

ENHANCED DEEP LEARNING TECHNIQUES TO DETECT TROJAN HORSE ATTACK IN CYBERSECURITY

*Major project report submitted
in partial fulfillment of the requirement for award of the degree of*

**Bachelor of Technology
in
Computer Science & Engineering**

By

SAGIN SANDOZ FERNANDO E	(20UECS0825)	(16803)
K AJAY KUMAR REDDY	(20UECS0493)	(15072)
GOLLAPALLI JAYANTH	(20UECS0339)	(16923)

*Under the guidance of
Dr. S. AMUTHA, M.E., Ph.D.,
ASSOCIATE PROFESSOR*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING**

**VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF
SCIENCE & TECHNOLOGY**

(Deemed to be University Estd u/s 3 of UGC Act, 1956)

**Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA**

May, 2024

ENHANCED DEEP LEARNING TECHNIQUES TO DETECT TROJAN HORSE ATTACK IN CYBERSECURITY

*Major project report submitted
in partial fulfillment of the requirement for award of the degree of*

**Bachelor of Technology
in
Computer Science & Engineering**

By

SAGIN SANDOZ FERNANDO E	(20UECS0825)	(16803)
K AJAY KUMAR REDDY	(20UECS0493)	(15072)
GOLLAPALLI JAYANTH	(20UECS0339)	(16923)

*Under the guidance of
Dr. S. AMUTHA, M.E., Ph.D.,
ASSOCIATE PROFESSOR*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING**

**VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF
SCIENCE & TECHNOLOGY**

**(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA**

May, 2024

CERTIFICATE

It is certified that the work contained in the project report titled "ENHANCED DEEP LEARNING TECHNIQUES TO DETECT TROJAN HORSE ATTACK IN CYBERSECURITY" by "SAGIN SANDOZ FERNANDO E (20UECS0825), K AJAY KUMAR REDDY (20UECS0493), GOLLA-PALLI JAYANTH (20UECS0339)" has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr. Sagunthala R&D

Institute of Science & Technology

May, 2024

Signature of Professor In-charge

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr. Sagunthala R&D

Institute of Science & Technology

May, 2024

DECLARATION

We declare that this written submission represents our ideas in our own words and where others ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

(SAGIN SANDOZ FERNANDO E)

Date: / /

(Signature)

(K AJAY KUMAR REDDY)

Date: / /

(Signature)

(GOLLAPALLI JAYANTH)

Date: / /

APPROVAL SHEET

This project report entitled ENHANCED DEEP LEARNING TECHNIQUES TO DETECT TROJAN HORSE ATTACK IN CYBERSECURITY by SAGIN SANDOZ FERNANDO E (20UECS0825), K AJAY KUMAR REDDY (20UECS0493), GOLLAPALLI JAYANTH (20UECS0339) is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners

Supervisor

Dr. S. Amutha, ME., Ph.D,
Associate Professor,.

Date: / /

Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our respected **Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (EEE), B.E. (MECH), M.S (AUTO),D.Sc., Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Chairperson Managing Trustee and Vice President.

We are very much grateful to our beloved **Vice Chancellor Prof. S. SALIVAHANAN**, for providing us with an environment to complete our project successfully.

We record indebtedness to our **Professor & Dean, Department of Computer Science & Engineering, School of Computing, Dr. V. SRINIVASA RAO, M.Tech., Ph.D.**, for immense care and encouragement towards us throughout the course of this project.

We are thankful to our **Head, Department of Computer Science & Engineering, Dr.M.S. MURALI DHAR, M.E., Ph.D.**, for providing immense support in all our endeavors.

We also take this opportunity to express a deep sense of gratitude to our **Internal Supervisor Dr. S. AMUTHA, M.E., Ph.D.**, for her cordial support, valuable information and guidance, she helped us in completing this project through various stages.

A special thanks to our **Project Coordinators Mr. V. ASHOK KUMAR, M.Tech., Ms. C. SHYAMALA KUMARI, M.E.**, for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

SAGIN SANDOZ FERNANDO E	(20UECS0825)
K AJAY KUMAR REDDY	(20UECS0493)
GOLLAPALLI JAYANTH	(20UECS0339)

ABSTRACT

Deep learning has emerged as a potent tool in bolstering efforts to detect and mitigate malware, offering automated feature extraction capabilities and promising potential in zero-day malware detection. However, this advancement is set against the backdrop of persistent cyber-attacks that pose significant threats to these systems. The lack of awareness of such attacks hampers the system's ability to identify and respond to them effectively, potentially leading to compromised or entirely crippled performance. To combat these challenges, cybersecurity experts are increasingly turning to deep learning models like Convolutional Neural Networks (CNNs) and hybrid techniques. These models have the capacity to enhance threat detection accuracy, reduce reaction times, and improve adaptability to evolving malware landscapes. Yet, fully harnessing the efficiency of deep learning in cybersecurity necessitates addressing critical issues. These include ensuring representative training datasets that capture diverse threat scenarios, minimizing false positives to enhance precision, and enhancing model adaptability to rapidly changing threats. Consequently, further research and development efforts in this domain are imperative. By refining deep learning methodologies and adapting them specifically for cybersecurity applications, we can optimize the efficacy of malware detection and mitigation strategies. This ongoing exploration is crucial for staying ahead of sophisticated cyber adversaries and safeguarding digital ecosystems from evolving threats. Therefore, a concerted focus on innovation and refinement is essential to fully leverage deep learning's potential in fortifying cybersecurity defenses. The performance metrics obtained from this model is 97% accuracy and 3% loss.

Keywords: Deep learning, Malware detection, Mitigation, Convolutional Neural Networks, Cyber-attacks

LIST OF FIGURES

4.1	Architecture Diagram	16
4.2	Data Flow	17
4.3	Activity Diagram	18
4.4	Class Diagram	19
4.5	Sequence Diagram	20
5.1	Grayscale Images As Input	25
5.2	Output As Confusion Matrix	26
5.3	Classification Code	27
5.4	Classification Code Result	28
5.5	Dataset As Input	29
5.6	Dataset Result	30
5.7	Accuracy Code	31
5.8	Accuracy Code Result	32
5.9		33
5.10	Classification Code Result	34
5.11	Confusion Matrix Of Detection	35
5.12	Scalar Detection	36
5.13	ROC Curve	37
6.1	Classification Code Result	44
7.1	Plagiarism Report	47
8.1	Detection Confusion Matrix	56
8.2	ROC Curve	57
8.3	Poster Presentation	58

LIST OF TABLES

6.1	Performance Metrics	38
6.2	Comparison of Existing System vs Proposed System	40

LIST OF ACRONYMS AND ABBREVIATIONS

AE	Autoencoder
APK	Android Package
CAV	Connected and Autonomous Vehicle
CNN	Convolutional Neural Networks
DIMD	Deep Image Mal Detect
DNN	Deep Neural Network
DoS	Denial of Service
DS	Dataset
DT	Decision Tree
ECDLP	Ensemble-Based Parallel Deep Learning
GB	Gigabyte
GBM	Gradient Boosting Machine
GPU	Graphics Processing Unit
KNN	K-Nearest Neighbor Classifiers
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IOT	Internet of Things
ISO	International Organization for Standardization
MacOS	Macintosh Operating System
ML	Machine Learning
MLA	Machine Learning Algorithm
PC	Personal Computer
PE	Portable Executable
PSO-BP	Particle Swarm Optimization-Back-Propagation

RAM	Random Access Memory
RF	Random Forest
RFA	Random Forest Algorithm
ROI	Return On Investment
RNN	Recurrent Neural Networks
SVM	Support Vector Machine
TPU	Tensor Processing Unit
WSN	Wireless Sensor Network
UI	User Interface
XGBoost	Extreme Gradient Boosting

TABLE OF CONTENTS

	Page.No
ABSTRACT	v
LIST OF FIGURES	vi
LIST OF TABLES	vii
LIST OF ACRONYMS AND ABBREVIATIONS	viii
1 INTRODUCTION	1
1.1 Introduction	1
1.2 Aim of the Project	1
1.3 Project Domain	2
1.4 Scope of the Project	2
2 LITERATURE REVIEW	4
3 PROJECT DESCRIPTION	9
3.1 Existing System	9
3.2 Proposed System	10
3.3 Feasibility Study	11
3.3.1 Economic Feasibility	12
3.3.2 Technical Feasibility	13
3.3.3 Social Feasibility	13
3.4 System Specification	14
3.4.1 Hardware Specification	14
3.4.2 Software Specification	14
3.4.3 Standards and Policies	15
4 METHODOLOGY	16
4.1 General Architecture	16
4.2 Design Phase	17
4.2.1 Data Flow Diagram	17

4.2.2	Activity Diagram	18
4.2.3	Class Diagram	19
4.2.4	Sequence Diagram	20
4.3	Algorithm & Pseudo Code	21
4.3.1	Algorithm	21
4.3.2	Pseudo Code	21
4.4	Module Description	23
4.4.1	Classification	23
4.4.2	Detection of Trojan Horse Attacks	23
4.5	Steps to execute the project	23
4.5.1	Step1 :Dataset	23
4.5.2	Step2 : Classification	24
4.5.3	Step3 : Detection	24
4.5.4	Step4 : ROC curve	24
5	IMPLEMENTATION AND TESTING	25
5.1	Input and Output	25
5.1.1	Input Design	25
5.1.2	Output Design	26
5.2	Types of Testing	27
5.2.1	Unit Testing	27
5.2.2	Integration Testing	29
5.2.3	System Testing	33
6	RESULTS AND DISCUSSIONS	38
6.1	Efficiency of the Proposed System	38
6.2	Comparison of Existing and Proposed System	39
6.2.1	Existing System	39
6.2.2	Proposed System	39
6.3	Sample Code	41
6.3.1	Output 1	44
7	CONCLUSION AND FUTURE ENHANCEMENTS	45
7.1	Conclusion	45
7.2	Future Enhancements	45

8	SOURCE CODE & POSTER PRESENTATION	48
8.1	Source Code	48
8.2	Poster Presentation	58
	References	59

Chapter 1

INTRODUCTION

1.1 Introduction

In recent years, the proliferation of Internet of Things (IoT) devices and applications has significantly contributed to modern society. The inter-connectedness enabled by IoT technologies has revolutionized daily life, impacting areas such as smart homes and industrial automation. However, this rapid integration of IoT also brings significant security concerns. One of the foremost challenges facing this industrial revolution is the threat posed by cybercriminals. These malicious actors exploit vulnerabilities in individual PCs and networks, aiming to steal confidential data for financial gain or disrupt systems through denial-of-service attacks. Among the arsenal of tools wielded by cybercriminals, malware stands out as a particularly insidious threat. Malware can infiltrate IoT devices, compromising their functionality and posing risks to users' privacy and security. As the IoT landscape expands, ensuring robust cybersecurity measures becomes imperative to protect against evolving threats. Effective security strategies must encompass proactive monitoring, timely updates, and robust authentication mechanisms to safeguard IoT ecosystems from malicious activities.

1.2 Aim of the Project

The project's primary objective is to create advanced deep learning models specifically designed for the detection of trojan horse malware. The focus will be on developing techniques that can proactively prevent malware infections through various strategies such as behavior analysis, anomaly detection, and robust access controls. By leveraging deep learning algorithms, the system will aim to identify and mitigate the presence of trojan horse malware effectively. Behavior analysis will involve studying patterns and actions of software to identify suspicious activities indicative of trojan behavior. Anomaly detection techniques will be utilized to recognize deviations from normal system behavior, flagging potential threats for further investiga-

tion. Additionally, implementing stringent access controls will restrict unauthorized software installations and system modifications, reducing the risk of trojan infiltration. Ultimately, this project aims to enhance cybersecurity measures by deploying sophisticated deep learning technologies that can combat trojan horse malware and mitigate associated risks effectively.

1.3 Project Domain

The project aims to develop advanced deep learning models tailored specifically for the detection and prevention of trojan horse malware within the context of Internet of Things (IoT) environments. This endeavor addresses the pressing need for robust cybersecurity measures in light of escalating cyber threats targeting interconnected IoT devices. The primary focus is on leveraging deep learning algorithms to proactively identify and mitigate trojan horse malware through innovative strategies such as behavior analysis, anomaly detection, and robust access controls. Behavior analysis will involve studying software patterns to identify suspicious activities indicative of trojan behavior, while anomaly detection techniques will recognize deviations from normal system behavior, flagging potential threats for further investigation. Additionally, stringent access controls will be implemented to restrict unauthorized software installations and system modifications, thereby reducing the risk of trojan infiltration. Ultimately, this project aims to deploy sophisticated deep learning technologies to combat trojan horse malware effectively and enhance cybersecurity resilience in IoT ecosystems.

1.4 Scope of the Project

The scope and objective of this project encompass the development of an advanced deep learning-based methodology tailored for malware detection and mitigation. The primary goal is to enhance the efficacy of existing methods by harnessing the capabilities of deep learning algorithms. This approach will focus on identifying intricate patterns indicative of malicious behavior and analyzing complex feature vectors associated with malware instances. By leveraging deep learning, the project seeks to improve the accuracy and efficiency of malware detection processes compared to traditional techniques. The methodology will involve training sophisticated neural networks to learn and recognize diverse malware signatures and behaviors, enabling

proactive identification and mitigation of potential threats. The project aims to contribute to the field of cybersecurity by advancing the state-of-the-art in malware detection through innovative deep learning techniques. Ultimately, the envisioned outcome is a robust and adaptable system capable of detecting and mitigating a wide range of malware variants, thereby bolstering the resilience of digital systems against evolving cyber threats.

Chapter 2

LITERATURE REVIEW

[1] Asma A. Al-Hashmi et.al., [2022] proposed that challenges of detecting malware variants that can evade traditional security measures through structural changes and obfuscation techniques. It highlights the limitations of static feature-based detection due to obfuscation and emphasizes the importance of dynamic or behavioral analysis for accurate detection. The study proposes a Deep-Ensemble and Multi-faceted Behavioral Malware Variant Detection Model that combines sequential deep learning and extreme gradient boosting techniques. Behavioral features extracted from dynamic analysis are used to develop effective representative patterns automatically. These patterns are then employed in an extreme gradient boosting-based classifier for decision-making, aiming to improve detection accuracy and reduce false negatives. The proposed model is validated through extensive experiments and demonstrates enhanced performance compared to existing methods in detecting malware variants reliably.

[2] Hanxun et al., [2020] proposed a comprehensive system architecture for worm detection based on deep learning, consisting of a data collection engine, CNN-based worm detection module, DNN-based worm signature automatic generation module, and a worm alarm module. The CNN-based worm detection method utilizes convolutional neural networks to detect worms by distinguishing worm payloads from normal payloads. On the other hand, the DNN-based worm signature generation module automatically generates worm signatures by learning the characteristics of worm payloads based on historic known worm traffic. The system aims to efficiently detect and respond to worm attacks by automating the process of identifying malware signatures without human intervention. The research emphasizes the effectiveness of deep learning models, particularly CNN and DNN, in enhancing intrusion detection systems and malware detection.

[3] Jeongwoo kim et.al., [2023] proposed the importance of malware classification in handling the proliferation of malware variants and the challenges associated with disassembly-based methods due to anti-disassembly techniques employed by sophisticated malware. The focus is on proposing a novel approach for malware family classification using binary files without disassembly. The method involves using a CNN-based model that integrates "malware images" and "structural entropies" extracted from binary files. These modalities, with different levels of granularity, complement each other and are combined using a cross-modal attention mechanism to improve classification accuracy. The model is validated using datasets from Kaggle, demonstrating superior performance compared to previous disassembly-based methods while eliminating the need for disassembling malware.

[4] Mohammed et.al., [2023] proposed that ensemble-based parallel deep learning (ECDLP) classifier for malware detection. The ECDLP classifier consists of an ensemble of deep neural networks (DNNs) and a meta learner. The DNNs are trained in parallel on different subsets of the dataset to increase diversity, and the results are merged using a stacked ensemble technique. The meta learner, a feed forward neural network with a hidden layer and an output layer activated using a sigmoid function, is trained using back propagation to learn how to merge the contributions of each individual DNN. The ECDLP classifier is optimized using a hybrid PSO-BP algorithm, which uses particle swarm optimization for global search and backpropagation for local search, and it is found to outperform all other methods in all four metrics: accuracy, precision, recall, and F1-measure. The study highlights the effectiveness of the proposed ensemble model in achieving the best performance in comparison with those from other ensemble machine learning algorithms.

[5] Muawia A. Elsadig. [2023] proposed that characteristics and performance of Wireless Sensor Networks (WSNs) are the main reasons for their rapid expansion in various fields. However, these networks are extremely susceptible to multiple security assaults, including Denial-Of-Service (DoS) attacks, which are among the most prevalent in these networks. This study sheds light on WSN restrictions, weaknesses, and security threats with a focus on DoS attacks. Recent techniques for DoS attack detection have been investigated thoroughly, highlighting their achievements and limitations. This provides valuable insight into the current state of recent research in this field. Accordingly, this study proposes a lightweight machine learning

detection approach based on a Decision Tree (DT) algorithm with the Gini feature selection method to detect DoS attacks in WSNs. An enhanced version of the WSN-DS dataset, developed by the author, was used to train and test the proposed approach. The proposed approach has shown good performance by achieving an accuracy rate of 99.5% with minimum overhead compared to Random Forest (RF), extreme gradient boosting (XGBoost), and k-Nearest Neighbor (KNN) classifiers. It only takes 9.7%, 13%, and 2% of the processing time required by FR, XGBoost, and KNN respectively, which indicates that our proposed approach significantly outperforms these classifiers in terms of processing time. It is noteworthy that RF achieved an accuracy that was somewhat superior; however, the proposed approach greatly surpassed RF by taking only 9.7% of the RF processing time, which is an important factor in meeting WSN constraints.

[6] Mulhem ibrahim et al., [2022] proposed that hybrid analysis, which combines static and dynamic analysis, is a powerful solution for detecting malware applications, providing the best of both approaches. However, it is also expensive and time-consuming. The authors used Androguard, a complete framework developed in Python for analyzing APK files, to extract features such as services, resources, dex files, and more. They then used machine learning algorithms, including Simple Logistic, SVM, J48, and Random Forest, to classify malware and achieved high accuracy rates. The authors also emphasize the importance of using a balanced dataset and avoiding features that may not be useful in detecting malware. They suggest that more trees in random forest classification and a depth of at least 16 appear to be better for detecting malware. Overall, the article highlights the potential of combining static and dynamic analysis with machine learning algorithms for effective malware detection.

[7] Nguyet quang et.al., [2022] The project highlights the increasing concern over phishing attacks and the limitations of existing detection techniques, leading researchers to focus on leveraging machine learning, particularly deep learning, for more effective phishing detection. A systematic literature review involving 81 papers is used to propose a taxonomy of deep learning algorithms for phishing detection. The study introduces the concepts of phishing and deep learning in cybersecurity, categorizes existing literature into different algorithmic categories, reviews state-of-the-art deep learning techniques, and analyzes their strengths and weaknesses. The

paper also discusses challenges faced by deep learning in phishing detection and suggests future research directions to address these issues. Finally, an empirical analysis evaluates the performance of various deep learning techniques, revealing common issues such as manual parameter-tuning, long training times, and suboptimal detection accuracy, which motivate further research efforts in the field.

[8] Omer aslan et.al., [2021] proposed that new malware classification framework based on deep learning algorithms emphasizes the effectiveness of hybrid analysis, combining static and dynamic analysis, for detecting malware applications. The study delves into the significance of feature extraction techniques, such as vision-based methods, to extract and visualize malware features. It discusses the categorization of extracted malware features using machine learning algorithms or heuristic techniques for detection and classification. The article highlights the importance of different malware detection and classification approaches, including signature-based methods, behavior-based techniques, and deep learning-based models. Overall, the research underscores the potential of utilizing deep learning algorithms and hybrid analysis for robust malware detection and classification, offering insights into the evolving landscape of cybersecurity threats and defense mechanisms.

[9] Sandeep gupta et.al., [2023] proposed that Connected and autonomous vehicles (CAVs) can fulfill the emerging demand for smart transportation on a global scale. Such innovations for transportation can bring manifold benefits, from fully autonomous driving services to proactive vehicle monitoring and traffic management. However, given the complexity involved in the deployment of CAVs, zero-tolerance safety, and security measures must be incorporated to avert vehicle immobilization, road accidents, disclosure of sensitive data, or any potential threats. In this article, we conceive a reference architecture for a CAVs ecosystem to derive a common attack taxonomy for the investigation of existing and emerging cyber threats. Subsequently, we discuss security mechanisms for the CAVs ecosystem that can be useful for the safe and secure transportation of passengers from one destination to another based on comprehensive studies of academic literature and industry white papers. Our work can provide valuable insights to security engineers and system architects for investigating security problems using a top-to-bottom approach and can aid in envisioning robust security solutions to ensure seamless CAVs operations.

[10] Xiaofei xing et.al., [2022] proposed that malware detection approach using autoencoder in deep learning, specifically AlexNet and Inception-V3, for feature extraction. The authors present a hybrid model combining deep learning algorithms and traditional machine learning algorithms to achieve a high accuracy of 99.3% for a 25-class malware classification task using data augmentation based on affine image transformation. The research also explores the feasibility of converting software binary data directly into a greyscale image, and using this to visually represent malicious actions in a greyscale image without setting up a dynamic runtime environment. The authors designed two model structures, AE-1 and AE-2, with AE-1 used for unsupervised feature extraction and AE-2 for supervised malware detection. The AE-2 network exhibited more stability in the experimental aspects of the classification task, and the AE-1 network was trained in an unsupervised manner with no software samples labelled.

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

The existing system employing deep learning methods like CNNs, RNNs, and hybrid models has consistently achieved malware detection accuracy of 94%. These advanced techniques automate feature extraction from malware data (such as PE files), enabling precise identification of threats. CNNs excel at capturing complex patterns in binary data, while RNNs analyze sequential behavior, enhancing accuracy even against dynamic malware. Hybrid models like scale mal net combine these strengths for real-time detection. Innovations like DIMD, using image-based techniques, further boost accuracy by leveraging deep learning's pattern recognition abilities. These advancements highlight deep learning's transformative role in strengthening cybersecurity against evolving threats. Ongoing research aims to optimize these methods for even higher accuracy and resilience.

Disadvantages of existing system :

1. **Reliance on Feature Engineering:** Deep learning models rely on feature engineering, feature learning, and feature representation techniques that require extensive domain-level knowledge-. This can be a disadvantage because once an attacker becomes aware of the features, the malware detector can be easily evaded.

2. **Data Availability and Quality:** Machine learning algorithms require data with a variety of malware patterns. However, publicly available benchmark data for malware analysis research is limited due to security and privacy concerns. The existing datasets have harsh criticisms, as they are often outdated, making it challenging to develop a generic machine learning-based malware analysis system that can be de-

ployed in real-time.

3. **Computational Expense:** The n-gram approach for static malware detection is computationally expensive, and the performance is considerably low. It is often difficult to apply domain-level knowledge to extract necessary features when building a machine learning model to distinguish between malware and benign files due to the inconsistent specifications and standards imposed by the Windows operating system.

4. **Handling Packed Malware:** Image processing techniques, which do not require disassembly or code execution, are faster compared to static and dynamic analysis. However, they may not handle packed malware effectively, as they rely on the layout and texture of malware binaries transformed into gray-scale images.

5. **Vanishing Gradient Problem:** During back propagation, the transition function tf in Recurrent Neural Networks (RNN) can encounter the vanishing gradient problem, leading to the decay of information through time. To overcome this, gradient clipping and gating mechanisms are introduced.

3.2 Proposed System

The proposed model for zero-day malware detection described in the text introduces a sophisticated approach that combines visualization techniques with deep learning architectures. By leveraging image processing techniques optimized for Machine Learning Algorithms (MLAs) and deep learning architectures, the model transforms malware data into visual representations suitable for analysis. It incorporates Convolutional Neural Networks (CNNs) for effective feature extraction and pattern recognition tasks. The hybrid approach integrates static analysis, dynamic analysis, and image processing within a unified framework, enabling comprehensive coverage of diverse malware behaviors. Visualization techniques enhance interpretability and aid in feature extraction. Ultimately, this model aims to achieve robust zero-day malware detection, capable of identifying previously unseen malware variants or behaviors in real-time deployments. Its scalability makes it suitable for proactive detection and mitigation of emerging threats, marking a significant advancement in cybersecurity.

Advantages of Proposed system :

1. **Enhanced Interpretability with Visualization Techniques :** The proposed system incorporates visualization techniques that improve the interpretability of model decisions. This aids in understanding the features and patterns used by the model for malware detection, making it easier to identify and analyze malicious behaviors.
2. **Robust Feature Extraction with Deep Learning Architectures :** Leveraging advanced deep learning architectures like Convolutional Neural Networks (CNNs) allows for robust feature extraction from malware data. This enables the model to automatically learn discriminative features, enhancing its ability to detect complex and evolving malware variants.
3. **Comprehensive Coverage of Malware Behaviors :** By integrating static analysis, dynamic analysis, and image processing, the proposed system offers a comprehensive approach to malware detection. This multi-faceted framework ensures coverage of diverse malware behaviors and characteristics, improving overall detection accuracy.
4. **Scalability and Real-Time Deployment :** The proposed model is engineered for scalability and real-time deployment, crucial for proactive detection and mitigation of emerging malware threats. Its design emphasizes efficient processing and analysis, making it suitable for deployment in dynamic and evolving cybersecurity environments.
5. **Adaptability to Zero-Day Threats :** With its focus on deep learning and hybrid techniques, the proposed system is well-equipped to detect zero-day malware threats. It can identify previously unseen malware variants or behaviors by leveraging learned representations from large datasets, enhancing the system's ability to stay ahead of evolving threats.

3.3 Feasibility Study

The feasibility study for the proposed project involves evaluating the practicality and viability of developing advanced deep learning models for trojan horse malware

detection within IoT ecosystems across multiple dimensions. From a technical standpoint, the project aims to leverage sophisticated deep learning algorithms like Convolutional Neural Networks (CNNs) for malware detection. This requires expertise in machine learning, image processing, and cybersecurity to effectively implement and optimize the models. Additionally, ensuring access to suitable hardware resources, such as GPUs for training deep learning models, and utilizing appropriate software tools like PyTorch or TensorFlow are essential for the successful execution of the proposed solution. Economically, developing and deploying advanced deep learning models for trojan horse malware detection necessitates investment in hardware infrastructure, software licenses, and skilled personnel. Evaluating the cost-benefit ratio and estimating the Return On Investment (ROI) based on the potential impact of mitigating malware threats within IoT environments are crucial considerations for assessing economic feasibility. Socially, addressing cybersecurity threats posed by malware is paramount for safeguarding individuals and organizations utilizing IoT devices. Enhancing cybersecurity measures through proactive malware detection and mitigation not only ensures public safety but also fosters trust in IoT technologies. Therefore, understanding the social impact and acceptance of deploying advanced deep learning solutions for cybersecurity is fundamental to the project's feasibility assessment.

3.3.1 Economic Feasibility

The economic feasibility of the proposed project implementing advanced deep learning models for trojan horse malware detection within IoT ecosystems involves conducting a comprehensive cost analysis and assessing potential Returns On Investment (ROI). This evaluation encompasses estimating the initial investment required for essential resources, including hardware such as GPUs, software such as deep learning frameworks, and human resources consisting of skilled data scientists and cybersecurity experts necessary to develop and deploy the models effectively. Additionally, the project will assess the potential benefits derived from mitigating malware threats, such as reduced data breaches, minimized system downtime, and associated costs. By quantifying these potential benefits, the project aims to justify the expenditure on implementing advanced deep learning solutions for trojan horse malware detection. This economic analysis will play a crucial role in determining the financial viability and long-term sustainability of the project, ensuring that the

anticipated returns outweigh the initial investment costs.

3.3.2 Technical Feasibility

The technical feasibility of the proposed project relies on evaluating essential resources, technologies, and expertise needed for developing and deploying advanced deep learning models tailored for trojan horse malware detection in IoT environments. This assessment includes ensuring access to crucial hardware resources like GPUs for efficient model training and optimization. Additionally, availability of key software tools such as PyTorch or TensorFlow is imperative to support the implementation and testing of sophisticated algorithms for malware detection. Evaluating the presence of skilled personnel with expertise in machine learning, image processing, and cybersecurity is vital for designing and validating effective deep learning algorithms. Expert professionals are essential for successful development and deployment of models capable of detecting trojan horse malware within complex IoT ecosystems. This comprehensive evaluation is crucial for determining the project's readiness to utilize advanced deep learning techniques in addressing cybersecurity challenges associated with IoT devices and applications, ensuring effective achievement of project objectives.

3.3.3 Social Feasibility

The social feasibility of deploying advanced deep learning models for trojan horse malware detection within IoT ecosystems involves evaluating several key aspects. Firstly, it entails assessing public perception, which includes gauging awareness and acceptance of cybersecurity measures aimed at safeguarding IoT devices from malware threats. Public trust and safety are paramount considerations, as enhancing cybersecurity through proactive malware detection helps build trust and ensures the safety of individuals and organizations relying on IoT technologies. Additionally, ethical considerations play a crucial role in this context, necessitating a thoughtful approach to address data privacy and security implications associated with the development and deployment of deep learning models for malware detection within IoT environments. By carefully examining these social factors, the project can assess the readiness and receptiveness of stakeholders towards implementing advanced cybersecurity solutions for IoT protection. This evaluation is essential for ensuring the successful adoption and impact of the proposed deep learning-based malware

detection system in real-world scenarios.

3.4 System Specification

3.4.1 Hardware Specification

- RAM : Minimum 4 GB
- Operating System: Windows, Linux, or Mac
- Memory Availability: Minimum 200 GB

3.4.2 Software Specification

Chrome

- Chrome: Latest version
- Python: 3.7
- Accelerator: GPU P100
- Packages: Keras, sklearn, skimage, glob, nilearn, matplotlib, cv2 (OpenCV), nibabel

Google Colab

- Operating System: Google Colab runs on virtual machines hosted on Google Cloud Platform. The underlying operating system is based on Linux.
- Browser: Google Colab is accessed through a web browser. It is compatible with most modern browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.
- Python Version: Google Colab typically provides the latest stable version of Python. As of now, it may offer Python 3.7 or a newer version.
- Hardware Accelerator: Google Colab allows you to use hardware accelerators such as GPUs (Graphics Processing Units) or TPUs (Tensor Processing Units). Common GPU options include NVIDIA Tesla K80, T4, P4, or P100, depending on availability.

- **Packages and Libraries:** Google Colab comes pre-installed with many popular Python packages and libraries. numpy, pandas, matplotlib, seaborn, scikit-learn, tensorflow, keras, torch, torchvision, opencv-python (cv2), scipy, nibabel, nilearn, skimage, glob
- **Other Tools:** Google Colab also provides access to additional tools and utilities for working with data, notebooks, and external storage. This includes integration with Google Drive for file storage and collaboration, and the ability to install additional packages using `!pip install`.

3.4.3 Standards and Policies

Google Colab

Google Colab is a cloud-based Jupyter notebook environment provided by Google as part of the Google Cloud Platform ecosystem. It offers a convenient and accessible platform for running Python code, particularly for data science, machine learning, and deep learning tasks. Users can leverage Colab's integration with Google Drive for seamless file management and collaboration. One of its key features is the ability to access powerful hardware accelerators like GPUs and TPUs, enabling faster execution of complex computations and machine learning models in a scalable environment.

Standard Used: ISO/IEC 27001

Google Drive

Google Drive is a file storage and synchronization service developed by Google. Launched on April 24, 2012, Google Drive allows users to store files in the cloud (on Google's servers), synchronize files across devices, and share files. In addition to a web interface, Google Drive offers apps with offline capabilities for Windows and macOS computers, and Android and iOS smartphones and tablets. Google Drive encompasses Google Docs, Google Sheets, and Google Slides, which are a part of the Google Docs Editors office suite that permits collaborative editing of documents, spreadsheets, presentations, drawings, forms, and more.

Standard Used: ISO/IEC 27001

Chapter 4

METHODOLOGY

4.1 General Architecture

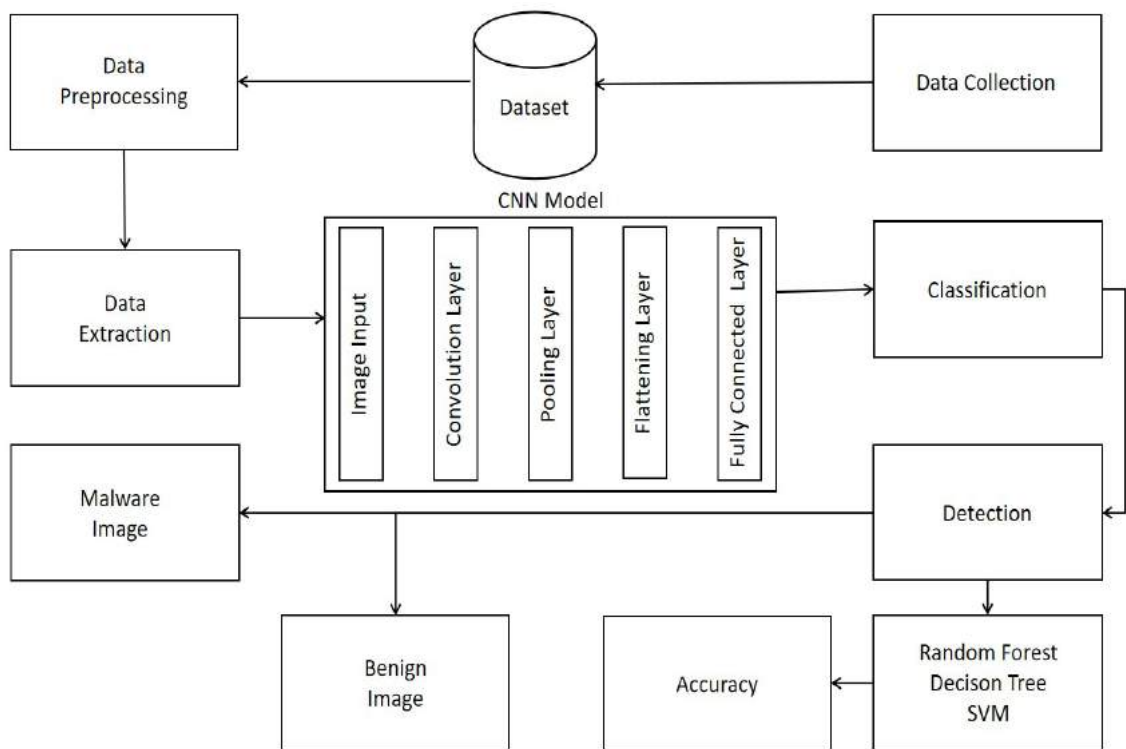


Figure 4.1: Architecture Diagram

Figure 4.1 shows that how the process is being made from importing the dataset and then training it using the CNN model for classification and produces confusion matrix then using CNN model to detect the malware affected images and produce another confusion matrix and in next step using RFA, DT, SVM algorithm code to determines the accuracy of the detected images.

4.2 Design Phase

4.2.1 Data Flow Diagram

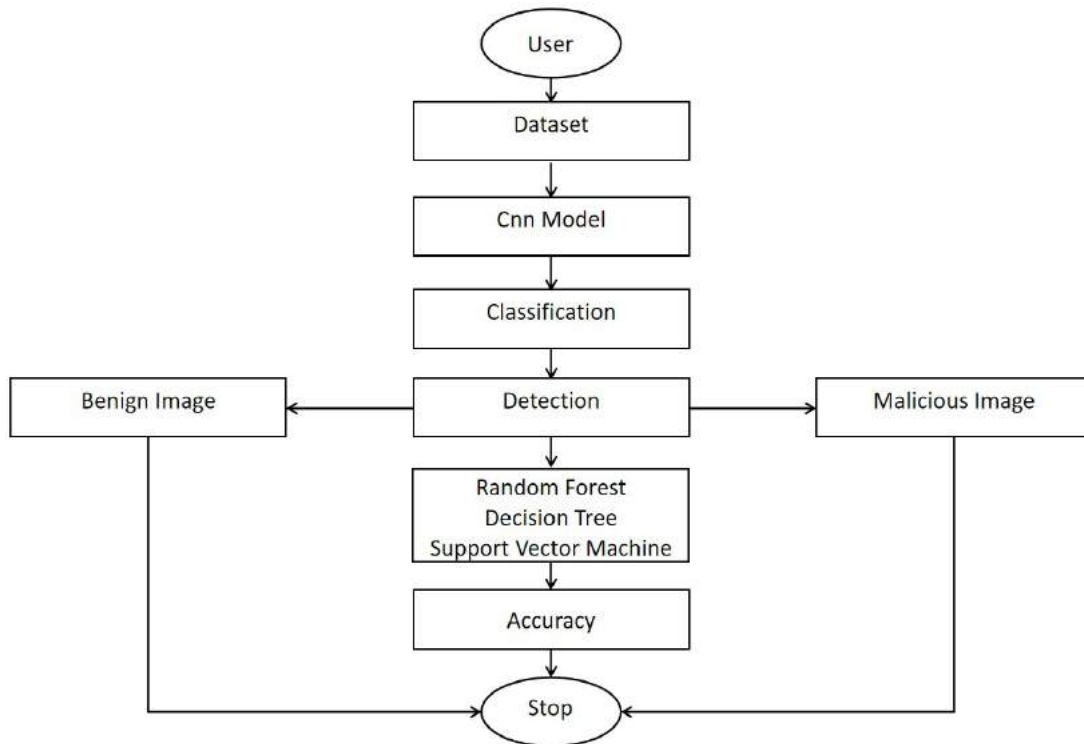


Figure 4.2: **Data Flow**

Figure 4.2 shows the outlines of process starting with dataset importation followed by training a CNN model for classification, resulting in a confusion matrix. Subsequently, the CNN model is utilized to detect malware-affected images, generating another confusion matrix. Next, an RFA algorithm evaluates the accuracy of detected images. This structured workflow integrates dataset handling, CNN classification, malware detection, and evaluation via the RFA, DT, SVM algorithm to assess image detection performance.

4.2.2 Activity Diagram

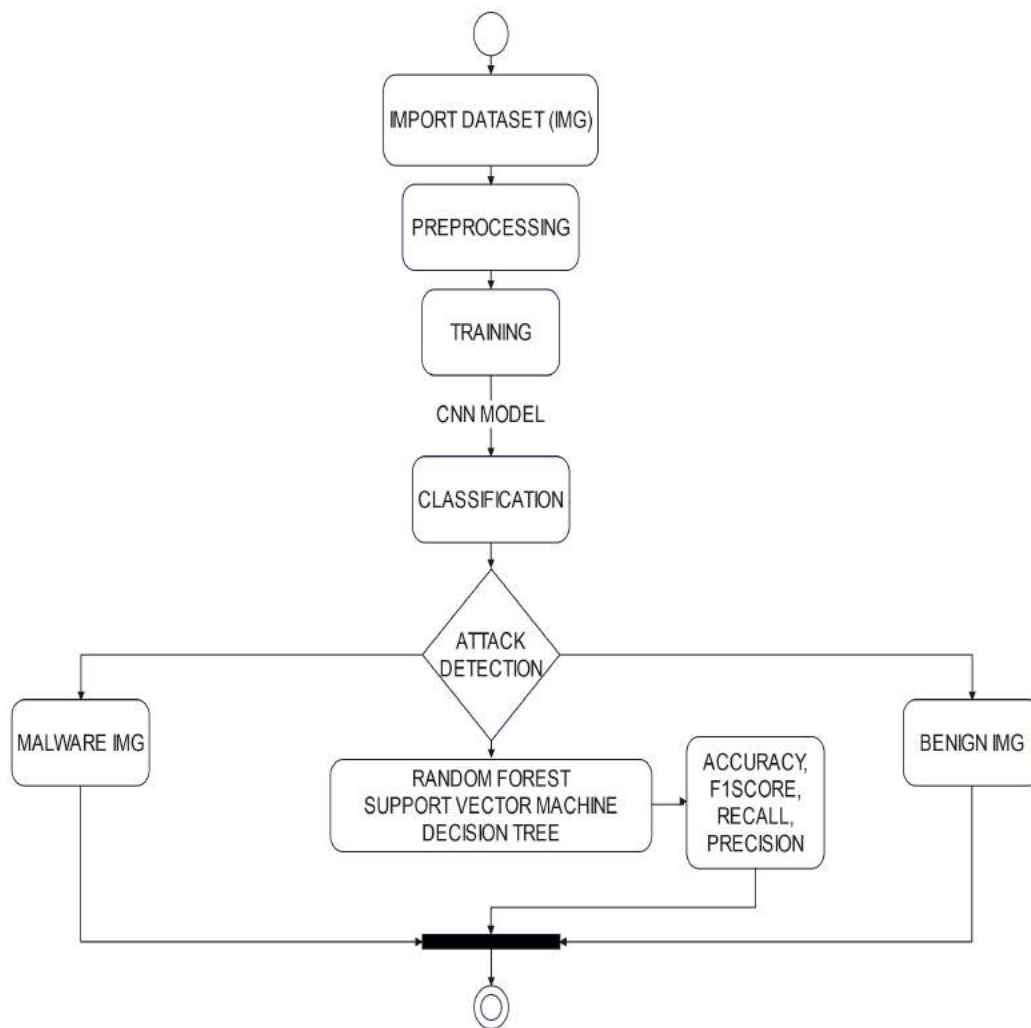


Figure 4.3: Activity Diagram

Figure 4.3 shows that detecting and mitigating attacks on an image dataset involves several stages. First, the image dataset is preprocessed and prepared for classification. Next, the Convolutional Neural Network (CNN) model is trained on the dataset to classify the images into normal or attack categories. The output of the CNN model is then used as input for the Random Forest model, which further classifies the attacks on how much affected. The Random Forest model also identifies the source IP addresses of the attacks. Finally, the mitigation stage involves taking appropriate action based on the classification results. This approach combines the strengths of both CNN and Random Forest models to provide a robust and accurate system for detecting and mitigating attacks on an image dataset.

4.2.3 Class Diagram

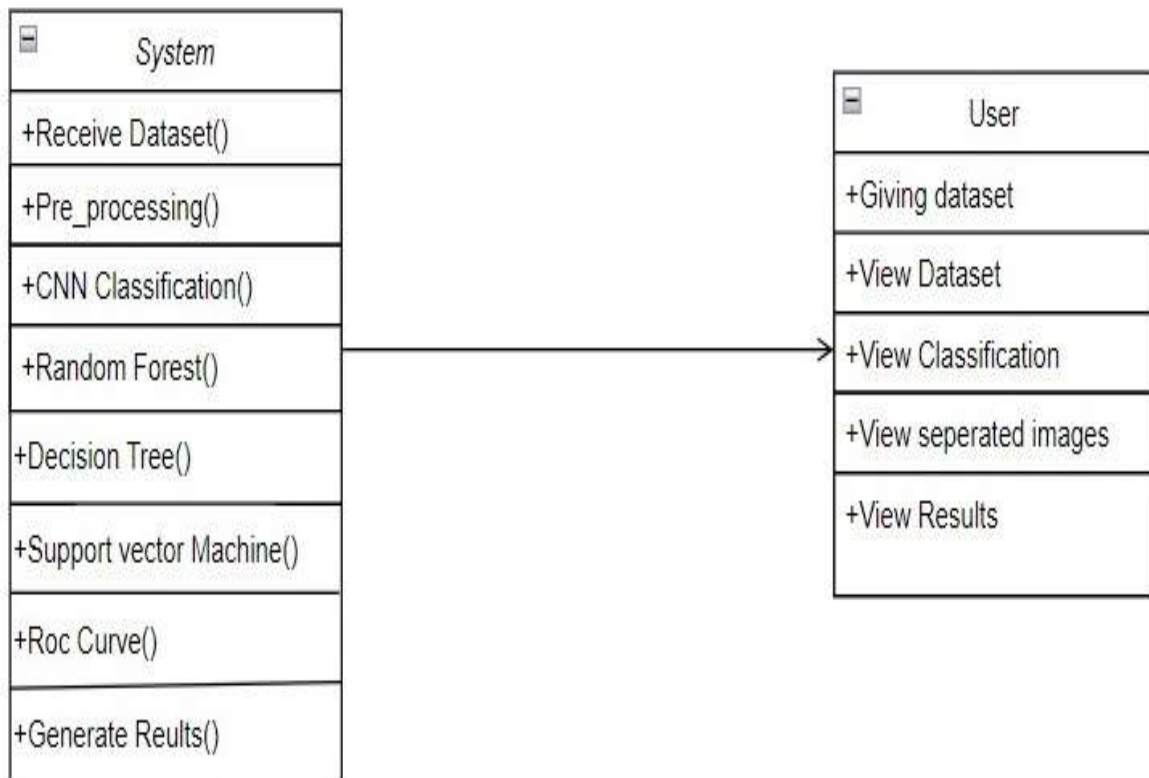


Figure 4.4: **Class Diagram**

Figure 4.4 shows the structure of a system by detailing the classes, their attributes, methods, and relationships, preprocessing involves tasks like data cleaning, normalization, and feature extraction to enhance the quality of the dataset. Subsequently, Convolutional Neural Networks (CNNs) are utilized for image classification, leveraging their ability to automatically extract relevant features from images. Following CNN classification, Random Forest is employed for detection, utilizing an ensemble learning method to improve accuracy by combining multiple decision trees. Moreover, Decision Trees and Support Vector Machines (SVMs) are applied for further classification tasks, each offering distinct advantages in handling different types of data. Finally, the Receiver Operating Characteristic (ROC) curve is utilized to evaluate the accuracy score of the models, providing a graphical representation of the trade-off between true positive rate and false positive rate.

4.2.4 Sequence Diagram

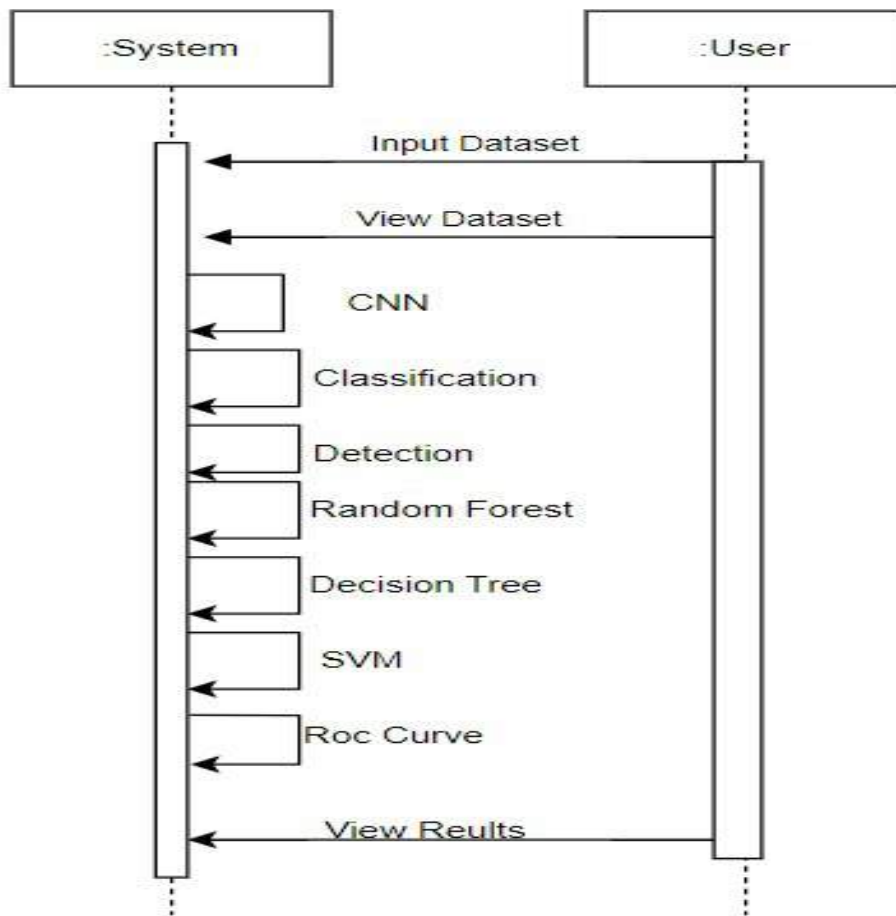


Figure 4.5: Sequence Diagram

Figure 4.5 shows the step-by-step process of processing a dataset for image classification and object detection. Initially, the dataset is preprocessed to enhance its quality. Next, Convolutional Neural Networks (CNNs) are applied for image classification, leveraging their ability to automatically extract relevant features from images. Following this, Random Forest is employed for object detection, utilizing an ensemble learning method to improve accuracy by combining multiple decision trees. Additionally, Decision Trees and Support Vector Machines (SVMs) are applied for further classification tasks, each offering distinct advantages in handling different types of data. The Receiver Operating Characteristic (ROC) curve is then utilized to evaluate the accuracy score of the models, providing a graphical representation of the trade-off between true positive rate and false positive rate.

4.3 Algorithm & Pseudo Code

4.3.1 Algorithm

```
1 Algorithm : CNN
2 STEP-1: Start
3 STEP-2: Import the img dataset.
4 STEP-3: Classify the dataset based on CNN model.
5 STEP-4: Detect if there is any malware in the img using CNN model.
6 STEP-6: Benign and malware affected images are separated.
7 STEP-7: Calculate the accuracy, f1score, recall, precision using random forest algorithm, decision
      tree algorithm and support vector machine algorithm.
8 STEP-8: Show ROC curve.
9 STEP-9: Display to the user.
10 STEP-10: Stop
```

4.3.2 Pseudo Code

```
1
2
3 import image_dataset
4
5 def classify_images(dataset):
6     create_cnn_model()
7
8     train_cnn_model(dataset)
9
10    for image in dataset:
11
12        predicted_label = cnn_model.predict(image)
13
14
15        image.metadata['label'] = predicted_label
16
17    return dataset
18
19 labeled_dataset = classify_images(image_dataset)
20
21 def detect_malware(labeled_dataset):
22
23     benign_images = []
24     malware_images = []
25     for image in labeled_dataset:
26         if image.metadata['label'] == 'benign':
27             benign_images.append(image)
28         else:
```

```

29         malware_images.append(image)
30
31     return benign_images, malware_images
32
33 benign_images, malware_images = detect_malware(labeled_dataset)
34
35 forest, decision_tree, and SVM algorithms
36 def calculate_metrics(benign_images, malware_images):
37     benign_features = extract_features(benign_images)
38     malware_features = extract_features(malware_images)
39
40     random_forest_model = train_random_forest(benign_features, malware_features)
41     decision_tree_model = train_decision_tree(benign_features, malware_features)
42     svm_model = train_svm(benign_features, malware_features)
43
44     accuracy_rf = random_forest_model.score(benign_features, malware_features)
45     flscore_rf = calculate_f1score(random_forest_model, benign_features, malware_features)
46     recall_rf = calculate_recall(random_forest_model, benign_features, malware_features)
47     precision_rf = calculate_precision(random_forest_model, benign_features, malware_features)
48
49     accuracy_dt = decision_tree_model.score(benign_features, malware_features)
50     flscore_dt = calculate_f1score(decision_tree_model, benign_features, malware_features)
51     recall_dt = calculate_recall(decision_tree_model, benign_features, malware_features)
52     precision_dt = calculate_precision(decision_tree_model, benign_features, malware_features)
53
54     accuracy_svm = svm_model.score(benign_features, malware_features)
55     flscore_svm = calculate_f1score(svm_model, benign_features, malware_features)
56     recall_svm = calculate_recall(svm_model, benign_features, malware_features)
57     precision_svm = calculate_precision(svm_model, benign_features, malware_features)
58
59     return accuracy_rf, flscore_rf, recall_rf, precision_rf, accuracy_dt, flscore_dt, recall_dt,
60         precision_dt, accuracy_svm, flscore_svm, recall_svm, precision_svm
61
62 accuracy_rf, flscore_rf, recall_rf, precision_rf, accuracy_dt, flscore_dt, recall_dt, precision_dt,
63     accuracy_svm, flscore_svm, recall_svm, precision_svm = calculate_metrics(benign_images,
64     malware_images)
65
66 plot_roc_curve(benign_images, malware_images)
67
68 print(f"Random Forest - Accuracy: {accuracy_rf}, F1 Score: {flscore_rf}, Recall: {recall_rf},
69     Precision: {precision_rf}")
70 print(f"Decision Tree - Accuracy: {accuracy_dt}, F1 Score: {flscore_dt}, Recall: {recall_dt},
71     Precision: {precision_dt}")
72 print(f"SVM - Accuracy: {accuracy_svm}, F1 Score: {flscore_svm}, Recall: {recall_svm}, Precision: {
73     precision_svm}")
74
75 stop

```

4.4 Module Description

4.4.1 Classification

Import the 8-Bit vector Grayscale image dataset from google drive using Google Collab. Preprocessing ensures that the data is in a suitable format for modeling. Split the dataset into training and validation sets. Using behavioral analysis the CNN identifies the affected images with trojan horse attacks and classifies non affected images also. In the confusion matrix, which gives the no. of benign and malware image is found. The user must provide input values for the certain fields in order to get results.

4.4.2 Detection of Trojan Horse Attacks

Import the 8-Bit vector Grayscale image dataset from google drive using Google Collab. Removing noise and damaged images from the dataset. Split the dataset into training and validation sets. Preprocessing ensures that the data is in a suitable format for modeling. Using behavioral analysis the dataset identifies the trojan affected images and them into benign and malware. The Confusion Matrix gives out the number of files affected by trojan horse malware and not affected by it. Random forest algorithm is used to display the accuracy for the number of trojan affected images found.

4.5 Steps to execute the project

4.5.1 Step1 :Dataset

- The Maling Dataset
- Sample Image Size 64X64
- Dataset Size - 7000 Images
- Import the maling dataset from drive.
- Spit it into two folders training and validation.
- Training the model.

4.5.2 Step2 : Classification

- Classify the imported dataset using cnn model.
- Classification is done based on name of the trojan.
- It generates the confusion matrix.

4.5.3 Step3 : Detection

- Using the CNN model, detection of the malware affected images and benign images is done.
- A confusion matrix is generated which gives visualisation of False Positives.
- Using performance metrics calculate accuracy, f1score,precision, recall using algorithms such as Random Forest algorithm, Decision Tree algorithm and Support Vector Machine algorithm from the dataset images.

4.5.4 Step4 : ROC curve

- Generate the ROC curve for the given 3 algorithms.
- Which gives the score for each training and predicted data values from the algorithm.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

5.1.1 Input Design

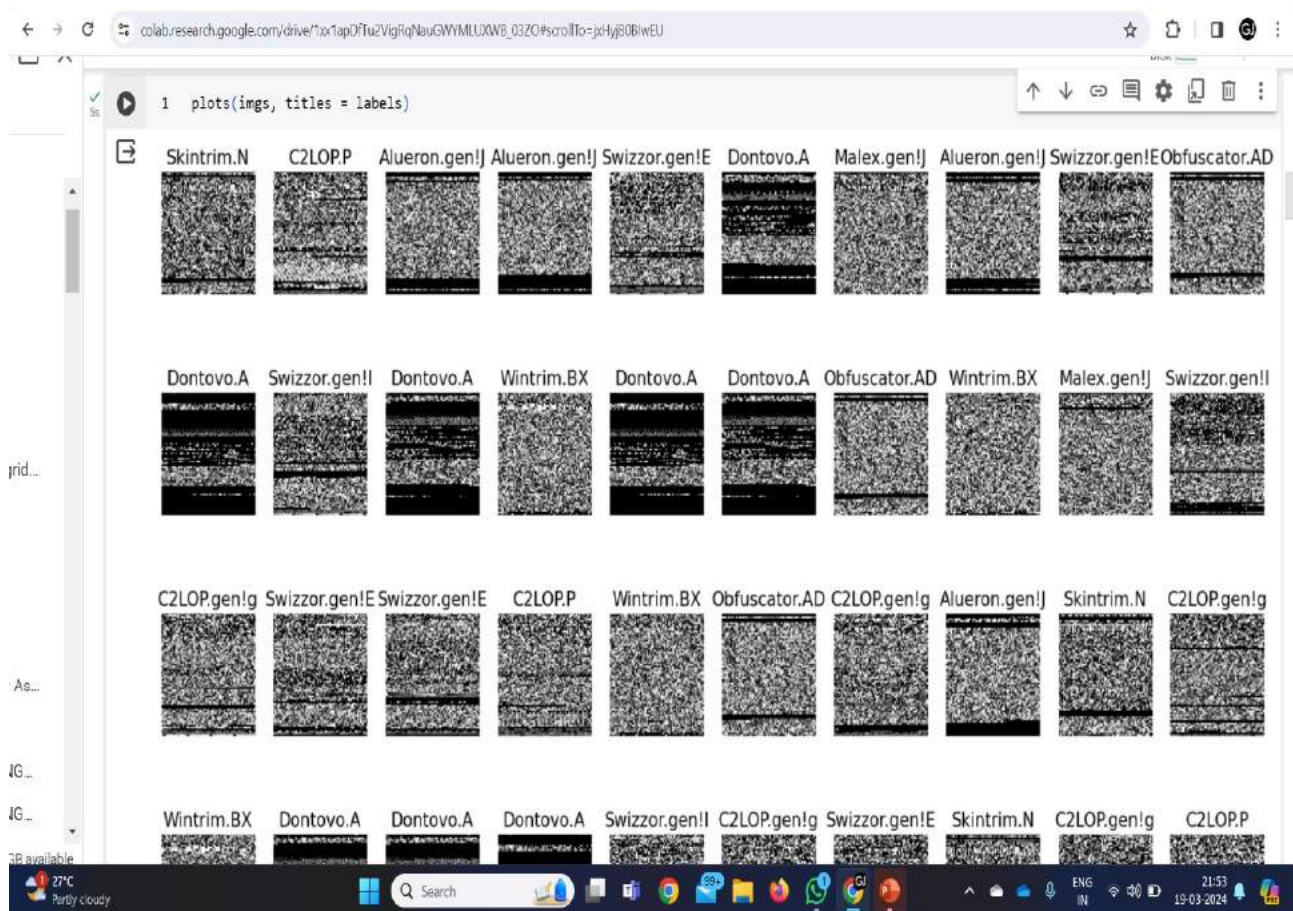


Figure 5.1: Grayscale Images As Input

Figure 5.1 shows to classify a dataset of grayscale images, the images can be input into a machine learning model such as a Random Forest algorithm to determine whether they are malware or benign. The model's output can be visualized in a confusion matrix, which shows the number of images correctly and incorrectly classified into each category. The matrix provides a detailed analysis of the model's performance and can help identify areas for improvement.

5.1.2 Output Design

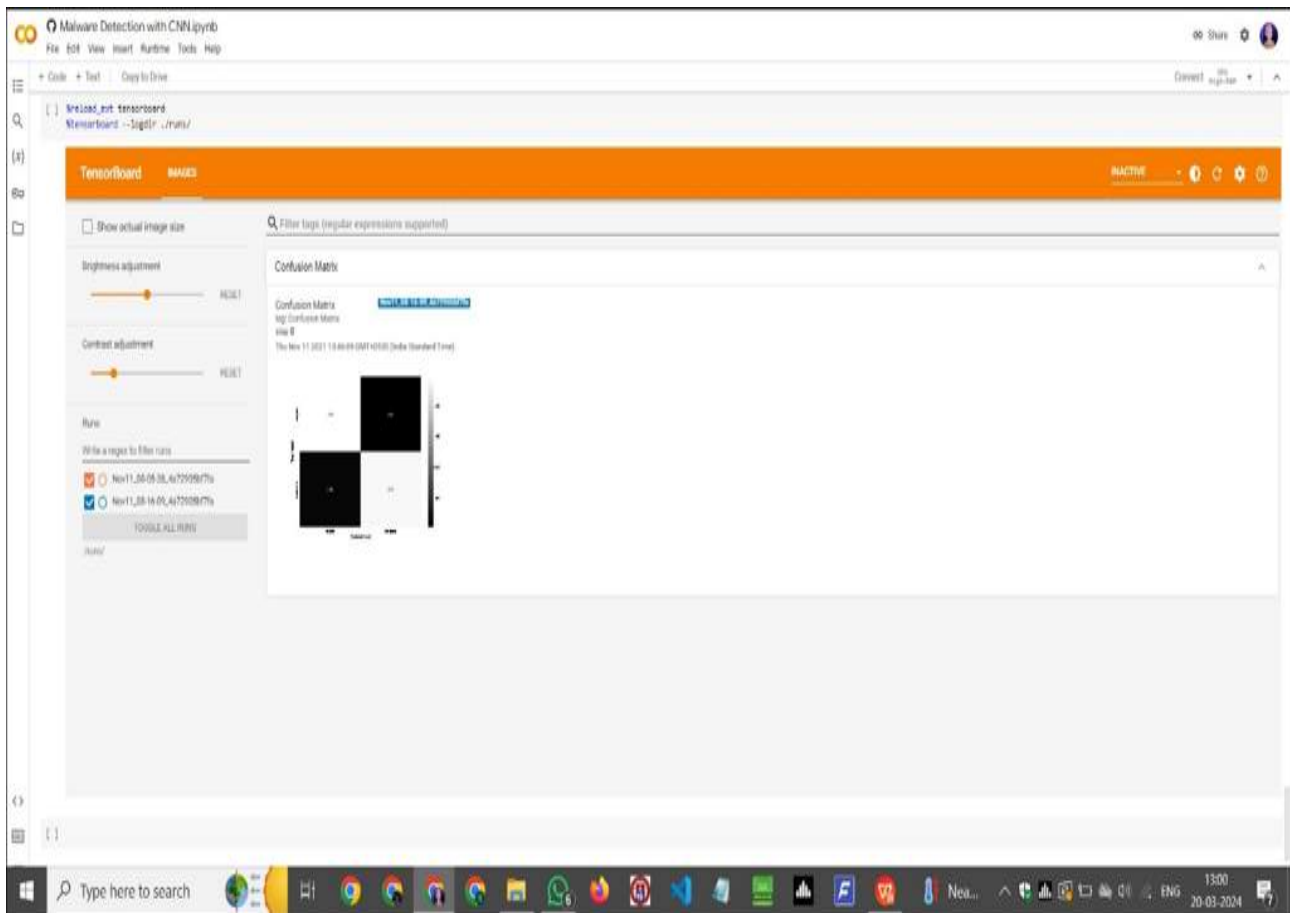


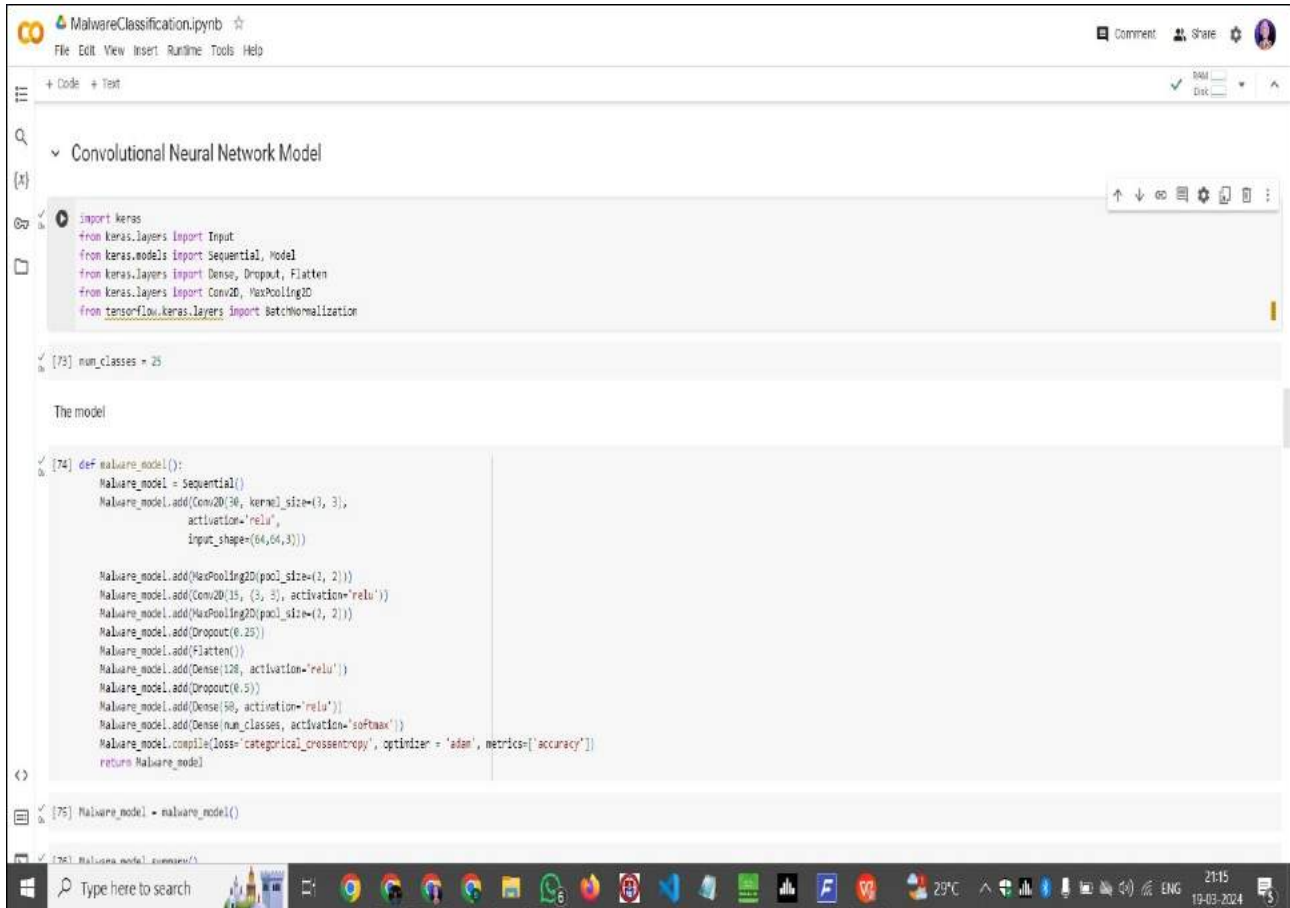
Figure 5.2: **Output As Confusion Matrix**

Figure 5.2 shows a 2x2 confusion matrix is a table that summarizes the performance of a binary classification model, such as a random forest model, in malware detection. It shows how many images were correctly and incorrectly classified as malware or benign, based on the model's score, which can be used to determine the threshold for classification. The matrix provides a clear and concise way to evaluate the model's performance and identify areas for improvement.

5.2 Types of Testing

5.2.1 Unit Testing

Input



```
import keras
from keras.layers import Input
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization

[73] num_classes = 25

The model

[74] def malware_model():
    malware_model = Sequential()
    malware_model.add(Conv2D(30, kernel_size=(3, 3),
        activation='relu',
        input_shape=(64,64,3)))

    malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    malware_model.add(Conv2D(15, (3, 3), activation='relu'))
    malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    malware_model.add(Dropout(0.25))
    malware_model.add(Flatten())
    malware_model.add(Dense(128, activation='relu'))
    malware_model.add(Dropout(0.5))
    malware_model.add(Dense(50, activation='relu'))
    malware_model.add(Dense(num_classes, activation='softmax'))
    malware_model.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics=['accuracy'])
    return malware_model

[75] Malware_model = malware_model()

[76] Malware_model.compile()
```

Figure 5.3: Classification Code

Figure 5.3 shows when grayscale images from a dataset are input into a classification code, the code analyzes the images and outputs a confusion matrix. This matrix shows the number of images that were correctly and incorrectly classified into different categories. The matrix provides a detailed analysis of the nature of the misclassifications made by the algorithm, allowing for an exploration of the prediction performance of the model. By highlighting which classes are predicted most reliably by the model, the confusion matrix enables the selection of prediction models that best suit specific tasks.

Test Result

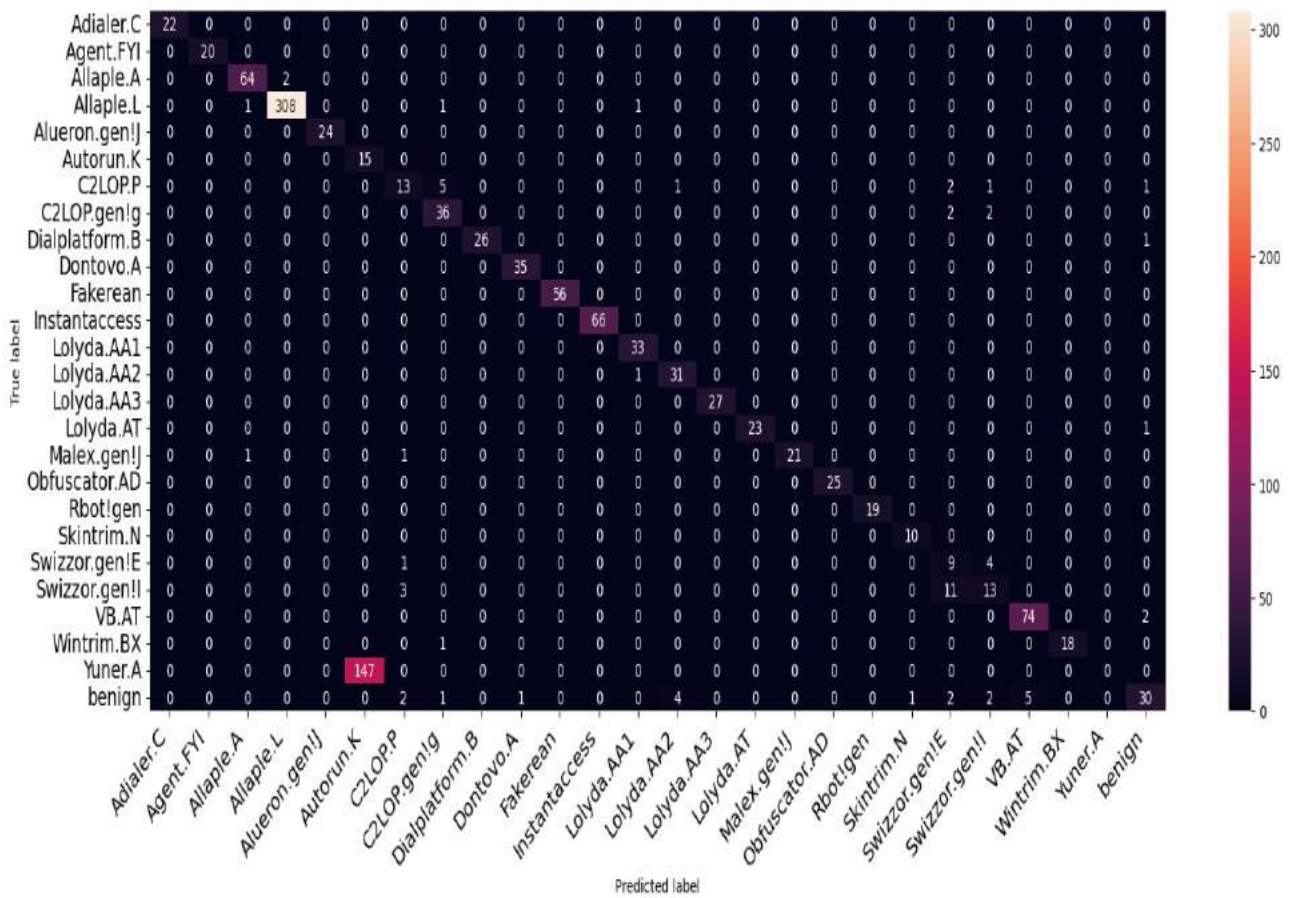


Figure 5.4: Classification Code Result

Figure 5.4 shows the output in the form of a confusion matrix provides a detailed breakdown of how different types of images are classified within a dataset of grayscale images. Each cell in the matrix represents the count or percentage of images that were correctly or incorrectly classified into specific categories by the machine learning or deep learning model. This visual representation allows for a comprehensive analysis of the model's performance across various classes, highlighting where the model excels and where it struggles in classifying different types of images accurately.

5.2.2 Integration Testing

Input 1

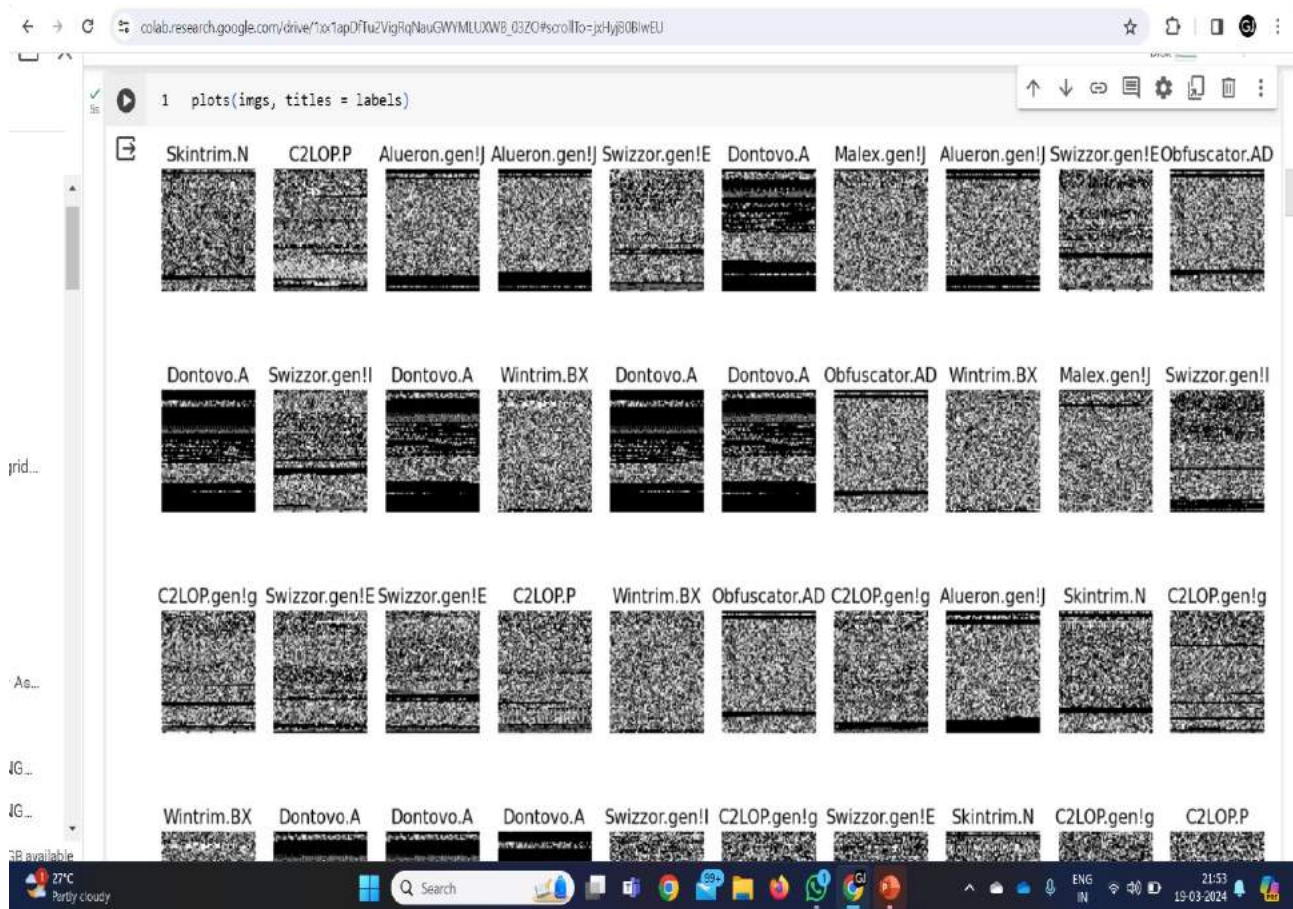


Figure 5.5: Dataset As Input

Figure 5.5 shows by inputting a dataset of grayscale images into a deep learning model trained for malware detection, the algorithm can analyze the images to determine the extent to which each image is affected by malware. This process involves extracting features from the grayscale images to classify them as either benign or malicious based on patterns and textures indicative of malware presence. The model's output provides insights into the level of malware impact on each image, aiding in effective malware detection and classification.

Test Result 1

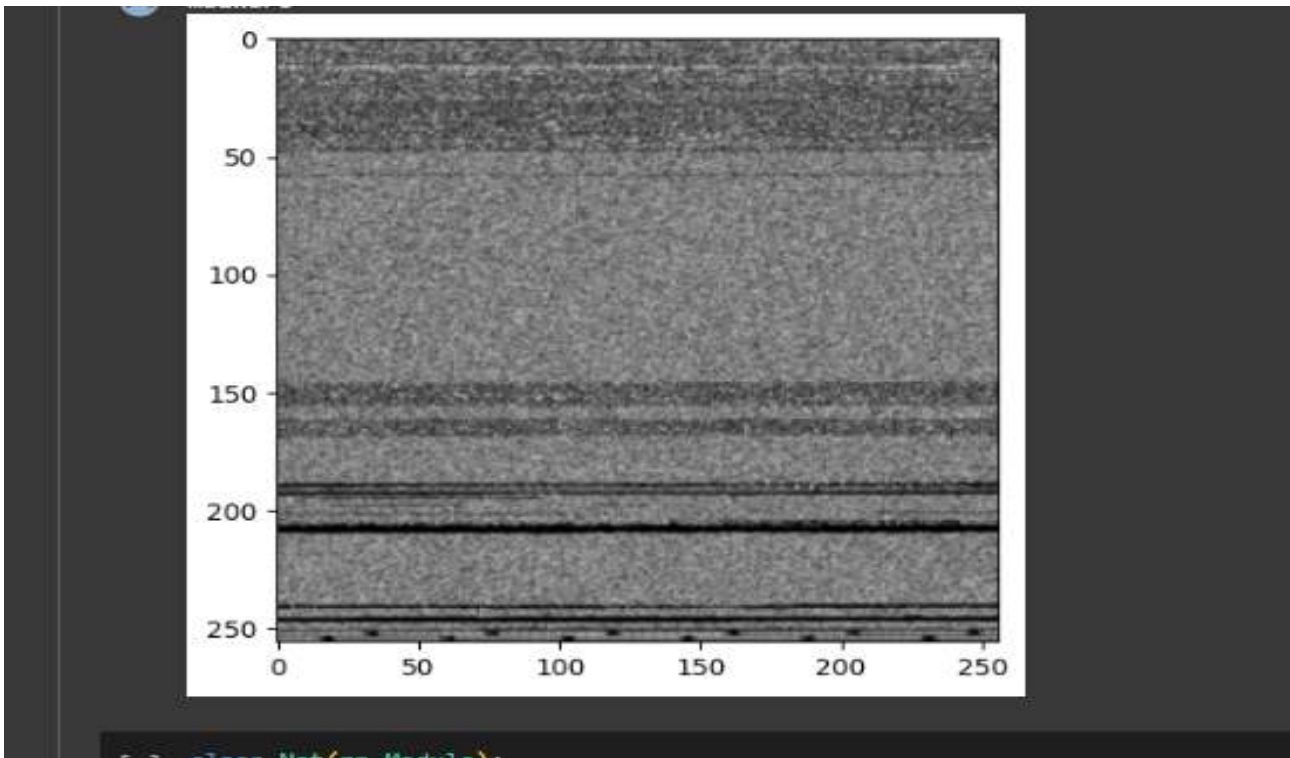
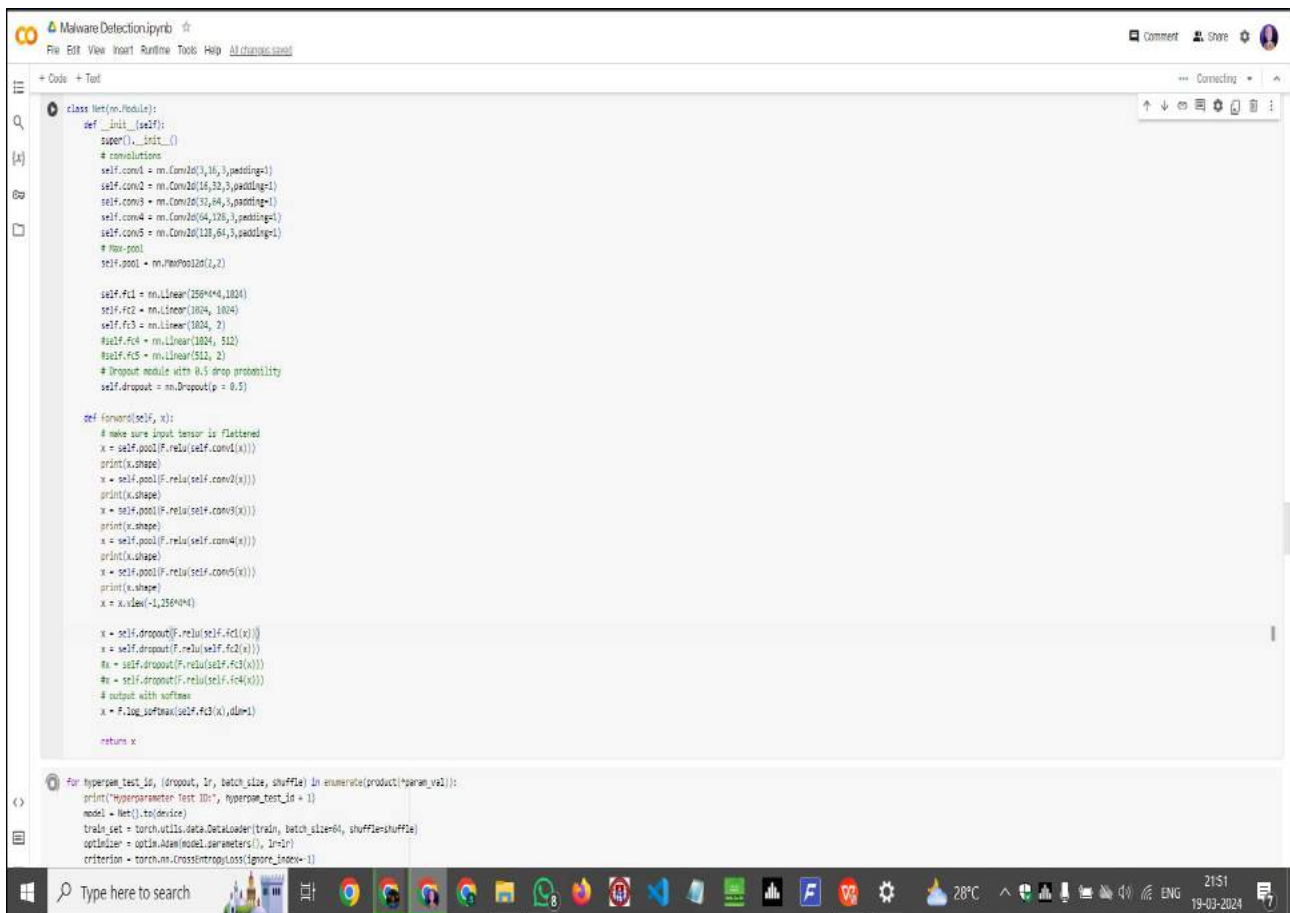


Figure 5.6: **Dataset Result**

Figure 5.6 shows output as grayscale images can be used to represent malware files in a visual format. These images can be generated by converting the binary content of the files into a hexadecimal representation, and then converting those hexadecimal values into pixel values in a grayscale image. The resulting images can then be analyzed using deep learning techniques, such as convolutional neural networks (CNNs), to classify the malware as benign or malicious. The pixel values in the grayscale images can indicate the frequency of different opcode sequences, which can be used as input features for malware detection. Dark lines or patterns in the grayscale images may indicate the presence of malware, as malware often has unique structural texture features that distinguish it from benign files. The accuracy of the random forest algorithm can be used to determine the likelihood that an image is malicious, with higher scores indicating a higher likelihood of malware.

Input 2



```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        # convolutions
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv4 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv5 = nn.Conv2d(128, 64, 3, padding=1)
        # Max pool
        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(256*4*4, 1024)
        self.fc2 = nn.Linear(1024, 1024)
        self.fc3 = nn.Linear(1024, 2)
        #self.fc4 = nn.Linear(1024, 512)
        #self.fc5 = nn.Linear(512, 2)
        # Dropout module with 0.5 drop probability
        self.dropout = nn.Dropout(p = 0.5)

    def forward(self, x):
        # make sure input tensor is flattened
        x = self.pool(F.relu(self.conv1(x)))
        print(x.shape)
        x = self.pool(F.relu(self.conv2(x)))
        print(x.shape)
        x = self.pool(F.relu(self.conv3(x)))
        print(x.shape)
        x = self.pool(F.relu(self.conv4(x)))
        print(x.shape)
        x = self.pool(F.relu(self.conv5(x)))
        print(x.shape)
        x = x.view(-1, 256*4*4)

        x = self.dropout(F.relu(self.fc1(x)))
        x = self.dropout(F.relu(self.fc2(x)))
        #x = self.dropout(F.relu(self.fc3(x)))
        #x = self.dropout(F.relu(self.fc4(x)))
        # output with softmax
        x = F.log_softmax(self.fc3(x), dim=1)

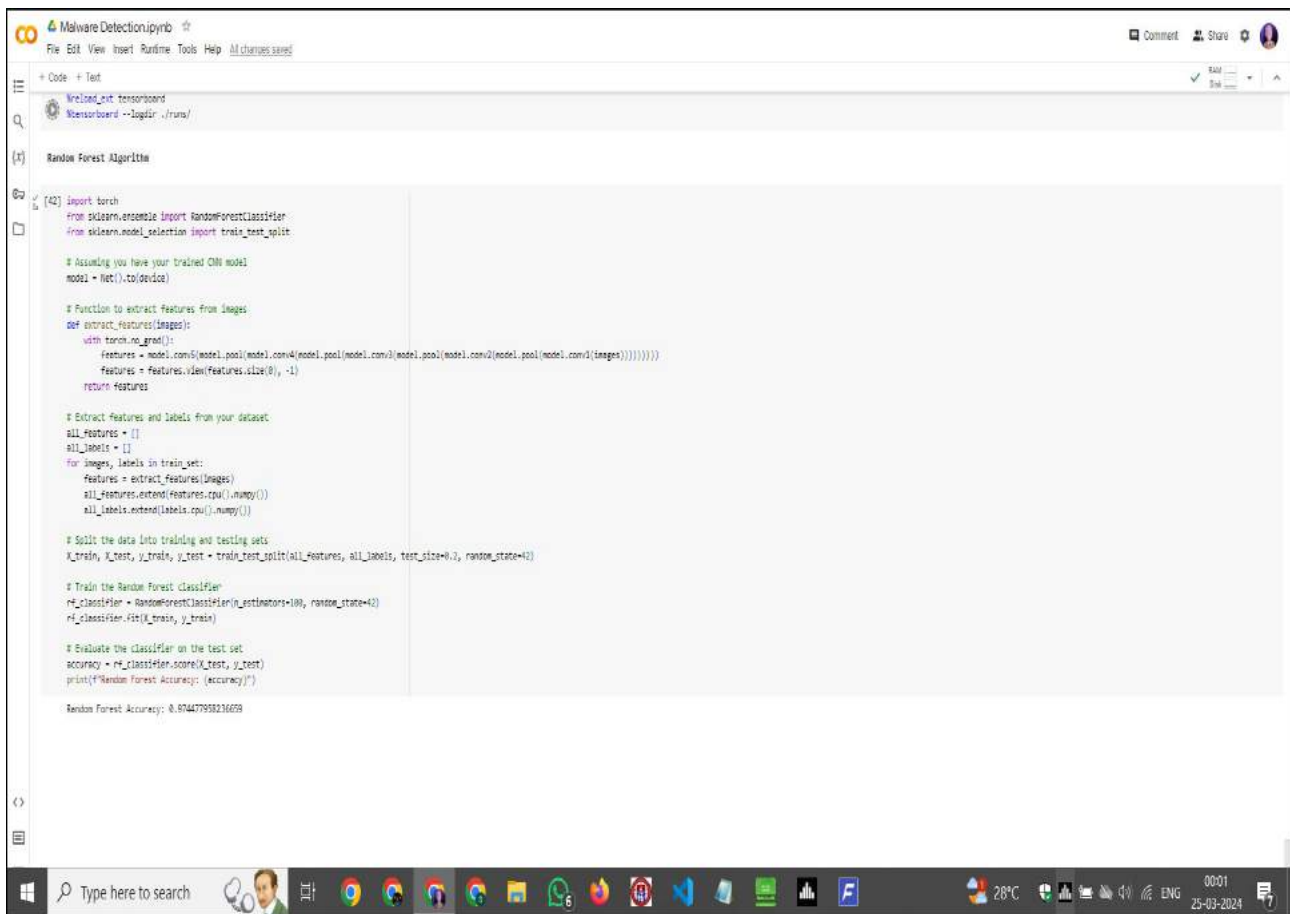
        return x

for hypersam_test_id, (dropout, lr, batch_size, shuffle) in enumerate(product('param_val2')):
    print("Hyperparameter Test ID:", hypersam_test_id + 1)
    model = Net()
    train_set = torch.utils.data.DataLoader(train, batch_size=batch_size, shuffle=shuffle)
    optimizer = optim.Adam(model.parameters(), lr=lr)
    criterion = torch.nn.CrossEntropyLoss(ignore_index=-1)
```

Figure 5.7: Accuracy Code

Figure 5.7 shows the input for Random Forest algorithm can be applied to grayscale images for classification, using techniques such as feature extraction and histogram analysis. The algorithm's accuracy in predicting grayscale image classes can be used to determine whether an image is malware or benign. This approach has been used in studies involving the colorization of grayscale aerial images, where the algorithm's regression capabilities were employed to predict color values for each pixel. However, the specific application of Random Forest to malware detection using grayscale images is not explicitly mentioned in the search results.

Test Result 2



```
MalwareDetection.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text
Kwladz_wit_tensorboard
Kwladz_wit_tensorboard --logdir ./runs/

Random Forest Algorithm

[42] In: import torch
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Assuming you have your trained CNN model
model = Net().to(device)

# Function to extract features from images
def extract_features(images):
    with torch.no_grad():
        features = model.conv5(model.pool(model.conv4(model.pool(model.conv3(model.pool(model.conv2(model.pool(model.conv1(images))))))))
        features = features.view(features.size(0), -1)
    return features

# Extract features and labels from your dataset
all_features = []
all_labels = []
for images, labels in train_set:
    features = extract_features(images)
    all_features.extend(features.cpu().numpy())
    all_labels.extend(labels.cpu().numpy())

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(all_features, all_labels, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

# Evaluate the Classifier on the test set
accuracy = rf_classifier.score(X_test, y_test)
print(f"Random Forest Accuracy: {accuracy}")

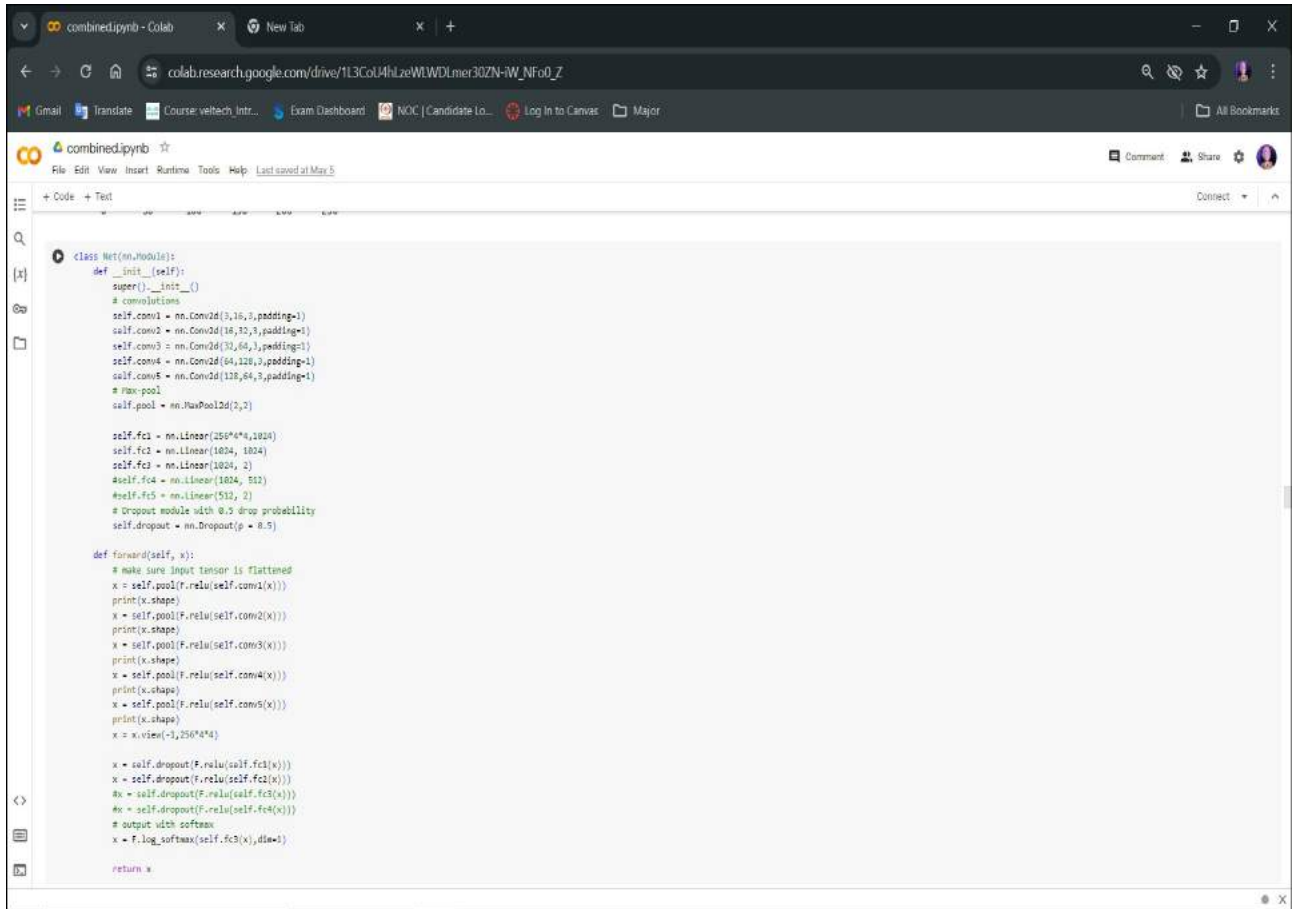
Random Forest Accuracy: 0.974477959236629
```

Figure 5.8: Accuracy Code Result

Figure 5.8 shows the Random Forest algorithm demonstrates high accuracy in malware detection, achieving scores of 98.7 and 99.27 in different studies. This exceptional performance indicates its effectiveness in distinguishing between malware and benign files, making it a robust choice for cybersecurity tasks. The algorithm consistently outperforms other techniques, showcasing its reliability and suitability for accurate classification. With near-perfect accuracy rates and strong performance across various datasets, Random Forest emerges as a dependable and efficient method for detecting malware.

5.2.3 System Testing

Input



```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        # convolutions
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv4 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv5 = nn.Conv2d(128, 64, 3, padding=1)
        # max-pool
        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(256*4*4, 1024)
        self.fc2 = nn.Linear(1024, 1024)
        self.fc3 = nn.Linear(1024, 2)
        self.fc4 = nn.Linear(1024, 512)
        self.fc5 = nn.Linear(512, 2)
        # Dropout module with 0.5 drop probability
        self.dropout = nn.Dropout(p = 0.5)

    def forward(self, x):
        # make sure input tensor is flattened
        x = self.pool(F.relu(self.conv1(x)))
        print(x.shape)
        x = self.pool(F.relu(self.conv2(x)))
        print(x.shape)
        x = self.pool(F.relu(self.conv3(x)))
        print(x.shape)
        x = self.pool(F.relu(self.conv4(x)))
        print(x.shape)
        x = self.pool(F.relu(self.conv5(x)))
        print(x.shape)
        x = x.view(-1, 256*4*4)

        x = self.dropout(F.relu(self.fc1(x)))
        x = self.dropout(F.relu(self.fc2(x)))
        #x = self.dropout(F.relu(self.fc3(x)))
        #x = self.dropout(F.relu(self.fc4(x)))
        # output with softmax
        x = F.log_softmax(self.fc3(x), dim=1)

        return x
```

Figure 5.9:

Figure 5.9 shows the system designed for both classification and detection tasks, likely focusing on image analysis. Users input data, possibly images, for classification and detection purposes. Upon receiving input, the system executes algorithms to analyze the images, potentially identifying objects, patterns, or anomalies relevant to the task. The output generated by the system likely includes information about the detected objects or identified features within the images, as well as any classifications or labels associated with them. Additionally, the output may highlight any images identified as being affected or manipulated in some manner, providing insights into potential issues or abnormalities within the data

Test Result 1

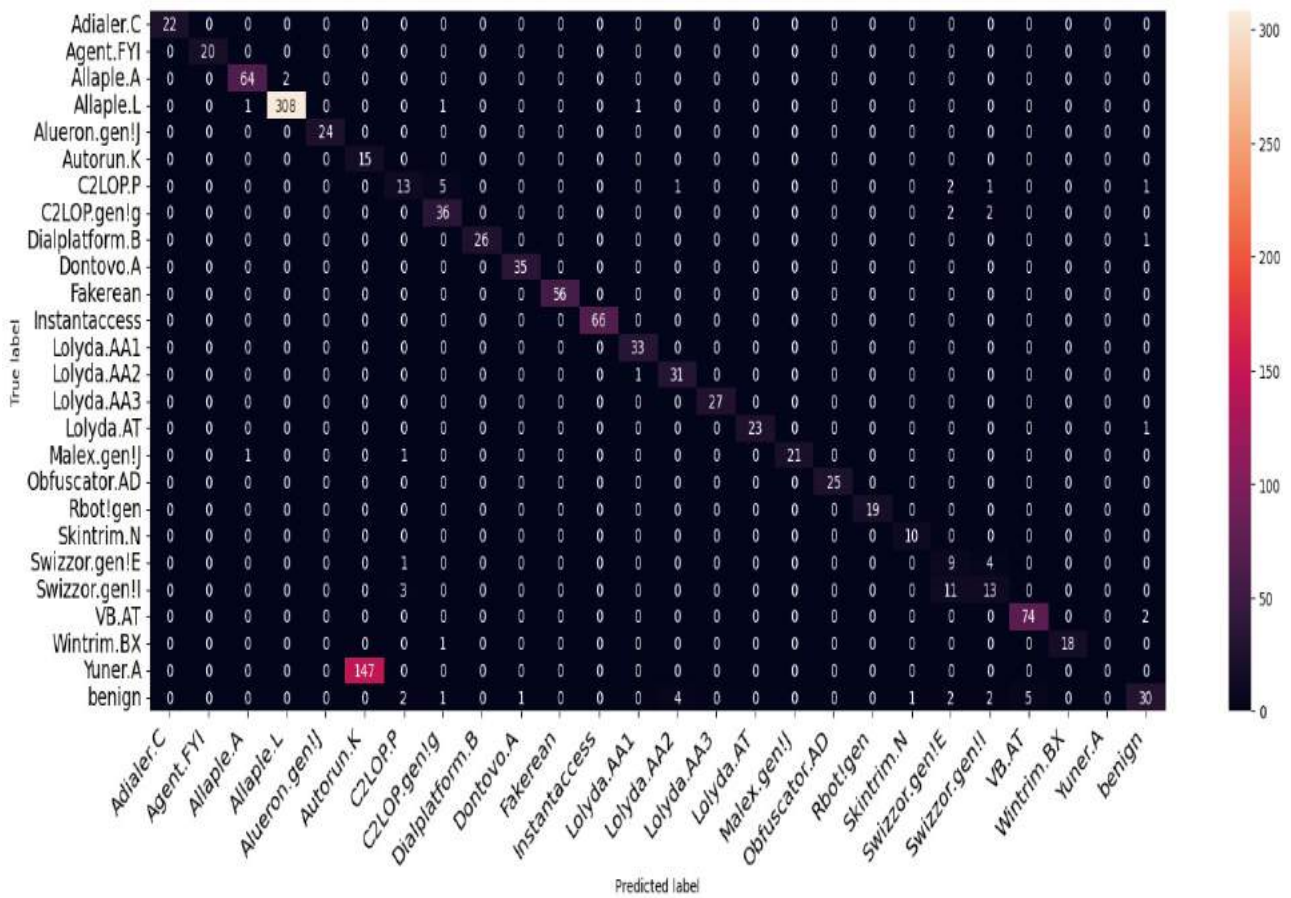


Figure 5.10: Classification Code Result

Figure 5.10 shows the output in the form of a confusion matrix provides a detailed breakdown of how different types of images are classified within a dataset of grayscale images. Each cell in the matrix represents the count or percentage of images that were correctly or incorrectly classified into specific categories by the machine learning or deep learning model. This visual representation allows for a comprehensive analysis of the model's performance across various classes, highlighting where the model excels and where it struggles in classifying different types of images accurately.

Test Result 2

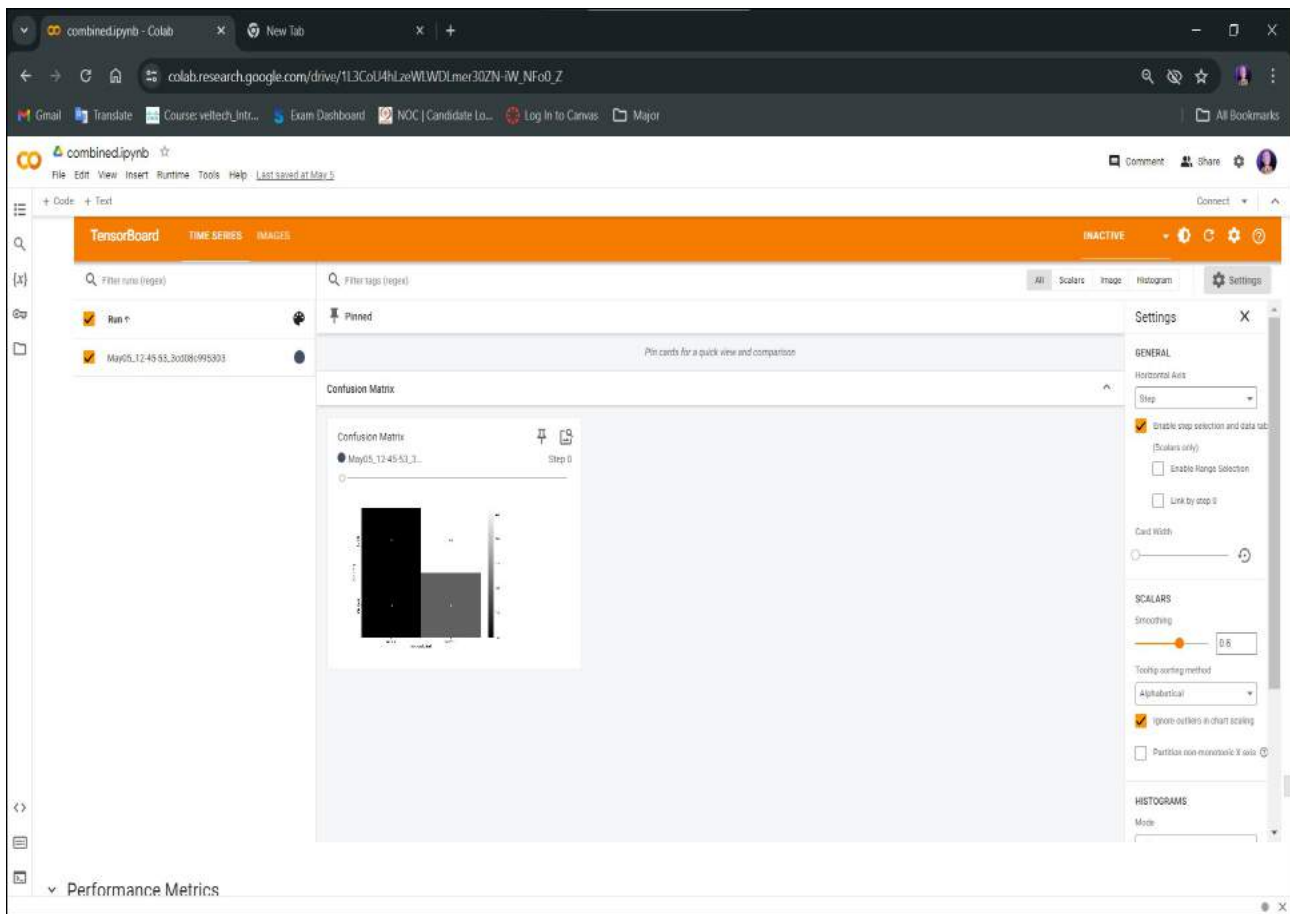


Figure 5.11: Confusion Matrix Of Detection

Figure 5.11 shows the confusion matrix represents the performance of a model in classifying instances as either "benign" or "malware." The key points are: The top left quadrant shows 0 true positives for the "benign" class, indicating the model did not correctly identify any instances as "benign." The top right quadrant shows 2.6 false negatives for the "benign" class, suggesting the model incorrectly classified 2.6 "benign" instances as "malware." The bottom left quadrant shows 0 false positives for the "malware" class, meaning the model did not incorrectly classify any "malware" instances as "benign." The bottom right quadrant shows 1 true positive for the "malware" class, indicating the model correctly identified 1 instance as "malware." The non-integer values in the confusion matrix are unusual, as confusion matrices typically display integer counts. This could imply the matrix is showing normalized or weighted values rather than raw counts, suggesting the model's performance is being evaluated using a more complex metric than simple accuracy.

Test Result 3

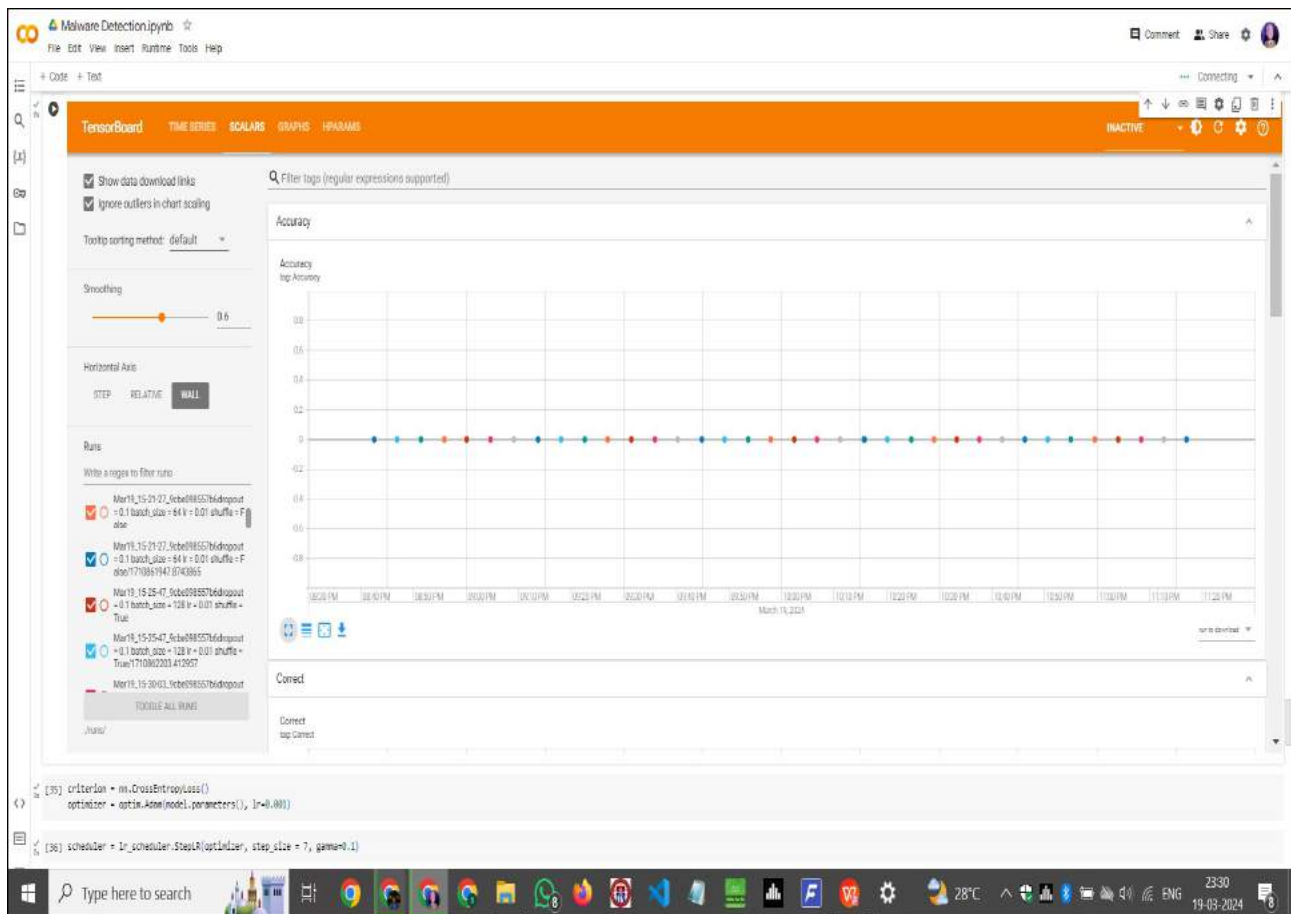
Figure 5.12: **Scalar Detection**

Figure 5.12 shows output of scalar detection of malware using grayscale images in Google Colab involves converting files to grayscale images, creating a dataset, training a CNN model, and achieving high accuracy in classifying malware vs benign files. This method leverages texture differences in binary representations for classification, offering advantages like detecting new malware variants efficiently. However, it may face challenges like computational complexity and vulnerability to adversarial attacks, requiring ongoing refinement for practical application in malware detection. Overall, this approach shows promise for accurate and automated malware detection but needs further development to enhance robustness and real-world usability.

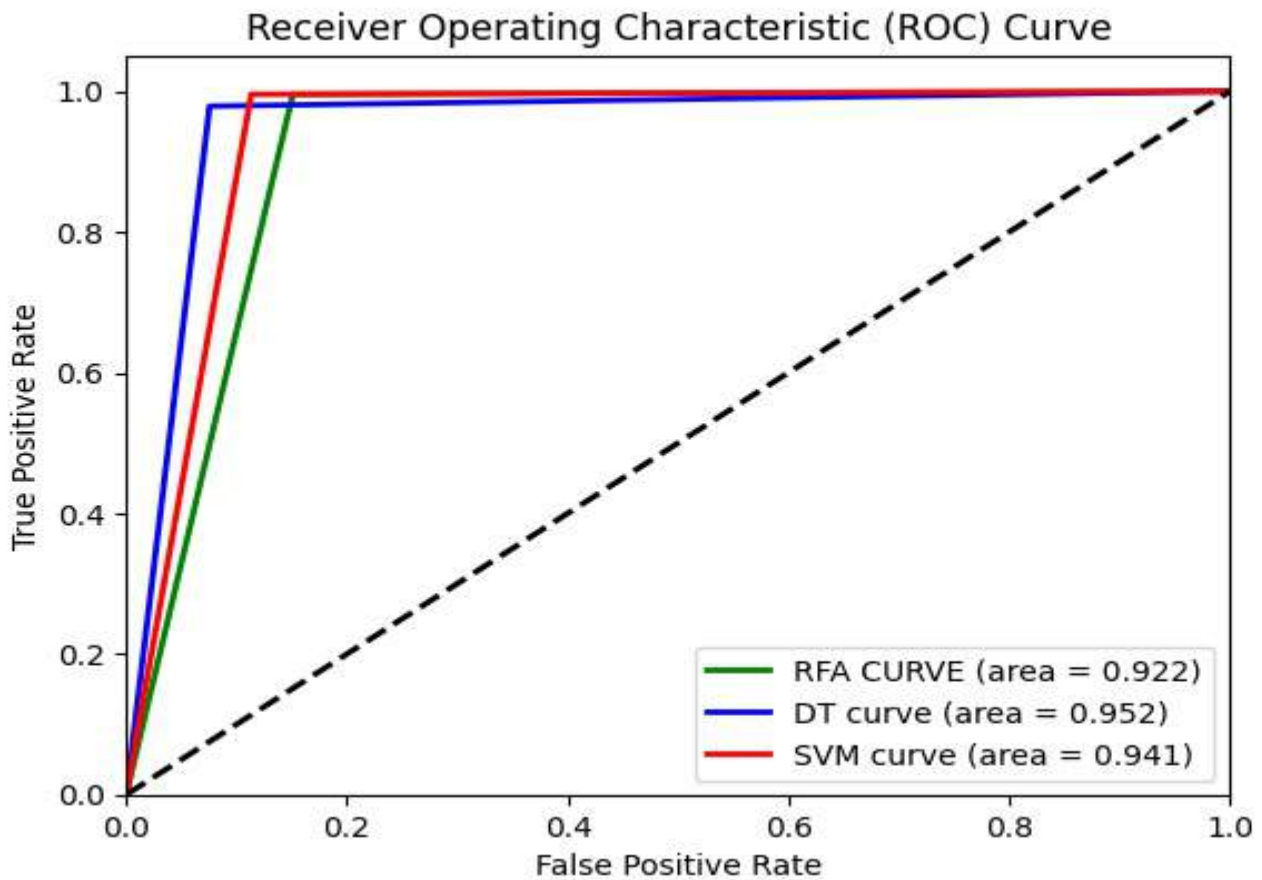


Figure 5.13: ROC Curve

Figure 5.13 shows the ROC curve compares the performance of three different classifiers: RFA, DT, and SVM. The green line represents the ROC curve for the RFA classifier, which has an AUC (Area Under the Curve) of 0.927, indicating excellent performance. The blue line represents the DT classifier with an AUC of 0.899, and the red line represents the SVM classifier with an AUC of 0.901. The dashed black line represents a random classifier, which serves as a baseline for comparison. The closer the ROC curve is to the top-left corner of the graph, the better the classifier's performance, as it indicates a higher true positive rate and lower false positive rate. Based on the AUC values, the RFA classifier outperforms the DT and SVM classifiers in this particular scenario. The ROC curve is a widely used tool in machine learning to evaluate the trade-offs between sensitivity (true positive rate) and specificity ($1 - \text{false positive rate}$) for binary classification problems.

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

The proposed model for zero-day malware detection combines visualization techniques with deep learning architectures to address evolving malware threats. It transforms malware data into visual representations using optimized image processing techniques, which are then fed into machine learning algorithms and deep learning architectures, particularly Convolutional Neural Networks (CNNs). By integrating static analysis, dynamic analysis, and image processing, the model comprehensively covers diverse malware behaviors. The incorporation of visualization techniques enhances interpretability and aids in feature extraction. The goal of this model is to achieve robust and intelligent zero-day malware detection, capable of identifying previously unseen malware variants in real-time deployments. Its scalability emphasizes proactive detection and mitigation of emerging malware threats, representing a significant advancement in cybersecurity. The performance metrics obtained from this model is 97% accuracy and 3% loss.

Algorithm	Accuracy(%)	Precision(%)	Recall(%)	F1-score(%)
RF	96.89	96.72	99.57	98.72
DT	96.20	97.88	97.4	97.67
SVM	97.24	97.11	99.57	98.33

Table 6.1: **Performance Metrics**

Table 6.1 shows the performance metrics like Accuracy, Precision, Recall, and F1 Score play crucial roles in assessing the performance of classification models. Accuracy measures the overall correctness of predictions made by a model. Precision focuses on the proportion of true positives among all positive predictions, indicating how many positive predictions are correct. Recall, also known as the true positive rate, evaluates how well the model identifies all positive instances. F1 Score, a metric that combines precision and recall, provides a balanced measure of a model's perfor-

mance, especially in scenarios where optimizing either precision or recall alone may compromise the model's effectiveness. These metrics are essential for understanding the trade-offs between precision and recall and are vital for evaluating the efficacy of machine learning models across various applications and datasets.

6.2 Comparison of Existing and Proposed System

6.2.1 Existing System

The existing system designed for detecting malware within affected images showcases a sophisticated approach by employing a diverse array of algorithms tailored for enhanced detection capabilities. By integrating a mix of machine learning methodologies including Random Forest, Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Convolutional Neural Network (CNN), and Naïve Bayes, the system can effectively discern and categorize various forms of malicious software. Each algorithm contributes distinct advantages to the detection process: Random Forest and Decision Tree excel in ensemble learning and interpretability, SVM specializes in identifying optimal hyperplanes, KNN relies on proximity-based classification, CNN is adept at image analysis, and Naïve Bayes offers simplicity and efficiency. This amalgamation of diverse algorithms enables the system to harness the unique strengths of each method, thereby bolstering its capacity to accurately and efficiently detect and classify malware present in affected images, ensuring a comprehensive and robust defense against malicious threats.

6.2.2 Proposed System

The proposed model for zero-day malware detection described in the text introduces a sophisticated hybrid approach that combines visualization techniques with deep learning architectures. By leveraging image processing techniques optimized for Machine Learning Algorithms (MLAs) and deep learning architectures, the model transforms malware data into visual representations suitable for analysis. It incorporates Convolutional Neural Networks (CNNs) for effective feature extraction and pattern recognition tasks. The hybrid approach integrates static analysis, dynamic analysis, and image processing within a unified framework, enabling comprehensive coverage of diverse malware behaviors. Visualization techniques enhance in-

interpretability and aid in feature extraction. Ultimately, this model aims to achieve robust zero-day malware detection, capable of identifying previously unseen malware variants or behaviors in real-time deployments. Its scalability makes it suitable for proactive detection and mitigation of emerging threats, marking a significant advancement in cybersecurity.

Aspect	Existing System	Proposed System
Malware Detection Techniques	Visual deep learning for cybersecurity.	Hybrid visual-deep learning approach.
Feature Engineering	Utilize automated feature extraction.	Uses visualization for interpretable, automated feature extraction.
Data Availability	Limited benchmark dataset availability.	Optimizes MLAs for visual analysis.
Handling Packed Malware	Struggles with packed malware.	Unified malware behavior analysis.
Interpretability of Model Decisions	Classical models lack interpretability.	Visual techniques enhance interpretability.
Scalability and Real-Time Deployment	Real-time deployment challenges.	Scalable real-time deployment.
Accuracy	94%	97%
Future Enhancements	Advanced ensemble learning.	Reinforcement learning investigation.

Table 6.2: Comparison of Existing System vs Proposed System

Table 6.2 shows the table includes a detailing information related to malware detection techniques, visual deep learning for cybersecurity, and a hybrid visual-deep learning approach. The tables shows difference of existing sysytem and proposed sysytem based on features, data availability challenges, handling packed malware, interpretability of model decisions, scalability, real-time deployment issues, and accuracy rates. It also outlines future enhancements for both approaches, focusing on advanced ensemble learning and reinforcement learning investigation. This comprehensive overview sheds light on the current strategies and potential advancements in malware detection and cybersecurity through the integration of visual techniques and deep learning architectures.

6.3 Sample Code

```
1 from google.colab import drive
2 drive.mount('/content/drive')
3 path_root = "/content/drive/MyDrive/malimg_dataset/validation"
4 from keras.preprocessing.image import ImageDataGenerator
5 batches = ImageDataGenerator().flow_from_directory(directory=path_root, target_size=(64,64),
6     batch_size=10000)
7 batches.class_indices
8 imgs, labels = next(batches)
9 imgs.shape
10 labels.shape
11 import matplotlib.pyplot as plt
12 def plots(ims, figsize=(20,30), rows=10, interp=False, titles=None):
13     if type(ims[0]) is np.ndarray:
14         ims = np.array(ims).astype(np.uint8)
15         if (ims.shape[-1] != 3):
16             ims = ims.transpose((0,2,3,1))
17     f = plt.figure(figsize=figsize)
18     cols = 10 # len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
19     for i in range(0,50):
20         sp = f.add_subplot(rows, cols, i+1)
21         sp.axis('Off')
22         if titles is not None:
23             sp.set_title(list(batches.class_indices.keys())[np.argmax(titles[i])], fontsize=16)
24         plt.imshow(ims[i], interpolation=None if interp else 'none')
25 import numpy as np
26 import pandas as pd
27 classes = batches.class_indices.keys()
28 perc = (sum(labels)/labels.shape[0])*100
29 plt.xticks(rotation='vertical')
30 plt.bar(classes, perc)
31 from sklearn.model_selection import train_test_split
32 X_train, X_test, y_train, y_test = train_test_split(imgs/255., labels, test_size=0.2)
33 X_train.shape
34 X_test.shape
35 y_train.shape
36 y_test.shape
37 import keras
38 from keras.layers import Input
39 from keras.models import Sequential, Model
40 from keras.layers import Dense, Dropout, Flatten
41 from keras.layers import Conv2D, MaxPooling2D
42 from tensorflow.keras.layers import BatchNormalization
43 def malware_model():
44     Malware_model = Sequential()
45     Malware_model.add(Conv2D(30, kernel_size=(3, 3),
46         activation='relu',
47         input_shape=(64,64,3)))
```

```

47
48     Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
49     Malware_model.add(Conv2D(15, (3, 3), activation='relu'))
50     Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
51     Malware_model.add(Dropout(0.25))
52     Malware_model.add(Flatten())
53     Malware_model.add(Dense(128, activation='relu'))
54     Malware_model.add(Dropout(0.5))
55     Malware_model.add(Dense(50, activation='relu'))
56     Malware_model.add(Dense(num_classes, activation='softmax'))
57     Malware_model.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics=['accuracy'])
58     return Malware_model
59 Malware_model = malware_model()
60 Malware_model.summary()
61 y_train.shape
62 import numpy as np
63 y_train_new = np.argmax(y_train, axis=1)
64 y_train_new
65 from sklearn.utils.class_weight import compute_class_weight
66 class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y_train_new), y=
    y_train_new)
67 class_weights_dict = dict(zip(np.unique(y_train_new), class_weights))
68 Malware_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, class_weight=
    class_weights_dict)
69 scores = Malware_model.evaluate(X_test, y_test)
70 print('Final CNN accuracy: ', scores[1])
71 import numpy as np
72 import pandas as pd
73 import numpy as np
74 y_pred_prob = Malware_model.predict(X_test)
75 y_pred = np.argmax(y_pred_prob, axis=1)
76 y_pred
77 y_test2 = np.argmax(y_test, axis=1)
78 y_test2
79 from sklearn import metrics
80 c_matrix = metrics.confusion_matrix(y_test2, y_pred)
81 import seaborn as sns
82 import pandas as pd
83 drive.mount('/content/gdrive')
84 def confusion_matrix(confusion_matrix, class_names, figsize = (10,7), fontsize=14):
85     """Prints a confusion matrix, as returned by sklearn.metrics.confusion_matrix, as a heatmap.
86
87     Arguments
88     -----
89     confusion_matrix: numpy.ndarray
90         The numpy.ndarray object returned from a call to sklearn.metrics.confusion_matrix.
91         Similarly constructed ndarrays can also be used.
92     class_names: list
93         An ordered list of class names, in the order they index the given confusion matrix.
94     figsize: tuple

```



```

95         A 2-long tuple, the first value determining the horizontal size of the ouputted figure,
96         the second determining the vertical size. Defaults to (10,7).
97     fontsize: int
98         Font size for axes labels. Defaults to 14.
99     """
100     df_cm = pd.DataFrame(
101         confusion_matrix, index=class_names, columns=class_names,
102     )
103     # Assuming 'confusion_matrix' contains your confusion matrix data
104     # df_confusion = pd.DataFrame(confusion_matrix)
105     # df_cm.to_csv('/content/gdrive/My Drive/confusion_matrix.csv', index=False)
106     fig = plt.figure(figsize=figsize)
107     try:
108         heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
109     except ValueError:
110         raise ValueError("Confusion matrix values must be integers.")
111     heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=
        fontsize)
112     heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize=
        fontsize)
113     plt.ylabel('True label')
114     plt.xlabel('Predicted label')
115     class_names= batches.class_indices.keys()
116     confusion_matrix(c_matrix, class_names, figsize = (20,7), fontsize=14

```

6.3.1 Output 1

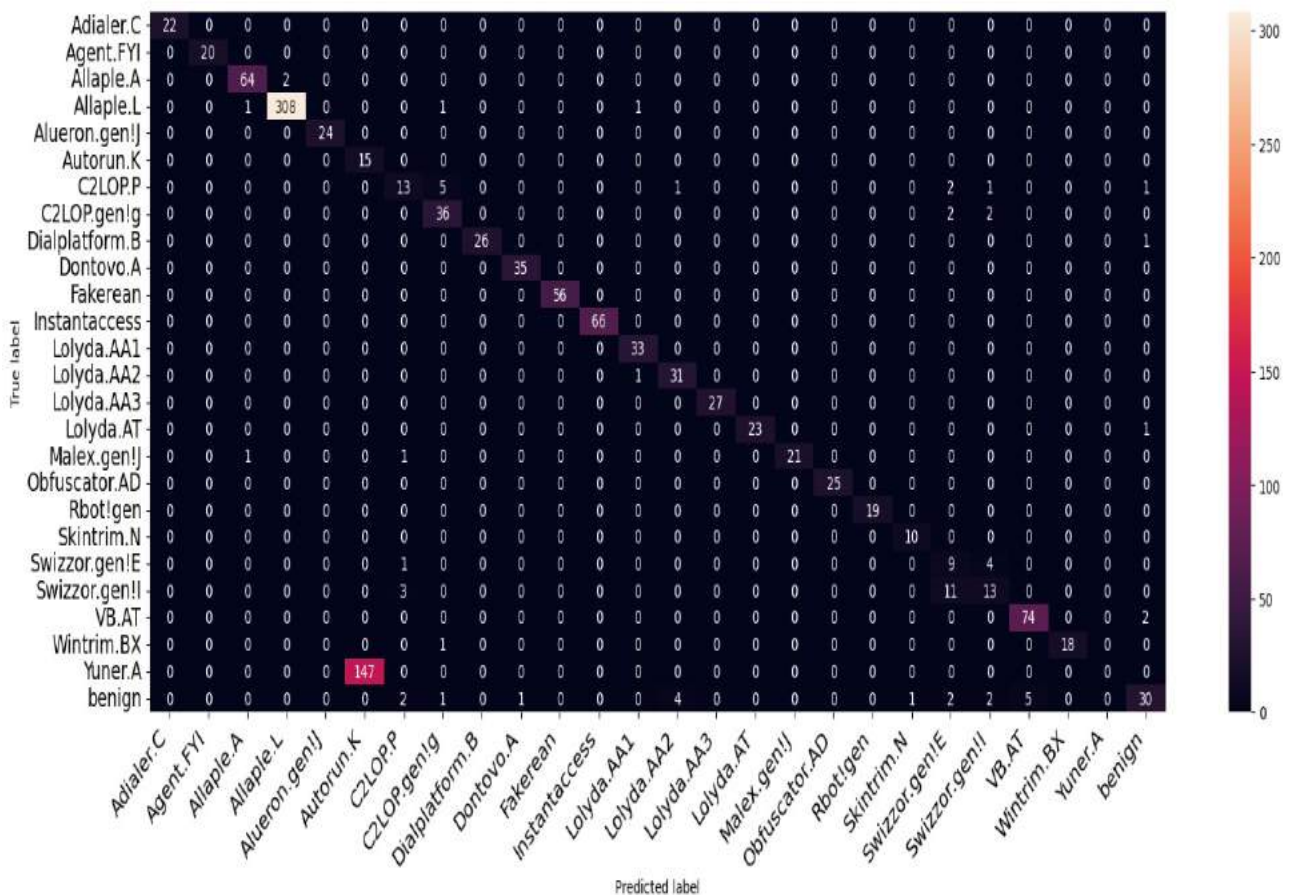


Figure 6.1: Classification Code Result

Figure 6.1 shows the output in the form of a confusion matrix provides a detailed breakdown of how different types of images are classified within a dataset of grayscale images. Each cell in the matrix represents the count or percentage of images that were correctly or incorrectly classified into specific categories by the machine learning or deep learning model. This visual representation allows for a comprehensive analysis of the model's performance across various classes, highlighting where the model excels and where it struggles in classifying different types of images accurately.

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

The proposed approach for predicting cyber hacking breaches leverages machine learning algorithms to proactively identify and forecast potential cyber threats. By integrating machine learning models, the system adapts and evolves in response to emerging cyber threats, enhancing its predictive capabilities. This novel approach advances the field of cybersecurity by establishing a proactive and dynamic paradigm for anticipating and mitigating cyber risks, strengthening digital ecosystems against malicious activities. The proposed system combines visualization techniques with deep learning architectures, transforming malware data into visual representations for analysis. It incorporates Convolutional Neural Networks (CNNs) for effective feature extraction and pattern recognition tasks. The hybrid approach integrates static analysis, dynamic analysis, and image processing, enabling comprehensive coverage of diverse malware behaviors. This model aims to achieve robust zero-day malware detection, capable of identifying previously unseen malware variants or behaviors in real-time deployments. The performance metrics obtained from this model is 97% accuracy and 3% loss.

7.2 Future Enhancements

Future enhancements could involve integrating more advanced machine learning models, such as deep learning algorithms, to further improve the accuracy and robustness of cyber hacking breach predictions. Additionally, incorporating real-time data streams and leveraging anomaly detection techniques could enhance the timeliness and effectiveness of threat detection. Furthermore, exploring ensemble learning methods to combine the strengths of multiple models could yield even more reliable

predictions. Moreover, developing automated response systems that can dynamically adjust security measures based on predictive insights would bolster proactive defense strategies. These enhancements aim to continually refine and strengthen cybersecurity defenses in the face of evolving cyber threats. **Real-time Monitoring:** Implement real-time monitoring capabilities to detect cyber hacking breaches as they occur or shortly thereafter. This involves integrating the prediction model into existing security systems and continuously analyzing incoming data streams for suspicious activity

•**Scalability and Performance:** Ensure that the prediction system can handle large volumes of data and maintain high performance under increasing loads. This may involve optimizing algorithms for parallel processing, leveraging distributed computing frameworks, and implementing efficient data storage and retrieval mechanisms.

•**Human-in-the-loop Systems:** Develop human-in-the-loop systems that combine the strengths of machine learning models with human expertise. This approach allows security analysts to review and validate predictions, incorporate domain knowledge, and provide feedback to improve model performance.

•**Integration of Advanced Algorithms:** Explore the integration of more advanced machine learning algorithms such as deep learning, ensemble methods, and gradient boosting algorithms. These algorithms may capture more complex patterns and relationships in the data, potentially leading to improved prediction accuracy. PLAGIARISM REPORT

Plagiarism Scan Report

Report Generated on: Apr 14, 2024



Content Checked for Plagiarism

Deep learning has emerged as a powerful tool for enhancing malware detection and mitigation efforts, offering automated feature extraction capabilities and the potential to detect zero-day malware. Cyberattacks, on the other hand, pose a serious threat to these systems. Without awareness of these attacks, the system cannot identify them, which might lead to impaired or completely crippled performance. Cybersecurity experts may increase threat detection accuracy, reaction times, and adaptability to changing malware landscapes by utilising deep learning models such as Convolutional Neural Networks (CNNs) and hybrid techniques. Maximising the efficiency of deep learning in cybersecurity applications requires addressing issues including representative training datasets, false positives, and model adaptability, underscoring the necessity of further research and development in this area.



Figure 7.1: Plagiarism Report

Chapter 8

SOURCE CODE & POSTER PRESENTATION

8.1 Source Code

```
1 import torch
2 import torchvision
3 import torch.nn.functional as F
4 from torchvision import datasets, transforms, models
5 import torch.optim as optim
6 import torch.nn as nn
7 from torch.optim import lr_scheduler
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import time
11 import os
12 import pathlib
13 import copy
14 from torch.utils.tensorboard import SummaryWriter
15 from google.colab import drive
16 drive.mount('/content/drive')
17 !pip install split-folders
18 import splitfolders
19 splitfolders.ratio('/content/drive/MyDrive/trojan_classes', output="output", seed=1337, ratio
    =(.8,0.2))
20 data_dir= pathlib.Path(os.path.normpath('/content/drive/MyDrive/trojan_classes'))
21 train_path = data_dir/"training"
22 val_path = data_dir/"validation"
23 malware_train = len(os.listdir(train_path/"malware"))
24 benign_train = len(os.listdir(train_path/"benign"))
25 malware_valid = len(os.listdir(val_path/"malware"))
26 benign_valid = len(os.listdir(val_path/"benign"))
27 print("Malware Train Images: {}".format(malware_train))
28 print("Benign Train Images: {}".format(benign_train))
29 print("Malware Validation Images: {}".format(malware_valid))
30 print("Benign Validation Images: {}".format(benign_valid))
31 transform = transforms.Compose([transforms.Resize((256,256)),
32                                 transforms.ToTensor(),
33                                 transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])])
34 train = datasets.ImageFolder(train_path, transform = transform)
```

```

35 val = datasets.ImageFolder(val_path, transform=transform)
36 batch_size = 128
37 train_set = torch.utils.data.DataLoader(train, batch_size=batch_size, shuffle = True)
38 val_set = torch.utils.data.DataLoader(val, batch_size = batch_size, shuffle=True)
39 classes = os.listdir(train_path)
40 print(classes)
41 def imshow(image, title = None):
42     image = image.numpy().transpose((1,2,0))
43     mean = np.array([0.485,0.456,0.406])
44     std = np.array([0.229,0.224,0.225])
45     image = image*std + mean
46     image = np.clip(image, 0, 1)
47     plt.imshow(image)
48     if title is not None:
49         print(title)
50     plt.pause(0.001)
51 images, labels = next(iter(train_set))
52 out = torchvision.utils.make_grid(images)
53 imshow(out, title=[classes[x] for x in labels])
54 imshow(images[1], classes[labels[0]])
55 class Net(nn.Module):
56     def __init__(self):
57         super().__init__()
58         # convolutions
59         self.conv1 = nn.Conv2d(3,16,3,padding=1)
60         self.conv2 = nn.Conv2d(16,32,3,padding=1)
61         self.conv3 = nn.Conv2d(32,64,3,padding=1)
62         self.conv4 = nn.Conv2d(64,128,3,padding=1)
63         self.conv5 = nn.Conv2d(128,64,3,padding=1)
64         # Max-pool
65         self.pool = nn.MaxPool2d(2,2)
66
67         self.fc1 = nn.Linear(256*4*4,1024)
68         self.fc2 = nn.Linear(1024, 1024)
69         self.fc3 = nn.Linear(1024, 2)
70         #self.fc4 = nn.Linear(1024, 512)
71         #self.fc5 = nn.Linear(512, 2)
72         # Dropout module with 0.5 drop probability
73         self.dropout = nn.Dropout(p = 0.5)
74
75     def forward(self, x):
76         # make sure input tensor is flattened
77         x = self.pool(F.relu(self.conv1(x)))
78         print(x.shape)
79         x = self.pool(F.relu(self.conv2(x)))
80         print(x.shape)
81         x = self.pool(F.relu(self.conv3(x)))
82         print(x.shape)
83         x = self.pool(F.relu(self.conv4(x)))
84         print(x.shape)

```

```

85         x = self.pool(F.relu(self.conv5(x)))
86         print(x.shape)
87         x = x.view(-1,256*4*4)
88
89         x = self.dropout(F.relu(self.fc1(x)))
90         x = self.dropout(F.relu(self.fc2(x)))
91         #x = self.dropout(F.relu(self.fc3(x)))
92         #x = self.dropout(F.relu(self.fc4(x)))
93         # output with softmax
94         x = F.log_softmax(self.fc3(x),dim=1)
95
96         return x
97 import torch
98 model = torch.nn.Linear(256, 10) # Example linear model with 256 input features and 10 output
    features
99 device = torch.device('cpu')
100 model = Net()
101 model.to(device)
102 if torch.cuda.is_available():
103     device = torch.device("cuda")
104 else:
105     device = torch.device("cpu")
106 print(device)
107 writer = SummaryWriter()
108 import torch
109 from torch.utils.data import DataLoader
110 from torch.utils.tensorboard import SummaryWriter
111 import os
112 os.system('CUDA_LAUNCH_BLOCKING=1')
113 os.environ['CUDA_LAUNCH_BLOCKING'] = '1'
114 from torch.utils.tensorboard import SummaryWriter
115 writer = SummaryWriter()
116 dataiter = iter(train_set)
117 images, labels = next(dataiter)
118 writer.add_graph(model, images.to(device))
119 writer.close()
120 writer.add_graph(model, images.to(device))
121 from torchsummary import summary
122 import torch
123 model = model.to(device)
124 summary(model, input_size= (3,256,256))
125 from itertools import product
126 parameters = dict(
127     dropout = [0.1, 0.3, 0.5],
128     lr = [0.01, 0.001],
129     batch_size = [32,64,128],
130     shuffle = [True, False]
131 )
132
133 param_val = [v for v in parameters.values()]

```



```

134 print(param_val)
135
136 for dropout, lr, batch_size, shuffle in product(*param_val):
137     print(dropout, lr, batch_size, shuffle)
138 def get_num_correct(preds, labels):
139     return preds.argmax(dim=1).eq(labels).sum().item()
140 import torch
141 import torch.nn as nn
142 import torch.nn.functional as F
143 import torch.optim as optim
144 from torch.utils.tensorboard import SummaryWriter
145 num_classes = 2
146 torch.autograd.set_detect_anomaly(True)
147 def filter_valid_samples(preds, labels, num_classes):
148     # Create a mask for valid labels
149     valid_mask = (labels >= 0) & (labels < num_classes)
150
151     # Filter out invalid samples
152     valid_preds = preds[valid_mask]
153     valid_labels = labels[valid_mask]
154
155     # Replace any remaining out-of-bounds labels with a default value
156     default_value = 0
157     valid_labels[valid_labels < 0] = default_value
158     valid_labels[valid_labels >= num_classes] = default_value
159
160     # Create a mask for valid predictions
161     valid_preds_mask = torch.any(valid_preds >= 0, dim=1)
162
163     # Filter out invalid predictions
164     valid_preds = valid_preds[valid_preds_mask]
165     valid_labels = valid_labels[valid_preds_mask]
166
167     return valid_preds, valid_labels
168 class Net(nn.Module):
169     def __init__(self):
170         super().__init__()
171         # convolutions
172         self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
173         self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
174         self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
175         self.conv4 = nn.Conv2d(64, 128, 3, padding=1)
176         self.conv5 = nn.Conv2d(128, 64, 3, padding=1)
177         # Max-pool
178         self.pool = nn.MaxPool2d(2, 2)
179
180         self.fc1 = nn.Linear(256*4*4, 1024)
181         self.fc2 = nn.Linear(1024, 1024)
182         self.fc3 = nn.Linear(1024, 2)
183         #self.fc4 = nn.Linear(1024, 512)

```

```

184         #self.fc5 = nn.Linear(512, 2)
185         # Dropout module with 0.5 drop probability
186         self.dropout = nn.Dropout(p = 0.5)
187
188     def forward(self, x):
189         # make sure input tensor is flattened
190         x = self.pool(F.relu(self.conv1(x)))
191         print(x.shape)
192         x = self.pool(F.relu(self.conv2(x)))
193         print(x.shape)
194         x = self.pool(F.relu(self.conv3(x)))
195         print(x.shape)
196         x = self.pool(F.relu(self.conv4(x)))
197         print(x.shape)
198         x = self.pool(F.relu(self.conv5(x)))
199         print(x.shape)
200         x = x.view(-1,256*4*4)
201
202         x = self.dropout(F.relu(self.fc1(x)))
203         x = self.dropout(F.relu(self.fc2(x)))
204         #x = self.dropout(F.relu(self.fc3(x)))
205         #x = self.dropout(F.relu(self.fc4(x)))
206         # output with softmax
207         x = F.log_softmax(self.fc3(x),dim=1)
208
209         return x
210 for hyperparam_test_id, (dropout, lr, batch_size, shuffle) in enumerate(product(*param_val)):
211     print("Hyperparameter Test ID:", hyperparam_test_id + 1)
212     model = Net().to(device)
213     train_set = torch.utils.data.DataLoader(train, batch_size=64, shuffle=shuffle)
214     optimizer = optim.Adam(model.parameters(), lr=lr)
215     criterion = torch.nn.CrossEntropyLoss(ignore_index=-1)
216     comment = f'dropout = {dropout} batch_size = {batch_size} lr = {lr} shuffle = {shuffle}'
217     writer = SummaryWriter(comment=comment)
218
219     for epoch in range(1):
220         total_loss = 0
221         total_correct = 0
222
223         for images, labels in train_set:
224             images, labels = images.to(device), labels.to(device)
225
226             preds = model(images)
227
228             valid_preds, valid_labels = filter_valid_samples(preds, labels, num_classes)
229
230             loss = criterion(valid_preds, valid_labels)
231             total_loss += loss.item()
232             total_correct += get_num_correct(valid_preds, valid_labels)
233

```

```

234         optimizer.zero_grad()
235         loss.backward()
236         optimizer.step()
237
238     writer.add_scalar("Loss", total_loss, epoch)
239     writer.add_scalar("Correct", total_correct, epoch)
240     writer.add_scalar("Accuracy", total_correct / len(train_set), epoch)
241
242     print("dropout:", dropout, "batch_size:", batch_size, "lr:", lr, "shuffle:", shuffle)
243     print("epoch:", epoch, "total_correct:", total_correct, "loss:", total_loss)
244
245     print("-----")
246
247     writer.add_hparams(
248         {"dropout": dropout, "lr": lr, "batch_size": batch_size, "shuffle": shuffle},
249         {
250             "accuracy": total_correct / len(train_set),
251             "loss": total_loss,
252         },
253     )
254
255 writer.close()
256 %reload_ext tensorboard
257 %tensorboard --logdir ./runs/
258 criterion = nn.CrossEntropyLoss()
259 optimizer = optim.Adam(model.parameters(), lr=0.001)
260 scheduler = lr_scheduler.StepLR(optimizer, step_size = 7, gamma=0.1)
261 history={'train_loss':[], 'valid_loss':[], 'train_acc':[], 'valid_acc':[]}
262 num_epochs=1
263
264
265 for epoch in range(num_epochs):
266     train_loss, train_correct=0.0,0
267     model.train()
268     for images, labels in train_set:
269
270         images, labels = images.to(device), labels.to(device)
271         optimizer.zero_grad()
272         output = model(images)
273         loss = criterion(output, labels)
274         loss.backward()
275         optimizer.step()
276         train_loss += loss.item() * images.size(0)
277         scores, predictions = torch.max(output.data, 1)
278         train_correct += (predictions == labels).sum().item()
279
280     valid_loss, val_correct = 0.0, 0
281     model.eval()
282     for images, labels in val_set:
283

```

```

284     images, labels = images.to(device), labels.to(device)
285     output = model(images)
286     loss=criterion(output, labels)
287     valid_loss+=loss.item()*images.size(0)
288     scores, predictions = torch.max(output.data,1)
289     val_correct+=(predictions == labels).sum().item()
290
291
292     train_loss = train_loss / len(train_set.sampler)
293     train_acc = train_correct / len(train_set.sampler)*100
294     valid_loss = valid_loss / len(val_set.sampler)
295     valid_acc = val_correct / len(val_set.sampler) * 100
296
297     print("Epoch:{}/{} \t average training loss:{:.4f} average validation loss:{:.4f} \t average
298           training acc:{:.2f} % average validation acc:{:.2f} %".format(epoch + 1, num_epochs,
299                                                                           train_loss
300                                                                           ,
301                                                                           valid_loss
302                                                                           ,
303                                                                           train_acc
304                                                                           ,
305                                                                           valid_acc
306                                                                           ))
307
308     history['train_loss'].append(train_loss)
309     history['valid_loss'].append(valid_loss)
310     history['train_acc'].append(train_acc)
311     history['valid_acc'].append(valid_acc)
312
313 import shutil
314
315 shutil.rmtree("./runs/")
316
317 nb_classes = 2
318
319
320 confusion_matrix = torch.zeros(nb_classes, nb_classes)
321 with torch.no_grad():
322     for i, data in enumerate(val_set, 0):
323         images, labels = data
324         images = images.to(device)
325         labels = labels.to(device)
326         outputs = model(images)
327         _, preds = torch.max(outputs, 1)
328         for t, p in zip(labels.view(-1), preds.view(-1)):
329             confusion_matrix[t.long(), p.long()] += 1
330
331 cm = confusion_matrix.numpy()
332 cm = cm / cm.sum(axis=1)
333
334 import seaborn as sns
335
336 classes = ("benign", "malware")
337
338 fig = plt.figure(figsize=(10, 7))

```

```

329 heatmap = sns.heatmap( cm, annot=True, cmap="gray" ,
330                         xticklabels = classes , yticklabels=classes )
331
332 plt.ylabel("True Label")
333 plt.xlabel("Predicted Label")
334 writer = SummaryWriter()
335 writer.add_figure("Confusion Matrix", fig)
336 writer.close()
337 %reload_ext tensorboard
338 %tensorboard --logdir ./runs/
339 import torch
340 from sklearn.ensemble import RandomForestClassifier
341 from sklearn.model_selection import train_test_split
342
343 # Assuming you have your trained CNN model
344 model = Net().to(device)
345
346 # Function to extract features from images
347 def extract_features(images):
348     with torch.no_grad():
349         features = model.conv5(model.pool(model.conv4(model.pool(model.conv3(model.pool(model.conv2(
350             model.pool(model.conv1(images))))))))))
351         features = features.view(features.size(0), -1)
352     return features
353
354 # Extract features and labels from your dataset
355 all_features = []
356 all_labels = []
357 for images, labels in train_set:
358     features = extract_features(images)
359     all_features.extend(features.cpu().numpy())
360     all_labels.extend(labels.cpu().numpy())
361
362 # Split the data into training and testing sets
363 X_train, X_test, y_train, y_test = train_test_split(all_features, all_labels, test_size=0.2,
364                                                     random_state=42)
365
366 # Train the Random Forest classifier
367 rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
368 rf_classifier.fit(X_train, y_train)
369
370 # Evaluate the classifier on the test set
371 accuracy = rf_classifier.score(X_test, y_test)
372 print(f"Random Forest Accuracy: {accuracy}")

```

Output 1

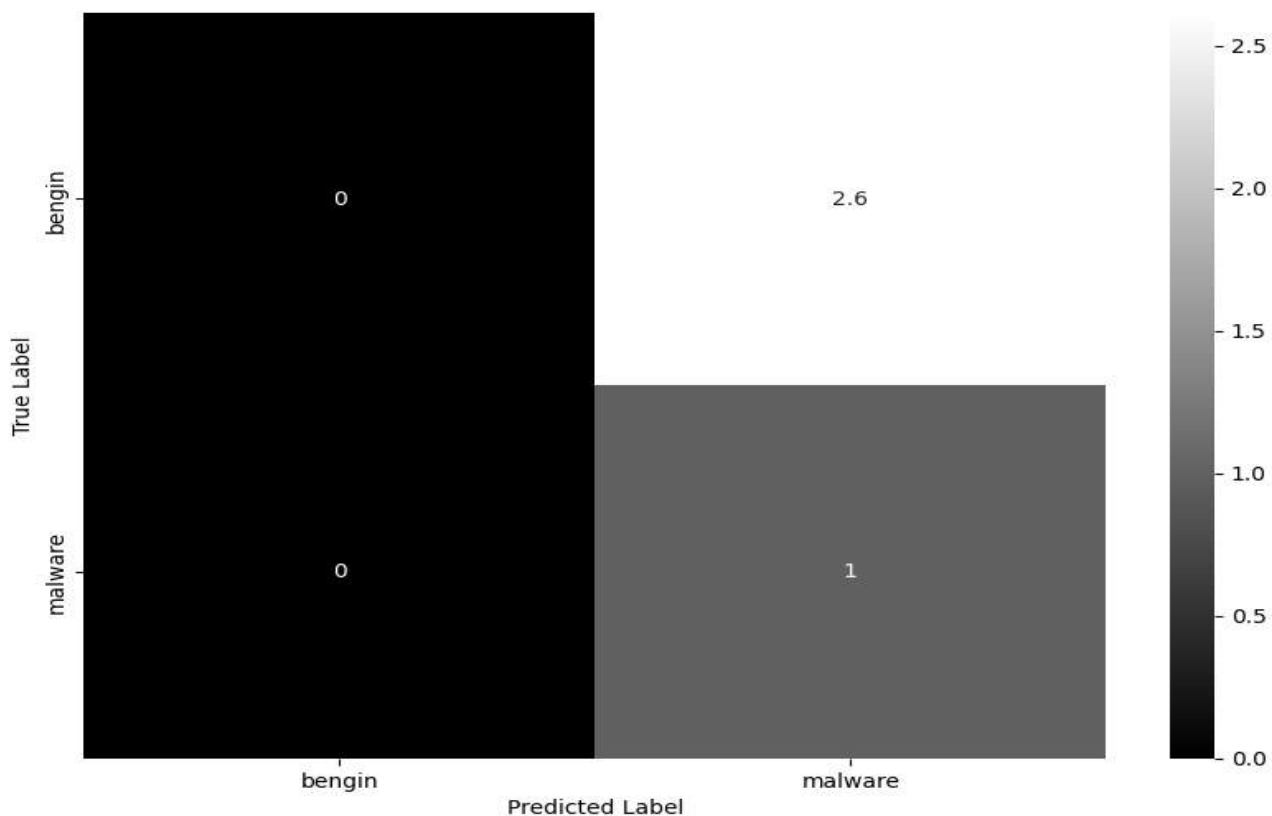


Figure 8.1: **Detection Confusion Matrix**

Figure 8.1 shows the confusion matrix represents the performance of a model in classifying instances as either "benign" or "malware." The key points are: The top left quadrant shows 0 true positives for the "benign" class, indicating the model did not correctly identify any instances as "benign." The top right quadrant shows 2.6 false negatives for the "benign" class, suggesting the model incorrectly classified 2.6 "benign" instances as "malware." The bottom left quadrant shows 0 false positives for the "malware" class, meaning the model did not incorrectly classify any "malware" instances as "benign." The bottom right quadrant shows 1 true positive for the "malware" class, indicating the model correctly identified 1 instance as "malware." The non-integer values in the confusion matrix are unusual, as confusion matrices typically display integer counts. This could imply the matrix is showing normalized or weighted values rather than raw counts, suggesting the model's performance is being evaluated using a more complex metric than simple accuracy.

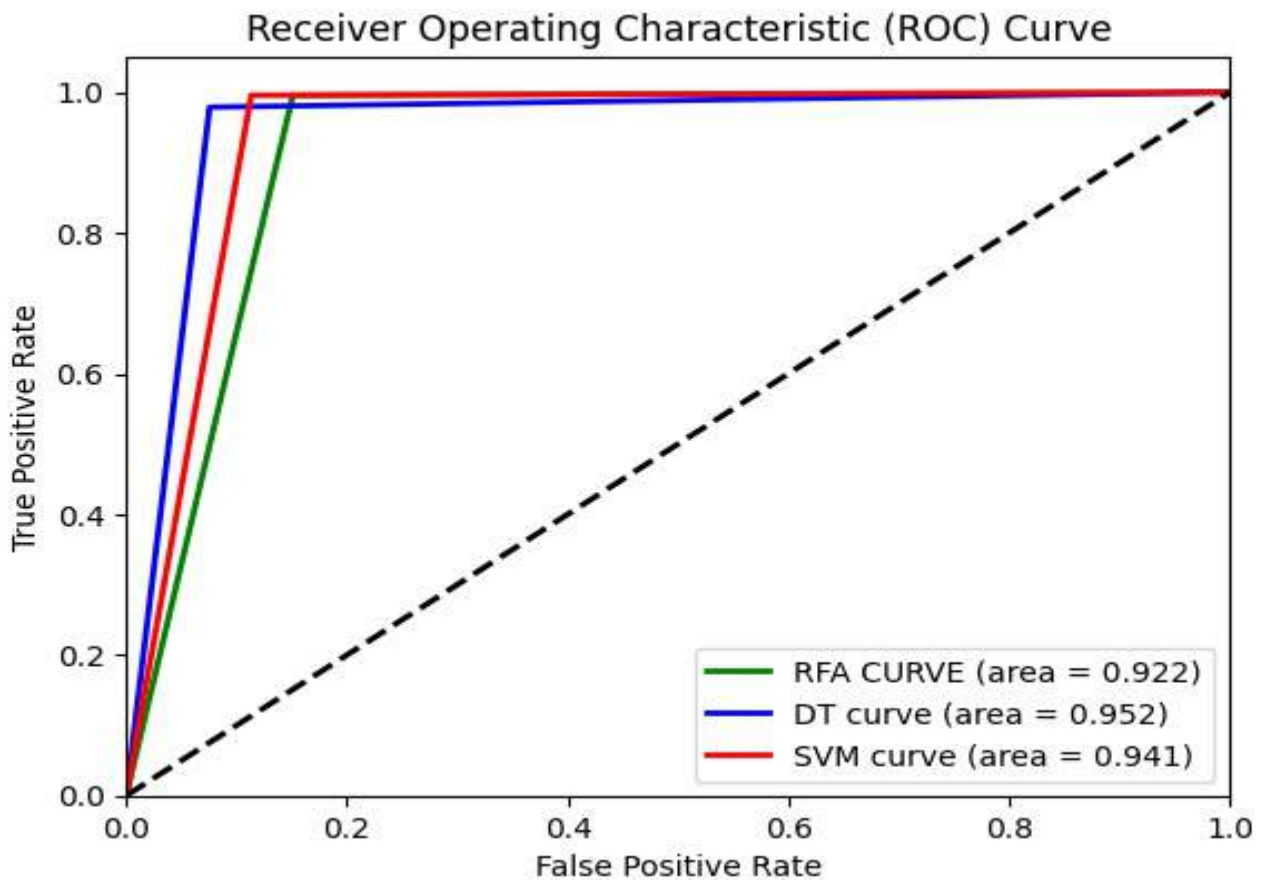


Figure 8.2: ROC Curve

Figure 8.2 shows the ROC curve compares the performance of three different classifiers: RFA, DT, and SVM. The green line represents the ROC curve for the RFA classifier, which has an AUC (Area Under the Curve) of 0.927, indicating excellent performance. The blue line represents the DT classifier with an AUC of 0.899, and the red line represents the SVM classifier with an AUC of 0.901. The dashed black line represents a random classifier, which serves as a baseline for comparison. The closer the ROC curve is to the top-left corner of the graph, the better the classifier's performance, as it indicates a higher true positive rate and lower false positive rate. Based on the AUC values, the RFA classifier outperforms the DT and SVM classifiers in this particular scenario. The ROC curve is a widely used tool in machine learning to evaluate the trade-offs between sensitivity (true positive rate) and specificity ($1 - \text{false positive rate}$) for binary classification problems.

8.2 Poster Presentation

ENHANCED DEEPLARNING TECHNIQUES TO DETECT TROJAN HORSE ATTACK IN CYBERSECURITY

Department of Computer Science and Engineering
School of Computing
1.156CS701-MAJOR PROJECT
INHOUSE
WINTER SEMESTER 2023-2024

Batch: (2020-2024)

INTRODUCTION

In recent years, the proliferation of Internet of Things (IoT) devices and applications has significantly contributed to the modern information society. The interconnectedness facilitated by IoT technologies has revolutionized various aspects of daily life, from smart homes to industrial automation. However, this rapid integration of IoT into our lives also brings forth significant security concerns. One of the foremost challenges facing the realization of the full potential of this industrial revolution is the threat posed by cybercriminals, these malicious exploit vulnerabilities in individual PCs and networks, aiming to steal confidential data for financial gain or disrupt systems through denial of service attacks. Additionally, we will explore the ethical, legal, and regulatory aspects surrounding cyberattacks and data breaches. This research not only serves as a valuable resource for cybersecurity professionals but also contributes to raising awareness among individuals and organizations about the importance of robust digital security. By shedding light on the ever-evolving world of cyber hacking breaches, we aim to empower stakeholders to fortify their defenses and safeguard their digital assets in an increasingly interconnected world.

RESULTS

The paragraph introduces a sophisticated hybrid model for zero-day malware detection that combines visualization techniques with deep learning architectures. This approach addresses the growing challenges posed by evolving malware threats by transforming malware data into visual representations for analysis. It utilizes advanced deep learning architectures like Convolutional Neural Networks (CNNs) for effective feature extraction and pattern recognition. The model integrates static analysis, dynamic analysis, and image processing within a unified framework, providing comprehensive coverage of diverse malware behaviors. Visualization techniques enhance interpretability and aid in feature extraction, contributing to robust and intelligent zero-day malware detection capable of identifying new variants in real-time. This model emphasizes scalability for proactive detection and mitigation of emerging threats, marking a significant advancement in cybersecurity.

STANDARDS AND POLICIES

Google Colab
Google Colab is a cloud-based Jupyter notebook environment provided by Google as part of the Google Cloud Platform ecosystem. It offers a convenient and accessible platform for running Python code, particularly for data science, machine learning, and deep learning tasks. Users can leverage Colab's integration with Google Drive for seamless file management and collaboration. One of its key features is the ability to access powerful hardware accelerators like GPUs and TPUs, enabling faster execution of complex computations and machine learning models in a scalable environment.
Standard Used: ISO/IEC 27001

Google Drive
Google Drive is a file storage and synchronization service developed by Google. Launched on April 24, 2012, Google Drive allows users to store files in the cloud (on Google's servers), synchronize files across devices, and share files. In addition to a web interface, Google Drive offers apps with offline capabilities for Windows and macOS computers, and Android and iOS smartphones and tablets. Google Drive encompasses Google Docs, Google Sheets, and Google Slides, which are a part of the Google Docs Editors office suite that permits collaborative editing of documents, spreadsheets, presentations, drawings, forms, and more.
Standard Used: ISO/IEC 27001

CONCLUSIONS

The proposed approach for predicting cyber hacking breaches leverages machine learning algorithms to proactively identify and forecast potential cyber threats. By integrating machine learning models, the system adapts and evolves in response to emerging cyber threats, enhancing its predictive capabilities. This novel approach advances the field of cybersecurity by establishing a proactive and dynamic paradigm for anticipating and mitigating cyber risks, strengthening digital ecosystems against malicious activities. The proposed system combines visualization techniques with deep learning architectures, transforming malware data into visual representations for analysis. It incorporates Convolutional Neural Networks (CNNs) for effective feature extraction and pattern recognition tasks. The hybrid approach integrates static analysis, dynamic analysis, and image processing, enabling comprehensive coverage of diverse malware behaviors. This model aims to achieve robust zero-day malware detection, capable of identifying previously unseen malware variants or behaviors in real-time deployments.

ACKNOWLEDGEMENT

Dr.S.Amutha, ME., Ph.D., / Associate Professor
Contact No - +91 9944163377
Mail ID - samutha@veltech.edu.in

TEAM MEMBER DETAILS

16803/Sagin Sundoz Fernando E
15072/K-Ajay Kumar Reddy
16923/G-Jayanth
9962321971
7995538412
93908 77629
vnu16803@veltech.edu.in
vnu15072@veltech.edu.in
vnu16923@veltech.edu.in

METHODOLOGIES

ABSTRACT

Deep learning has emerged as a powerful tool for enhancing malware detection and mitigation efforts, offering automated feature extraction capabilities and the potential to detect zero-day malware. However, cyberattacks are a major threat to these systems, if the system is unaware of the existence of these attacks, it won't be able to detect them, and performance may be disrupted or disabled altogether. By leveraging deep learning models like Convolutional Neural Networks (CNNs) and hybrid approaches, cybersecurity professionals can improve threat detection accuracy, response times, and adaptability to evolving malware landscapes. Addressing challenges such as representative training datasets, false positives, and model adaptiveness is essential for maximizing the effectiveness of deep learning in cybersecurity applications, highlighting the need for ongoing research and development in this field.

Chart : Classification of malware based on number of images

Figure 8.3: Poster Presentation

References

- [1] Asma A. Al-Hashmi, Fuad A. Ghaleb, A. Al-Marghilani, Abdulsamad E. Yahya, Shouki A. Ebad, Muhammad Saqib M.S, and Abdulbasit A. Darem. “Deep-Ensemble and Multifaceted Behavioral Malware Variant Detection Model,” Open access journal of Computer and Electronic Engineering, Vol. 10, , pp.42762-42777,2022.
- [2] Hanxun et al., “A worm detection system based on deep learning,” Open access journal of Department of Computer Science, vol. 8, pp. 205444, 2020.
- [3] Jeongwoo kim, Joon-young paik, and Eun-sun cho. “Attention-Based Cross-Modal CNN Using Non-Disassembled Files For Malware Classification,” Open access journal of Computer Science and Engineering, Vol. 11, , pp.22889-22903,2023.
- [4] Mohammed, Kok swee sim, Shing Chiang tan, and Chee peng lim. “An Ensemble-Based parallel Deep Learning Classifier With PSO-BP Optimization For Malware Detection,” Open access journal of Engineering and Technology, Vol. 11, , pp.76330-76346,2023.
- [5] Muawia A. Elsadig. “Detection of Denial-of-Service Attack in Wireless Sensor Networks: A Lightweight Machine Learning Approach,” Open access journal of scientific Research, Vol. 11, , pp.83537-83551,2023.
- [6] Mulhem ibrahim, Bayan issa, and Muhammed basheer jasser, “A method for automatic android malware detection based on static analysis and deep learning,” Open access Journal of Department of computer Engineering Software Engineering, vol. 10, , pp. 117334-117352, 2022.
- [7] N.Abdal gawad, A.sajun, Y.kaddoura, A.zualkernan. “Generative Deep Learning to Detect Cyberattacks For The IOT-23 Dataset,” Open access journal of Department of Computer Science and engineering, Vol 10, , pp.6430-6441,2022.
- [8] Nguyet quang do, Ali selamat, Ondrej krejcar, Enrique Herrera-viedma, and Hamido fujita. “Deep Learning for Phishing Detection: Taxonomy, Current Challenges and Future Directions,” Open access journal of Informatics and Management, Vol. 10, , pp.36429-36463,2022.

- [9] Nikolaos peppes, Theodoros alexakis, Emmanouil daskalakis, Konstantinos demestichas, and Evgenia adamopoulou. “Malware Image Generation and Detection Method Using Dcgans and Transsfer Learning,” Open access journal of Department of Agriculture Economics and rural development, vol 11, , pp. 105872-105884, 2023.
- [10] Omer aslan, and Abdullah asim Yilmaz, “ A new malware classification framework based on deep learning algorithms,” Open access journal of Computer Engineering Department, vol. 9, , pp. 87936-87951, 2021.
- [11] S.Abiajh roseline, S.geetha, Seifedine kadry, and Yunyoung nam. “Intelligent Vision-Based Malware Detection and Classification Using Deep Random Forest Paradigm,” Open acess journal School of Computer Science and Engineering, Vol 8, ,pp.206303-206324,2020.
- [12] Sandeep gupta, Carsten maple, and Roberto passerone. “ An Investigation Of Cyber-Attacks And Security Mechanisms For Connected And Autonomous Vehicles,” Open acesss journal of Information Engineering and Computer Science, Vol. 11, , pp.90641-90669,2023.
- [13] Xiaofei xing, Xiang jin, Haroon elahi, Hai jiang, and Guojun wang, “A malware detection approach using autoencoder in deep learning,” Open access journal Computer Science and Information Technologies, vol. 10, , pp.25696-25706, 2022.
- [14] Young-seob jeong, Sang-min lee, Jong-hyun kim, Jjiyoung woo, and Ah reum kang. “Malware Detection Using Byte Streams Of Different File Formats,” Open acess journal of Department of Computer Engineering, vol 10, , pp.51041-51047, 2022.
- [15] Youngghoon ban, Sunjun lee, Dokyung song, Haehyun cho, and Jeong Hyun yi. “Fam: Featuring Android Maleware For Deep Learning-Based Familial Analysis,” Open acess journal of School of Software Convergence, Vol 10, , pp.20008-20018,2022.