ANDROID MALWARE DETECTION USING DYNAMIC FEATURE ANALYSIS

SADIA RASHID

01-241211-009

A thesis submitted in fulfillment of the
Requirements for the award of the degree of
Master of Science (Software Engineering)

Department of Software Engineering

BAHRIA UNIVERSITY ISLAMABAD

January, 2024

APPROVAL FOR EXAMINATION

Scholar's Name: _Sadia Rashid________ Registration No. 01-241-211009__

Program of Study: __MS (Software Engineering) __

Thesis Title: Android Malware Detection Using Dynamic Features Analysis__

It is to certify that the above scholar's thesis has been completed to my satisfaction and, to my belief, its standard is appropriate for submission for examination. I have also conducted a plagiarism test of this thesis using HEC-prescribed software and found a similarity index __17%_ that is within the permissible limit set by the HEC for the MS degree thesis. I have also found the thesis in a format recognized by the BU for the MS thesis.

Principal Supervisor's

Signature:____

Date: ____

Name:____

AUTHOR'S DECLARATION

I, Sadia Rashid hereby state that my MS thesis titled "___Android Malware Detection using Dynamic Feature Analysis_" is my work and has not been submitted previously by me for taking any degree from this university Bahria University Islamabad, or anywhere else in the country/world.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw/cancel my MS degree.

Name of scholar: _Sadia Rashid (01-241211-009)

Date: ___________________________________

PLAGIARISM UNDERTAKING

I, _Sadia Rashid_____, solemnly declare that the research work presented in the thesis titled "__Android Malware Detection using Dynamic Feature Analysis__" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero-tolerance policy of the HEC and Bahria University towards plagiarism. Therefore I as an Author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred to/cited.

I undertake that if I am found guilty of any formal plagiarism in the above-titled thesis even after the award of my MS degree, the university reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of scholars are placed who submitted plagiarized thesis.

Scholar / Author's Sign: ________________________

Name of the Scholar: Sadia Rashid (01-241211-009)

DEDICATION

To my bright future

ACKNOWLEDGEMENT

I would start by thanking ALLAH Almighty, with gratitude for giving me strength in every aspect of life and helping me in this thesis as well.

I wish to express appreciation to my thesis supervisor, Dr. Tamim Ahmed Khan, for his guidance throughout this thesis.

I want to acknowledge my efforts that I didn't give up even when the odds were against me and lastly completed this thesis on time.

ABSTRACT

Android is a widely used operating system but with its success, it also faces issues with malware threats. With advanced technology, malware is also becoming complex making it challenging to detect. Android malware detection techniques are used to detect malware on Android operating systems. There are mainly two types of detection techniques which are static and dynamic analysis. This thesis is focused on the dynamic analysis of Android malware.

The use of machine learning or deep learning for malware detection requires datasets. These datasets Malware Researchers developed many machine learning models and deep learning models to detect Android malware considering static as well as dynamic features-based datasets. We consider features extracted from system executions and we perform multi-class classification of malware categories in this study, by considering all malware classes (Adware, Backdoor, File Infector, PUA, Ransomware, Riskware, Scareware, Trojan, Trojan-Banker, Trojan-Dropper, Trojan-SMS, Trojan-Spy and zero-day) by using deep learning algorithm as it outperforms machine learning in high dimensional data.

We use the CIC-AndMal-2020 dataset, the newly developed dataset for multi-class classification. It consists of 13 prominent malware classes and 141 features. We also perform statistical analysis on the dataset (p-value analysis and correlation) to identify the relationship between features and statistical analysis of the model (bias and variance) to arrive at the optimal model. We evaluate the proposed algorithm using performance metrics i.e. accuracy, precision, recall, ROC-AUC analysis, and F1-Score and, finally, compare our results with existing studies. Our results outperform previous dynamic analysis results.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CIC	-	Canadian Institute of Cybersecurity
ML	-	Machine Learning
DL	-	Deep Learning
DNN	-	Deep Neural Network
LSTM	-	Long Short-Term Memory
OS	-	Operating System
AI	-	Artificial Intelligence
APK	-	Android Package Kit
AUC-ROC	-	Area under ROC Curve
NB	-	Naive Bayes
DT	-	Decision Tree
RF	-	Random Forest
SL	-	Simple Logistic
TPR	-	True Positive Rate
FPR	-	False Positive Rate
FNR	-	False Negative Rate

PCA	-	Principal Component Analysis
API	-	Application Programming Interface
CNN	-	Convolutional Neural Network
RNN	-	Recurrent Neural Network
P Value	-	Probability Value
SMOTE	-	Synthetic Minority Oversampling Technique
ReLU	-	Rectified Linear Unit
Tan h	-	Hyperbolic Tangent

CHAPTER 1

INTRODUCTION

In this modern age, society is aware of computer security threats. Rapid technological advancements have also made enhancements in the mobile phone industry. With the increase in mobile phones, the market share value of the mobile industry has increased by 8% in 2023 as compared to 2022 [1]. According to Global Stats, Android has become the widely used OS for being an open-source architecture [2].

However, it is pertinent to highlight that a survey conducted and presented in [3]. that there is a malware present on every 1 out of 20 devices running Android Operating System (O/S). Malware developers are constantly developing complex malware that is undetectable. These developers used multi-platform development tools that make it difficult to determine the level of maliciousness therefore this is a significant challenge to detect potential threats. The increase in Android malware threats in the 2022 year is presented in Fig 1.1.

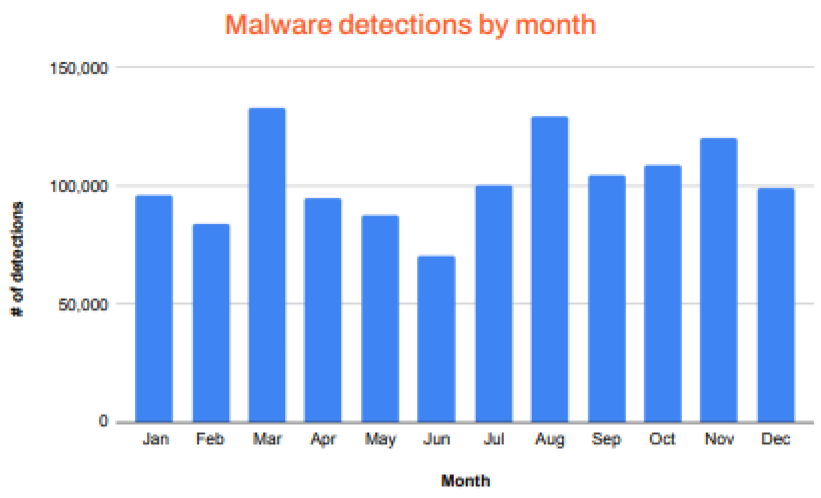


Figure 1. 1 Adopted from [3]

To overcome this challenge, Android malware detection techniques are developed to protect Android devices from malware. Android malware analysis consists of three approaches, static analysis, dynamic analysis, and hybrid analysis. The static analysis [4], consists of signature, permission, and component-based investigation. It analyzes applications to detect malicious behavior without executing it in real real-time environment. The dynamic analysis [5], consists of behavioral analysis of the application by executing the application in an android emulator or real-time environment.
Researchers have developed several machine learning techniques to detect malicious applications, these techniques are used to identify hidden patterns of previously encountered malicious apps. Machine learning (ML) uses a set of features with a corresponding class to train the model which classifies between benign and malicious apps. This classification is utilized to identify malware and classify it into corresponding malware families. , deep learning (DL) is a subset of machine learning and appeared as a mechanism that shows remarkable results in traditional artificial intelligence (AI) tasks related to computer vision, natural language processing (NLP), and speech processing fields. It is also emerging as an active area of research [6].

1.1 Motivation

The motivation of this thesis is to improve the Android malware analysis mechanism using dynamic-based features. The primary focus of the study is to perform a multi-class classification of Android malware by developing a deep learning-based algorithm to improve the Android malware analysis mechanism. We use newly developed CIC-AndMal-2020 dataset and we select the significant set of features by performing a statistical analysis on features. To enhance the Android malware detection mechanism machine learning (ML) and deep learning (DL) techniques are implemented but most of them perform binary classification or consider fewer number of malware classes. The dataset mattered a lot in the detection mechanism of Android malware. The CIC-AndMal-2020 dataset is a newly generated dataset that consists of 14 malware categories and 141 features. Most of the research consider fewer number of malware classes to detect Android malware.

1.2 Research Gap

Android is the in-demand operating system and with an increase in its users, the security of Android smartphones is important to keep out malware attacks from acquiring sensitive information. To alleviate this threat several initiatives have been undertaken to analyze malware. There are three types of android malware analysis techniques which are static analysis, dynamic analysis and hybrid analysis [7]. Static analysis extracts detailed information from APK without executing it and utilizes requested permission to identify threats in applications but detecting malware exclusively with permission may not guarantee a high detection rate [8]. The dynamic analysis performs behavioral modeling of the APK on run time by executing the code on an Android emulator or device. The advantage of dynamic analysis over static analysis is in code obfuscation and real-time monitoring, [9][10][11]. The hybrid analysis combines both static features and dynamic features to detect android malware but majority of the studies has consider few number of malware classes (Adware, Ransomware, Scareware, and Trojan-SMS malware categories) to perform multiclass classifications [7] [12][13][14]. Malware is anticipated to get more complex in the future and malware developers employ novel encryption methods that can make the detection of malware difficult [15]. Malware developers use various strategies to avoid static analysis (encryption, code obfuscation, etc.) but dynamic analysis is tolerant to such strategies and provides more insight into malware which can improve detection capabilities [16][17]. Researchers have developed several machine learning (ML) and deep learning (DL) based algorithms focusing dynamic analysis to identify malware categories but earlier studies were focused on binary classification [18][19][20][21][22][23][24][25]. With the rapid advancement in technology, malware also become more sophisticated to detect, due to this data scientists developed datasets that contain samples of malware categories and malware families [26][27]. Researchers analyze newly generated datasets using machine learning (ML) and deep learning (DL) based techniques but the majority among them considered fewer number of features (mostly API and memory features) as compared to features present in newly developed datasets or less number of malware classes to detect malicious activities [28][29][30] [31][32]. Deep learning algorithms are more effective than machine learning algorithms in the scenario that data is large and high dimensional [33] [34]. There is a need for deep learning (DL) based models that can efficiently classify a variety of dynamic features and malware categories with a multi-class classification.

1.3 Problem Statement

Malware analysis datasets are large in size due to several features and available examples. Deep learning-based models generally outperform machine learning techniques when datasets of larger size are available. The majority of existing studies consider a limited number of features and malware categories using static analysis-based datasets. Android malware analysis using datasets based on feature extraction from dynamic analysis is more robust as compared to static analysis resulting in datasets. We need to analyze how we can consider a broader range of malware categories, use dynamic analysis resulting datasets, and consider deep-learning models with optimal configuration to improve results.

1.4 Research Questions

RQ 1: How can we consider more dynamic features for multiclass classification?

RQ 2: How can we use deep learning models for multiclass classification to improve the accuracy and performance of Android malware detection considering dynamic features analysis?

1.5 Research Objectives

1. To consider the dynamic features to detect Android malware and perform multiclass classification.
2. To develop a deep learning model that can improve accuracy (performance) by considering 13 malware categories for multi-classification.

1.6 Contribution of the study

This section of the thesis describes the contributions resulting from the proposed study

- The CIC-AndMal-2020 is a newly developed dataset to perform multi-class classification.
- We perform statistical analysis on the dataset by performing p-value analysis and identifying the correlation of features of the dataset.

- This study performs multi-class classification using a deep learning algorithm by considering all features and malware categories of the dataset.
- This study also performs statistical analysis of the model by identifying the bias and variance by experimenting with different hyper parameters, hidden layers, and activation functions.
- We evaluate the performance of the proposed algorithms by identifying accuracy, precision, AUC-ROC analysis, recall, and F1-score.
- We compare the results of the best-performing model with existing studies.

1.7 Thesis Outline

The arrangement of this thesis is described below.

- **Chapter 2** addresses the literature review related to Android malware concerning dynamic analysis based on deep learning and machine learning algorithms.
- **Chapter 3** addresses the research methodology, implementation, and research processes regarding the proposed study.
- **Chapter 4** addresses the details of implementation and results. We also performed a comparative analysis with existing studies.
- **Chapter 5** addresses the conclusion and future work.

CHAPTER 2

LITERATURE REVIEW

This chapter presented the literature review on Android malware detection using dynamic analysis techniques. This research primarily concentrates on the multiclass classification of dynamic feature analysis based on deep learning. The algorithms, validation methods, and performance evaluation metrics used in dynamic analysis based on ML and DL were discussed. Lastly, we presented existing datasets. This chapter concludes with an open research gap.

2.1 Android Malware Detection Techniques

Android malware detection analysis consists of three approaches i.e., static, dynamic, and hybrid analysis. The static analysis analyzes code to detect malicious behavior without executing the APKs in an Android virtual device or real-time device. It extracts detailed information (signatures) from APK (Android Application kit). The execution of code in an Android virtual device or real-time device is considered by Dynamic Analysis. It analyzes the behavior of the APK file on run time. The advantages of dynamic analysis include dynamic code loading detection and record of run time application behavior [5].

2.2 Android Malware Detection w.r.t Dynamic Analysis

This section discusses the existing studies concerning Android malware detection using dynamic analysis.

2.2.1 Android Malware Detection Based on Machine Learning

There are several approaches suggested to discover Android malware analysis based on ML. These approaches are further distinguished into multi-class and binary classification.

2.2.1.1 Binary Classification

Authors develop a dataset and apply a set of ML classification techniques which are NB, Decision Tree (J48), RF, SL, and k-star [35]. The dataset consists of 11000 Android applications out of which 6971 are malware samples and the rest are benign samples the study extracted 123 dynamic permissions from the dataset and performed binary classification on it. The experimental results achieved an accuracy of 99.7% with the SL machine-learning technique. This study developed the largest dataset till then and gives the highest accuracy of previously proposed approaches but lacks discussion about multi-class classification.

Authors develop a dynamic syscall-capture system that extracts system call traces from the application during their interaction with a phone on runtime [23]. Due to this, 50 malicious and 50 benign applications are collected by them from the Android malware genome project and Google Play Store respectively to process them into aggregate datasets. The experimental results showed that Decision Tree Algorithms gave 85% and Random Forest gave 88% of accuracy.

A comparative analysis was performed by authors on emulator-based vs device-based Android malware detection [36]. The study shows 23.8% more efficient analysis of Android applications and extraction of features from mobile devices as compared to emulated environments. The dataset consists of 2444 Android applications, out of which 1222 malware samples were collected from the Genome project and benign samples were collected from Intel Security. The study performed information gain to extract the top 100 features which consist of API calls from the dataset. The author applied machine learning algorithms that as RF, NB, Multilayer Perceptron, SL, J48 decision tree, PART, and SVM (linear) for binary classification. The experimental results gave 0.926 F-Measure for phone-based analysis, 93.1% TPR, and 92% FPR with the RF classifier.

Authors proposed host host-based detection system called ServiceMonitor, which dynamically detects malware on mobile phones [37]. The ServiceMonitor used the Markov chain model to analyze how applications request services from mobile phones and transform them into feature vectors. The proposed study developed a dataset composed of 8034 malware samples obtained from Androzo, Drebin, and Genome Project. The benign samples which were of 10024 in size were collected from Google Play. This study performed Principal Component Analysis (PCA) to extract 200 features from a dataset having maximum variance. The RF classifier is used for binary classification. 96.7% accuracy was achieved in the experimental results.

The authors proposed the selection of relevant attributes for improving locally extracted features using classical feature selectors (SAILS), which is a feature selection technique [38]. The proposed study used a dataset composed of 4949 samples, 2475 were benign and 2474 were malware applications from Drebin. The features consist of system calls extracted during the executions of applications. The experimental results show reformed values for evaluation metrics as compared to conventional feature selection methods for ML classifiers that is LR, CART, RF, XGBoost, and DNN with a recall rate of 24.79 %to 92.2%, and a true positive rate is reported from 95.2 and 99.79%.

A dynamic analysis technique in Android malware detection known as SelFDroid was proposed by the authors in [39]. The suggested approach consists of the collection of data, extraction, and selection of features, and classification process. The study collected 200 Android applications to develop a dataset, 100 were malware samples collected from Androzo datasets and 100 were benign applications collected from APKPure. The system calls, Network Packets, CPU Usage, and Battery Usage features were extracted from applications in an emulated environment. This study used a Sequential Minimal Optimization (SMO) classifier to evaluate results and achieved 91.7% accuracy, 93.1% precision, 90.0 recall, and 8.3 error rate.

The authors proposed a dynamic analysis approach called DATDroid [40]. In this methodology, the authors collected 200 dataset samples from which 100 samples were benign, collected from the APKPure market and 100 were malicious which were collected from Genome Project. The five features were extracted which included system call, errors and time of system call process, CPU usage, memory, and network packets. The experimental results gave an accuracy of 91.7%, a precision of 93.1%, and a recall rate of 90.0% by using the Random Forest algorithm.

The authors proposed a two-stage mixed detection model which consists of RF and Markov models. The RF model is to detect ransomware [41]. The study extracted 30967 features out of a total of 6393 by dynamic analysis from 7 categories that is Windows API call stats, Registry keys operations, file system operations, Strings, File extensions, Directory operations, and Dropped file extensions. The machine learning algorithm i.e. RF is used to achieve an accuracy of 97.3% with 4.8% FPR and 1.5% FNR.

A detailed analysis is being done to analyze all categories of dynamic analysis and their features by using different filter and wrapper methods [18]. The CIC-AndMal-2020 dataset is being used. This paper is focused on the dynamic analysis part of the dataset. The Information Gain and Backward elimination methods are used to find the categories of features with the highest significance which are Memory and API. Constant feature and Genetic Algorithm techniques are used to identify a feature of high significance from the API and Memory category and hence 35 features are extracted (11 from Memory and 24 from the API category). Random Forest is applied for classification and the number of trees is set to 5. The F1 score of the system is increased to 0.82 after the removal of irrelevant features.

The author presented entropy-based dynamic behavior analysis for Android malware detection is being done on six classes of features extracted from the CIC-AndMal-2020 dataset [11]. The study extracted 141 dynamic features i.e. 105 API features, 2 battery features, 4 network features, 1 process feature, and 6 logcast features after executing APK files in the sandbox environment. The Shannon entropy is computed for every malware sample before and after rebooting to understand behavior changes. The RF classifier is used to classify malware categories with 0.984 precision and 0.983 recall values.

The author proposed web based framework, which detects malware from Android devices MLDroid [21]. The study collected 56,871 samples from AndroMalShare and 1200 Android Malware Genome projects and developed a dataset consisting of 50,000 unique samples. The study extracts 1532 permissions and 310 API calls from the dataset. The features are then divided into 30 unique feature sets. The feature selection approaches are Gain-ratio, Chi-Squared test, Information gain, OneR, PCA, and Logistic Regression applied to the feature. The study also applied feature subset selection approaches which are Correlation-based, Rough set, Consistency subset evaluation approach, and Filtered subset evaluation on distinct feature sets. The study implemented 21 different ML algorithms which are SVM, NB, RF, MLP, LR, BN, AB, decision tree (DT), KNN, deep neural network (DNN), self-organizing map (SOM), Kmean, farthest first clustering (FF), filtered clustering (FC), density-based clustering (DB), J48? YATSI (Y-J48), SMO YATSI (Y-SMO), MLP YATSI (Y-MLP), best training ensemble approach (BTE), majority voting ensemble approach (MVE), and nonlinear ensemble decision tree forest approach (NDTF) to find the suitable model to detect android malware. The experimental results show that models developed by considering features selected by the feature selection approach as an input can detect malware more effectively rather than models developed by using all extracted feature sets and gave the highest accuracy of 98.8%.

The author proposed an Android malware detection model based on APIs and permission features [19]. Two datasets are used, The first one is the Malgenome dataset, which consists of 3800 Android applications (1260 malware apps, 2539 benign apps, and 3800 applications with permissions and API calls) and the second one is the Maldroid dataset which consists of 11599 samples categorized as benign, adware, banking malware, and mobile riskware. A detailed analysis is being done on existing ML classification models from which KNN, SVM, NB, and DT are selected to perform classification. The result shows an accuracy of 86% on the Maldroid dataset and a precision of 99% on the Malgenome dataset with SVM.

Authors have proposed a dynamic analysis framework, called Android, which can detect malware and classify malware families based on multiple types of dynamic features [42]. The 2 datasets are used, the first one consists of the Drebin dataset, which consists of 5560 malicious applications and 8806 benign applications gathered from different sources i.e. Google Play and Androzo called M1, and the second one consists of 5000 benign and 5000 malicious applications gathered from Androzo called M2. EnDroid extracts 58709 total features from the dataset and chi-square feature selection method is used to select 5000 features from it. Multiple ML models are applied for classification and Stacking outperforms others giving the F-Measure of 0.9521 for M1 and 0.9642 for M2.

2.2.1.2 Multi-Class Classification

The systematic and functional approach is being use identify malware classes and families on dynamic layers [43]. The CIC-AndMal-2020 dataset is being used. The paper focused on the dynamic analysis (before rebooting) part of the dataset. 141 features are being extracted from 6 characteristics (Memory, API, Network, Battery, Logcat, and Process) for dynamic analysis. The comparison between machine learning techniques (Decision tree (J48), NB, SVM, AB, LR, KNN, RF, and Multilayer Perceptron (MLP)) is being done to evaluate the effectiveness of the proposed model. RF shows an accuracy of 96.86% and becomes the most effective classifier than others in the malware categorization phase while in malware family classification RF shows an accuracy of 99.65%. After comparison, the proposed model (Random Forest) shows better accuracy than existing models by accurately classifying 180 malware families and 14 malware categories.

The study developed an ensemble-based machine-learning model for multiclass classification to detect Android malware [31]. The study used the CCCS-CIC-AndMal-2020 dataset. The dataset consists of 14 malware classes and 141 features. The study only considers 12 malware classes and 45 features out of 141 for multi-class classification. The study used an ensemble ML model composed of RF, KNN, MLP, Decision Trees, SVM, and LR classifiers. The model used a voting mechanism to identify malware classes. The experimental results show an accuracy of 95% of accuracy.

The study developed a framework for dynamic analysis called EnDroid [44]. EnDroid framework is composed of an ensemble-based machine learning algorithm. The study considered a dataset to show the effectiveness of the proposed framework. Dataset 1 consists of 8806 benign applications and 5213 malicious applications gathered from Androzoo and applications collected from AndroZoo. The framework extracts 58709 dynamic behavior from dataset 1 but after applying chi-square only 5000 remain as features. The features consist of system calls. The experimental results show that the M1 dataset has achieved 0.9735 F-measure, and 0.9682 AUC, whereas on the M2 dataset, 0.9830 F-measure and 0.9702 AUC are achieved.

Table 2. 1 Selected Studies of ML

Ref	**Dates**	**Machine Learning**	**Features**	**Dataset**	**Performance Metrics**	**Limitations**
Islam et al. 2023 [31]	2023	Ensemble Model	API, Memory, and Network	CIC-AndMal-2020	Acc 95%	Only consider 45 features out of 141 and performed ML
HashSem et al. [43]	2022	Random forest		CIC-AndMal-2020	Acc 96.86%	Performed ML
[18]	2022	Random forest	APIs and Memory	CIC-AndMal-2020	F1 0.82	Performed binary classification with ML

Amer et al. [19]	2022	KNN, SVM, NB, and decision tree	APIs and permission	Malgenome (1260 malware apps, 2539 benign apps) and Maldroid(1159 samples)	Acc 86% on the Maldroid dataset and precision of 99% on the Malgenome dataset.	Performed binary classification
[20]	2021	Random forest	Memory, API, Network, Battery, logcat, and Process.	CIC-AndMal-2020	Precision 0.984 and 0.983 recall values	Permorfed binary classification
Thangaveloo et al. [39]	2021	Sequential Minimal Optimization (SMO)	System calls, CPU usage, memory usage, and network packets.	Genome Project by Zhou and Jiang	Accuracy 91.7, Precision 93.1, Recall 90.0, Error Rate 8.3	binary classification is performed with ML
Ananya et al.[38]	2020	LR, CART, RF, XGBoost, and DNN	System calls	A total of 4949 samples comprising 2475 benign and 2474 Drebin applications	Recall rate of 24.79 %to 92.2% and a true positive rate is reported from 95.2 and 99.79%.	binary classification is performed with ML
Hwang et al. [41]	2020	Random Forest	API Calls	Sample size of 2,507 from ransomware and 3,886 from the normalware group.	Acc 97.3%, 4.8% FPR and 1.5% FNR	binary classification is performed with ML
Thangaveloo et al. [40]	2020	RF	System calls, CPU usage, memory usage, and network packets.	Genome Project by Zhou and Jiang	accuracy of 96.67%	binary classification is performed with ML
Salehi et al. [37]	2019	Random forest	200 features	9560 samples, from 194 different families, obtained from AndroZoo, Drebin, and	96.7% Accuracy	binary classification is performed with ML

				Malware Genome datasets		
Mahindru et al. [35]	2017	Simple Logistic	123 unique permissions (Default permissions, Development tools, Hardware control, Network communications, Phone Calls, Services that cost money, storage, System tools, accounts, messages, and Personal information)	11,000 samples (out of which 6,971 are malware samples and the rest were benign)	Accuracy 99.7%	Performed binary classification
Alzaylaee et al. [36]	2017	Random Forest	API Calls	Android malware genome project and Intel Security (McAfee Labs).	0.926 F-measure 93.1% TPR 92% FPR	Binary Classification Performed
Bhatia et al. [23]	2017	Random Forest	System calls	50 malicious samples from Genome Project and 50 benign samples from the Google Play Store	88% Accuracy	Performed Binary classification

2.2.2 Android Malware Detection Based on Deep Learning

Numerous deep learning-based methodologies have been suggested to identify Android malware. These approaches are further divided into binary classification and multi-class classification.

2.2.2.1 Binary Classification

Authors have proposed a dynamic analysis method named the Component Traversal method, which can automatically execute the entire Android application code [45]. The study has extracted Linux kernel system calls from executing applications with the help of dynamic analysis. The DL architecture-based model Stacked AutoEncoder (SAEs) is used to learn and identify Android malware patterns. For the experimental study, the dataset collected from Comodo Cloud Security Center consists of 1500 benign apps and 1500 Android malware. A comparison between traditional ML models and the DL model is performed. The DL model outperforms the ML model giving an accuracy of 93.68%.

Authors proposed a DL-Droid, a deep learning-based model that can detect Android malware through dynamic analysis [46]. The dataset consists of 31,125 Android applications that have been used out of which 11505 were malware samples and the rest 19620 were benign. The dataset is obtained from Intel Security (McAfee Labs). Information Gain is used to determine the ranking of features i.e. API calls, Intents, and Permissions. A comparative study is performed between the proposed model and 7 ML models i.e. SVM Linear, SVM RBF, NB, SL, PART, RF, and J48 Decision Tree. DL-Droid outperforms other classifiers by giving an accuracy of 97.76% with dynamic features.

The study developed a DL-based NLP model to identify Android malware [47]. The study developed an internet-based real-world dataset from Drebin and Google Play Store, which is composed of 3567 malicious applications and 3536 benign samples. The study extracts semantic information from system calls as features. The study used an LSTM classifier and developed an NLP model for binary classification. The experimental results of the model achieved 96.6%, 91.3%, 93.7%, and 9.3% of recall, precision, accuracy, and low FPR respectively.

The study performed CNN-based android malware detection [48]. The study also developed a technique of transformation that converts a series of event logs into flattened data with two-dimensional features as CNN works best with flattened data. The study collected non-malignant applications from Google Play and malware application samples were collected from a private company consisting of 423 different kinds of malware categories to develop their dataset. The dataset consists of 17,000 benign and 17,000 malicious applications. The features consist of API Calls. The experimental results show 93.012% accuracy and 12.9% FNR.

The study developed a DL-based CNN model [49]. The dataset consists of 5560 applications, 3536 trusted, and 3564 malware belonging to 179 different families. The system calls were used as features. The experimental results give an accuracy ranging between 0.85 and 0.95.

The study developed a DL-based hybrid model composed of CNN and RNN for malware classification [50]. The authors developed a dataset by running malware samples on a cuckoo sandbox. The API calls are considered features and used as behavior modeling. The study used a numeric label against each malware sample from the VirusTotal web service. The VirusTotal consists of different antivirus and each antivirus signature has its methodology to label the malware sample. The experimental results show 85.6% precision and 89.4% recall which gives better results than ML models.

The study proposed a tool called VizMal, which can picture the implementation steps of Android applications and emphasize the traces of potentially malicious behavior [51]. The dataset consists of non-malignant apps from the Google Play Store and malware apps from Drebin. It consists of 500 Android applications, which consist of 250 non-malware applications and 250 of the remaining applications are malware. The features consist of system calls. The study performed a questionnaire to validate the tool. The experimental results show that VizMal basing LSTM gave better results than other ML classifiers by giving 0.098 FPR and 0.551 FNR.

The study proposed a maxNet framework that consists of RNN architecture [52]. The proposed algorithm can detect malware as soon as it occurs. The study developed a dataset that consisted of 361,265 samples. The experimental results show the 0.96 F1-Score and 96.2% TPr at 1.6% FPr.

2.2.2.2 Multi-Class Classification

The study proposed a DL-based framework to identify Android malware [24]. The study considered network features of the dataset CICAndMal2017 for Android malware analysis. The dataset consists of Adware, Ransomware, Scareware, and SMS Malware categories. The study developed a hybrid model based on CNN and LSTM. The experimental results give the accuracy of 99.79%, 98.90%, and 97.29% on binary, category, and family classification respectively.

The study performed a semi-supervised learning technique for DNN called pseudo-label on the set of labeled and unlabeled observations [32]. The study developed their dataset called CICMalDroid2020, which consists of 17341 samples of five malware categories that are Adware, Banking, SMS, Riskware, and Benign. The experimental results show 97.84% F1-Score and 2.76% FPR.

The study proposed a Two-level Filtering learner Algorithm (TLFL) algorithm to effectively identify mislabeled samples and remove them to enhance the malware detection mechanism [30]. The study used the CCCS-CICandMal-2020 dataset but only considered 4 categories that are Adware, Riskware, Trojan, and Zero-Day. The study only considers 2465 out of 9504 features using ExtraTreesClassifier. The study conducted two experiments, the first experiment only considered Adware, Riskware, and Trojan categories by comparing CNN with the TLFL algorithm which gave an accuracy of 93.4 and 96.7% respectively. The second experiment considered Adware, Riskware, Trojan, and Zero-Day categories by comparing CNN with the TLFL algorithm which gave an accuracy of 83% and 90.61% respectively.

The study proposed the method BIR-CNN to identify Android malware [28]. The proposed system consists of a CNN with batch normalization and inception residual. The study only considered network features from the CICAndMal2017. The dataset is composed of 429 malware samples and 5065 benign. The experiment result shows an accuracy of 99.73% in binary classification, an accuracy of 99.53% in multi-class classification by considering Adware, Ransomware, Scareware, and SMSmalware categories, and an accuracy of 94.38% with 35 malicious families’ classifier 94.38%.

The study proposed a hybrid model that is a Deep Learning classifier called DL Dense Classifier with LSTM [29]. The study used the CIC-AndMal-2020 dataset by considering 12 malware categories that are Adware, Backdoor, File Infector, PUA, Ransomware, Riskware, Scareware, Trojan, Trojan-Banker, Trojan-Dropper, Trojan-SMS, Trojan-Spy, and 141 features. The results from the experiment give an accuracy of 91%.

Table 2. 2 Selected Studies of DL

Ref	**Dates**	**Deep Learning**	**Features**	**Dataset**	**Performance Metrics**	**Limitations**
Gulbarga et al. [29]	2023	DL+LSTM	API, Memory, Network, Battery and logcat	CIC-AndMal-2020	Acc 91% Recall 90% Precision 92% F-Score 91%	Didn't consider Zero Day Category
Liu et al. [28]	2022	BIR-CNN	Network	CIC-AndMal-2020	Acc 99.73% (binary) Acc 99.53% (multi class) Acc 94.38% (35 families)	Only consider Adware, Ransomware, Scareware, and SMSmalware categories
Allogmani et al. [30]	2022	Ensemble Algorithm	2456 features	CICandMal-2020	Acc 96.7% (Adware, Riskware and Trojan) Acc 90.61% (Adware, Riskware, Trojan, and Zero-Day)	Only consider Adware, Riskware, Trojan, and Zero-Day categories
Gohari et al. [24]	2021	CNN+LSTM	Network	CICAndMal2017	Acc of 99.79%, 98.90%, and 97.29% on binary, category, and family classification respectively.	Only consider Adware, Ransomware, Scareware, and SMS Malware
Mahdavifar et al. [32]	2020	DNN	17341 features	CICMalDroid2020	97.84% F1-Score and 2.76% FPR.	Only consider Adware, Banking, SMS, Riskware and Benign

Lorenzo et al. [51]	2020	LSTM	System calls	250 non-malware applications from Google Play and 250 malware applications from Drebin	0.098 FPR and 0.551 FNR	Binary Classification Performed
Alzaylaee et al. [46]	2020	DL-Droid	API calls, Intents, and Permissions	Intel Security (McAfee Labs)	Accuracy 97.76%	Binary Classification
Xiao et al. [47]	2019	LSTM	System calls	3567 malicious applications from Drebin and 3536 benign samples from Google Play	Recall of 96.6% with precision of 91.3%, accuracy of 93.7% and low FPR of 9.3%.	Binary classification
Gronát et al.[52]	2019	RNN	1383 features	dataset consisting of 361,265 samples	0.96 F1-Score and 96.2% TPr at 1.6% FPr.	Binary classification
Hou et al. [45]	2017	Stacked AutoEncoder (SAEs)	System Calls	Comodo Cloud Security Center 1500 benign apps and 1500 Android malware	Accuracy of 93.68%.	Binary Classification
Martinelli et al. [49]	2017	CNN	System calls	3536 trusted and 3564 malware belonging to 179 different families	accuracy ranging between 0.85 and 0.95	Binary classification
Kolosnjaji et al. [50]	2016	CNN + RNN	API Calls		85.6% on precision and 89.4% on recall	Performed Binary classification
Yeh et al. [48]	2016	CNN	API Calls	17,000 benign and 17,000 malicious applications	93.012% accuracy and 12.9% FNR	Binary classification

2.3 Android Malware Detection w.r.t Hybrid Analysis

This section discusses the existing studies concerning Android malware detection using hybrid analysis.

2.3.1 Machine Learning

There are several approaches suggested to discover Android malware using hybrid analysis based on ML.

2.3.1.1 Multiclass Classification

The author proposed Tree Augmented naïve Bayes which is TAN TAN-based hybrid detection system [7]. The study considers 1650 malware Apps from Drebin, AMD, Androozo, and GitHub repositories and 1650 benign Apps from Androozo and Google Play. The study considers Trojan spy, Trojan SMS, Backdoor, Ransomware, and Adware. The study considers API, permissions for static analysis, and system calls for dynamic analysis. The study performed static analysis on samples using the APK tool and then APKs were dynamically analyzed. The experimental study shows 99% accuracy.
The study proposed a novel framework for wrapping feature selection with the combination of Random forest and greedy stepwise (RFGreedySW) [13]. The study used the CIC-InvesAndMal2019 dataset. The static features are permissions and intents while dynamic analysis are API calls and logs. The preprocessing of the feature is done with the proposed framework and later ML models (RF, DT, and SVM RBF) are applied to detect Android malware categories. The study considers Adware, PremuimSMS, Ransomware, Scareware, and SMS malware. The experimental study shows that RF gives an accuracy of 75% on dynamic layer and DT gives an accuracy of 91.8% on static analysis.

2.3.2 Deep Learning

There are several approaches suggested to discover Android malware using hybrid analysis based on DL. These approaches are further divided into binary and multiclass classification.

2.3.2.1 Binary Classification

The author proposed a deep learning-based hybrid analysis malware detection algorithm [25]. The hybrid DL model consists of DBN and GRU. The study used a benign dataset which consist of 7000 samples from Google Play and APKpure. The malware dataset consists of 6298 samples which is further divided into obfuscated malware gathered from

PRAGuard (collection of MalGenome and Contagio minidump datasets) and non-obfuscated malware from VirusShare. The study extracted 351 features which are 303 static features (124 resource features and 179 semantic features) and 48 dynamic features (API functions). The experimental results show an accuracy of 96.82 %.

2.3.2.2 Multiclass Classification

The author proposed a deep learning-based hybrid analysis malware detection model which is DroidDetectMW [12]. The study considers the CICAndMal2017 dataset. The dataset consists of Adware, Ransomware, Scareware, and SMS malware categories. The static features include command strings, API calls, intents, and permissions and dynamic features are system calls, cryptographic activities, dynamic approvals, and information leakage. The study used feature selection techniques which are chi-square, fisher score, and information gain to select relevant features from static and two-stage fuzzy metaheuristic dynamic analysis. The comparative analysis is done with ML models which are KNN, SMO, SVM, RF, DT, NB, and MLP, and the proposed model gives the best result out of them of 95.4% accuracy in malware categories and 88% accuracy in malware families.

The author proposed deep learning-based hybrid analysis to detect Android malware by developing Res7LSTM [14]. The study used a hybrid model which consists of ResNet (Residual Network) and LSTM. The study used the CICAndMal2017 dataset. The APKs are first statically analyzed by binary classification and then resulting malware APKs from it dynamically analyzed by performing multiclass classification. The study performs binary classification with static analysis and multiclass classification with dynamic analysis. The experimental study shows an accuracy of 94.04%.

Table 2. 3 Selected Studies of Hybrid Analysis

Ref	**Dates**	**DL**	**ML**	**Features**	**Dataset**	**Performance Metrics**	**Limitations**
Taher et al. [12]	2023	EHHO-ANN		command strings, API calls, intents and	CIC-AndMal-2017	Acc 95.4% in malware categories	Only consider Adware, Ransomware, Scareware,

				permissions (static) system calls, cryptographic activities, dynamic approvals, and information leakage (dynamic)			and SMSmalware categories
Santosh K. et al [13]	2022		RF, DT, and SVM RBF	permissions and intents (static) and API calls and logs (dynamic)	CIC-InvesAndMal2019	accuracy of 75% on (dynamic) DT acc of 91.8% (static)	Only consider Adware, PremuimSMS, Ransomware, Scareware, SMSmalware
Ding et al. [14]	2021	ResNet + LSTM			CICAndMal2017	Accuracy 94.04%.	Only consider Adware, Ransomware, Scareware, and SMSmalware categories
Surendran et al. [7]	2020		Tree Augmented Naïve Bayes	API, permissions for static analysis, and system calls for dynamic analysis	1650 malware and 1650 benign Apks from Drebin, AMD, Androozo, Github repositories	99% accuracy	Only consider Trojan spy, Trojan-SMS, Backdoor, Ransomware and Adware
Lu et al. [25]	2020	DBN + GRU		124 resource features 179 semantic features and 48 dynamic features (API functions)	7000 samples from Google Play and APKpure. 6298 samples from PRAGuard	Acc 96.82 %.	Binary classification performed

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

In this section, we concentrate on the preprocessing of the dataset. A suitable dataset is relatively important for the detection of malware. The limitations of available studies are discussed in chapter 2. To overcome the limitations, we develop a deep learning model for multiclass classification.

This chapter explains the preprocessing steps of the dataset as well as the development of a deep learning model. Additionally, the feature selection processes are also presented in this chapter.

The preprocessing phase includes the cleaning of data, transformation, and normalization, extracting, and selecting features. It is the prime phase and it needs a great amount of effort and time. It not only augments the efficiency of the model system but also affects the storage of the system [53].

3.2 Research Methodology

We performed applied research methodology which is presented in Figure 3.1. In this applied research, the problems regarding CIC dataset cleaning, feature selection, and multi-class classification of malware categories are discussed and solutions are implemented.

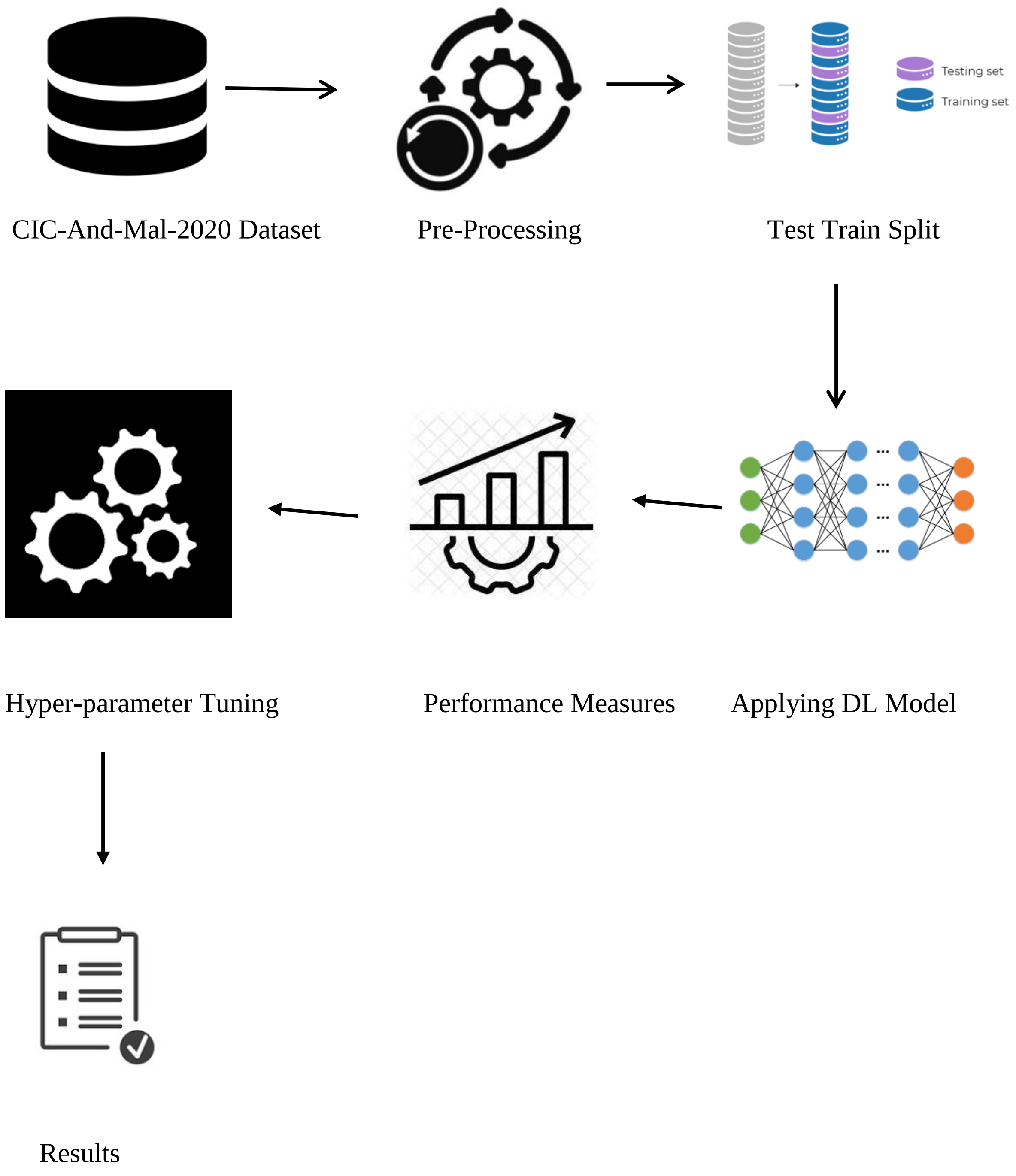


Figure 3. 1 Proposed Methodology

3.3 Proposed Architecture

The proposed architecture is presented in Figure 3.2. Following are the research of proposed research.

1. Analysis of existing proposed research of malware detection concerning ML and DL is performed.
2. The problems are identified regarding the selection of features and multi-class classification of malware android classes.
3. The Android malware detection system for multi-class classification using deep learning has been developed.
4. Existing studies based on deep learning and machine learning are studied.
5. Pre-processing of the dataset is performed which consists of data cleaning.
6. Statistical analysis of the dataset is performed to obtain the features.
7. Features are passed as input to the DL algorithm. This study implemented a DNN.
8. Evaluation of the models is performed with different evaluation measures.

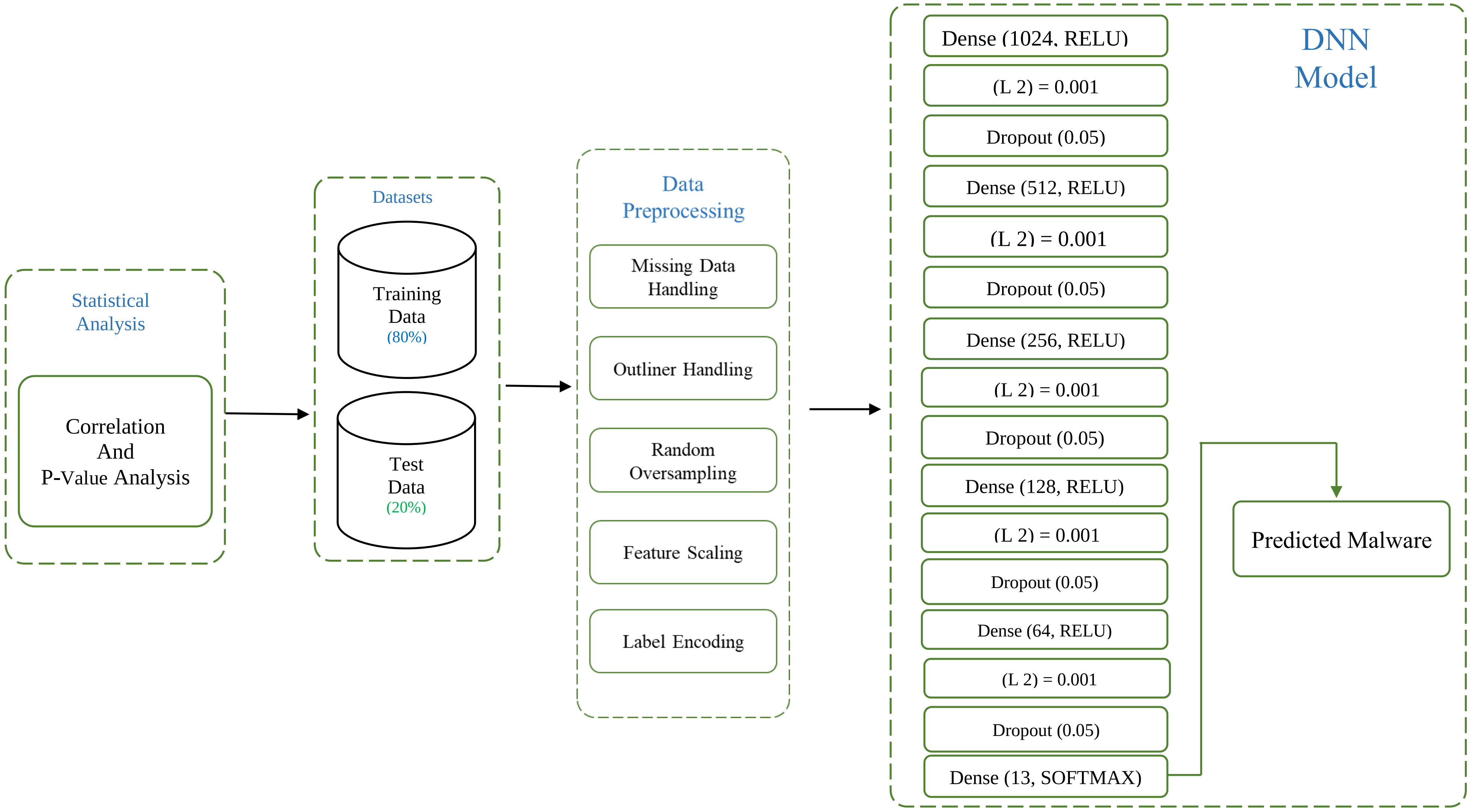


Figure 3. 2 Proposed Deep Neural Network Architecture

3.4 Dataset

We use CIC-And-Mal Dataset 2020, which is a publicly available dataset developed by the Canadian Institute for Cybersecurity. The dataset consists of 200k benign and 200k malware samples with 14 malware categories and 191 distinguished malware families. We only consider the dynamic analysis part of the dataset which contains 141 features that are further classified into logcat (6 features), API (105 features), Memory (23 features), battery (2 features), Network (4 features) and Process (1 feature). The dataset consists of 53439 samples that are classified into 14 malware types including No_Category, Riskware, Adware, Trojan, Zero_day, Ransomware, Trojan_Spy, Trojan_SMS, Potentially Unwanted Apps (PUA), Scareware, File Infector, Trojan_Banker, Trojan_Dropper. The division of malware categories labeled in the CIC dataset is shown in Figure 3.3.

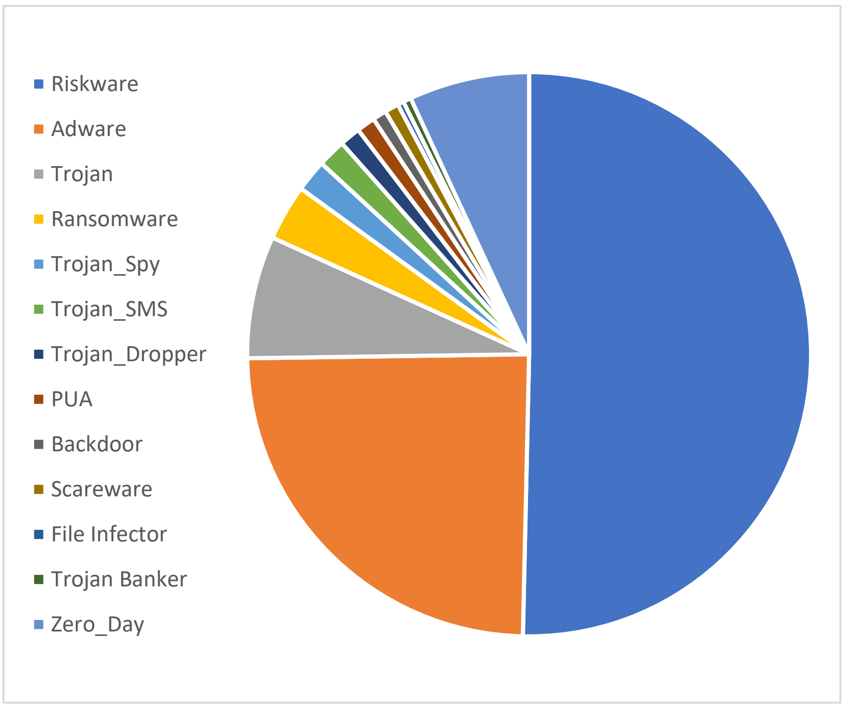


Figure 3. 3 Division of various malware categories in the dataset

3.5 Preprocessing Phase

The pre-processing phase is an important part of data analysis and deep learning. It includes normalization, data transformation, label encoding, and removal of outliers from raw data.

3.5.1 Missing Data Handling

The use of missing value imputation is important as a dataset with missing values cannot present accurate results. Imputation techniques are applied to fill in missing values of the dataset to make it more comprehensive for the results of model training. The objective is to reduce the impact of data that is missing on the validity of dataset results. We divided the dataset into 10 data frames, each data frame consisting of 10 columns or features to find the missing values from the dataset. We have used the mean approach from ScikitLearn Library [54]. We have applied the "isnull().sum()" method to identify the mean of the data frames. The mean of all data frames is zero resulting in no missing values present in the dataset.

3.5.2 Data Normalization

Normalization is the process of transforming different variables of the dataset into a standardized scale so that each feature can contribute equally to the training of the model [55]. We have performed both feature scaling and outlier removal at the dataset.

3.5.2.1 Feature Scaling

In this study, we applied MinMax Scaler from ScikitLearn Library. MinMax Scaler takes the maximum value as 0 and the minimum value as 1 and linearly scales the rest of the features in between them using the formula as shown below.

$$X_{normalized} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

3.5.2.2 Outlier Handling

The observation or finding that differs from the majority of the data present in the dataset is called an outlier [56]. As we are working on a dynamic nature dataset which is CIC-AndMal-2020, it is highly skewed in nature. We have applied the Z-score to detect outliers and later removed them using the formula as shown below. The vast difference can be seen in Table 3.1 after applying outlier handling.

Malware Category	**Number of Samples with Outliers**	**Number of Samples After Removing Outliers**
Zero Day	2146	1344
Trojan _Spy	1039	642
Trojan	4025	1935
Scareware	424	179
Ransomware	1550	1017
PUA	625	284
File_Infector	119	95
Trojan_Banker	123	81
Trojan_Dropper	733	494
Adware	5142	2930
Backdoor	546	296
Riskware	6792	3946

$$X_{standardized} = \frac{X - \mathrm{mean}(X)}{std(X)}$$

Table 3. 1 Outlier Handling of Dataset

3.5.3 Label Encoding

Label encoding uses the Keras library to transform categorical labels into numerical values through one hot encoding process [57]. The CIC dataset has malware categories as labels which are converted into numerical values so that the machine can understand it for multi-class classification.

3.5.4 Random Oversampling

The CIC-And-Mal 2020 dataset is highly imbalanced. Riskware has the most frequency in the dataset whereas Trojan-Banker has the least frequency. The 1:58 is the ratio present between them. Such imbalance figures affect the training of the model which eventually will affect the prediction results. The approach to tackle the class imbalance problem is to resample the dataset randomly. We have applied the Synthetic Minority Oversampling Technique which is also known as SMOTE to balance the dataset. It generates artificial instances from the minority class of the dataset to balance the class division of the dataset [58]. The division of Android malware samples is shown in Table 3.1 and Figure 3.5 shows the effect of applying SMOTE.

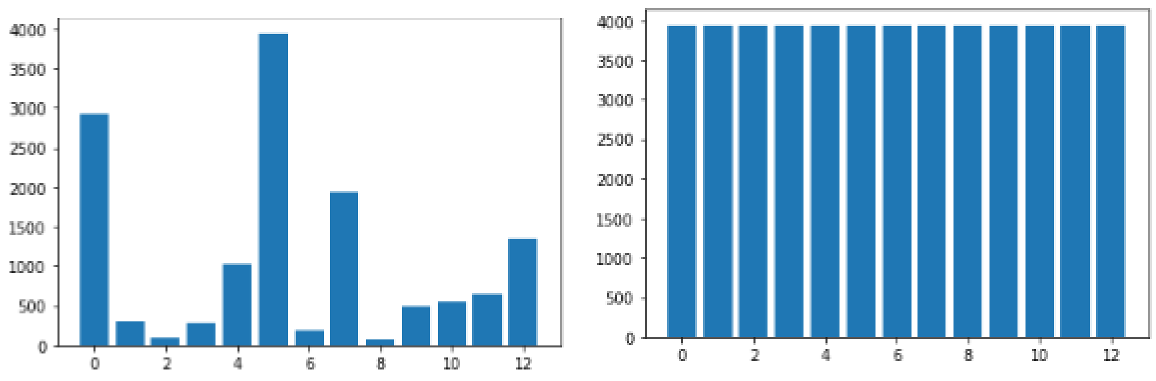


Figure 3. 4 Results of Applying SMOTE on the Dataset

3.6 Train and Test Splitting

The training and testing split for cross-validation is accepted as the gold standard in the training of ML models [59]. For a fair comparison, the dataset in deep learning is divided into two parts: training and testing data or it can be called validation data. We use evaluation metrics to assess the model on testing data after training it on training data. We divided the dataset into 80 20 percent ratio of training and testing data respectively.

3.7 Model Development Phase

3.7.1 Deep Learning Algorithm

In this phase, we applied a DL-based algorithm which is a Deep Neural Network (DNN) on the CIC-And-Mal 2020 dataset. To evaluate the performance and find the best possible results, we applied different numbers of layers, hyper parameters, and activation functions to develop the model. We iterate these steps until we get satisfying results. We selected DNN algorithm to find optimal model for multi classification of malware classes.

3.7.2 Model Parameters

Activation functions, the number of layers, and hyperparameter tuning constitute the model parameter.

3.7.2.1 Hyperparameter Tuning

The hyper parameter tuning is the most important phase of deep learning model development. There is no set standard to select the hyperparameter, it is mostly based on the nature and volume of the training dataset. To select the suitable parameters the constraints of tradeoffs and limitations of memory were to be also considered [60]. In this research, we have executed several experiments to get satisfactory results, we analyze the parameter's values before recording the findings. As the size of the dataset increased after applying SMOTE, it took 1 to 2 days to train the single model.

The hyperparameter consists of the following.

1) **Batch Size:** The amount of training samples used in one iteration is defined by the batch size. The training dataset is divided into multiple batches and based on the error calculated on each batch the model is updated [61]. We apply different batch sizes from 64 up to 330. We performed several experiments and concluded 320 batch size as it gives the best possible results. Although the larger the batch size the more amount of memory space and processing time it will need
2) **Epochs:** The epoch indicates one pass through which the model passes through the entire training data and calculates the loss function. Each epoch consists of one forward and backward propagation. After each epoch, the model updated its parameters according to the loss function [62]. In our research, we used the EarlyStopping function to determine suitable epoch numbers for the training of the

model and to prevent overfitting of the model [63]. The EarlyStopping consists of two parts.

- **Monitor Metric:** It consists of a metric that we have to monitor which is in our case loss function.
- **Patience Level:** It consists of a value that we set and based on which it analyzes sequential epochs with no enhancement on monitored metrics. We set the value of 10.
- **Early Stopping Condition:** When the condition is met according to the patience parameter, the training of the model stops immediately.

3) **Drop Out:** It is used to avoid overfitting in neural networks [64]. We experiment with different dropout values from 0.05 up to 0.2.
4) **Optimizer Function:** The Optimizer function is used to calculate the loss function in backpropagation. In our research, we use Adam as an optimizer function.

3.7.2.2 Hidden Layers

The complexity of neural networks depends on the number of layers. In our study, we have experimented with 2 layers up to 9 layers architecture and analyzed the performance of models based on hidden layers.

3.7.2.3 Activation Function

The activation function is applied to each neuron of the neural network. It is a mathematical operation that describes the output of neurons concerning the weight of input features. The generally applied activation functions in Deep Neural Networks are sigmoid, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU).

Sigmoid: It is usually used in the output layer of classification either binary or multiclass classification. It transforms a vector of numbers into probability distribution to predict results and its range encompasses (0, 1) [65]. It is defined as the following expression.

$$S(x) = \frac{1}{1 + e^{-1}}$$

Rectified Linear Unit (ReLU): ReLU improves DNN learning on multi-dimensional input. The benefit of ReLU in its activation functions is that it doesn't require any costly computations, it just performs multiplication and comparison [66]. We use ReLU on hidden layers of DNN, it trains significantly deep networks as compared to Sigmoid and Tanh. It is calculated by the following expression.

$$RELU\ (x) = \max(0, x)$$

Hyperbolic Tangent: It is defined as a ratio of hyperbolic cosine and sine tangent [67].

$$\text{Tan h} = \frac{e^{x} - e^{-x}}{e^{x} - e^{-x}}$$

3.8 Evaluation Measures

In this research, we are implementing multi-class classification of malware classes and we evaluate the proposed model using standard evaluation metrics that are F1-Score, recall, confusion matrix, accuracy, precision, and AUC-ROC analysis. Detailed explanations and standard formulas to measure them are as follows.

3.8.1 Accuracy

Accuracy gives a ratio of how frequently the model classifies correct instances which makes the overall performance of the model. It is measured by the below equation.

$$Accuracy = \frac{true\ Positive + True\ Negative}{Total\ Instances}$$

3.8.2 Precision

Prediction emphasizes on ratio of accurately positive predictive instances to total positive instances. It is measured by the following metric.

$$Precision = \frac{True\ Positive}{True\ Positive + false\ Positive}$$

3.8.3 Recall

Recall metric is the ratio of accurately classified positive occurrences divided by the total occurrence of positively identified. It is crucial when false-negative instances are more significant than false negatives. It is calculated by the following formula.

$$Recall = \frac{True\ Positive}{True\ Positive + false\ Negative}$$

3.8.4 F1- Score

The mean of precision and recall value is the F1-score metric. It has ranges of 0 to 1. 1 represents optimal precision and recall, and 0 represents a value of recall and precision will be as low as possible. The mathematical expression of the F1-score is as follows.

$$F1 - Score\ = \frac{2 * Rec * Prec}{Rec + Prec}$$

3.8.5 AUC-ROC Analysis

The area Under the Reciever Operating Characteristic Curve is known as AUC-ROC. The curve is a graphical representation of the classification algorithm. It is the ratio of the True positive rate and the False Positive Rate[68].

3.8.6 Confusion Matrix

The Confusion matrix consists of a table that gives the synopsis of many actual and expected values that the model of classification generates for multi-class classification performance.

CHAPTER 4

RESULTS AND EVALUATION

This section describes the DL-based detection of malware models concerning dynamic analysis. The model consists of a DNN algorithm. We considered the dynamic analysis part of the CIC-AndMal-2020 dataset and performed featured selection on it. We performed correlation-based feature selection. This chapter consists of implementation details of model development and evaluation of that model.

4.1 Statistical Analysis of the Dataset

4.1.1 P-Value Analysis

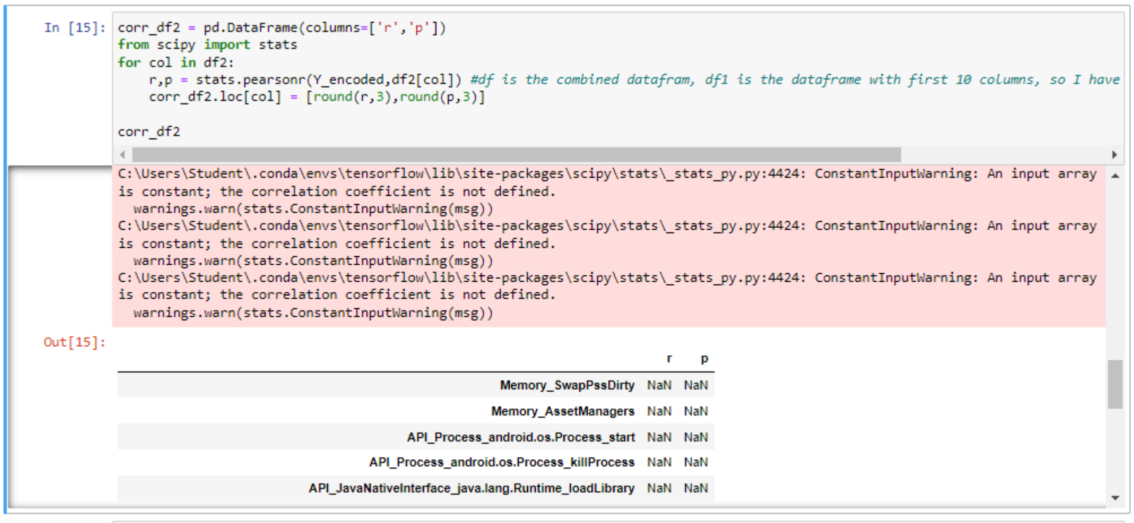
We perform a p-value analysis on the dataset. P value stands for probability value and it signifies whether there is any association between two variables or not. The association depends on the p-value, if the value is more than 0.05 then there is the likelihood to be no association and vice versa. Under statistics gold standard, states that a p-value lower than 0.05 is considered significant [69].

4.1.2 Correlation

The dataset that has features with similar programming can correlate. Correlation is the method to identify the relationship between two variables. We have performed the Spearman correlation *(p)* and Pearson correlation *(r)* to measure the strength and direction of the linear relationship between features. It can range from -1 to 1.

Correlation coefficients that range from 0.7 to 0.9 are considered highly correlated. To train the model the redundant values need to be removed beforehand as they can affect the results later. The value of >= ± 0.75 indicates a high correlation and allows us to remove redundancy [70].

We find most of the features correlated except for a few. The python gives a warning that is shown in Figure 4.1. According to the error message input array of the columns is constant which means that 27 of the features out of 141 have constant values for samples. So according to correlation and probability value, we dropped them out of the dataset and performed experiments on the rest of the 114 features which is presented in Figure 4.2.



```
In [15]: corr_df2 = pd.DataFrame(columns=['r','p'])
         from scipy import stats
         for col in df2:
             r,p = stats.pearsonr(Y_encoded,df2[col]) #df is the combined datafram, df1 is the dataframe with first 10 columns, so I have
             corr_df2.loc[col] = [round(r,3),round(p,3)]

         corr_df2
```

```
C:\Users\Student\.conda\envs\tensorflow\lib\site-packages\scipy\stats\_stats_py.py:4424: ConstantInputWarning: An input array
is constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(msg))
C:\Users\Student\.conda\envs\tensorflow\lib\site-packages\scipy\stats\_stats_py.py:4424: ConstantInputWarning: An input array
is constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(msg))
C:\Users\Student\.conda\envs\tensorflow\lib\site-packages\scipy\stats\_stats_py.py:4424: ConstantInputWarning: An input array
is constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(msg))
```

Out[15]:

	r	p
Memory_SwapPssDirty	NaN	NaN
Memory_AssetManagers	NaN	NaN
API_Process_android.os.Process_start	NaN	NaN
API_Process_android.os.Process_killProcess	NaN	NaN
API_JavaNativeInterface_java.lang.Runtime_loadLibrary	NaN	NaN

Figure 4. 1 Correlation of dropped features

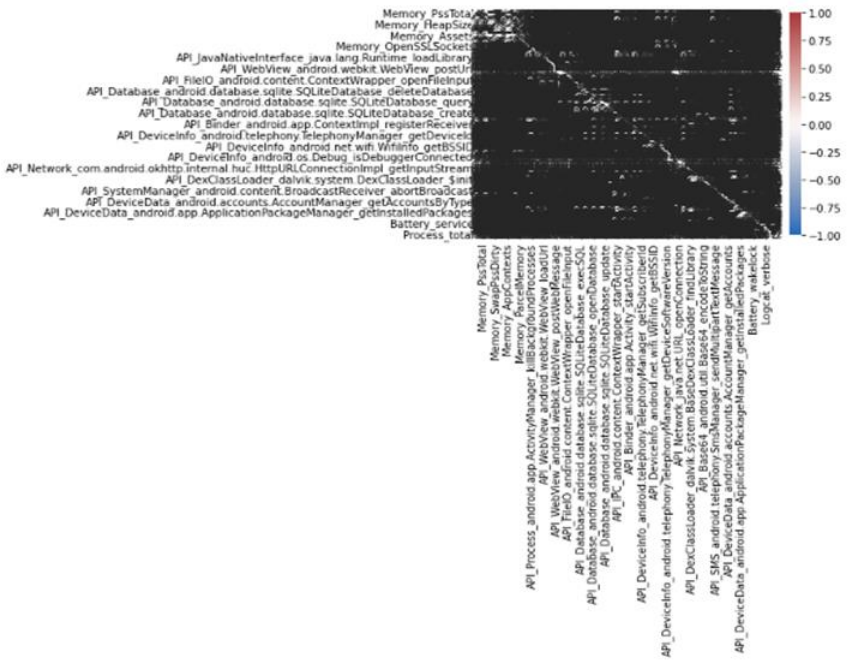


Figure 4. 2 Correlation of 114 features

4.2 Experimental Setup

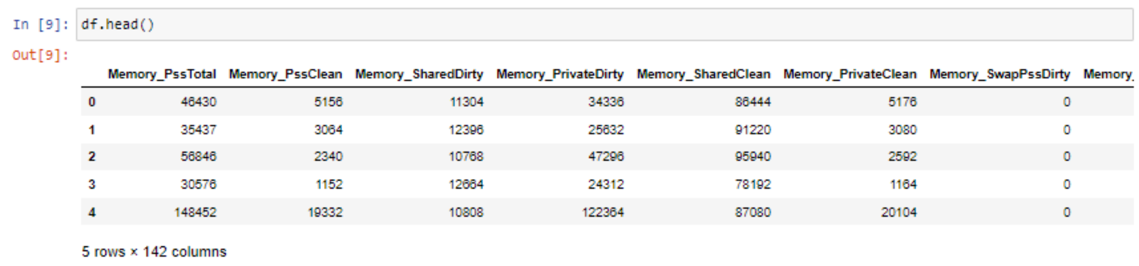
We utilize Jupyter Notebook for the execution of experiments of the proposed algorithm. Python is commonly used with ML and DL as Python provides numerous built-in libraries for visualization, classification, and analysis of data. Sklearn is a machine learning library that is used to perform machine learning calculations. The NumPy library deals with multi-dimensional arrays which is important for ML numerical operations and it also stores data at run time. Panda library helps in data manipulation by providing data as data frames. Tensor flow library is developed by Google which we used to develop the deep neural network. It is a deep-learning library that is utilized to develop neural networks. We also used Keras in the development of model development. Keras is an API for neural networks developed to act as an interface in developing neural networks for several frameworks of deep learning.

4.2.1 Pre-Processing Phase

This phase consists of data cleaning by removing outliers and data normalization.

4.2.1.1 Dataset

We consider dynamic analysis part of the CIC-AndMal-2020 dataset. Which consists of 141 features and 13 malware categories that are Backdoor, Trojan, PUA, Scareware, Trojan-Dropper, Trojan-Spy, Zero-day, Ransomware, Trojan-SMS, Scareware, Riskware, File Infector, and Adware. Figures 4.3 and 4.4 explain the structure and description of a dataset.

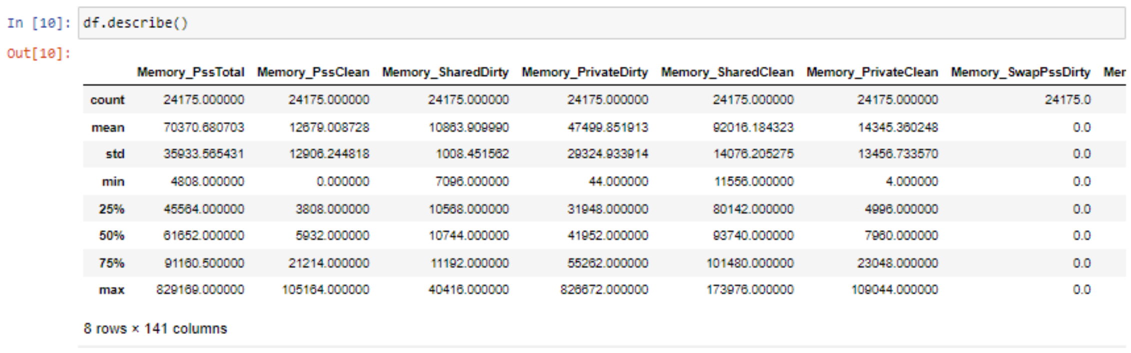


```
In [9]: df.head()
Out[9]:
```

	Memory_PssTotal	Memory_PssClean	Memory_SharedDirty	Memory_PrivateDirty	Memory_SharedClean	Memory_PrivateClean	Memory_SwapPssDirty	Memory_
0	46430	5156	11304	34336	86444	5176	0	
1	35437	3064	12396	25632	91220	3080	0	
2	56846	2340	10768	47296	95940	2592	0	
3	30576	1152	12664	24312	78192	1164	0	
4	148452	19332	10808	122364	87080	20104	0	

5 rows × 142 columns

Figure 4. 3 Dataset Structure



```
In [10]: df.describe()
Out[10]:
```

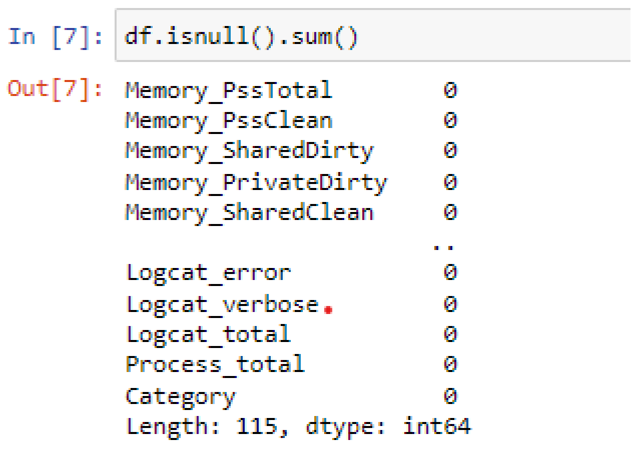
	Memory_PssTotal	Memory_PssClean	Memory_SharedDirty	Memory_PrivateDirty	Memory_SharedClean	Memory_PrivateClean	Memory_SwapPssDirty	Mer
count	24175.000000	24175.000000	24175.000000	24175.000000	24175.000000	24175.000000	24175.0	
mean	70370.680703	12679.008728	10863.909990	47499.851913	92016.184323	14345.360248	0.0	
std	35933.565431	12906.244818	1008.451562	29324.933914	14076.205275	13456.733570	0.0	
min	4808.000000	0.000000	7096.000000	44.000000	11556.000000	4.000000	0.0	
25%	45564.000000	3808.000000	10568.000000	31948.000000	80142.000000	4996.000000	0.0	
50%	61652.000000	5932.000000	10744.000000	41952.000000	93740.000000	7960.000000	0.0	
75%	91160.500000	21214.000000	11192.000000	55262.000000	101480.000000	23048.000000	0.0	
max	829169.000000	105164.000000	40416.000000	826672.000000	173976.000000	109044.000000	0.0	

8 rows × 141 columns

Figure 4. 4 Description of Dataset

4.2.1.2 Missing Data Handling

We first performed missing data handling by identifying null values present in the dataset. We did not find any null values in the dataset as presented in Figure 4.5.

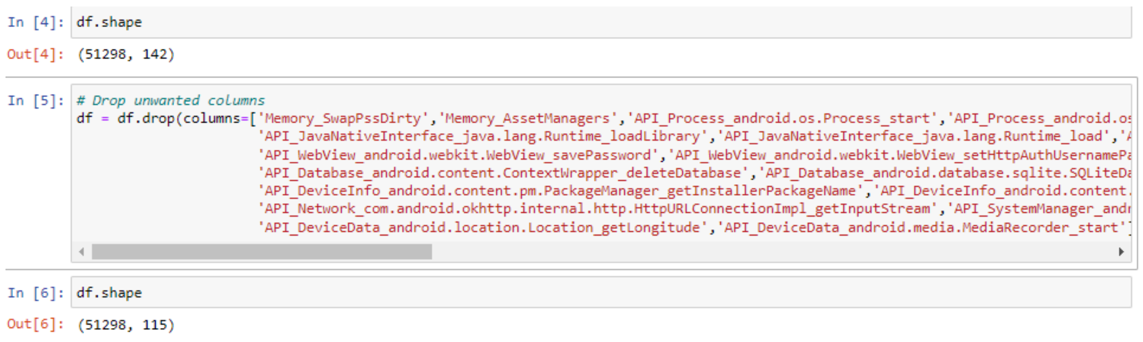


```
In [7]: df.isnull().sum()

Out[7]: Memory_PssTotal        0
        Memory_PssClean        0
        Memory_SharedDirty     0
        Memory_PrivateDirty    0
        Memory_SharedClean     0
                              ..
        Logcat_error           0
        Logcat_verbose         0
        Logcat_total           0
        Process_total          0
        Category               0
        Length: 115, dtype: int64
```

Figure 4. 5 Dataset without Null Values

4.2.1.3 Dropped Features

According to correlation we dropped 27 features out of 141 performed experiments and trained the proposed model on the remaining 114 features as presented in Figure 4.6.



```
In [4]: df.shape
Out[4]: (51298, 142)

In [5]: # Drop unwanted columns
        df = df.drop(columns=['Memory_SwapPssDirty','Memory_AssetManagers','API_Process_android.os.Process_start','API_Process_android.os
                              'API_JavaNativeInterface_java.lang.Runtime_loadLibrary','API_JavaNativeInterface_java.lang.Runtime_load','
                              'API_WebView_android.webkit.WebView_savePassword','API_WebView_android.webkit.WebView_setHttpAuthUsernamePa
                              'API_Database_android.content.ContextWrapper_deleteDatabase','API_Database_android.database.sqlite.SQLiteDa
                              'API_DeviceInfo_android.content.pm.PackageManager_getInstallerPackageName','API_DeviceInfo_android.content.
                              'API_Network_com.android.okhttp.internal.http.HttpURLConnectionImpl_getInputStream','API_SystemManager_andr
                              'API_DeviceData_android.location.Location_getLongitude','API_DeviceData_android.media.MediaRecorder_start']

In [6]: df.shape
Out[6]: (51298, 115)
```

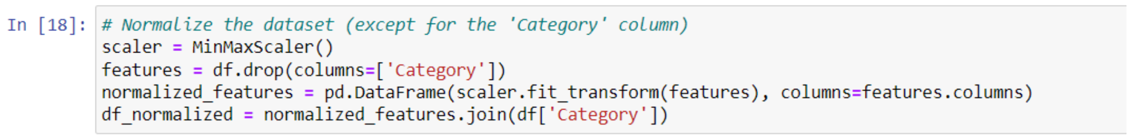
Figure 4. 6 Dropped Features

4.2.1.4 Data Normalization

The dataset is in the form of an alphanumeric state so we performed normalization on it to transform it into a standard scale.

4.2.1.4.1 Feature Scaling

We applied the MinMax scaler to transform features between 0 and 1 as presented in Figure 4.7.

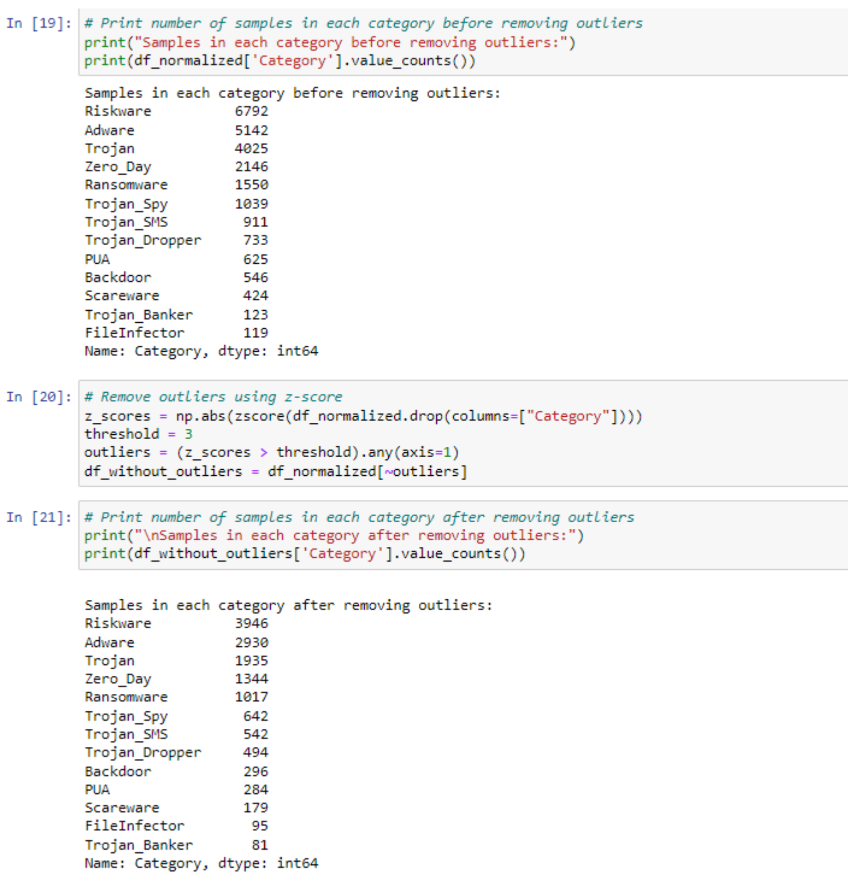


```
In [18]: # Normalize the dataset (except for the 'Category' column)
         scaler = MinMaxScaler()
         features = df.drop(columns=['Category'])
         normalized_features = pd.DataFrame(scaler.fit_transform(features), columns=features.columns)
         df_normalized = normalized_features.join(df['Category'])
```

Figure 4. 7 Feature Scaling of Dataset

4.2.1.4.2 Outlier Handling

The CIC-AndMal-2020 is highly skewed in nature so we performed outlier handling to tackle this problem as presented in Figure 4.8.

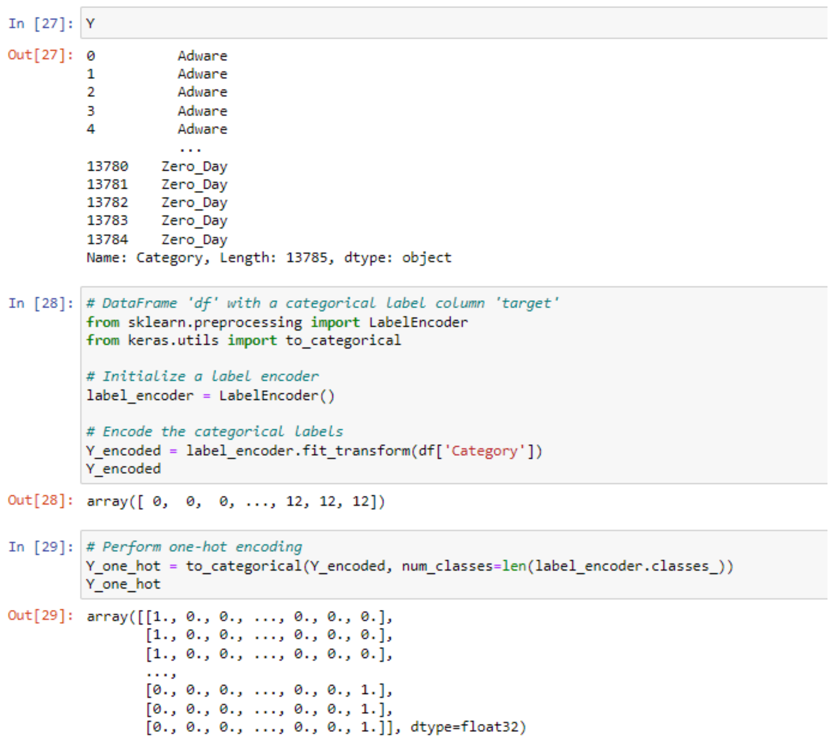


```
In [19]: # Print number of samples in each category before removing outliers
         print("Samples in each category before removing outliers:")
         print(df_normalized['Category'].value_counts())

         Samples in each category before removing outliers:
         Riskware          6792
         Adware            5142
         Trojan            4025
         Zero_Day          2146
         Ransomware        1550
         Trojan_Spy        1039
         Trojan_SMS         911
         Trojan_Dropper     733
         PUA                625
         Backdoor           546
         Scareware          424
         Trojan_Banker      123
         FileInfector       119
         Name: Category, dtype: int64

In [20]: # Remove outliers using z-score
         z_scores = np.abs(zscore(df_normalized.drop(columns=["Category"])))
         threshold = 3
         outliers = (z_scores > threshold).any(axis=1)
         df_without_outliers = df_normalized[~outliers]

In [21]: # Print number of samples in each category after removing outliers
         print("\nSamples in each category after removing outliers:")
         print(df_without_outliers['Category'].value_counts())


         Samples in each category after removing outliers:
         Riskware          3946
         Adware            2930
         Trojan            1935
         Zero_Day          1344
         Ransomware        1017
         Trojan_Spy         642
         Trojan_SMS         542
         Trojan_Dropper     494
         Backdoor           296
         PUA                284
         Scareware          179
         FileInfector        95
         Trojan_Banker       81
         Name: Category, dtype: int64
```

Figure 4. 8 Outlier Handling of Dataset

4.2.1.4.3 Label Encoding

We performed Label encoding to transform categorical data into numerical data as presented in Figure 4.9.



```
In [27]: Y
Out[27]: 0        Adware
         1        Adware
         2        Adware
         3        Adware
         4        Adware
                   ...
         13780    Zero_Day
         13781    Zero_Day
         13782    Zero_Day
         13783    Zero_Day
         13784    Zero_Day
         Name: Category, Length: 13785, dtype: object

In [28]: # DataFrame 'df' with a categorical label column 'target'
         from sklearn.preprocessing import LabelEncoder
         from keras.utils import to_categorical

         # Initialize a label encoder
         label_encoder = LabelEncoder()

         # Encode the categorical labels
         Y_encoded = label_encoder.fit_transform(df['Category'])
         Y_encoded
Out[28]: array([ 0,  0,  0, ..., 12, 12, 12])

In [29]: # Perform one-hot encoding
         Y_one_hot = to_categorical(Y_encoded, num_classes=len(label_encoder.classes_))
         Y_one_hot
Out[29]: array([[1., 0., 0., ..., 0., 0., 0.],
                [1., 0., 0., ..., 0., 0., 0.],
                [1., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 1.],
                [0., 0., 0., ..., 0., 0., 1.],
                [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

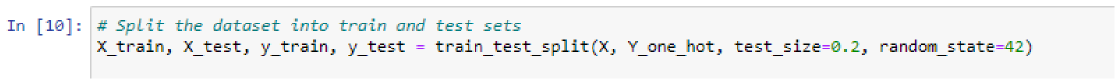
Figure 4. 9 Label Encoding of Malware Categories

4.3 Model Development

The model development phase consists of the following steps.

4.3.1 Train Test Split

The dataset is split into 80:20 ratios for train data and test data respectively.



```
In [10]: # Split the dataset into train and test sets
         X_train, X_test, y_train, y_test = train_test_split(X, Y_one_hot, test_size=0.2, random_state=42)
```

Figure 4. 10 Tran Test Split

4.3.2 Algorithm for Deep Neural Network

The DNN model that is proposed, is composed of 114 features as input to input layer. It consists of 5 hidden layers. The first hidden layer has 1024 neurons and 5 hidden layers consist of 64 neurons. The value of the output layer is 13 as we are performing multi-class classification.

4.3.2.1 DNN Algorithm

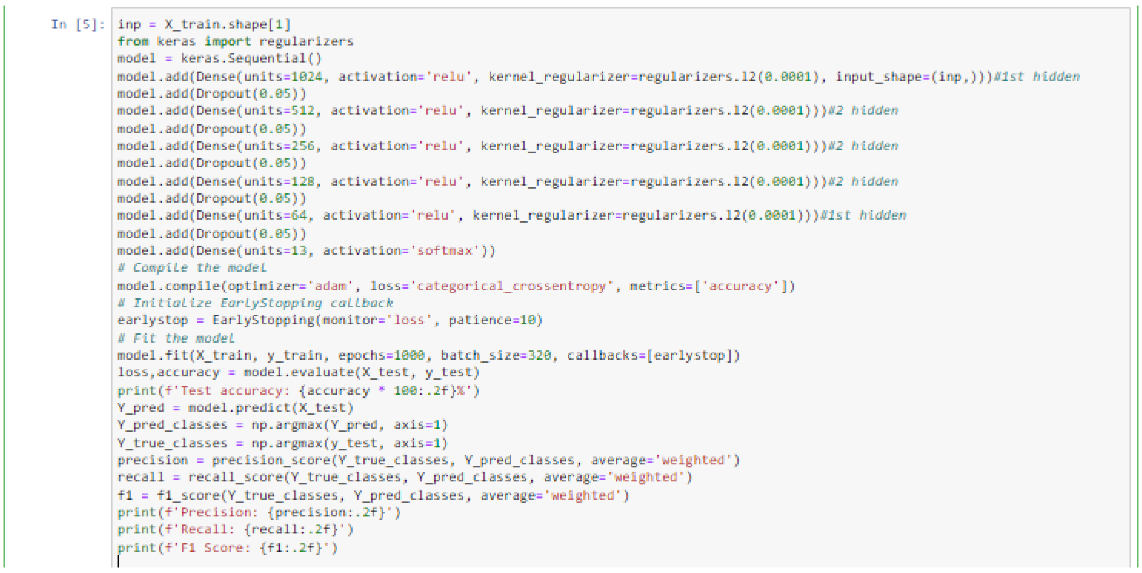
```
$\textbf{Input}: (FS)_n$ as a list of features and labels$\textbf{Output}$: Multi-class classification of malware categories

DNN Model: Create a deep neural network with hidden layers represented as L1…Ln
L1 : inp → Dense (1024, 'relu', 'l2(0.0001)') → Dropout (0.05)
L2 : Dense (512, 'relu', 'l2(0.0001)') → Dropout (0.05)
L3 : Dense (256, 'relu', 'l2(0.0001)') → Dropout (0.05)
L4 : Dense (128, 'relu', 'l2(0.0001)') → Dropout (0.05)
L5 : Dense (64, 'relu', 'l2(0.0001)') → Dropout (0.05)
Output Layer
Dense (13, 'softmax')
Training:
     model.compile(optimizer='adam','loss=categorical_crossentropy',
     metrics=['accuracy'])
     Early stopping: monitor 'loss',patience =10
    Train(NN, X_train, y_train, 'epochs','batch_size=320',callbacks=[earlystop])
Evaluate(Model, X_test,Y_test)

Return Multi-class classification of malware categories based on performance metrics
```

4.4 Results

The implementation of DNN is done by Python 3.10 version and Keras framework. Furthermore, TensorFlow, Numpy, Sklearn, and Keras Turner were implemented. Jupyter Notebook from Anaconda is used as an environment to develop the proposed algorithm while Matplot library is used for visualization. We execute over 100 experiments with a combination of various hidden layers (neurons), and hyper-parameters. The observations are as follows.

4.4.1 DNN Results

To examine the impact of epochs, batch size, dropout, optimizer, layer count, and activation functions. Different results have been acquired using the DNN algorithm.



```
In [5]: inp = X_train.shape[1]
        from keras import regularizers
        model = keras.Sequential()
        model.add(Dense(units=1024, activation='relu', kernel_regularizer=regularizers.l2(0.0001), input_shape=(inp,)))#1st hidden
        model.add(Dropout(0.05))
        model.add(Dense(units=512, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))#2 hidden
        model.add(Dropout(0.05))
        model.add(Dense(units=256, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))#2 hidden
        model.add(Dropout(0.05))
        model.add(Dense(units=128, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))#2 hidden
        model.add(Dropout(0.05))
        model.add(Dense(units=64, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))#1st hidden
        model.add(Dropout(0.05))
        model.add(Dense(units=13, activation='softmax'))
        # Compile the model
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        # Initialize EarlyStopping callback
        earlystop = EarlyStopping(monitor='loss', patience=10)
        # Fit the model
        model.fit(X_train, y_train, epochs=1000, batch_size=320, callbacks=[earlystop])
        loss,accuracy = model.evaluate(X_test, y_test)
        print(f'Test accuracy: {accuracy * 100:.2f}%')
        Y_pred = model.predict(X_test)
        Y_pred_classes = np.argmax(Y_pred, axis=1)
        Y_true_classes = np.argmax(y_test, axis=1)
        precision = precision_score(Y_true_classes, Y_pred_classes, average='weighted')
        recall = recall_score(Y_true_classes, Y_pred_classes, average='weighted')
        f1 = f1_score(Y_true_classes, Y_pred_classes, average='weighted')
        print(f'Precision: {precision:.2f}')
        print(f'Recall: {recall:.2f}')
        print(f'F1 Score: {f1:.2f}')
```

Figure 4. 11 DNN Model Evaluation

Epochs: We have used the early stopping function to find out the best possible combination of epochs to give the combination of the best possible results. We have given the maximum number of values 1000 to the epochs so that the early stopping function will train the model as much as possible and stop at when its conditions are met. Table 4.1 explains the results.

Table 4. 1 Effect of Epochs on Dataset

Layers	**Epochs**	**DNN**			
		Accuracy	**Precision**	**Recall**	**F1-Score**
5	203/1000	93.9 %	94 %	94 %	94 %

Batch Size: We experimented with different batch sizes to evaluate the impact of batch size. The minimum batch size we consider is 64 and 320 batch size gives the best results. Table 4.2 explains the results.

Table 4. 2 Effect of Batch Size on Dataset

Layers	**Batch Size**	**DNN**			
		Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1-S (%)**
5	64	92	93	93	93
5	128	91.2	92	92	92
5	256	92.84	93	93	93
5	320	93.9	94	94	94

Drop Out: We examine the effect of dropout with different dropout rates. We experimented with distinct dropouts presented in Table 4.3.

Table 4. 3 Effect of Drop Out on Dataset

Layers	**Drop Out**	**DNN**			
		Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1-S (%)**
5	0.05	93.9	94	94	94
5	0.1	93.8	93	93	93
5	0.2	92.38	92	92	92
5	0.3	90.1	90	90	90

Layers: The effect of several layers on the model has been examined. The results of experiments with different numbers of layers are presented in Table 4.4

Table 4. 4 Effect of Layers on Dataset

Layers	**Training Accuracy**	**DNN**			
		Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1-S (%)**
3	92.9	92.4	93	93	93
4	96.16	93.3	93	93	93
5	96.3	93.9	94	94	94
6	94.8	91.5	92	92	92
7	94.8	92	93	93	93

4.4.2 Discussion

We analyze the effect of parameters of the DNN algorithm by performing several experiments that give significant predictions to elaborate the results.

4.4.2.1 Deep Neural Network

The proposed DNN algorithm has obtained effective findings by modifying the parameters of the model based on experimental results.

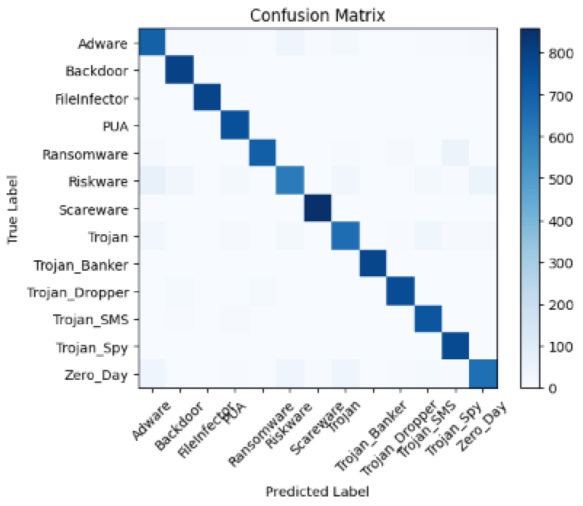
Number of Layers Effect on Model: The number of hidden layers has significantly impacted the training accuracy of the model. When we increase the number of hidden layers, the value of accuracy, precision, recall, and F1-Score also increases or decreases accordingly. We use a geometric sequence with a ratio of 2 in adjusting the value of neurons in models. The minimum number of neurons we use is 64 and the maximum is 4096. We selected 5 5-layer models out of all the experiments we conducted as it give better results of evaluation metrics than others. The 5-layer model has a training accuracy of 96.3 as compared to 3 layers which have a training accuracy of 92 and on 6 layers model the training accuracy started to decrease from 96 to 94 as well as accuracy, precision, recall and F1-score value also decreased as presented in Table 4.4. We achieve a high accuracy of 93.9 at 5 layers which consist of neurons that are 1024, 512, 256, 128, and 64 respectively.

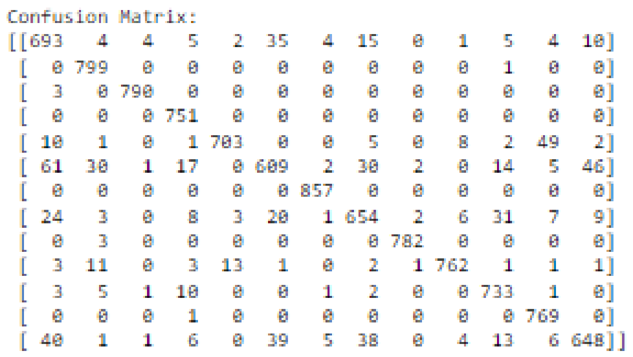
Hyper Parameter Effect: The earlystopping function is used to identify the number of epochs for each experiment. When we gave the standard value of epochs it did not give a satisfactory result. By setting a patience value of 10 on the early stopping function and a maximum value of 1000 we determine the number of epochs for each experimental model.

The model has significantly affected by batch size. The accuracy of the model increases with the increase in batch size before it surpasses threshold size. Later on, the accuracy of the model starts dropping. In this research, we utilize 320 as a batch size that provides significantly improved results than others. We utilize a geometric sequence with a ratio of 2 in adjusting batch size values in experiments. The minimum count of batch size we experimented with is 64 up to 512. We select the batch size of 320 which gives recall, precision, and F1-Score of 94% and accuracy of 93.9% respectively as shown in Table 4.2. We use rate of dropout rate of 0.5,0.1,0.2 and 0.3. We obtained better results on 0.05 dropout that is 93.9, 94,94,94 of accuracy, recall, precision, and F1-Measure which are presented in Table 4.3 respectively. The sigmoid function is used in the output layer and in the hidden layers we use relu.

4.4.3 Confusion Matrix

The proposed model correctly classifies malware categories in their respective classes. Figure 4.12 shows that the proposed approach performs distinguished for multi-class classification. The proposed model can classify multi-class malware categories efficiently while maintaining overall distinguished performance.





```
Confusion Matrix:
[[693   4   4   5   2  35   4  15   0   1   5   4  10]
 [  0 799   0   0   0   0   0   0   0   0   1   0   0]
 [  3   0 790   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0 751   0   0   0   0   0   0   0   0   0]
 [ 10   1   0   1 703   0   0   5   0   8   2  49   2]
 [ 61  30   1  17   0 609   2  30   2   0  14   5  46]
 [  0   0   0   0   0   0 857   0   0   0   0   0   0]
 [ 24   3   0   8   3  20   1 654   2   6  31   7   9]
 [  0   3   0   0   0   0   0   0 782   0   0   0   0]
 [  3  11   0   3  13   1   0   2   1 762   1   1   1]
 [  3   5   1  10   0   0   1   2   0   0 733   1   0]
 [  0   0   0   1   0   0   0   0   0   0   0 769   0]
 [ 40   1   1   6   0  39   5  38   0   4  13   6 648]]
```

Figure 4. 12 Confusion matrix of the DNN Model

4.4.4 AUC-ROC Analysis

The ratio of the True Positive Rate and False Positive Rate is defined as the ROC Curve. The ROC curve shows that the proposed algorithms accurately classify between TPR and FPR. The AUC value is 1 of the proposed algorithm which means it is a perfect classifier. Figure 4.13 presents the results.

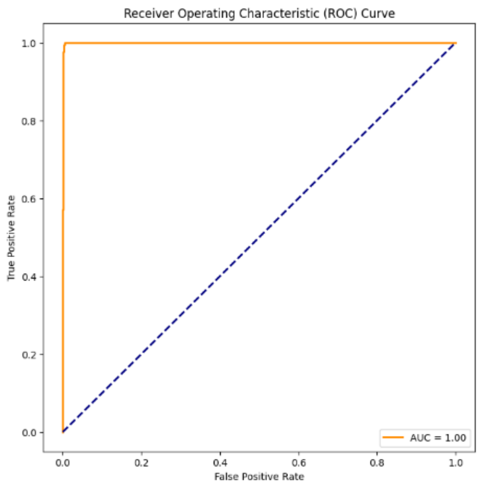


Figure 4. 13 ROC-AUC Analysis of DNN

4.5 Statistical Analysis of Model

This division discusses the statistical analysis of the proposed DNN algorithm. The experiments have been performed with several combinations of hyper parameters and layers to find the best possible model which neither has overfitting nor under fitting.

4.5.1 Bias and Variance

To identify the best possible model we performed k-fold cross-validation on models where the value of k =10. Through cross-validation, we find bias and variance of all experimented models and plot it to get the model that is neither over fitted nor under fitted. According to the bias and variance graph, the intersection point is close to point 5 so we select the 5-layer model which is presented in Figure 4.14

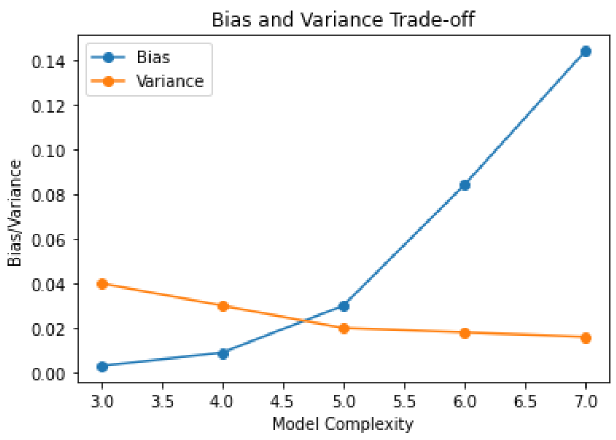


Figure 4. 14 Trade-off of Variance and Bias

4.6 Comparing Results with Existing Approaches

This division discusses the comparison of the proposed model with the existing studies. Few studies applied the multi-class classification to detect Android malware concerning dynamic analysis. One new malware category from the CIC-AndMal-2020 dataset which is not detected before is detected in the proposed model. We perform a comparison analysis of the proposed DNN model with existing models. The majority of existing studies considered few malware classes or features. The proposed study surpasses existing studies by performing statistics analysis of the dataset and model with a proficient deep learning algorithm as presented in Table 4.5.

Table 4. 5 Comparison with Existing Approaches

Study	**Features**	**Algorithm**	**Malware Categories**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 (%)**
Proposed	API, Memory, Network, Battery and Logcat	DNN	13	93.9	94	94	94
[29]	API, Memory, Network, Battery and Logcat	DL+ LSTM	12	91	92	90	91
[31]	API, Memory, Network	ML-Ensemble	12	91	92	91	91
[28]	Network	BIR-CNN	4	99.53	-	-	-
[30]	Network	TLFL Algo	4	90.61	-	-	-

CHAPTER 5

CONCLUSION

This section consists of future work and the conclusion of the thesis.

5.1 Conclusion

The Android is most demanding operating system. With its popularity, it also faces threats against Android malware. Researchers used detection approaches to identify malicious activities in APKs. It consists of two types of detection approaches which are dynamic analysis and static analysis. This study is focused on the dynamic analysis of Android malware. Researchers have proposed many ML and DL-based algorithms to identify malware but the majority of the research is focused on binary classification but with technology advancements, there is a need to identify malware categories with diverse features.

This study proposed a deep learning-based deep neural network algorithm that can detect 13 malware classes (Backdoor, Trojan, PUA, Scareware, Trojan-Dropper, Trojan-Spy, Zero-day, Ransomware, Trojan-SMS, Scareware, Riskware, File Infector, and Adware). We train this model on CIC-AndMal-2020 which is a newly developed dataset via CIC. It consists of 141 features and 13 malware categories. We performed statistical analysis of the dataset by performing p-value analysis and correlation. Based on it we dropped 27 features out of 141 features and trained the model on the remaining 114 features (Memory, API, Logcat, Battery, and Process). We compare our research with existing studies which outperform them by giving an accuracy of 93.9% of accuracy, 94% of accuracy, 94% of accuracy, and 94% of f1 score.

5.2 Future Work

In this research, we performed multi-class classification with a deep learning-based algorithm on the newly developed dataset by the Canadian Institute of Cybersecurity i.e. CIC-And-Mal-2020. Later on, we will develop our dataset by analyzing APKs in an emulated environment and performing dynamic analysis. We can also identify zero-day attacks with different deep learning algorithms as there is a need to identify newly generated malware.

REFERENCES

[1] Stat Counter Global Stats, “Desktop vs Mobile vs Tablet Market Share Worldwide Sept 2022 - Sept 2023.” [Online]. Available: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet

[2] Stat Counter Global Stats., “Mobile Operating System Market Share Worldwide.” [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[3] “2023 Global Mobile Threat Report.”

[4] Y. Pan, X. Ge, C. Fang, and Y. Fan, “A Systematic Literature Review of Android Malware Detection Using Static Analysis,” *IEEE Access*, vol. 8, pp. 116363–116379, 2020, doi: 10.1109/ACCESS.2020.3002842.

[5] A. Kapratwar, “Static and Dynamic Analysis for Android Malware Detection,” San Jose State University, San Jose, CA, USA, 2016. doi: 10.31979/etd.za5p-mqce.

[6] R. Vinayakumar, K. P. Soman, P. Poornachandran, and S. Sachin Kumar, “Detecting Android malware using Long Short-term Memory (LSTM),” in *Journal of Intelligent and Fuzzy Systems*, IOS Press, 2018, pp. 1277–1288. doi: 10.3233/JIFS-169424.

[7] R. Surendran, T. Thomas, and S. Emmanuel, “A TAN based hybrid model for android malware detection,” *Journal of Information Security and Applications*, vol. 54, Oct. 2020, doi: 10.1016/j.jisa.2020.102483.

[8] H. Kang, J. W. Jang, A. Mohaisen, and H. K. Kim, “Detecting and classifying android malware using static analysis along with creator information,” *Int J Distrib Sens Netw*, vol. 2015, 2015, doi: 10.1155/2015/479174.

[9] K. McLaughlin and IEEE Computer Society., *2018 16th Annual Conference on Privacy, Security and Trust (PST) : August 28-30, 2018, Belfast, Northern Ireland, United Kingdom*.

[10] A. Kapratwar, F. Di Troia, and M. Stamp, “Static and dynamic analysis of android malware,” in *ICISSP 2017 - Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, SciTePress, 2017, pp. 653–662. doi: 10.5220/0006256706530662.

[11] I. T. Ahmed, N. Jamil, M. M. Din, and B. T. Hammad, “Binary and Multi-Class Malware Threads Classification,” *Applied Sciences (Switzerland)*, vol. 12, no. 24, Dec. 2022, doi: 10.3390/app122412528.

[12] F. Taher, O. AlFandi, M. Al-kfairy, H. Al Hamadi, and S. Alrabaee, “DroidDetectMW: A Hybrid Intelligent Model for Android Malware Detection,” *Applied Sciences (Switzerland)*, vol. 13, no. 13, Jul. 2023, doi: 10.3390/app13137720.

[13] G. P. G. & S. K. Santosh K. Smmarwar, “A Hybrid Feature Selection Approach-Based Android Malware Detection Framework Using Machine Learning Techniques,” in *Cyber Security, Privacy and Networking*, May 2022.

[14] C. Ding, N. Luktarhan, B. Lu, and W. Zhang, “A hybrid analysis-based approach to android malware family classification,” *Entropy*, vol. 23, no. 8, Aug. 2021, doi: 10.3390/e23081009.

[15] R. Sihwail, K. Omar, and K. A. Z. Ariffin, “A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis,” vol. 8, pp. 4–6, 2018.

[16] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, “Dynamic malware analysis in the modern era—A state of the art survey,” *ACM Comput Surv*, vol. 52, no. 5, Sep. 2019, doi: 10.1145/3329786.

[17] A. Moser, C. Kruegel, and E. Kirda, “Limits of static analysis for malware detection,” in *Proceedings - Annual Computer Security Applications Conference, ACSAC*, 2007, pp. 421–430. doi: 10.1109/ACSAC.2007.21.

[18] S. Khalid and F. B. Hussain, “Evaluating Dynamic Analysis Features for Android Malware Categorization,” in *2022 International Wireless Communications and Mobile Computing, IWCMC 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 401–406. doi: 10.1109/IWCMC55113.2022.9824225.

[19] E. Amer *et al.*, “Using Machine Learning to Identify Android Malware Relying on API calling sequences and Permissions,” 2022.

[20] D. S. Keyes, B. Li, G. Kaur, A. H. Lashkari, F. Gagnon, and F. Massicotte, “EntropLyzer: Android Malware Classification and Characterization Using Entropy Analysis of Dynamic Characteristics,” in *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge, RDAAPS*

2021, Institute of Electrical and Electronics Engineers Inc., May 2021. doi: 10.1109/RDAAPS48126.2021.9452002.

[21] A. Mahindru and A. L. Sangal, “MLDroid—framework for Android malware detection using machine learning techniques,” *Neural Comput Appl*, vol. 33, no. 10, pp. 5183–5240, May 2021, doi: 10.1007/s00521-020-05309-4.

[22] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, “Emulator vs real phone: Android malware detection using machine learning,” in *IWSPA 2017 - Proceedings of the 3rd ACM International Workshop on Security and Privacy Analytics, co-located with CODASPY 2017*, Association for Computing Machinery, Inc, Mar. 2017, pp. 65–72. doi: 10.1145/3041008.3041010.

[23] T. Bhatia and R. Kaushal, “Malware Detection in Android based on Dynamic Analysis.” [Online]. Available: https://www.idc.com/prodserv/smartphone-os-market-share.jsp

[24] M. Gohari, S. Hashemi, and L. Abdi, “Android Malware Detection and Classification Based on Network Traffic Using Deep Learning,” in *2021 7th International Conference on Web Research, ICWR 2021*, Institute of Electrical and Electronics Engineers Inc., May 2021, pp. 71–77. doi: 10.1109/ICWR51868.2021.9443025.

[25] T. Lu, Y. Du, L. Ouyang, Q. Chen, and X. Wang, “Android malware detection based on a hybrid deep learning model,” *Security and Communication Networks*, vol. 2020, 2020, doi: 10.1155/2020/8863617.

[26] Canadian Institute for Cybersecurity, “Android Malware Dataset (CIC-AndMal2017).” Accessed: Dec. 30, 2023. [Online]. Available: https://www.unb.ca/cic/datasets/andmal2017.html

[27] Canadian Institute for Cybersecurity (CIC), “CCCS-CIC-AndMal-2020.” Accessed: Dec. 30, 2023. [Online]. Available: https://www.unb.ca/cic/datasets/andmal2020.html

[28] T. Y. Liu, H. Q. Zhang, H. X. Long, J. Shi, and Y. H. Yao, “Convolution neural network with batch normalization and inception-residual modules for Android malware classification,” *Sci Rep*, vol. 12, no. 1, Dec. 2022, doi: 10.1038/s41598-022-18402-6.

[29] M. I. Gulbarga, S. Cankurt, N. Shaidullaev, and A. L. Khan, “Deep Learning (DL) Dense Classifier with Long Short-Term Memory Encoder Detection and

Classification against Network Attacks," in *Proceedings - 2023 17th International Conference on Electronics Computer and Computation, ICECCO 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/ICECCO58239.2023.10146604.

[30] E. Allogmani and D. P. Josyula, "Two-level Filtering Learner Applied to a Noisy Malware Dataset," in *Proceedings - 2022 6th International Conference on Intelligent Computing and Control Systems, ICICCS 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 171–178. doi: 10.1109/ICICCS53718.2022.9788176.

[31] R. Islam, M. I. Sayed, S. Saha, M. J. Hossain, and M. A. Masud, "Android malware classification using optimum feature selection and ensemble machine learning," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 100–111, Jan. 2023, doi: 10.1016/j.iotcps.2023.03.001.

[32] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning," in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, IEEE, Aug. 2020, pp. 515–522. doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094.

[33] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553. Nature Publishing Group, pp. 436–444, May 27, 2015. doi: 10.1038/nature14539.

[34] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning", doi: 10.1007/s12525-021-00475-2/Published.

[35] A. Mahindru and P. Singh, "Dynamic permissions based android malware detection using machine learning techniques," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Feb. 2017, pp. 202–210. doi: 10.1145/3021460.3021485.

[36] M. K. Alzaylaee, S. Y. Yerima syerima, and qubacuk Sakir Sezer, "EMULATOR vs REAL PHONE: Android Malware Detection Using Machine Learning."

[37] M. Salehi, M. Amini, and B. Crispo, "Detecting malicious applications using system services request behavior," in *ACM International Conference Proceeding*

Series, Association for Computing Machinery, Nov. 2019, pp. 200–209. doi: 10.1145/3360774.3360805.

[38] A. Ananya, A. Aswathy, T. R. Amal, P. G. Swathy, P. Vinod, and S. Mohammad, “SysDroid: a dynamic ML-based android malware analyzer using system call traces,” *Cluster Comput*, vol. 23, no. 4, pp. 2789–2808, Dec. 2020, doi: 10.1007/s10586-019-03045-6.

[39] “SelFDroid: Selected Features in Dynamic Analysis Techniques for Android malware Detection,” 2021. [Online]. Available: https://www.researchgate.net/publication/356208018

[40] R. Thangaveloo, W. W. Jing, C. K. Leng, and J. Abdullah, “DATDroid: Dynamic analysis technique in android malware detection,” *Int J Adv Sci Eng Inf Technol*, vol. 10, no. 2, pp. 536–541, 2020, doi: 10.18517/ijaseit.10.2.10238.

[41] J. Hwang, J. Kim, S. Lee, and K. Kim, “Two-Stage Ransomware Detection Using Dynamic Analysis and Machine Learning Techniques,” *Wirel Pers Commun*, vol. 112, no. 4, pp. 2597–2609, Jun. 2020, doi: 10.1007/s11277-020-07166-9.

[42] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, “A novel dynamic android malware detection system with ensemble learning,” *IEEE Access*, vol. 6, pp. 30996–31011, 2018, doi: 10.1109/ACCESS.2018.2844349.

[43] A. Hashem, E. Fiky, M. A. Madkour, and A. El Shenawy, “Android Malware Category and Family Identification Using Parallel Machine Learning”, doi: 10.22059/jitm.2022.88133.

[44] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, “A novel dynamic android malware detection system with ensemble learning,” *IEEE Access*, vol. 6, pp. 30996–31011, 2018, doi: 10.1109/ACCESS.2018.2844349.

[45] S. Hou, A. Saas, L. Chen, and Y. Ye, “Deep4MalDroid: A deep learning framework for android malware detection based on Linux kernel system call graphs,” in *Proceedings - 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops, WIW 2016*, Institute of Electrical and Electronics Engineers Inc., Jan. 2017, pp. 104–111. doi: 10.1109/WIW.2016.15.

[46] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, “DL-Droid: Deep learning based android malware detection using real devices,” *Comput Secur*, vol. 89, Feb. 2020, doi: 10.1016/j.cose.2019.101663.

[47] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, “Android malware detection based on system call sequences and LSTM,” *Multimed Tools Appl*, vol. 78, no. 4, pp. 3979–3999, Feb. 2019, doi: 10.1007/s11042-017-5104-0.

[48] C. W. Yeh, W. T. Yeh, S. H. Hung, and C. T. Lin, “Flattened data in convolutional neural networks: Using malware detection as case study,” in *Proceedings of the 2016 Research in Adaptive and Convergent Systems, RACS 2016*, Association for Computing Machinery, Inc, Oct. 2016, pp. 130–135. doi: 10.1145/2987386.2987406.

[49] F. Martinelli, F. Marulli, and F. Mercaldo, “Evaluating Convolutional Neural Network for Effective Mobile Malware Detection,” in *Procedia Computer Science*, Elsevier B.V., 2017, pp. 2372–2381. doi: 10.1016/j.procs.2017.08.216.

[50] B. H. Kang and Q. Bai, Eds., *AI 2016: Advances in Artificial Intelligence*, vol. 9992. in Lecture Notes in Computer Science, vol. 9992. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-50127-7.

[51] A. De Lorenzo, F. Martinelli, E. Medvet, F. Mercaldo, and A. Santone, “Visualizing the outcome of dynamic analysis of Android malware with VizMal,” *Journal of Information Security and Applications*, vol. 50, Feb. 2020, doi: 10.1016/j.jisa.2019.102423.

[52] P. Gronát, J. Aldana-Iuit, and M. Bálek, “MaxNet: Neural network architecture for continuous detection of malicious activity.”

[53] J. J. Davis and A. J. Clark, “Data preprocessing for anomaly based network intrusion detection: A review,” *Computers and Security*, vol. 30, no. 6–7. pp. 353–375, Sep. 2011. doi: 10.1016/j.cose.2011.05.008.

[54] *2016 4th International Symposium on Digital Forensic and Security (ISDFS).* IEEE, 2016.

[55] D. Singh and B. Singh, “Investigating the impact of data normalization on classification performance,” *Appl Soft Comput*, vol. 97, Dec. 2020, doi: 10.1016/j.asoc.2019.105524.

[56] Harika Bonthu, “Detecting and Treating Outliers | Treating the odd one out.” Accessed: Dec. 18, 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/

[57] et al F. Chollet, “Keras.” Accessed: Dec. 18, 2023. [Online]. Available: https://keras.io/

[58] A. Fernández, S. García, F. Herrera, and N. V Chawla, “SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary,” 2018.

[59] J. Tan, J. Yang, S. Wu, G. Chen, and J. Zhao, “A critical look at the current train/test split in machine learning,” Jun. 2021, [Online]. Available: http://arxiv.org/abs/2106.04525

[60] R. Bardenet, M. Brendel, B. Kégl, M. Sebag, and S. Fr, “Collaborative hyperparameter tuning,” 2013.

[61] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, “Don’t Decay the Learning Rate, Increase the Batch Size,” Nov. 2017, [Online]. Available: http://arxiv.org/abs/1711.00489

[62] D. Thomas, C. Maraston, R. Bender, and C. Mendes De Oliveira, “THE EPOCHS OF EARLY-TYPE GALAXY FORMATION AS A FUNCTION OF ENVIRONMENT.”

[63] et al F. Chollet, “EarlyStopping.” Accessed: Dec. 25, 2023. [Online]. Available: https://keras.io/api/callbacks/early_stopping/

[64] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” 2014.

[65] J. Han and C. Moraga, “The Influence of the Sigmoid Function Parameters on the Speed of Backpropagation Learning.”

[66] F. Farhadi, *Learning activation functions in deep neural networks*. Ecole Polytechnique, 2017.

[67] B. Karlik and A. V. Olgac, “Volume (1): Issue (4).”

[68] “AUC”.

[69] R. A. Fisher, *Statistical methods for research workers*. Oliver and Boyd: Edinburgh, 1925.

[70] J. Han, J. Pei, and H. Tong, *Data mining: Concepts and techniques*. Morgan kaufmann., 2022.

APPENDIX

Table 1 P Value of 141 Features of Dataset

Dynamic Features	P value	Dynamic Features	P value
Memory_PssTotal	0.345	API_Command_java.lang.Runtime_exec	0.0
Memory_PssClean	3.433 e-87	API_JavaNativeInterface_java.lang.Runtime_loadLibrary	1.0
Memory_SharedDirty	0	API_JavaNativeInterface_java.lang.Runtime_load	1.0
Memory_PrivateDirty	2.006 e-10	API_WebView_android.webkit.WebView_loadUrl	0.0
Memory_SharedClean	1.877e-24	API_WebView_android.webkit.WebView_loadData	1.848e-06
Memory_PrivateClean	2.097e-70	API_WebView_android.webkit.WebView_loadDataWithBaseURL	1.859e-99
Memory_SwapPssDirty	1.0	API_WebView_android.webkit.WebView_addJavascriptInterface	1.213e-207
Memory_HeapSize	3.556e-153	API_WebView_android.webkit.WebView_evaluateJavascript	0.999
Memory_HeapAlloc	4.0617e-40	API_WebView_android.webkit.WebView_postUrl	1.0
Memory_HeapFree	2.1687e-33	API_WebView_android.webkit.WebView_postWebMessage	1.0
Memory_Views	0	API_WebView_android.webkit.WebView_savePassword	1.0
Memory_ViewRootImpl	0	API_WebView_android.webkit.WebView_setHttpAuthUsernamePassword	1.0
Memory_AppContexts	0	API_WebView_android.webkit.WebView_getHttpAuthUsernamePassword	1.0
Memory_Activities	0.0	API_WebView_android.webkit.WebView_setWebContentsDebuggingEnabled	0.5736
Memory_Assets	3.5075e-268	API_FileIO_libcore.io.IoBridge_open	0.0
Memory_AssetManagers	1.0	API_FileIO_android.content.ContextWrapper_openFileInput	0.0
Memory_LocalBinders	0.0	API_FileIO_android.content.ContextWrapper_openFileOutput	5.667e-99
Memory_ProxyBinders	0.0	API_FileIO_android.content.ContextWrapper_deleteFile	0.0
Memory_ParcelMemory	0.0	API_Database_android.content.ContextWrapper_openOrCreateDatabase	5.253e-07
Memory_ParcelCount	0.0	API_Database_android.content.ContextWrapper_databaseList	1.0
Memory_DeathRecipients	0.0	API_Database_android.content.ContextWrapper_deleteDatabase	1.0
Memory_OpenSSLSockets	1.0516e-31	API_Database_android.database.sqlite.SQLiteDatabase_execSQL	0.0

Memory_WebViews	0.0	API_Database_android.database.sqlite.SQLiteDatabase_deleteDatabase	1.0
API_Process_android.os.Process_start	1.0	API_Database_android.database.sqlite.SQLiteDatabase_getPath	4.0003e-308
API_Process_android.app.ActivityManager_killBackgroundProcesses	5.0670e-05	API_Database_android.database.sqlite.SQLiteDatabase_insert	2.39004e-14
API_Process_android.os.Process_killProcess	1.0	API_Database_android.database.sqlite.SQLiteDatabase_insertOrThrow	0.0014
API_Database_android.database.sqlite.SQLiteDatabase_insertWithOnConflict	5.1502e-12	API_Database_android.database.sqlite.SQLiteDatabase_openDatabase	0.0
API_Database_android.database.sqlite.SQLiteDatabase_openOrCreateDatabase	0.9674	API_Database_android.database.sqlite.SQLiteDatabase_query	9.9662e-233
API_Database_android.database.sqlite.SQLiteDatabase_queryWithFactory	9.715e-221	API_Database_android.database.sqlite.SQLiteDatabase_rawQuery	2.4132e-232
API_Database_android.database.sqlite.SQLiteDatabase_rawQueryWithFactory	0.0	API_Database_android.database.sqlite.SQLiteDatabase_update	1.3272e-168
API_Database_android.database.sqlite.SQLiteDatabase_updateWithOnConflict	1.32723e-168	API_Database_android.database.sqlite.SQLiteDatabase_compileStatement	0.0
API_Database_android.database.sqlite.SQLiteDatabase_create	1.0	API_IPC_android.content.ContextWrapper_sendBroadcast	0.0
API_IPC_android.content.ContextWrapper_sendStickyBroadcast	1.0	API_IPC_android.content.ContextWrapper_startActivity	0.0
API_IPC_android.content.ContextWrapper_startService	0.0	API_IPC_android.content.ContextWrapper_stopService	0.0303
API_IPC_android.content.ContextWrapper_registerReceiver	0.0	API_Binder_android.app.ContextImpl_registerReceiver	0.0
API_Binder_android.app.ActivityThread_handleReceiver	0.0	API_Binder_android.app.Activity_startActivity	0.0
API_Crypto_javax.crypto.spec.SecretKeySpec_$init	1.3909e-83	API_Crypto_javax.crypto.Cipher_doFinal	1.6204e-63
API_Crypto-Hash_java.security.MessageDigest_digest	0.0	API_Crypto-Hash_java.security.MessageDigest_update	0.0
API_DeviceInfo_android.telephony.TelephonyManager_getDeviceId	0.0	API_DeviceInfo_android.telephony.TelephonyManager_getSubscriberId	0.0
API_DeviceInfo_android.telephony.TelephonyManager_getLine1Number	1.596e-23	API_DeviceInfo_android.telephony.TelephonyManager_getNetworkOperator	8.6237e-143
API_DeviceInfo_android.telephony.TelephonyManager_getNetworkOperatorName	0.0	API_DeviceInfo_android.telephony.TelephonyManager_getSimOperatorName	1.0642e-79
API_DeviceInfo_android.net.wifi.WifiInfo_getMacAddress	0.0	API_DeviceInfo_android.net.wifi.WifiInfo_getBSSID	0.0074
API_DeviceInfo_android.net.wifi.WifiInfo_getIpAddress	0.0	API_DeviceInfo_android.net.wifi.WifiInfo_getNetworkId	0.994
API_DeviceInfo_android.telephony.TelephonyManager_getSimCountryIso	2.181e-111	API_DeviceInfo_android.telephony.TelephonyManager_getSimSerialNumber	0.0
API_DeviceInfo_android.telephony.TelephonyManager_getNetworkCountryIso	0.0	API_DeviceInfo_android.telephony.TelephonyManager_getDeviceSoftwareVersion	2.662e-83

Memory_WebViews	0.0	API_Database_android.database.sqlite.SQLiteDatabase_deleteDatabase	1.0
API_Process_android.os.Process_start	1.0	API_Database_android.database.sqlite.SQLiteDatabase_getPath	4.0003e-308
API_Process_android.app.ActivityManager_killBackgroundProcesses	5.0670e-05	API_Database_android.database.sqlite.SQLiteDatabase_insert	2.39004e-14
API_Process_android.os.Process_killProcess	1.0	API_Database_android.database.sqlite.SQLiteDatabase_insertOrThrow	0.0014
API_Database_android.database.sqlite.SQLiteDatabase_insertWithOnConflict	5.1502e-12	API_Database_android.database.sqlite.SQLiteDatabase_openDatabase	0.0
API_Database_android.database.sqlite.SQLiteDatabase_openOrCreateDatabase	0.9674	API_Database_android.database.sqlite.SQLiteDatabase_query	9.9662e-233
API_Database_android.database.sqlite.SQLiteDatabase_queryWithFactory	9.715e-221	API_Database_android.database.sqlite.SQLiteDatabase_rawQuery	2.4132e-232
API_Database_android.database.sqlite.SQLiteDatabase_rawQueryWithFactory	0.0	API_Database_android.database.sqlite.SQLiteDatabase_update	1.3272e-168
API_Database_android.database.sqlite.SQLiteDatabase_updateWithOnConflict	1.32723e-168	API_Database_android.database.sqlite.SQLiteDatabase_compileStatement	0.0
API_Database_android.database.sqlite.SQLiteDatabase_create	1.0	API_IPC_android.content.ContextWrapper_sendBroadcast	0.0
API_IPC_android.content.ContextWrapper_sendStickyBroadcast	1.0	API_IPC_android.content.ContextWrapper_startActivity	0.0
API_IPC_android.content.ContextWrapper_startService	0.0	API_IPC_android.content.ContextWrapper_stopService	0.0303
API_IPC_android.content.ContextWrapper_registerReceiver	0.0	API_Binder_android.app.ContextImpl_registerReceiver	0.0
API_Binder_android.app.ActivityThread_handleReceiver	0.0	API_Binder_android.app.Activity_startActivity	0.0
API_Crypto_javax.crypto.spec.SecretKeySpec_$init	1.3909e-83	API_Crypto_javax.crypto.Cipher_doFinal	1.6204e-63
API_Crypto-Hash_java.security.MessageDigest_digest	0.0	API_Crypto-Hash_java.security.MessageDigest_update	0.0
API_DeviceInfo_android.telephony.TelephonyManager_getDeviceId	0.0	API_DeviceInfo_android.telephony.TelephonyManager_getSubscriberId	0.0
API_DeviceInfo_android.telephony.TelephonyManager_getLine1Number	1.596e-23	API_DeviceInfo_android.telephony.TelephonyManager_getNetworkOperator	8.6237e-143
API_DeviceInfo_android.telephony.TelephonyManager_getNetworkOperatorName	0.0	API_DeviceInfo_android.telephony.TelephonyManager_getSimOperatorName	1.0642e-79
API_DeviceInfo_android.net.wifi.WifiInfo_getMacAddress	0.0	API_DeviceInfo_android.net.wifi.WifiInfo_getBSSID	0.0074
API_DeviceInfo_android.net.wifi.WifiInfo_getIpAddress	0.0	API_DeviceInfo_android.net.wifi.WifiInfo_getNetworkId	0.994
API_DeviceInfo_android.telephony.TelephonyManager_getSimCountryIso	2.181e-111	API_DeviceInfo_android.telephony.TelephonyManager_getSimSerialNumber	0.0
API_DeviceInfo_android.telephony.TelephonyManager_getNetworkCountryIso	0.0	API_DeviceInfo_android.telephony.TelephonyManager_getDeviceSoftwareVersion	2.662e-83

API_DeviceInfo_android.content.pm.PackageManager_getInstalledApplications	1.0	API_DeviceInfo_android.content.pm.PackageManager_getInstalledModules	1.0
API_DeviceInfo_android.content.pm.PackageManager_getInstalledPackages	1.0	API_Network_java.net.URL_openConnection	0.0
API_Network_org.apache.http.impl.client.AbstractHttpClient_execute	1.0	API_Network_com.android.okhttp.internal.huc.HttpURLConnectionImpl_getInputStream	0.0
API_Network_com.android.okhttp.internal.http.HttpURLConnectionImpl_getInputStream	1.0	API_DexClassLoader_dalvik.system.BaseDexClassLoader_findResource	1.6166e-194
API_DexClassLoader_dalvik.system.BaseDexClassLoader_findResources	3.0938e-91	API_DexClassLoader_dalvik.system.BaseDexClassLoader_findLibrary	0.0
API_DexClassLoader_dalvik.system.DexFile_loadDex	1.893e-19	API_DexClassLoader_dalvik.system.DexFile_loadClass	0.0003
API_DexClassLoader_dalvik.system.DexClassLoader_$init	0.0	API_Base64_android.util.Base64_decode	1.0
API_Base64_android.util.Base64_encode	0.0	API_Base64_android.util.Base64_encodeToString	0.0
API_SystemManager_android.app.ApplicationPackageManager_setComponentEnabledSetting	0.0	API_SystemManager_android.app.NotificationManager_notify	1.4419e-12
API_SystemManager_android.telephony.TelephonyManager_listen	1.3255e-212	API_SystemManager_android.content.BroadcastReceiver_abortBroadcast	1.0
API_SMS_android.telephony.SmsManager_sendTextMessage	1.9869e-264	API_SMS_android.telephony.SmsManager_sendMultipartTextMessage	1.0
API_DeviceData_android.content.ContentResolver_query	8.1905e-178	API_DeviceData_android.content.ContentResolver_registerContentObserver	0.0
API_DeviceData_android.content.ContentResolver_insert	0.0022	API_DeviceData_android.content.ContentResolver_delete	6.9825e-25
API_DeviceData_android.accounts.AccountManager_getAccountsByType	7.8505e-08	API_DeviceData_android.accounts.AccountManager_getAccounts	3.5013e-272
API_DeviceData_android.location.Location_getLatitude	1.0	API_DeviceData_android.location.Location_getLongitude	1.0
API_DeviceData_android.media.AudioRecord_startRecording	0.999	API_DeviceData_android.media.MediaRecorder_start	1.0
API_DeviceData_android.os.SystemProperties_get	0.0	API_DeviceData_android.app.ApplicationPackageManager_getInstalledPackages	0.0
API__sessions	0.0	Network_TotalReceivedBytes	1.0
Network_TotalReceivedPackets	3.2433e-236	Network_TotalTransmittedBytes	1.0
Network_TotalTransmittedPackets	1.4726e-200	Battery_wakelock	0.0
Battery_service	0.0	Logcat_debug	2.9875e-18
Logcat_info	0.00096	Logcat_warning	5.9366e-13
Logcat_error	4.1375e-23	Logcat_verbose	3.5425e-12
Logcat_total	0.0497	Process_total	2.569e-98