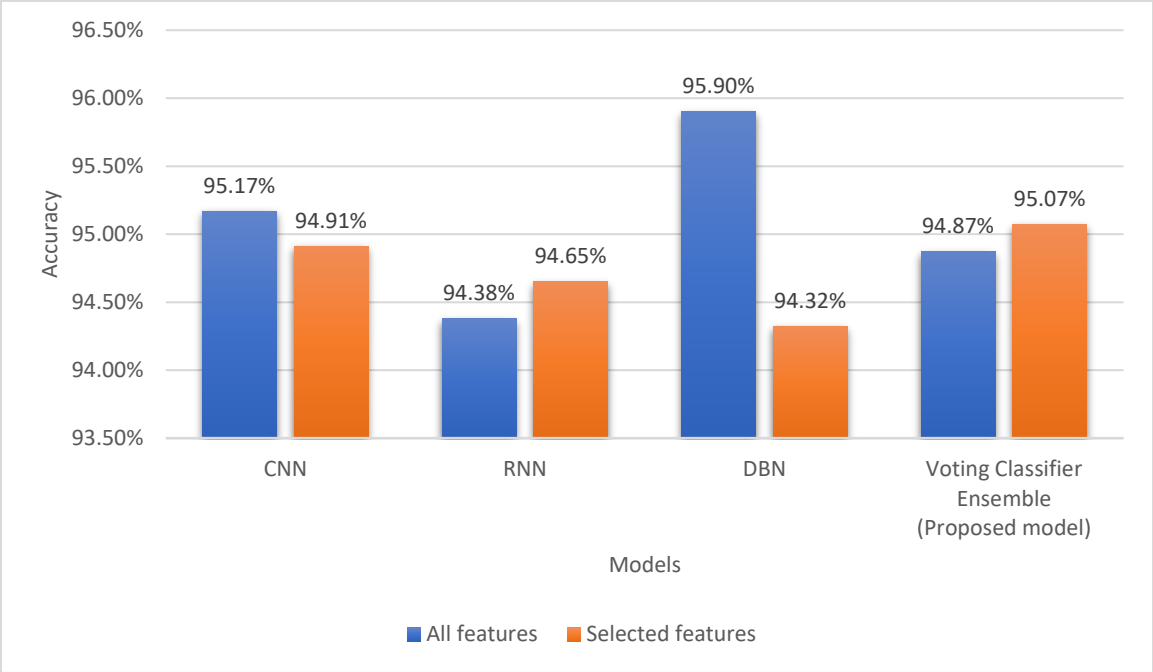


Fig. 24: Comparative analysis of Training Accuracy achieved by different models

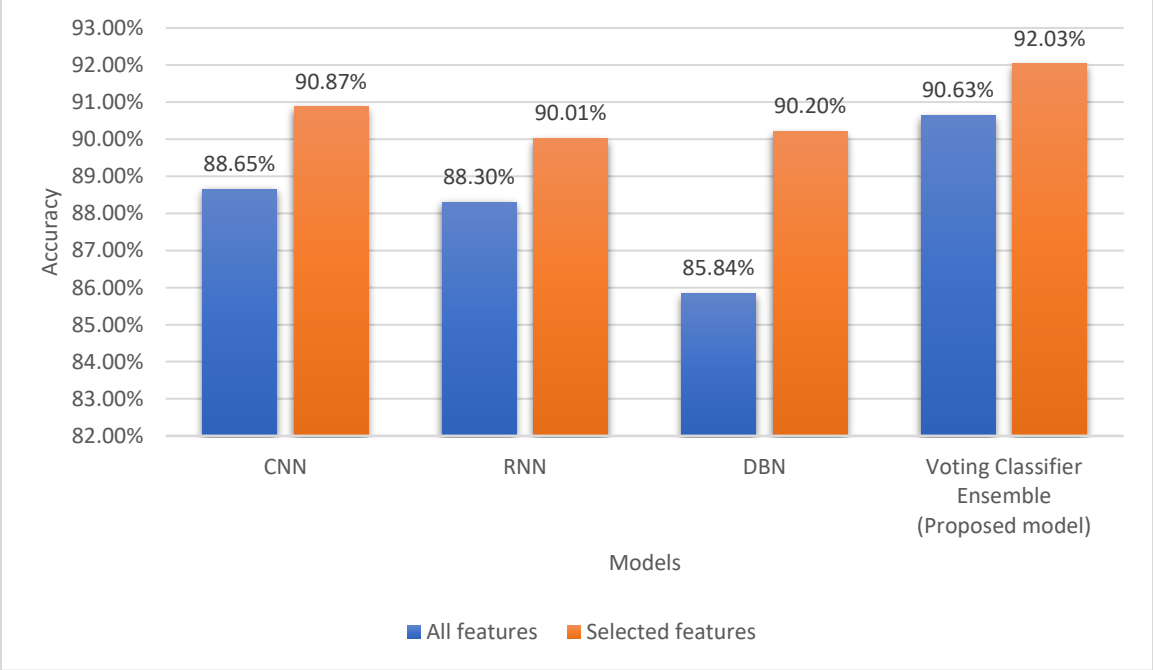


Fig. 25: Comparative analysis of Validation Accuracy achieved by different models

The remaining evaluation metrics of the models and the ensemble voting classifier using all features can be referred from Table 4 and the evaluation metrics of the models and the ensemble voting classifier using selected features has been represented in Table 5.

Table 4: Evaluation metrics for all models with all features

S No.	Models (All features, 11598 instances)			
	Name	*Precision*	*Recall*	*F-Score*
1	CNN Model	90.00%	89.00%	88.50%
2	RNN Model	87.00%	88.00%	88.30%
3	DBN Model	80.00%	75.00%	85.40%
4	Voting Classifier Ensemble (Proposed Model)	88.00%	87.00%	90.60%

Table 5: Evaluation metrics for all models with selected features

S No.	Models (98 selected features, 11598 instances)			
	Name	*Precision*	*Recall*	*F-Score*
1	CNN Model	89.00%	88.00%	90.70%
2	RNN Model	87.00%	88.00%	90.10%
3	DBN Model	90.00%	90.00%	90.20%
4	Voting Classifier Ensemble (Proposed Model)	91.00%	91.00%	92.20%

Fig. 26 and Fig. 27 provide an analysis of the Precision scores and Recall scores respectively achieved by the proposed model as compared to other models using all features in first case and selected features in the second case.

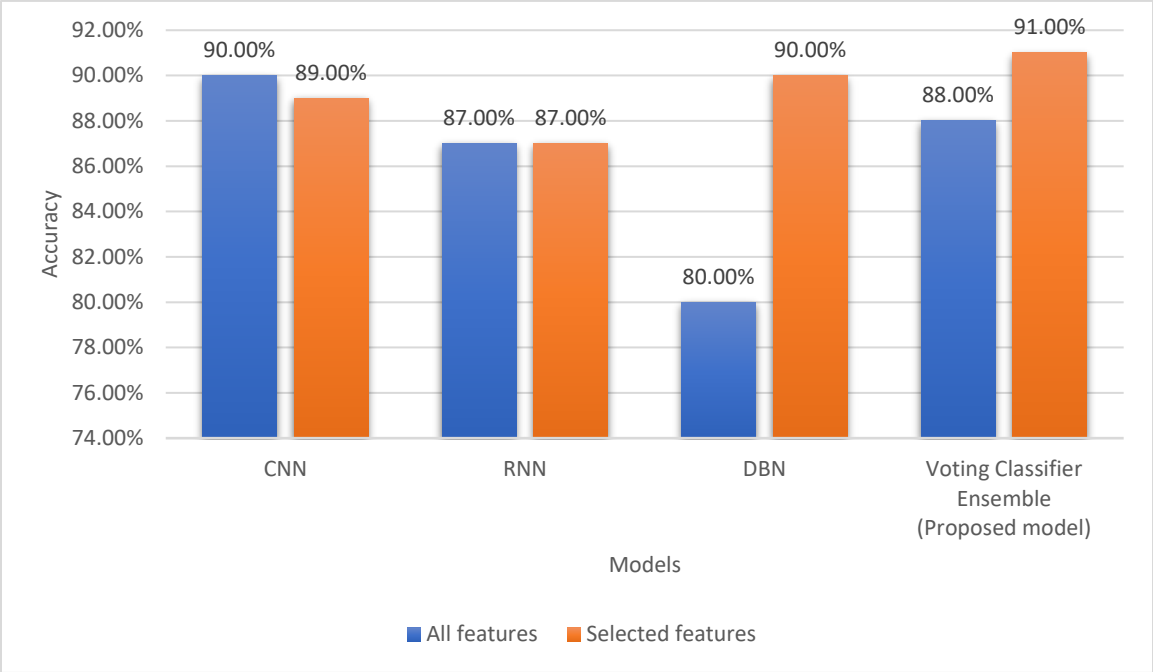


Fig. 26: Comparative analysis of Precision scores achieved by different models

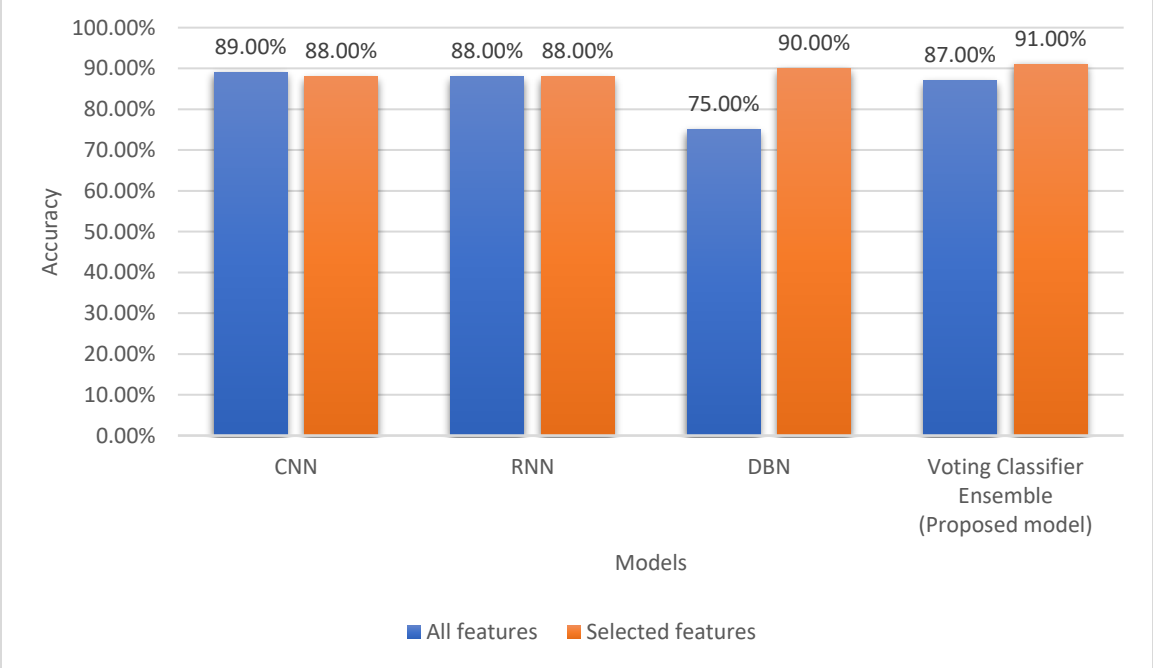


Fig. 27: Comparative analysis of Recall scores achieved by different models

The F-scores achieved by the proposed model in comparison with other models using all features and selected features respectively can be analyzed through Fig. 28.

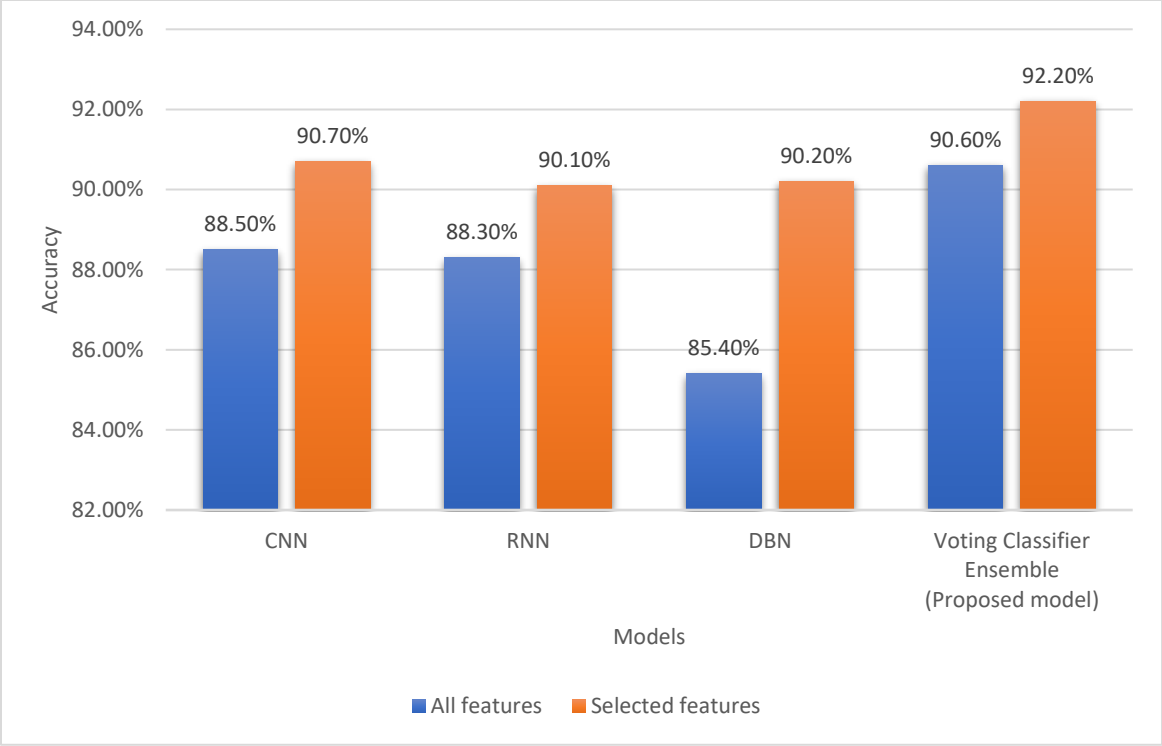


Fig. 28: Comparative analysis of F-scores achieved by different models

From the comparative analysis of various performance metrics mentioned in Table 1, Table 2, Table 3, and Table 4 (namely, Accuracy, Precision, Recall and F-score), as represented by Fig. 24 & 25, Fig. 26, Fig. 27 and Fig. 28 respectively, it can be deduced that the voting classifier ensemble model results in increased accuracy results and it has been found that by using selected features instead of the complete dataset, the accuracy of the model increases.

By using Max Voting method, the mode of predictions from different models is calculated and hence even if any one of the prediction models is wrong, the majority vote of the models is selected as the final output.

5. CONCLUSION

With increased usage of IoT based Android devices, security and protection from malware have become an inevitable issue. In order to design a malware-protection system, there is a need for techniques to detect malware. The traditional detection methods, based on the signature and permissions have become ineffective with the increased complexity of malware due to techniques like code obfuscation and encryption. In this paper, the implementation as well as analysis of various deep learning methods, CNN, RNN, and DBN, and proposed a voting classifier ensemble model has been carried out in order to enhance the performance in terms of malware detection. The results show that the performance of the model with the voting classifier is enhanced, which is further increased using feature selection. It has been found that the usage of the voting classifier with selected features dataset, has an improved training accuracy of 95.07% and validation accuracy of 92.03%.

6. FUTURE SCOPE

For future scope, a comparison of various other models can also be done to find the most suitable model for malware detection, and also the categories of classification should be increased with a larger dataset for a more accurate training of the models. In the near future, a review of several other models should be performed to determine the most appropriate model for malware identification, and the categories of classification can be expanded with a wider dataset for more precise model training.

As of this moment, the proposed framework has been proven to be a valuable tool for the computer security. For the near future, it has been planned to integrate more classification algorithms to it. Future research entails exploration of these variations with new features that could be added to the existing data. Furthermore, multiple number of Feature Engineering Methods can be employed and hence, comparisons can be carried out, in order to find the optimal features for the best training as well as testing results.

REFERENCES

[1] Gavel, S., Joshi, P., Tiwari, S., & Raghuvanshi, A. S. (2021). An Optimized Hybrid Clustering Method Using Salp Swarm Optimization with K-Means. In *Nanoelectronics, Circuits and Communication Systems* (pp. 247-254). Springer, Singapore.

[2] Wang, A., Liang, R., Liu, X., Zhang, Y., Chen, K., & Li, J. (2017, March). An inside look at IoT malware. In *International Conference on Industrial IoT Technologies and Applications* (pp. 176-186). Springer, Cham.

[3] Aksakalli, I. K. (2019). Using convolutional neural network for Android malware detection. *Information Technologies, Management and Society*.

[4] Cengil, E., & ÇINAR, A. (2016). A new approach for image classification: convolutional neural network. *European Journal of Technique*, *6*(2), 96-103.

[5] Deep learning-Convolutional Neural Network, E-source: (Accessed on 14th April 2021) https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

[6] Sawle, P. D., & Gadicha, A. B. (2014). Analysis of malware detection techniques in Android. *International Journal of Computer Science and Mobile Computing*, *3*(3).

[7] Biswas, P., Charitha, R., Gavel, S., & Raghuvanshi, A. S. (2019, April). Fault Detection using hybrid of KF-ELM for Wireless Sensor Networks. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)* (pp. 746-750). IEEE.

[8] Sun, L., Li, Z., Yan, Q., Srisa-an, W., & Pan, Y. (2016, October). SigPID: significant permission identification for android malware detection. In *2016 11th international conference on malicious and unwanted software (MALWARE)* (pp. 1-8). IEEE.

[9] El Boujnouni, M., Jedra, M., & Zahid, N. (2015, December). New malware detection framework based on N-grams and support vector domain description. In *2015 11th international conference on information assurance and security (IAS)* (pp. 123-128). IEEE.

[10] Wüchner, T., Cisłak, A., Ochoa, M., & Pretschner, A. (2017). Leveraging compression-based graph mining for behavior-based malware detection. *IEEE Transactions on Dependable and Secure Computing*, *16*(1), 99-112.

[11] Bhattacharya, A., & Goswami, R. T. (2017). Comparative analysis of different feature ranking techniques in data mining-based android malware detection. In *Proceedings of the 5th international conference on frontiers in intelligent computing: theory and applications* (pp. 39-49). Springer, Singapore.

[12] Souri, A., & Hosseini, R. (2018). A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-centric Computing and Information Sciences*, *8*(1), 1-22.

[13] Elhadi, A. A., Maarof, M. A., & Osman, A. H. (2012). Malware detection based on hybrid signature behaviour application programming interface call graph. *American Journal of Applied Sciences*, *9*(3), 283.

[14] Fan, C. I., Hsiao, H. W., Chou, C. H., & Tseng, Y. F. (2015, July). Malware detection systems based on API log data mining. In *2015 IEEE 39th annual computer software and applications conference* (Vol. 3, pp. 255-260). IEEE.

[15] Wang, P., & Wang, Y. S. (2015). Malware behavioural detection and vaccine development by using a support vector model classifier. *Journal of Computer and System Sciences*, *81*(6), 1012-1026.

[16] Fraley, J. B., & Figueroa, M. (2016, March). Polymorphic malware detection using topological feature extraction with data mining. In *SoutheastCon 2016* (pp. 1-7). IEEE.

[17] Sun, M., Li, X., Lui, J. C., Ma, R. T., & Liang, Z. (2016). Monet: a user-oriented behavior-based malware variants detection system for android. *IEEE Transactions on Information Forensics and Security*, *12*(5), 1103-1112.

[18] Sun, H., Wang, X., Buyya, R., & Su, J. (2017). CloudEyes: Cloud-based malware detection with reversible sketch for resource-constrained internet of things (IoT) devices. *Software: Practice and Experience*, *47*(3), 421-441.

[19] Tang, Y., Xiao, B., & Lu, X. (2010). Signature tree generation for polymorphic worms. *IEEE transactions on computers*, *60*(4), 565-579.

[20] Zarni Aung, W. Z. (2013). Permission-based android malware detection. *International Journal of Scientific & Technology Research*, *2*(3), 228-234.

[21] Ding, Y., Zhang, X., Hu, J., & Xu, W. (2020). Android malware detection method based on bytecode image. *Journal of Ambient Intelligence and Humanized Computing*, 1-10.

[22] Lee, S., Dolby, J., & Ryu, S. (2016, September). HybriDroid: static analysis framework for Android hybrid applications. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 250-261). IEEE.

[23] Pektaş, A., & Acarman, T. (2017). Classification of malware families based on runtime behaviors. *Journal of information security and applications*, *37*, 91-100.

[24] Palumbo, P., Sayfullina, L., Komashinskiy, D., Eirola, E., & Karhunen, J. (2017). A pragmatic android malware detection procedure. *Computers & Security*, *70*, 689-701.

[25] Zhu, H. J., Jiang, T. H., Ma, B., You, Z. H., Shi, W. L., & Cheng, L. (2018). HEMD: a highly efficient random forest-based malware detection framework for Android. *Neural Computing and Applications*, *30*(11), 3353-3361.

[26] Ganesh, M., Pednekar, P., Prabhuswamy, P., Nair, D. S., Park, Y., & Jeon, H. (2017, July). CNN-based android malware detection. In *2017 International Conference on Software Security and Assurance (ICSSA)* (pp. 60-65). IEEE.

[27] Jha, S., Prashar, D., Long, H. V., & Taniar, D. (2020). Recurrent neural network for detecting malware. *Computers & Security*, *99*, 102037.

[28] Liu, Y., Tantithamthavorn, C., Li, L., & Liu, Y. (2021). Deep Learning for Android Malware Defenses: a Systematic Literature Review. *arXiv preprint arXiv:2103.05292*.