IXNet: Interpretable and Explainable Deep Attentive Network-based Model for Android Malware Detection

Khalid Mohammed Abdi

A Thesis Submitted to

The Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September 2022

Adama, Ethiopia

IXNet: Interpretable and Explainable Deep Attentive Network-based Model for Android Malware Detection

Khalid Mohammed Abdi

Advisor: Akey Sungheetha(Ph.D)

A Thesis Submitted to

The Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September 2022

Adama, Ethiopia

Declaration

I hereby declare that this Master Thesis entitled "**IXNet: Interpretable and Explainable Deep Attentive Network-based Model for Android Malware Detection"** is my original work. That is, it has not been submitted for the award of any academic degree, diploma, or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation.

Khalid Mohammed	_____	_____
Name	Signature	Date

Recommendation

I, the advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled "**IXNet: Interpretable and Explainable Deep Attentive Network-based Model for Android Malware Detection**" prepared under my guidance by **Khalid Mohammed** submitted in partial fulfillment of the requirements for the degree of Masters of Science in Computer Science and Engineering. Therefore, I recommend the submission of a revised version of the thesis to the department following the applicable procedures.

Akey Sungheetha (Ph.D) _____ _____

Major Advisor Signature Date

Approval Page

I, the advisor of the thesis entitled "**IXNet: Interpretable and Explainable Deep Attentive Network-based Model for Android Malware Detection"** and developed by **Khalid Mohammed**, hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Akey Sungheetha (Ph.D) _____ _____

Major Supervisor Signature Date

We, the undersigned, members of the Board of Examiners of the thesis by **Khalid Mohammed** have read and evaluated the thesis entitled "**IXNet: Interpretable and Explainable Deep Attentive Network-based Model for Android Malware Detection"** and examined the candidate during the open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master of Science in Computer Science Engineering

_____ _____ _____

Chairperson Signature Date

_____ _____ _____

Internal Examiner Signature Date

_____ _____ _____

External Examiner Signature Date

Finally, approval and acceptance of the thesis are contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____ _____ _____

Department Head Signature Date

_____ _____ _____

School Dean Signature Date

_____ _____ _____

Office of Postgraduate Studies, Dean Signature Date

ACKNOWLEDGEMENT

Alhamdulillah! First and foremost, I would like to thank Allah, the Almighty, for His mercy and blessing. Despite my best efforts, this wouldn't have been fruitful without the continuous help and kind support of many individuals.

I would like to sincerely thank my advisor Dr. Akey Sungheetha, for all of her help and insightful suggestions during the course of the study.

I owe Dr. Mesfin Abebe the utmost respect and genuine gratitude for his guidance and invaluable assistance.

My biggest respect and appreciation go to my colleagues at Information Network Security Administration (INSA) and ASTU-CSE AI SIG members for their incomparable support and helpful recommendations.

Finally, I would like to offer my deepest and warmest gratitude to my families and friends, whom I am deeply indebted to, for their unlimited support and encouragement throughout my academic endeavors. As always, they have been my never-ending source of inspiration all along.

Last but not the least, I would like to thank myself for believing in me, and for having the courage to chase my dreams.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF CODE SNIPPETS

LIST OF EQUATIONS

LIST OF ACRONYMS

Acronyms	Definition
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
CNN	Convolutional Neural Network
DBN	Deep Belief Network
DL	Deep Learning
DPA	Data Protection Act
FN	False Negatives
FP	False Positives
FPR	False Positives Rate
GDPR	General Data Protection Regulation
IML	Interpretable Machine Learning
LBRP	Layer-wise Backward Relevance Propagation
LIME	Local Interpretable Model-Agnostic Explanations
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
OS	Operating System
RNN	Recurrent Neural Network
ReLu	Rectified Linear Unit
SHAP	SHapley Additive exPlanations
TN	True Negatives
TP	True Positives
TPR	True Positives Rate
XAI	Explainable Artificial Intelligence
XGBoost	eXtreme Gradient Boosting

ABSTRACT

The diversity, sophistication, and availability of malware, specifically Android malware, pose an enormous threat to the security of mobile devices. A number of deep learning approaches were proposed to mitigate this issue since deep learning models are suitable for large and complex data. These solutions, however, often put more emphasis on the accuracy of detection rather than detection interpretability. This reckons that there are no adequate studies concerned with understanding the models' learning process and logic of inference to the predictions. Thus, there is a need to develop a transparent Android malware detection tool without undermining accuracy. This study aims to address this issue by developing an interpretable and explainable deep attentive network, IXNet, which consists of parallel components for detection and interpretation. The interpreter component performs instance-wise feature selection providing a focal point for more tuned decisions, and the detector component uses the selected features to make the hidden feature representation and final detection. Generally, the proposed network traces the learning process and decision path made by the detector to provide an interpretation of the decision made. The learned concepts are extracted using two mechanisms; Multi-head Self-attention and Layer-wise Backward Relevance Propagation. The attention mechanism traces the internal working principle during the training, and the decision-making process of the model during the testing phase, whereas the relevance propagation attends to the learning progress of the model during the training session in real-time. Several experiments were done using two datasets; Drebin and MalMem, where the proposed network-based model performed splendidly on both datasets. The experiments conducted show that the model achieves an impressive accuracy of 98.89% with a 1.56% false positive rate on Drebin and an accuracy of 99.98% with a 0.05% false positive rate on MalMem, outperforming its counterpart models, XGBoost, Xmal, and TabNet, while maintaining an excellent level interpretation. Regarding transparency, the model provides the relevant information needed to explain its learning and decision-making process. The subjective evaluation also show transparency promotes the acceptance of AI-based models.

***Keywords**: Android Malware Detection, Interpretable Deep Learning, Explainable Deep Learning, Explainable Artificial Intelligence, XAI, Instance-wise Feature Selection, Layer-wise Backward Relevance Propagation, Interpretable Deep Attentive Network*

CHAPTER ONE

INTRODUCTION

1.1. Background of the Study

Since the dawn of time, collection, preservation, and transfer of information have been the cornerstone of the civilization we realize today. Communication of information between people of the same or different eras has had its inconvenience. However, over time, smooth and seamless communication media with no dimensional barriers emerged. Thus, the genuineness of this communication concerns all parties involved. In today's day and age, the internet has become the mainstream information exchanging platform for all kinds of people, data, and media all over the world. The capabilities provided in this platform are available for all users regardless of their intentions. The need to secure networks and devices has intensified due to the rise of attack vectors such as the Internet of Things (IoT) and personalized mobile devices, such as smartphones, tablets, personalized digital assistants, and wearable devices.

Mobile devices are becoming convenient ways of communication and information sharing due to their usability. Nowadays, services, from paying bills to managing big companies' financial transactions, are one click away with these devices; thus, making them essential in our day-to-day life. Users keep a wealth of sensitive and private information on these gadgets, which are often connected to the outside world through the internet. Most mobile devices run on Android, the most popular OS for portable devices (dominating 70% of the global mobile OS market since 2016)[1]. The number of applications developed for this platform is also increasing in staggering quantities. These circumstances with mobile devices becoming ubiquitous created a big realm for attackers to target them. This begs for the protection of these devices from all possible forms of attack.

Coupled with the shortage of cyberthreat expert personnel, the very dynamic and complex nature of the cybersecurity landscape makes it infeasible to anticipate, prepare and protect information and network infrastructures from every possible attack (either known or Zero-day

[1] https://gs.statcounter.com/os-market-share/mobile/worldwide

attacks). In the cyber realm, malware is the most common tool used in cyberattacks making it a vicious form of cyberthreat (Carrier et al., 2022). The threat of malware in one place in the world is not to be ignored by the rest. Sooner than later, it will come knocking on our doorstep. With the increase of new malwares and their variants every second[2], the time and resources needed to analyze every single sample and its variants, and update the malware knowledge base regarding imminent attacks and attack actors is impractical. This problem can be addressed with the use of automated systems to develop an ultimate weapon against malware. To help this, many different machine learning approaches had been implemented and performed very well. But, the increase in the volume of data necessitates the need for more efficient models, this is where Deep Learning comes in.

Recently, the success of AI in a range of sectors and applications has led to its popularity for complex and big data-based problems. With its ability to learn high-dimensional concepts, Deep Learning is the front-runner in this domain of problems. Some of the approaches used in Natural Language Processing and Computer Vision are being applied in other areas to duplicate their success (Niu et al., 2021). However, due to its black-box nature, deep learning is underutilized in high-stakes areas like healthcare, banking, criminal/justice, and cybersecurity (Saraf et al., 2020; Khoulimi et al., 2022; Ribeiro et al., 2016). Domain experts in these industries require an abstract representation of findings, decision processes, and the overall internal workings of a model. They require some form of testimony in order to implement AI-based solutions in real-world scenarios (Mathews, 2019; Ahuja, 2021). This suspicion stems from the very nature of deep learning, in which the internal working principle and decision-making process are hidden from end-users, leading to the cynical view of domain experts (Fan et al., 2021). Thus, this leads to the need for a mechanism to describe the internal working principle, learning process, and decision-making step/logic as well as contextualization of results (**globally** and **locally**).

Furthermore, as most of the source feeds used as data sources originate from western countries, automated systems that use these sources must act in accordance with regulatory requirements set by these entities such as GDPR[3], the European General Data Protection Regulation policy adopted in 2018 (Voigt and Von dem Bussche, 2017). One of the requirements of GDPR is the

[2] https://www.av-test.org/en/statistics/malware/
[3] https://gdpr-info.eu/

need for transparency in automated systems that use data from their assets. The "right of explanation" as stated by the GDPR requires the development of models that are open about their internal working principle and decision-making logic. To help this issue, it is comparative to design transparent deep learning models which can determine the key factors influencing the process of each decision (Melis et al., 2018).

This work proposes an approach to integrate transparency into deep learning models for a better understanding of results and comprehension of the learning process and internal working phenomena of deep learning models. This helps in justifying results and creating a knowledge base and its abstractions for human comprehension. The proposed solution can also pave the way for the ultimate Android malware detection and analysis method by providing a means to select the most salient features to be used as a criterion to identify malicious applications from benign, regardless of the techniques used. The output can also be used by security analysts and malware threat hunters to analyze the reason behind the detection to be effective in identifying the trends used by malware developers for future threat prediction and prevention.

1.2. Motivation of the Study

With recent developments in the internal and external political instabilities in Ethiopia, the threat of cyberwarfare is eminent. In this half-fiscal year alone, 3400 attacks have been attempted, which is three times larger than the number of attacks in the same period of the last year[4]. The recent increase in the number of attempted attacks on our country's information infrastructures calls for effective means of cyber defense and threat intelligence analysis. Information Network Security Administration (INSA), the largest cybersecurity institution in our country, is working on integrating AI to automate existing systems. In general, for the purpose of efficiency and effectiveness, these automated systems need a central knowledge base with essential information derived using both dynamic and static analysis from malwares and their variants.

With the risks involved, the major challenge in this process is ensuring the fidelity of the automated intelligent systems and providing reasonable testimonials for gaining trust in them. This is why most current defense-in-depth security layers in our country and around the world

[4] https://www.insa.gov.et/web/guest/w/news-31

are made up of manual and signature-based security mechanisms as they are easier to understand. Therefore, there is a need that can be satisfied with the use of eXplainable AI (XAI). It helps in justifying results and creating a knowledge base in human comprehensible abstractions.

Some of the motivations of this study are:

- Achievement of deep learning in many fields such as CV and NLP
- Need for a standard way to interpret and explain Deep Learning models
- Shortage of qualified professionals in the field of cybersecurity
- The increased feed of android malware available for the research community
- The necessity of building a concrete knowledge base about malware

1.3. Statement of the problem

In the domain of cybersecurity, the basis for the cyber-defense of a country is a knowledge base for managing cyberthreats known to the cybersecurity landscape worldwide. The notion of populating a knowledge base manually is exhaustive and impractical. Thus, a more efficient way can be achieved with the implementation of automated systems which can extract the underlying similarities by detecting trends and techniques used by attackers. This can then be used to effectively detect, identify and attribute existing and new malwares, and their variants. Success in the detection is not the only primary objective in malware detection and analysis; presenting an interpretable and explainable model is also comparably important. Ideally, security analysts should be presented with the detection and classification results as well as how and why those decisions were made. An inherent way of discovering quiescent glitches in data during the training phase must also be incorporated into these models.

Nevertheless, the models proposed in most recent publications are neither explainable nor interpretable, making them a black box for analysts seeking to comprehend the model's decision path for accurate detection and classification. Meanwhile, some recent research works have attempted to mitigate this issue. Earlier efforts attempted to explain the detection results of deep learning models using various surrogate models, while others proposed an explainable model that can express its decision-making process often after the training phase. There is a void

created by prior approaches to deliver comprehensible deep learning models. These works have not looked at the idea of integrating the notions of Interpretability and Explainability in order for more sensible informativeness. Rather, they tend to mix up the two concepts and end up tangling. They also have not taken the advantage of exploiting the latent knowledge that is manifested throughout the model's learning phase.

The solution proposed here is the use of a multi-head self-attention and layer-wise backward relevance propagation to make the model inherently interpretable and explainable. The learning process of the model is recorded to provide a valid debugging tool. The decision path made by the model is traced to provide a better interpretation of the decision made. The proposed model also incorporates attention-based prominent feature selection and supervised deep learning for accurate detection and classification using obfuscated and hybrid features.

1.4. Research Questions

- This study was conducted to address the following research questions.
  - **RQ1:** How to develop a transparent android malware detection model without compromising detection accuracy?
  - **RQ2:** What are the effects of attention mechanisms-based feature selection and interpretation on the overall performance and transparency of a deep learning model?
  - **RQ3:** What is the effect of transfer learning employed in ensemble-like deep learning networks?

1.5. Objective of the study

1.5.1. General Objective

- ❖ The general objective of this study is to develop an interpretable and explainable deep attentive network-based model for accurate android malware detection.

1.5.2. Specific Objectives

- ❖ To realize the general objective, the following specific objectives are needed to be met.
  - ✓ To collect android malware datasets, that used different analysis techniques, from trusted public online repositories and transform them properly.
  - ✓ To design a sequential deep attentive network and develop an accurate Android malware detection model using the designed network.
  - ✓ To evaluate the proposed model using different standard metrics.
  - ✓ To implement a multi-head self-attention mechanism for inherent interpretability and explainability.
  - ✓ To implement layer-wise backward relevance propagation for surveilling the learning process of the proposed model.
  - ✓ To evaluate the interpretability of the proposed model using a controlled environment.
  - ✓ To compare the interpretability and explainability of the proposed model with other analogous models.

1.6. Scope and Limitation of the Study

1.6.1. Scope of the Study

This study attempts to address the challenge of improving the transparency and human comprehensibility of Deep Learning models. It primarily focuses on the interpretability and explainability of deep learning models for Android malware detection in the realm of cybersecurity. The proposed model is tested only for binary classification (detection). The proposed model is trained and tested using features extracted from the static and dynamic analysis of obfuscated and non-obfuscated applications.

1.6.2. Limitation of the Study

The study does not encompass other types of malwares other than Android malware. Android malware classification is not covered as well. The proposed model is not tested for multilabel or multiclass classifications. As the intended purpose of this study is for threat intelligence analysis for malware knowledge base construction, the primary focus is effectiveness and not efficiency. Moreover, the possibility of adversarial or any other evasion attacks and their effects on the model is not explored. An analysis of the degree of fidelity of the interpretations to the proposed model is not included in this study.

1.7. Significance of the Study

An accurate malware detection system is without a doubt an ultimate tool in the fight against android malware, and malware in general. What will be more effective is the presence of interpretable tools that can describe the correlation of different malwares based on their features and behaviors. In addition to giving a local and global understanding of how a deep learning model makes decisions, the proposed approach can assist security analysts in understanding potential vulnerabilities of automated systems to well-crafted evasion attacks and adversarial.

When the proposed study is finalized, it will provide:

- ✓ A Step forward in the search for the ultimate Android malware detection tool.
- ✓ An approach to extract the most salient features to be used as a criterion for detection.
- ✓ An answer to not only the “which”, but also the “how” and “why” of a detection.
- ✓ A way of predicting the trend of malwares and malware-oriented attacks.
- ✓ Ideally**,** this study can be extended to other malwares as well.

1.8. Beneficiaries of the Study

With the proper optimization and tuning, when the proposed solution is implemented, it will have numerous benefits for the professionals listed below.

- ***Security and Risk Analysts:*** analyze malwares and their variants to find solutions and to describe an event in a more sensible way.
- ***Malware Hunters:*** inspect patterns and functionalities of malware of a large number of Android applications at once.
- ***CTI Researchers***: monitor the trend used by malware developers and predict potential future scenarios with more accuracy.

1.9. Application of Expected Output

As the proposed model describes its results in a more sensible way, it can be applied in domains that need human-comprehensible deep learning applications. The output of the proposed thesis work can be applied in:

- Cybersecurity Institutions
  - Cyber Threat Intelligence (CTI) Analysis
  - Cyber Threat Hunting
  - Cyber Security Risk Analysis
- Cyber Security Enterprises and Vendors
- Cyber threat sharing platforms
- Other high-stake sectors with similar Scenarios

1.10. Organization of the Thesis

This thesis is composed of six chapters in total. The first Chapter mainly discussed the introduction, background, statement of the problem, and research questions and objectives of the study. Furthermore, it discusses the study's scope and limitations, significance, beneficiaries, and application. The remaining thesis is organized as follows:

- Chapter two walks through the background literature, related works, and a summary of related works. It also identifies the gaps in the related works to be filled by this work.
- Chapter three reviews the research methodologies such as research approach, dataset preparation techniques, required development tools, and evaluation metrics used.
- Chapter four explains the architectural design, modeling algorithms, problem formulation, and working principle of the proposed model.
- Chapter five discusses the experimentation environment setup, the proposed model implementation, the hyperparameters used, and a preview of the experimentation classes.
- Chapter six reports the result of the experiment and its comparisons with the result of previous works. It also analyzes the meaning of the findings and how the research questions have been answered.
- Finally, the last section concludes the research work and indicates future research directions, and offers recommendations.

CHAPTER TWO

LITERATURE REVIEW

2.1. Overview

This chapter presents the concepts of the fundamental elements and the current knowledge body of science regarding the topics discussed in the study. It has three sections; Introduction, Background literature, and Related works.

2.2. Background Literature

2.2.1. Cybersecurity

From the first malware attack to the first attempt at cyberwar, the digital world has shown to be very susceptible and prone to causing a global-scale catastrophe. Cybersecurity is one of the high-stakes domains where indifference and mistakes can have devastating consequences, from evading the privacy of individuals, organizations, and countries to crumbling the global economy, emanating abomination-level disasters, and the rise of worldwide wars. As evidenced by their impact level, cyberattacks are increasing in number and sophistication[5]. In comparison to 50% in 2020, 79% of detected attacks had a severe impact, 32% were "critical," and 47% were "high". The sophistication and networked cybercrime demonstrated by new attacks indicate the potential danger in terms of the risks involved, as well as the need for serious measures that must be implemented.

Cybersecurity is a discipline concerned with protecting information, systems, network infrastructures, and technologies of the digital world from digital attacks[6]. These attacks often target extortion of money, access of sensitive private information, or disruption of routine business operations (Khoulimi et al., 2022). The shifting nature of security vulnerabilities is one of the most difficult aspects of cybersecurity (Khoulimi et al., 2022). The cost of alert fatigue due to inefficient alert prioritization has also exacerbated the challenge of cybersecurity in organizations (Ho et al., 2012), with one in five alerts being false alarms[7]. Furthermore,

[5] https://www.infosecurity-magazine.com/news/breach-volumes-2021-exceed-2020/
[6] https://www.cisco.com/c/en/us/products/security/what-is-cybersecurity.html
[7] https://www.securitymagazine.com/articles/97260-one-fifth-of-cybersecurity-alerts-are-false-positives

cybersecurity risk oversight is difficult due to new regulations that result in incurring hefty fines for organizations that encounter cybersecurity breaches. They are required by privacy laws like the Data Protection Act (DPA)[8] of 2018 and the General Data Protection Regulation (GDPR)[9] to implement appropriate security measures.

Although having updated intelligence about cyberthreats is nearly impractical, it is vital to safeguard information and other assets against cyberattacks, which can take various forms. Malware, phishing, advanced persistent threats (APTs), and zero-day vulnerabilities are examples of cyberthreats. Of the cybersecurity dangers outlined, Malware is the only one that may be used as a vector for other threat attacks (Mathews, 2019).

2.2.2. Android Malware

In today's connected world, given the vast amounts of private, sensitive, and credential information mobile devices are entrusted with, they have developed into an enticing target for malware developers (Razgallah et al., 2021). Android malware is malicious software designed to infiltrate systems used by mobile devices, that run on a Linux-based operating system known as Android, causing leakage of confidential information or abnormal behavior. Currently, with the vast number of malicious apps on the internet, it is regarded as one of the most serious security menaces (Liu et al., 2022). Malware Cybersecurity faces enormous challenges as a result of malware's diversity, sophistication, and accessibility. (Gibert et al., 2020).

The use of antivirus has been the first line of defense for more than a decade. But these antiviruses mainly use signature-based methods (Stamp, 2021). Signature-based detection matches the signature of an application to a signature database of known malwares. This approach can only recognize known and prevalent malwares, making it prone to variants, unknown, and zero-day malwares. To mitigate these limitations and address modern risks effectively, other alternative approaches such as heuristic-based and behavioral-based (anomaly) detections had been developed (Tian et al., 2020). By using these methods, some engines are able to detect malwares without needing a signature. But, as the engines are looking for malicious code or behavior, they are prone to evasion techniques (Carrier et al., 2022).

[8] https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted
[9] https://gdpr-info.eu/

Evasion techniques such as Obfuscation, Environmental Awareness and Time-based evasion, remain the main obstacles to malware detection (Carrier et al., 2022). **Obfuscation**, originally developed to protect intellectual property, is one of the most popular malware detection evasion techniques that make programs obscure to analyze without altering their core functionality (Stuart, 2020). Given the recent sophistication of obfuscation techniques used by malware variants, malware detection requires special attention now more than ever (Marais et al., 2021).

Over the past few years, malware detection has been a focus of active cybersecurity research (Marais et al., 2021). There is a need to effectively detect obfuscated malware samples using approaches through hybrid analysis features (Stamp, 2021). It is anticipated that features from the static and dynamic analysis will offer a model harmonious information that will help it have a more complete understanding of a sample. (Islam et al., 2013; Pei, Yu, Tian, et al., 2020) have shown that more robust and effective classifiers can be developed using features extracted from both analyses. Furthermore, earlier works have also found that malwares have a hard time evading detection when analyzed from multiple feature scopes (Islam et al., 2013). For instance, (Yuxin, 2019; Ahmadi, 2016) used both approaches to detect malwares. The process of building a knowledge base from raw features extracted with these analysis techniques results in the need for expertise and exhaustive effort. These circumstances eventually call for the realization of automation in Android malware detection (Khoulimi et al., 2022).

2.2.3. Automation in Android Malware Detection

The manual feature selection is not an intuitive decision to make and should be based on the extracted features from the applications (Yuxin & Siyi, 2017). There is no guarantee that manually selected features can indicate the most destructive malware (Mahdavifar et al., 2020). The assumption that a specific class of features can always be used to get the most accurate results makes models more susceptible to adversarial attacks (Stuart, 2020). Although researchers have been trying to explore ways for identifying prominent features and effective means for feature selection, they often focus on a specific malware family, type, or feature only, which makes these solutions ungeneralizable.

By using a cutting-edge dominant feature selection algorithm, (Gera, 2021) suggested a hybrid approach to feature extraction to detect Android ransomware. In comparison to the existing static, dynamic, and hybrid approaches, the proposed hybrid solution outperformed them all

with a 99.85 percent accuracy and zero false positives rate while taking 60 salient features into account. Additionally, it demonstrates how the combination of all dynamic features aids in more accurate malware identification

Current antivirus vendors' signature-based and heuristic-based detection techniques necessitate constant analysis and update on new malware threats, which is a laborious, error-prone, and time-consuming task (Wen & Yu, 2017). Therefore, AI-based automated Android malware detection tools are required to detect known and zero-day malware threats (Liu et al., 2022). The presence of high-volume data streams has led the automation endeavor to employ artificial intelligence and machine learning techniques (Mathews, 2019).

2.3. Artificial Intelligence (AI)

Artificial Intelligence has come a long way since its modest genesis in 1943, now becoming an omnipresent and pervasive trend. Thanks to recent developments in machine learning and deep learning techniques, it has finally established itself as a fundamentally transformative technology (Stamp, 2021).

2.3.1. Machine Learning (ML)

Machine Learning is a data-driven subfield of Artificial Intelligence area devoted to developing intelligent machines by processing a large amount of data (prior examples) to discover pattern abstraction that exists within (Stamp, 2021). However small, any machine learning technique will need the help of a subject expert to identify the features that are necessary to lessen data complexity and make patterns obvious to learning algorithms. (Gera et al., 2021).

2.3.2. Deep Learning (DL)

In a nutshell, Deep learning is a special type of machine learning that focuses on self-improvement by analyzing data to form abstract intermediate representations using Artificial Neural Networks (ANNs), which are abstractions of the human brain and its processing (Stamp, 2021). Deep learning has a multi-layer architecture made up of layers of neurons that can be excited or inhibited using activation functions. The signals between neurons are regulated by the learnable weight and bias parameters. The architecture facilitates the learning process

through layer-wise pretraining and back-propagation to tune the weights and bias of neurons and minimize error using a loss function (LeCun et al., 2015).

In deep learning, the necessity of domain-specific knowledge and exhaustive feature engineering is highly diminished due to the progressive nature of learning high-level representations of features (Mahdavifar et al., 2020). Additionally, deep learning will continue to produce better results as the dataset size increases, whereas models produced by other shallow machine learning techniques will relatively reach saturation point more swiftly, meaning that adding more data won't produce better models. The success of deep learning applications in the fields of Natural Language Processing, Robotics, and Computer Vision makes it the primary choice in complex and high-stake domains from shallow machine learning techniques.

2.3.3. Deep Learning in Android Malware Detection

In domains with complex and enormous data such as malware threat detection, deep learning-based solutions are primary choices (Stamp, 2021). The evolving nature of malware will render traditional machine learning models, such as those developed by Wen and Yu (2017), ineffective for malware detection issues (Gibert et al., 2020). This serves as the motivation for creating a deep learning-based malware detection architecture (Liu et al., 2021; LeCun et al., 2015).

In most recent works, deep learning algorithms have been used to address the malware detection issue (Gibert et al., 2019; Elayan & Mustafa, 2021; Bourebaa & Benmohammed, 2020). Some tried using hybrid DL and an ensemble of ML and DL models (Lu et al., 2020). These works demonstrated their effectiveness in detecting malwares and their variants (Wang et al., 2018). In (Zhao, 2019), the authors proposed a DAE-CNN hybrid model to boost the performance and accuracy of SVM by 5%. Other notable works also include the detection of obfuscated malware using deep learning algorithms. The approach proposed by (Pei et al., 2019), showed that the model was able to score a 99% F1-Score for the obfuscated and non-obfuscated samples. The detection model presented by (Lu et al., 2020) consists of hybrid deep learning models and has been shown to perform better compared to shallow ML models. It is claimed to detect Android malware obfuscation technology with more than 96% accuracy. They suggested the number of samples and the representativeness of the obtained malware features is necessary.

2.3.4. Challenge of Deep Learning Models in High-stake Sectors

High-stake sectors usually encompass areas in which a great deal of risk or serious consequences are involved, such as healthcare, criminal justice, finance, and cybersecurity (Saraf et al., 2020). The main issue related to deploying deep learning models in high-stakes sectors is the unavailability of techniques that explain the predictions of the models (Mathews, 2019). In healthcare, for instance, because they affect patients' health, doctors need to validate and justify the reasoning behind deep learning models' predictions and the contributions of various factors that led to them (Choi et al., 2016). In clinical informatics also (Shickel et al., 2017) claim linear models predominate in applied clinical informatics as deep learning models are difficult to interpret. In cybersecurity, with an increasing number of threats and malware sophistications, the threat landscape presents a setback for traditional detection methods, promoting the use of deep learning models (Khoulimi et al., 2022).

Unexpected issues are being produced by the black-box decision-making models currently in use (Saraf et al., 2020). This may be directly linked to mistakes inherited from some sort of human bias found in the dataset used to train the models (Parnas, 2017). Past incidents, like Google[10] and COMPAS[17], show the necessity of interpreting these models. In 2020, Google Vision Cloud, Google's CV service, has been shown to have a racial bias towards dark-skinned people. Studies also show that COMPAS is susceptible to race and gender biases[11]. These studies stress the need for validating AI models before deploying them into real world, especially in high stake sectors. Moreover, some responsible ML legislation, such as GDPR, has been proposed recently to be fulfilled by technologies that use or have used European Source at some point. The GDPR introduces, for the first time, a right of explanation of the underlying rationale with AI-based automated systems (Goddard, 2017). In order to make sure these regulations are met, means of explaining black-box models is necessary (Melis et al., 2018).

After all, how can users be confident it if the learning process and rationale behind decisions cannot be properly verified by subject-matter experts? How can bias be identified and mistakes be corrected if the models are not understood well? For these reasons, a trending research subject known as Explainable Artificial Intelligence (XAI) was born (Saraf et al., 2020).

[10] https://algorithmwatch.org/en/google-vision-racism/
[11] https://www.nytimes.com/2017/10/26/opinion/algorithm-compas-sentencing-bias.html

2.4. Explainable Artificial Intelligence (XAI)

ML-based approaches typically have the drawback of being black-box (Kinkead et al., 2021; Iadarola et al., 2021). XAI refers to techniques that allow human experts to understand the results of AI applications (Gunning, 2017). As stated by the pioneers[12] themselves, "Explainable Artificial Intelligence (XAI) is a program that aims to create a suite of ML techniques that produce more human-understandable models while maintaining a high level of accuracy". (Miller, 2019) also defined interpretation as "the degree to which a human can understand the cause of a decision". If a model's decisions are simpler for a human to understand than its counterpart's predictions, the model is more interpretable. (Gilpin et al., 2018). This enables users to understand, trust, and manage AI applications properly (Mathews, 2019).

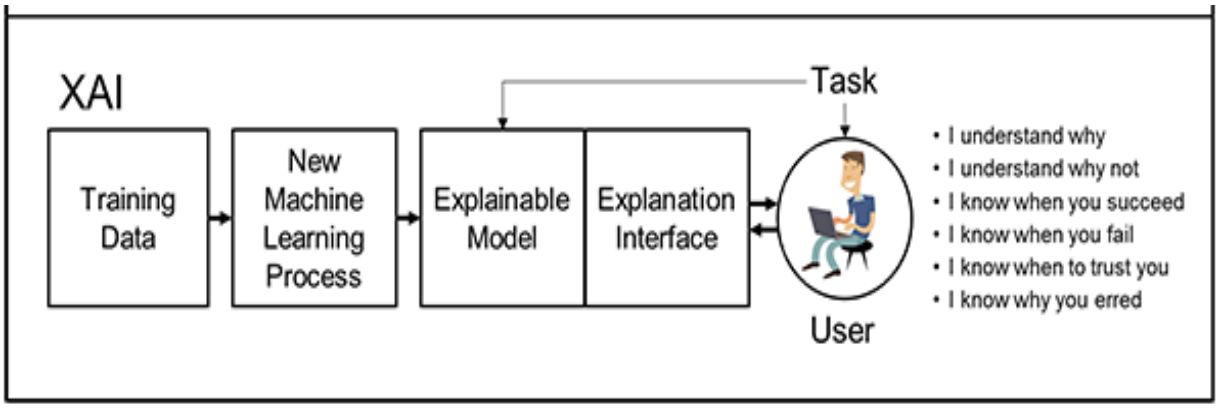


Figure 1. 1: XAI Concept Adopted from DARPA official website[13]

Artificial Intelligence transparency is a relatively new and unsolved problem in AI (Vilone & Longo, 2020; Mathews, 2019; Melis et al., 2018). It has received more attention lately as a result of the increased use of AI-based models in real the world. Transparency of a decision can be quantified at; ***Local*** (Instance level), explanation for each decision, and ***Global*** (Batch level), aggregated explanation for the whole sample batch. There are two types of approaches used in XAI, Extrinsic and Intrinsic[14].

[12] https://www.darpa.mil/program/explainable-artificial-intelligence
[13] https://www.darpa.mil/program/explainable-artificial-intelligence
[14] https://christophm.github.io/interpretable-ml-book/other-interpretable.html

2.4.1. Extrinsic (Post-hoc) XAI

Extrinsic or Post-hoc methods rely on an analysis done after the algorithm finishes its training and execution enhancing the considered model from a black-box to gray-box. One prominent category in this approach is Model-agnostic. These methods often use global surrogate models that are trained using the predicted labels from a black-box model, thus explaining how it makes its prediction (Ribeiro et al., 2016; Vilone & Longo, 2020). Here, notable examples are Local Interpretable Model-agnostic Explanations (LIME) and SHapley Additive exPlanation (SHAP). They are considered to be state-of-the-art model-agnostic XAI frameworks by many works of literature[15].

Although such model-agnostic methods are intuitive, the downside is the need for another model that always performs less than the black-box model in question. This results in less confidence in the credibility of the explanations (Rudin, 2019). Furthermore, rather than drawing conclusions about the data, the surrogate model does so regarding the black box model. The computation complexity needed also magnifies the need for intrinsic XAI models.

2.4.2. Intrinsic (Inherent) XAI

Some machine learning models are inherently interpretable, for instance, it is simple to extract decision rules from a decision tree. Unfortunately, most machine learning models, including random forests, gradient-boosted machines, and neural networks, cannot be directly interpreted. Intrinsic methods describe internal working principles of themselves inherently, providing some insights into what went into the decision-making process (Gilpin et al., 2018). The assumptions made and the concepts learned by models can be properly investigated thus ensuring their genuineness (Melis et al., 2018). Identification of noises and irrelevant features with improper importance can be used to further guide the data processing used in the training, thus improving the model (Saraf et al., 2020). Class-Activation Mapping (CAM) and Gradient-Weighted CAM are two popular intrinsic interpretable methods (Rahali et al., 2020).

[15] https://fr.coursera.org/lecture/cdss3/attention-and-explainability-prwMO

2.4.3. Interpretability Verses Explainability in XAI

In the theme of machine learning, interpretability and explainability have arisen as a response to the need for techniques to identify and take corrective measures against biases and mistakes in the concepts learned by models, as well as to be able to present results and decision steps in a human-understandable way. The two terminologies are usually used interchangeably in many recent works which are inaccurate and misleading (Rudin, 2019).

Explainability: tries to summarize the causes of decisions by describing the features used to make the decision and how much contribution they have to the decision. These may give some understanding of the reasons without explicitly stating how these contributions are mapped to the exact decision, thus making it impossible to recreate the decision-making process. Moreover, the process may be affected by biases or wrong data points. Mostly, explanations are made by other surrogate models outside of the model thus making it generalizable.

Interpretability: tries to describe how a decision is made by the model by tracing back to the overall process of decision-making in the model. A decision made by an interpretable model can be recreated or can be made to a new sample using the same interpretations. Here the degree of importance given to each feature can be evaluated by users (domain experts) to know for sure if biases or mistakes are properly taken care of, and if a model is working as expected (Fan et al., 2021). Its downside is the reason why a feature is considered more important than others cannot be determined using interpretability.

- To summarize the difference between Explainability and Interpretability:
  - ***Explainability*** summarizes why the model made the decision it did, but not how it mapped to that exact decision, nor if it is biased or not. It is extrinsic and model-agnostic.
  - ***Interpretability*** describes how it made its decision, but not why the criteria it used are sensible. It is more intrinsic and model-specific.
  - **Explainability** is complementary to **Interpretability** in the sense that one provides the information gap of the other.

2.4.4 Intrinsic Interpretability vs Accuracy Tradeoff

Intrinsic interpretability is a valuable utility to guide the research direction to explain how to improve models in tasks where they perform better than humans. But most ML models are focused on accuracy rather than interpretability. Even those that tried, claim that it compromises the models' accuracy and performance. For feed-forward neural networks, (Li et al., 2019) stated that maintaining interpretability and performance is still an issue to be addressed.

Generally, it is widely accepted that there is a tradeoff between inherent interpretability and accuracy in developing transparent machine learning models. But there are some studies that show otherwise (Choi et al., 2016; Li et al., 2019). According to these studies, the mechanisms used for interpretation can compensate for and even enhance model performance. The application of Attention-based mechanisms, for example, has contributed to reducing the overall computation needed, thus increasing the performance of models (Niu et al., 2021). As the mechanisms select salient features and feed them to the next step, the number of computations decreases (Arik & Pfister, 2019). In such circumstances, more interpretability leads to better overall accuracy, thus tearing down the tradeoff covenant (Choi et al., 2016; Guo et al., 2019).

2.4.5. Explainable Deep Learning (XDL)

Lately, there appears to be a rise in interest in XAI techniques and tools. Some works of literature have tried to propose inherent interpretable models in various domains (Arik & Pfister, 2019; Li et al., 2019). In order to provide interpretable classification results for image classification, (Zeiler & Fergus, 2014) proposed a Deconvolutional Network, which identifies the components of an image that are crucial for classification. For sequential data, (Choi et al., 2016) have proposed RETAIN, Attention-based RNN, to provide interpretable classification results by computing the contribution of each factor in the medical records to the predicted diagnoses. This study has shown the use of the Attention-based interpretability mechanism contributes to the rise in performance, undermining the accuracy-interpretability tradeoff.

One of the popular techniques that grant explainability to highly complex deep neural networks is ***Layer-wise Relevance Propagation (LRP)***. It operates by propagating the prediction backward in the neural network using a set of purposely designed propagation rules (Montavon et al., 2019). XAI is essential for model debugging, detecting fairness issues, and regulatory compliance in high-stake applications (Saraf et al., 2020).

Stakeholders, users, and experts would prefer to have a way to visualize and interpret the behavior of machine learning models (Wu et al., 2021; Iadarola et al., 2021). In practice, the capacity to interpret outcomes and better process the data during the subsequent iteration can overwhelm the performance of deep learning algorithms. (Wu et al., 2021). However, there is still much to learn about how effective current XAI frameworks are, particularly in high-stake applications (Kinkead et al., 2021). There aren't enough studies to properly explain how these models predict things and how well the other XAI techniques work (Ahuja, 2021). The doubt about using these frameworks for high-risk decisions can be addressed with inherent interpretable models (Rudin, 2019). This approach has diverted the attention of most XAI researchers recently (Guo et al., 2019; Owens et al., 2022).

2.4.6. XDL in Android Malware Detection

A detection system must not only alert users to potentially harmful activity, but also adequately explain why that activity was detected (Han et al., 2019; Li et al., 2019). Given an input, the black-box DL model produces output through a series of actions that are hardly humanly understandable, thus they are unlikely to be practical (Khoulimi et al., 2022). The model's interpretability determines how simple it is for analysts to control, evaluate, and correct the performance of a black-box model (Melis et al., 2018; Kinkead et al., 2021; Iadarola et al., 2021). However, the current AI-based detectors do not necessarily incorporate result-interpreting tools (Marais et al., 2021). This can result in devastating results, and thus security analysts often need to justify detection results. As a result, they have favored solutions that are simpler to understand, such as rule-based and signature-based systems, over AI-based ones because they are simpler to adjust, improve, and manage (Mathews, 2019).

In the context of malware detection and analysis, the rapid growth in the number and sophistication of android malware and the expense of its consequences, as well as the importance of the stakes involved, are indicators to ponder DL and XAI/IAI (Stamp, 2021). Malware analysts' efforts to examine them from scratch and build a knowledge base of malware samples are reduced by the interpretations of a deep learning-based malware detector. The interpretability and explainability of deep learning models for Android malware detection are, however, rarely examined in the literature (Alani & Awad, 2022; Kumar et al., 2018; Yan et al., 2021). Nevertheless, there are some works that focus in this area.

2.5. Related Works

Hereon, some works that approached the interpretability of android malware detection to investigate the contribution of features in the final result are discussed. These works provided a means to grasp the hidden process of deep learning models.

The first groundbreaking work in the explainable detection of Android malware was done by (Arp et al., 2014). The authors suggest DREBIN, a simple method for detecting Android malware that allows for the direct identification of malicious apps on a smartphone. DREBIN addressed the interpretability problem in such a way as to identify the features relevant to detection storing the most dominant weights rendering the application to be malicious. The explanations offered for each detection reveal pertinent characteristics of the detected malware. They also provided a public online dataset, Drebin, which is considered as the benchmark dataset for android malware detection.

The model proposed by (Shanshan Wang et al., 2016) examines the covert causes of harmful outcomes. Users can then look into the role that each feature played in the outcome and gain an understanding of what factors led to the final choice. Another general approach was provided by (Melis et al., 2018) by utilizing a gradient-based approach to identify the most significant local features This approach achieves explainable malware detection on Android as it draws focus to the general traits that the model learned to distinguish between legitimate and malicious applications. This approach is applicable to any black-box machine-learning model. (Marais et al., 2021), on the other hand, by incorporating attention mechanisms into detection models, they made it possible to determine which portion of the files exhibits suspicious behavior.

Apart from the challenge of explainability in android malware detection, the features found in the dataset of these explainable models often consist of a small portion of the applications, often from static analysis (Wu et al., 2021). The Drebin used features from static analysis, outperforming other machine learning approaches. It detects 94 percent of the malware samples accurately at a false-positive rate of one percent, while the other approaches scored between 10%–50% detection rate at this FPR (False Positive Rate). Despite its success, the approach lacks dynamic inspection manifesting the inherent static analysis limitations. In order to identify Android malware with high accuracy, the proposed method by (Shanshan Wang et al., 2016)

used network traffic analysis. The detection model performs well and achieves a 99 percent average detection rate. Another potential way of approaching malware detection is through the conversion of the binary files into gray-scale images. With this approach, (Marais et al., 2021) suggest a model that achieves an accuracy rate of 88, and an 85 percent accuracy in identifying whether a sample is packed or encrypted.

A DNN rule extraction method to identify malicious network traffic from mobile applications was suggested by (Yan et al., 2021), and they managed to achieve 98.55 percent accuracy. For the purpose of representing rules taken from hidden and output layers, the method combines hidden-output trees. Although implicitly, it is implied by their work that these trees can be used to provide an explanation for the detection result, the method to accomplish this goal, however, has not been discussed.

(Kinkead et al., 2021) suggested a novel CNN-based method to identify key parts that are considered significant in the opcode sequence of an Android application that seem to help with malware detection. They also compared CNN-highlighted locations with those identified as significant by the cutting-edge explainability method LIME. The approach, however, only considers the instruction sets of the application, ignoring the information found in other features as well. Additionally, this model is also susceptible to evasion attacks.

The paper (Iadarola et al., 2021) proposed a technique to extend interpretability to models designed for the detection and family identification of Android malware. Using a visual representation of the decision phase and a numerical assessment of the prediction's reliability in relation to similar predictions, the method for detecting Android malware allows a generic user to interpret and evaluate the prediction. For the purpose of designing explainable deep learning model architectures, the method relies on the representation application in terms of images. They considered the Grad-CAM adopted from the works of (Selvaraju et al., 2017), to help explain why a model predicts in a certain way, and to provide a clear and simple way for the analyst to understand. Moreover, they also demonstrated how the analyst can take explainability into account to evaluate various models. The experiments done on the proposed solution illustrated an average accuracy of 96.5 percent. The approach does not try to increase the efficiency of models and also it is restricted only to adding explainability. The model is not also tested against evasion techniques as well.

An Interesting approach was taken by (Li et al., 2021) to compile the interpretability of the logistic regression on multi-layer feedforward networks while maintaining their performance. In order to explain the results, the paper proposed a brand-new network called Galaxy Transformer. It operates in different levels of abstraction of assembly code, with an interpretable feed-forward network for result explanation. The explanations are reckoned dynamically by calculating the influence of each feature on the prediction and estimating the weighted importance of each feature. The model allegedly possesses the interpretability of a logistic regression model with the modeling ability of a multi-layer neural network and, according to the authors. The approach aims to assist malware analysts in identifying malicious payloads and recognizing recurring patterns in malware samples. Unfortunately, the paper considered only the sequence of assembly code, for detection.

An interpretable ANN model (Xmal) with a tailored attention mechanism, implemented using a Multi-layer Perceptron (MLP) model, was proposed in the paper (Wu et al., 2021) to interpret the malicious exhibited by Android apps. In terms of interpretability, the proposed model outperformed LIME and Drebin, the two most widely used tools for explainability. However, their strategy falls short of adequately describing how all malicious behaviors are used. Only the API calls of applications were examined in their work to determine whether they were malicious. The model is unable to produce features that correspond to every malicious behavior in the sample. They recommended the use of a multi-head attention mechanism to enhance the explainability.

2.5.1. Summary of Related Works

Table 2. 1: Summary of Related Works

Authors	Objective	Methods	Feature Set	Gap
Bonzhi Wu, e t al. (2021)	Interpret malicious behaviors of Android applications	MLP-based customized attention layer	APIs and Permissions	▪ Considered Static feattures ▪ Only explainability to some extent ▪ No multi-head attention
Iadarola G, et al. (2021)	Extend interpretability to Android malware detection models	Images representation of DEX files With Grad-CAM	Opcode	▪ Only opcodes (Static analysis) ▪ Not inherent ▪ Restricted to adding explainability ▪ No robust against obfuscated android malwares
Miles Q. Li, et al. (2021)	Enable nonlinear models to achieve interpretability without compromising performance	Transformer & Multi-layer feedforward neural network	Assembly code	▪ Considered only sequence of assembly code (Static analysis) ▪ Only explainability to some extent
Kinkead, et al. (2021)	Identify important locations in opcode sequence which contribute to detection	CNN	Opcode	▪ Considered only sequence of assembly code (Static analysis) ▪ Only explainability
Anli Yan, et al. (2021)	To detect malicious network traffic and understand the detection process of DNN	DNN tree-based rule extraction	Network traffic	▪ Employs only explainability ▪ Complexity ▪ Not inherent

These earlier studies aim to address the problem of improving human comprehension and transparency of deep learning models. The majority of these efforts try to explain the detection result of deep learning models using model-agnostic methods. These methods are very computationally-expensive and time-consuming. While others try to create an explainable model that can describe its own results (Wu et al., 2021). Even though this approach is better than the previous, it still lacks the information which can be tracked by using inherent interpretability mechanisms.

Some works (Li et al., 2019; Arik & Pfister, 2019) proposed a better alternative in which malware detection deep learning models can be interpretable by incorporating attention mechanisms such as Transformers to give the models inherent interpretability. This approach can be used to know how a model makes the decision process, but not why. Furthermore, in most works, only features extracted by static analysis are used for detection and classification, which makes them vulnerable to obfuscated Android malware.

From the literature review, it is apparent to point out that none of the pieces of literature reviewed so far have tried to compose the two concepts, Interpretability and Explainability, for achieving more wholesome AI transparency. This approach is the ideal direction in the pursuit of achieving XAI. Moreover, they all try to explain the decision-making process after the training phase of deep learning processes. There is no single work that tried to integrate a mechanism for debugging and surveilling deep learning models during the training phase. This approach addresses the problem of deep learning model biases during the learning process of the model, which would be an ideal time to actually monitor and debug models.

This paper proposes to solve the aforementioned gaps through the integration of interpretability and explainability to deep learning models. This is achieved with the implementation of attention mechanisms for the interpretation and accurate Android malware detection. As for the interpretation of the learning process of deep learning models, layer-wise relevance propagation is employed.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1. Chapter Overview

In this chapter, the research methodology used for the successful completion of this study is discussed, including the dataset preparation, experimentation methodology, and evaluation methodology used. This portion is necessary to ensure the reproducibility of the research.

3.2. Research Approach

A research approach consists of detailed methods of data collection, data analysis, potential solution proposition, experimentation, result interpretation, and evaluation. It is, therefore, based on the nature of the research problem being addressed. This research starts with questions and objectives that need to be realized at the end of the research process and thus can be addressed as quantitative research.

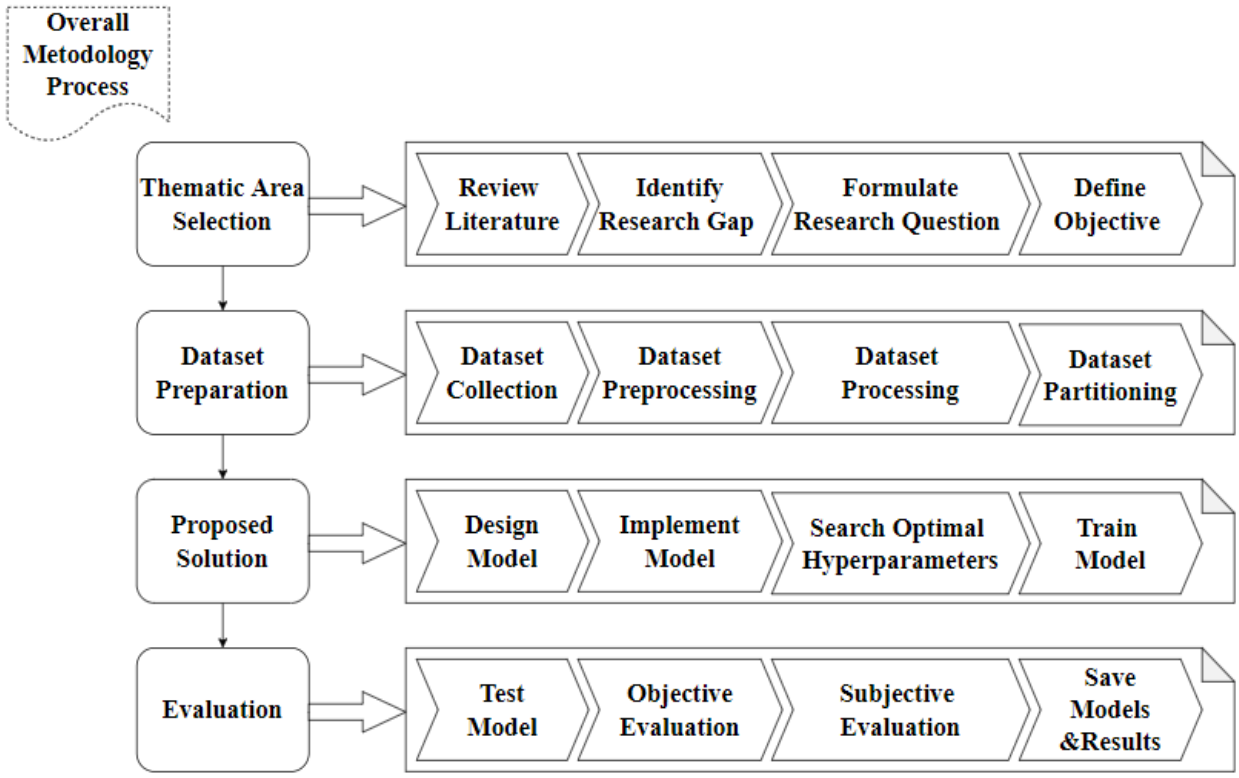


Figure 3. 1: Overall Research Methodology Process Flow

3.4. Data Preparation Methodology

The approaches used in the data preparation methodology are essentially divided into two categories: Data collection and Data Analysis.

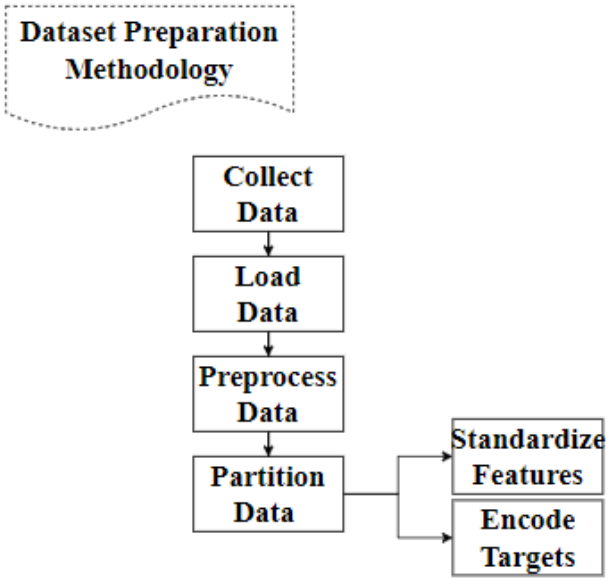


Figure 3. 2: Dataset Preparation Methodology

3.4.1. Data Collection Methodology

In essence, the quality of deep learning models depends on the training data used, and obtaining good and reliable representative training data presents a great deal of issues in the research of Android malware detection (Gilbert et al., 2020; Pei et al., 2020).

For the purpose of this study, datasets containing benign and malware applications are collected from two online public repositories, the Institute for System Security (ISS), Technische Universität Braunschweig, and the Canadian Institute for Cybersecurity (CIC) of the University of New Brunswick. These repositories are the primary source of datasets used in most cybersecurity-related research. As the datasets are the latest and benchmarked datasets with features extracted through different techniques, they are used to perceive the performance of the proposed model in accordance. The description of the collected datasets can be summarized as follows.

1. **Drebin**

Drebin is a publicly available static android malware dataset by the Institute for System Security (ISS)[16], to promote Android malware research (Arp et al., 2014). The dataset contains static features extracted using static analysis from a total of 15036 android applications, of which 5,560 applications are malicious.

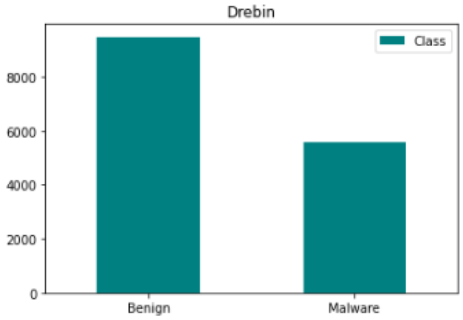


Figure 3. 3: Drebin Dataset Target Classes

2. **CIC-MalMem-2022**

MalMem is an obfuscated android malware dataset provided by the Canadian Institute for Cybersecurity (CIC) [17] from real-world simulations. It was developed to test memory analysis techniques for the detection of obfuscated malware. Using malware that is common in the real world, the dataset was made to be as accurate a representation of a real-world situation as much as possible (Carrier et al., 2022). It offers a balanced dataset containing a total of 58,596 records with equal benign and malicious memory dumps.

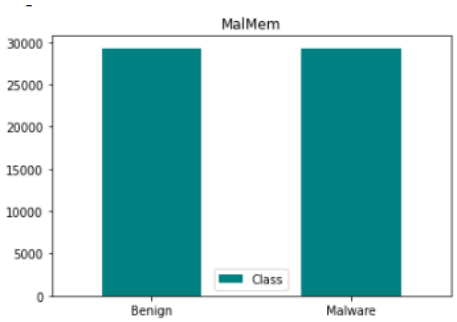


Figure 3. 4: CIC-MalMem 2022 Dataset Target Classes

[16] https://www.sec.cs.tu-bs.de/~danarp/drebin/
[17] http://205.174.165.80/CICDataset/CICMalMem2022/

❖ **Artificially Generated Dataset**

This dataset is a randomly generated tabular data used as part of a controlled environment setup. It is used to assess and validate the accurate interpretation of the proposed network. Numpy's Random Sampling (numpy.random) library was used to achieve an arbitrary number of features and data instances. The library produces pseudo-random numbers using combinations of BitGenerator and Generator to sample from different statistical distributions[18]. This dataset is used to demonstrate the correct quantification of the said "coefficient" or relative feature importance of input variables. The data in each instance are all randomly generated.

3.4.2. Data Pre-processing Methodology

NumPy and Pandas libraries were used for importing, cleaning, and manipulating datasets.

- **Data Cleaning:** The datasets have been cleaned with the following techniques using the powerful tools provided by pandas:
    - ✓ Renaming Columns for simplicity
    - ✓ Replacing nan values
    - ✓ Replacing unsupported invalid data types
- **Dataset Partitioning:** The datasets are partitioned into training, validating, and testing datasets with the Pandas library. Initially, this split is done in the ratio of 70:10:20.
- **Categorical Encoding:** is the process of representing non-numeric data into numeric values. This is necessary as deep learning algorithms only work with numeric representations.
    - **One-hot Encoding:** ideal for a set of data where there is no natural ordering between categorical values. It represents the categorical values using binary values (0, 1).
- **Standardization:** ensures the training stability and performance of a model, and prevents them from behaving in an unexpected way. Typically, this is done transform the dataset into a normally distributed dataset by reducing the mean to 0 and scaling the variance to 1[19]. In this work, Sklearn's StandardScaling is used for standardization.

[18] https://numpy.org/doc/stable/reference/random/index.html
[19] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

3.5. Experimentation Methodology

The experimentation methodology compromises the steps taken starting from designing to recording progress results. This includes designing the proposed network, implementing a model based on the network, training, testing and evaluating the model. At last, the model and the result from the evaluation are saved for analysis.

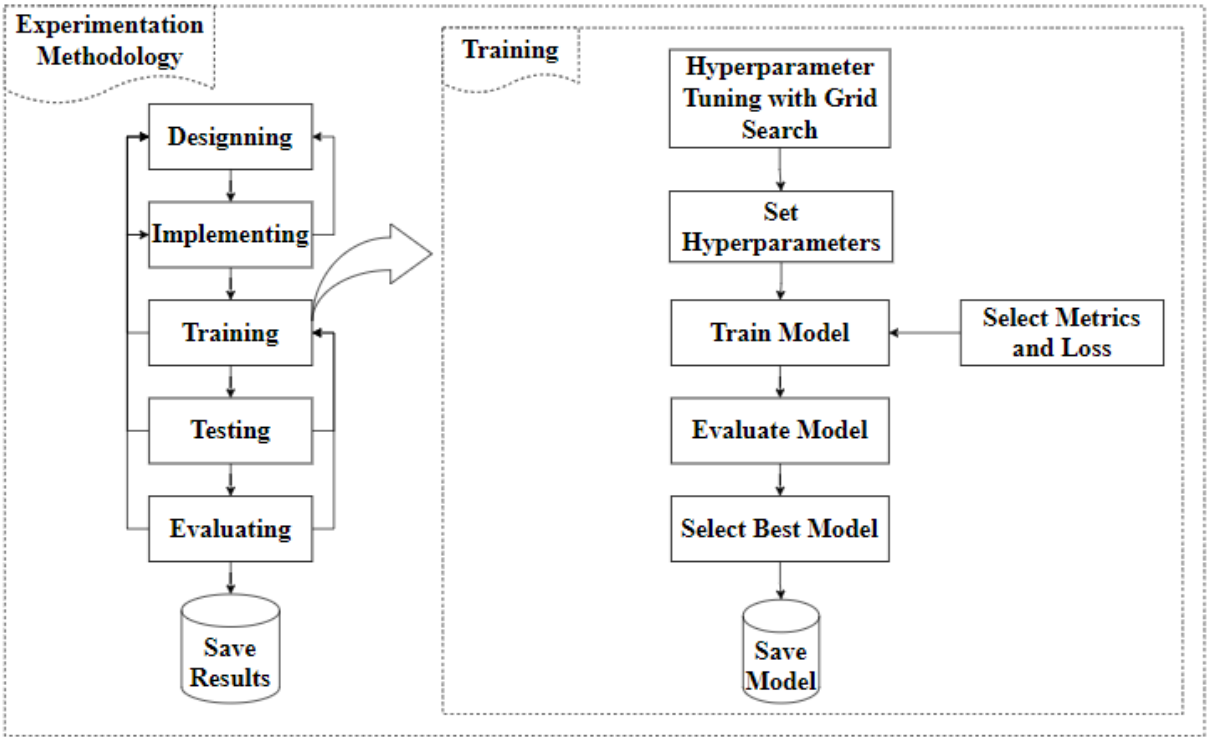


Figure 3. 5: Experimentation Methodology

3.5.1. Conceptual Framework

- Explainable machine learning models can be achieved by tracing every decision made by the model (Wu & Chen et al, 2020).
- Selecting prominent features from both static and dynamic approaches is necessary to fully analyze the structure and behavior of android malware be for effective android malware detection (Gera et al, 2021).
- Supervised deep learning can be used to achieve accurate Android malware detection (Zhao et al., 2019).

3.5.2. Baseline Works

The performance of the proposed model is compared to recent methods using the evaluation methodology. The recent works implemented alongside the proposed model are:

1. **XGBoost (eXtreme Gradient Boosting)**[20]: method proposed by (Chen & Guestrin, 2016), is the most popular ML algorithm for tabular datasets.

2. **Xmal**[21]: a novel explainable android malware detection model proposed by (Wu et al., 2021) to identify malware as well as point out its malicious behavior.

3. **TabNet**[22]: a novel interpretable deep learning architecture proposed by (Arik & Pfister, 2019) for tabular data. It employs sequential attention to determine which features to base each step's decisions on.

3.6. Hardware and Software Requirements

- The hardware and software tools used for the successful completion of this study are discussed.

3.6.2. Hardware Requirements

In the table below, the hardware tools used in the research are listed.

Table 3. 1: Hardware Requirements

Tools	Description	Usage
Laptop	Lenevo E-51 Core-i7 7th Generation with 16GB RAM	Main accessory through overall research
Hardisk	1TB of HDD hard disk	To save models and datasets
Desktop	HP HP 290 G4 Microtower P	To train the model
GPU	GeForce RTX2070 Super 6GB RAM	To process the datasets

[20] https://xgboost.readthedocs.io
[21] https://github.com/wubozhi/Xmal
[22] https://github.com/google-research/google-research/tree/master/tabnet

3.6.1. Software Development Requirements

Software Development Tools are collections of software and libraries necessary to provide a suitable environment for the overall development process. They are used for designing, implementing, visualizing, debugging, testing, deploying, and maintaining the whole application software. The software tools that are used in this study are listed in the table below.

Table 3. 2: Software Requirements

Tools	Version	Description
Python[23]	3.9.1	High-level, interpreted, and interactive programming language popular in AI applications for its simplicity and rich resources
TensorFlow[24]	2. 9.1	Open-source library for AI applications
Keras[25]	2.9.0	Popular high-level API used in AI
Google Colab[26]	-	Online working environment with GPU & TPU
Scikit-learn[27]	1.1.2	Library with various ML algorithms, metrics, and other tools
Pandas[28]	1.4.3	Powerful data analysis, manipulation, and visualization tool
NumPy[29]	1.23.0	Python library for computation on multi-dimensional data
Matplotlib[30]	3.5.2	Comprehensive multi-platform data visualization library
Tensorboard[31]	2.10.0	Visualization for ML experimentations in TensorFlow
Google Drive[32]	-	Personal Cloud Storage and File Sharing Platform
diagrams.net[33]	-	Free online diagram software designing tool

[23] https://www.python.org/
[24] https://www.tensorflow.org/
[25] https://keras.io/
[26] https://colab.research.google.com/
[27] https://scikit-learn.org/
[28] https://pandas.pydata.org/
[29] https://numpy.org/
[30] https://matplotlib.org/
[31] https://www.tensorflow.org/tensorboard
[32] https://drive.google.com/
[33] https://app.diagrams.net/

3.7. Evaluation Methodology

The type of evaluation methodology and metrics used are crucial for a reliable assessment of machine learning models. Evaluation metrics are applied to track and measure model performance during both training and testing, and then rank models in accordance. Oftentimes, a single metric may not give a full picture of the result at hand as these metrics may have different implications. Thus, using a subset of the metrics is necessary to have a concrete evaluation of the models.

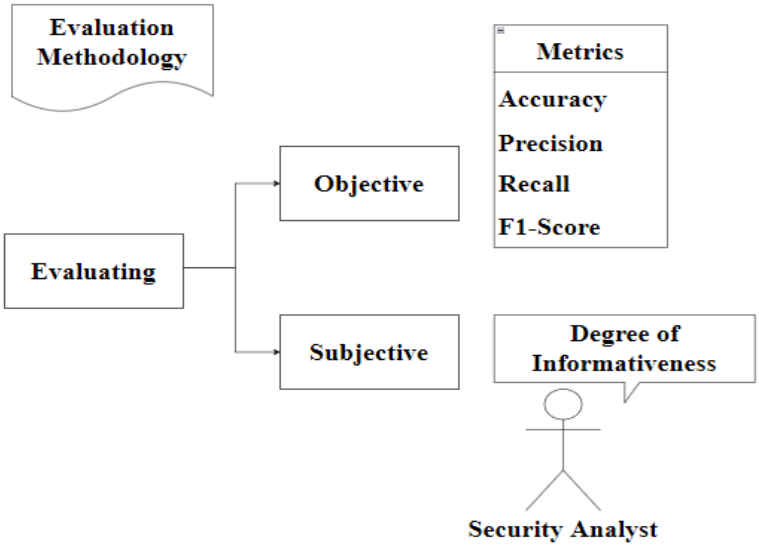


Figure 3. 6: Evaluation Methodology

Subjective evaluation: an evaluation technique that determines the quality of model descriptions in terms of informativeness. It is assessed by human domain expertise or end-users using prepared surveying methods such as questionnaires. The proposed model is evaluated by security experts at INSA.

Objective evaluation: an evaluation technique in which the overall performance of a model is assessed by different mathematical evaluation metrics.

Evaluation of the proposed model was conducted using both subjective and objective evaluation techniques using a variety of metrics for a better understanding of the results. The comparative analysis between the model and other contrasting models is also based on the following metrics.

A. **Confusion matrix**: tabular visualization of ground-truth versus model predictions.

Table 3. 3: Confusion Matrix

Confusion Matrix		**Ground Truth**	
		P	**N**
Prediction Result	**P**	TP	FP
	N	FN	TN

B. **Accuracy**: the percentage of correct predictions out of the total number of predictions.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (3.1)$$

C. **Precision**: the percentage of correct positive predictions out of all the positive predictions.

$$Precision = \frac{TP}{TP+FP} \quad (3.2)$$

D. **Recall/Sensitivity/TPR**: the percentage of correctly predicted sample from a class.

$$Recall = \frac{TP}{TP+FN} \quad (3.3)$$

E. **F1 score**: popular metric appropriate for an imbalanced dataset.

$$F1 - score = 2 * \frac{Precision*Recall}{Precision+Recall} \quad (3.4)$$

F. FPR: the proportion of incorrectly classified negative instances.

$$FPR = \frac{FP}{TN+FP} \quad (3.5)$$

G. FNR: the proportion of incorrectly classified positive instances.

$$FNR = \frac{FN}{TP+FN} \quad (3.6)$$

Table 3. 4: Evaluation Metrics Terminologies

		Predicted	
		Positive	**Negative**
Actual	**Positive**	True positives (TP)	False negatives (FN)
	Negative	False positives (FP)	True negatives (TN)

CHAPTER FOUR

DESIGN OF THE PROPOSED NETWORK

4.1. Chapter Overview

This chapter describes the overall design process of the proposed solution. The rest of the chapter is divided into three sections; Proposed Solution, Network Architecture, and Network Design.

4.2. Proposed Solution

The proposed solution is to use an interpretable and explainable deep attentive network for instance-wise prominent feature selection and accurate Android malware detection. Inspired by the works of (Arik & Pfister, 2019), the network is designed to describe the working principle and the learning process itself while performing prominent feature learning and selection and supervised classification, thus android malware detection. It uses a sequential instance-wise feature selection approach to choose relevant features at each step. This design enables two kinds of interpretation; explainability and interpretability. During the testing/deployment phase, the explainability of the network is done at different levels; step, local and global. Meanwhile, the interpretability is provided through layer-wise propagation.

The proposed solution bases its ground on the TabNet encoder architecture proposed by (Arik & Pfister, 2019). The encoder consists of a Feature Transformer block, an Attentive Transformer, and Feature Selector (Masker) block. The salient features are selected as a result of sparse normalization of the coefficients using Sparsemax. The feature selection mask also provides interpretable information about the functionality of the network for each step, and the masks can be combined to determine the importance of a feature globally.

The approach used in the paper is sequential of attentive networks that resemble the ensemble approach used in shallow machine learning algorithms. The attentive networks, Steps, perform fair voting to decide the final prediction. Given its impressive technical proposal, the use of a shared layer in TabNet appears to be illogical. It follows an unnecessary, complex, and rather deleterious approach by mixing both bagging and boosting ensemble learning in the name of Transfer Learning. The knowledge transfer is performed using a shared layer which subjugates the dependent layers into a more uniform representation. This leads to significant deviation from

the idea of ensemble learning, which is to create a more diverse and independent representation of input subspace for each classifier to make a fair and independent assumptions necessary for fair and independent decision-making (voting) process. In the designed network, IXNet, however, the transfer of knowledge occurs with the use of continuous feature selection across the network's steps rather than a fine-tuned weight parameter of a layer as in TabNet. This way the network makes assumptions about the current sample batch based on the enhanced data given at that instance rather than the assumption made by previous layers.

4.3. Network Architecture

The architecture has two parallel components: the interpretation and detection components. The interpreter component contains Attentive blocks, Selector blocks, and the Presentation subcomponent. The detection component, on other hand, contains Transformer blocks and a Classifier block. Batch normalization is employed to regulate the learning process of the network, and dropout to avoid possible overfitting.

The architecture can be perceived as an abstraction of a series of step operations performed on the input data. Each network step is a block of components from both the interpreter and detector components of the network. There are a predefined number of steps in the network each of which employ feature selection and feature transformation. In a step, the masked input representation from the previous step, if available, is used to enable further focus on relevant features. Each step gets its own vote in the final classification and interpretation.

The proposed network, just like TabNet, mimics an ensemble classification as each step votes to make the final decision. The output from each step is also aggregated as step importance to influence the voting procedure. The more aggregated output, the more step importance, in turn the more the veto power. The attention masks learned at each step and iteration are recorded to provide an observable depiction of the phenomena inside the network. The decision behind each feature selection (masking) at each stage can be validated or understood using the interpretation presented at different stages of the network. The number of steps is a hyperparameter option; increasing the steps increases the learning capacity of the network, but also increases training time, memory usage, and the probability of overfitting.

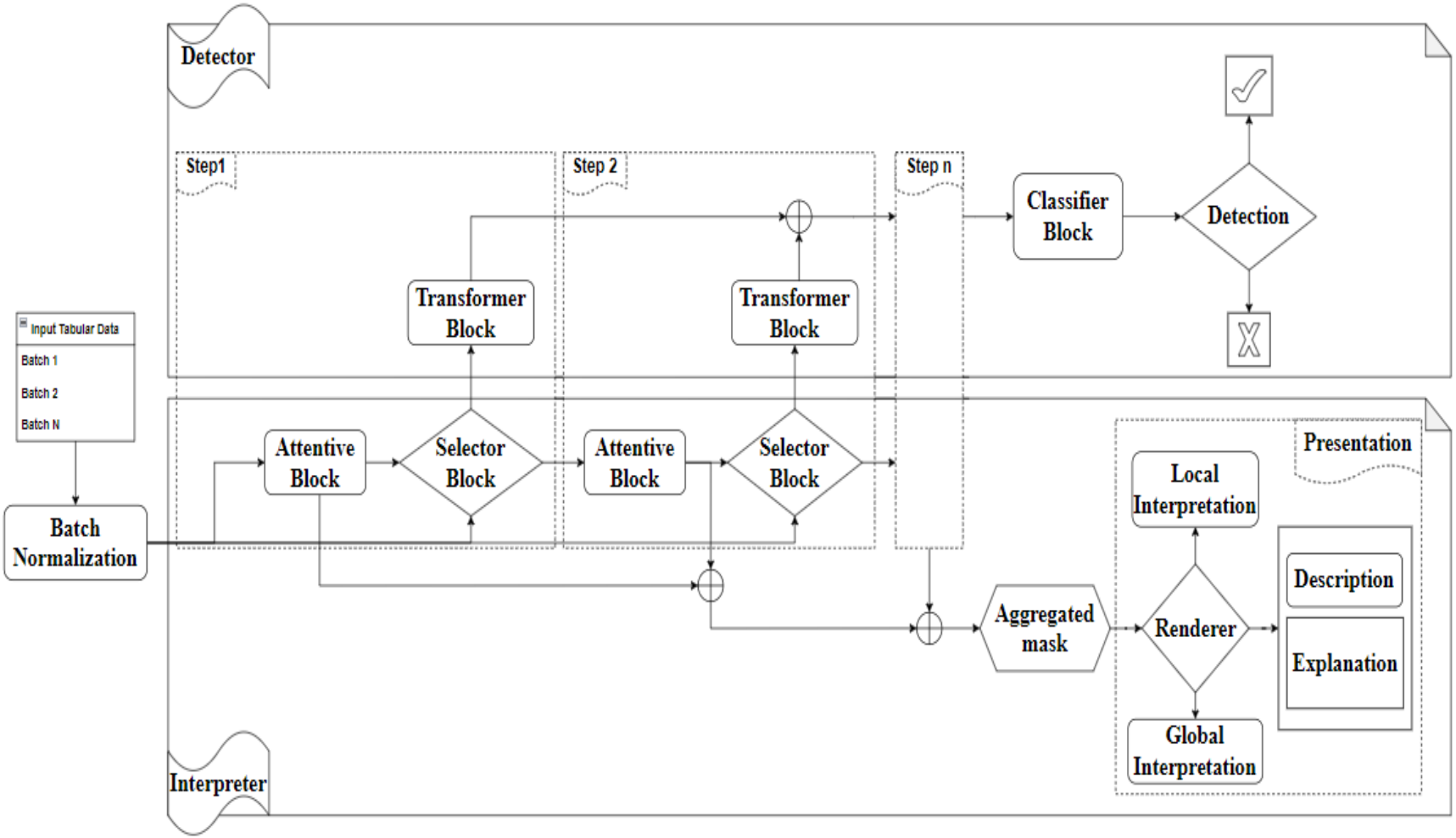


Figure 4. 1: Overall architecture of the proposed IXNet Network

❖ The major components are described as follows:

4.3.1. Interpreter Component

The interpreter component contains Attentive blocks, Selector blocks and The Presentation subcomponent. The number of Attentive and Selector blocks is equal to the number of steps.

1. Attentive Block

This block contains a Multi-head Attention layer, Average Pooling layer, and Batch Normalization layer. The attentive block accepts normalized input data and returns the alignment score of each value against themselves using a self-attention mechanism after normalizing it. The self-attention is employed using a Multi-head Attention layer.

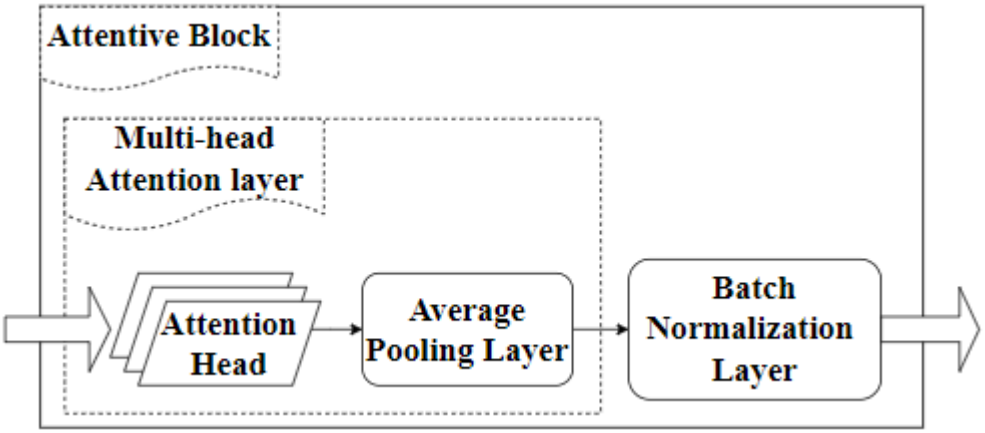


Figure 4. 2: Attentive Block

- The **Multi-head Attention layer** contains multiple attention heads that return alignment scores in different representations.
- The average value of the returned scores is computed using an **Average pooling layer**.
    - ❖ Average pooling layer is a mean reduction algorithm
- Finally, the average value is normalized using a Batch Normalization layer and returned as the output of this layer.

2. Attention Layer

The attention layer is basically two fully connected layers with Rectifier Linear unit (ReLu) and hyperbolic tangent (tanh) activation functions respectively. The layers have trainable kernels (weight variables) but no bias variables. The first layer has an arbitrary number of neurons while the second has neurons exactly as many neurons as the number of input features.

- The block accepts vector representation of the input sample. The dot product of the input with the first kernel weight and then passed through a ReLu function.
- Then, the dot-product of the output from the ReLu with the second kernel weight is fed to a tanh function.
- Then, a Batch normalization layer is employed to normalize the output.
- The normalized value is multiplied by the prior scales.
- Finally, the alignment score is calculated by feeding the result to a Softmax or Sparsemax function, and then the result is returned.

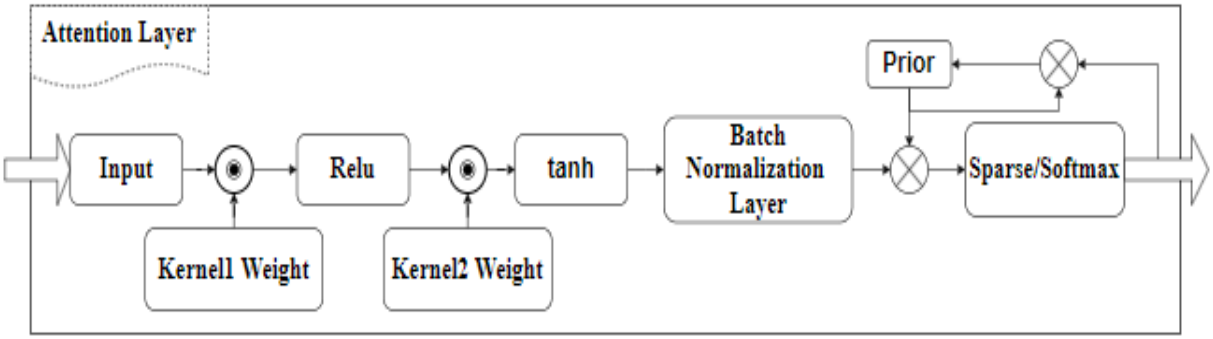


Figure 4. 3: Attention Layer

3. Selector Block

This block is basically a multiplication-normalization block. The output from the Attentive block and the normalized input is multiplied element-wise to mask less relevant features using the score provided by the Attentive block. This block enables the network to mask less dominant input values, thus giving instance-wise feature selection ability to the proposed network.

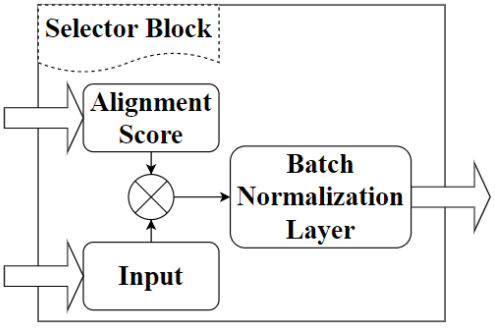


Figure 4. 4: Selector Block

4.3.2. Detector Component

The detection component contains Transformer blocks and a Classifier block made up of a single fully connected (FC) layer. The number of Transformer blocks is equal to the number of steps of the network.

1. Transformer Block

The transformer block is made up of two fully connected (FC) layers, a Dropout layer, GLU layer, and ReLu. The block accepts the selected (masked) feature vector representations of the input sample.

- The input is fed to an FC layer and its output is passed through a Dropout layer.
- The output from the dropout layer is fed to another FC layer.
- A Batch normalization layer then receives the output vector from the second FC layer and then, the output value is fed to the GLU subblock.
- Finally, the output from the GLU is then passed through a ReLu and outputted.

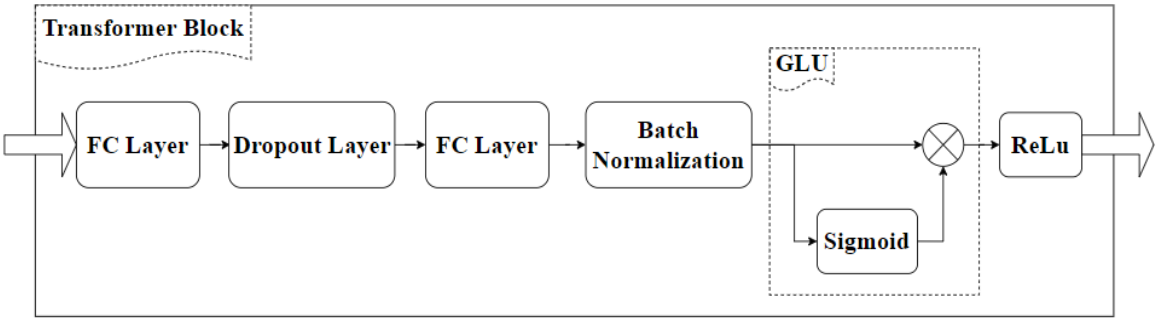


Figure 4. 5: Transformer Block

2. Classifier Block

This block is the final decision-making layer. It only contains a single fully connected layer.

- It Accepts the outputs of Transformer blocks from each step and aggregates them.
- Then, the aggregated value is fed to an FC layer for the final classification decision.

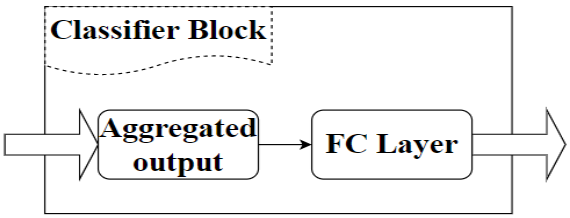


Figure 4. 6: Classifier Block

4.4. Network Design

The design process of the proposed network is explained here. The concepts and algorithms used for the design, the model problem formulation, and the working principle of the network is summarized as follows.

4.4.1. Network Building Algorithms

The proposed network is mainly designed using the notions of Attention, fully connected layer, Dropout, Batch Normalization, activation functions, Loss functions. These concepts are described below.

A. Attention: Inspired by how the human brain tends to concentrate on the relevant parts when processing a sheer volume of data at once, the concept of attention in deep learning has arguably gained fair popularity. Proposed by (Bahdanau et al., 2014), it first originated in NLP to address the problem of modeling lengthy sequences in encoder-decoder-based neural machine translation systems. By considering the context variable as an output of attention pooling, the model only aligns with the portions of the input sequence that are pertinent to the current prediction. The fundamental concept in the Attention mechanism is weighted mean reduction, the quantization of the contribution of each element to the mean[34]. In deep learning, it is a vector of importance weights that estimates how strongly one data point correlates with others.

Introducing the ideas of local attention and global attention, (Luong et al., 2015) also redefined the calculation of alignment score in three different approaches: the dot-product method, the concatenation method, and the general method. The most common attention mechanism is a dot product and Softmax used in recent works (Arik & Pfister, 2019).

$$attention(s, h) = Va^T tanh(Wa[s: h]) \quad (4.1)$$

where both Va and Wa are attention weight matrices as learnable parameters in the alignment model.

[34] https://dmol.pub/dl/attention.html

Self-attention: is a special attention mechanism that scores the sets of values directly with themselves. First introduced by (Cheng et al., 2016), self-Attention is defined as "the mechanism of relating different positions of a single sequence or sentence in order to gain a more vivid representation".

Multi-head Attention*:* enables the model to jointly attend to data from various vector representation subspaces at various positions. Each head's individual vectors are combined into a single vector, which is then linearly projected into the initial dimension's subspace. The most famous example of multi-head self-attention is Transformer.

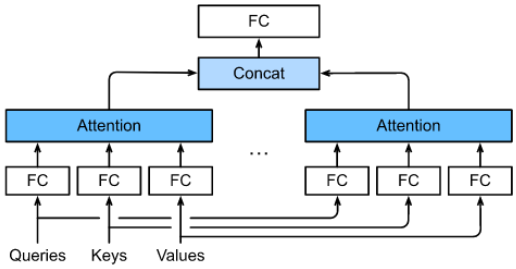


Figure 4. 7: Multiread Attention Adopted from (Vaswani et al., 2017)

Transformers: are novel methods of processing sequences without the need for Recurrent Neural Networks, as proposed by (Vaswani et al., 2017). The transformer, made up of a multi-head self-attention mechanism, concentrates on data points in various linear subspaces. It served as the foundation for cutting-edge models for language representation. Based on key, query, and value, this work provided a highly general and inclusive definition of attention.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{4.2}$$

The development of deep neural networks has led to widespread adoption of the attention mechanism across a range of application domains. In the work of (Niu et al., 2021), the potential of attention to bring interpretability to deep learning was spotlighted.

B. **Activation Functions**: introduce non-linearity to artificial neural networks for the realization of the modeling of non-linear relations in ANNs. The most common activation functions used include:

Sigmoid: smoothes the input values between [0, 1]. Mathematically it is expressed as follows:

$$\delta(x) = \frac{1}{1+e^{-x}} \tag{4.3}$$

ReLu: converts all negative numbers to zero and outputs non-negative values. Mathematically it is expressed as follows:

$$Relu(x) = max(0, x) \tag{4.4}$$

Softmax: normalizes its input value into a probabilistic distribution summation of one within the range between zero and one. It is used as a classifier instead of random activation. Mathematically it is expressed as:

$$\sigma(\vec{z})_i = \frac{e^{zj}}{\sum k_{j=1}\, e^{zj}} \tag{4.5}$$

where $z^{\rightarrow}$ is the input vector and K is the number of classes.

Sparsemax: a softmax-like activation function that produces a sparse vector of probabilities. Although it achieves a performance similar to Softmax, it applies a more compact selective approach. Mathematically it is expressed as:

$$sparsemax(z) = arg_{p\in\Delta k-1} min\ ||p - z||^2 \tag{4.6}$$

GLU: outputs the product of the input value with its Sigmoid function output. Adopted from the work by (Dauphin et al., 2016), it provides non-linearity for the Transformer block as implied by (Arik & Pfister, 2019).

- $GLU(x) = x \times sigmoid(x)$ (4.7)

C. Fully connected layer (FC): a neural network where every neuron is connected to every neuron in the next layer. They are typically found in ANNs as hidden layers.

D. **Dropout**: discards the output of a portion of the hidden nodes at each layer by freezing/ disconnecting some neurons randomly during training. It is used mainly as a countermeasure to avoid overfitting in machine learning models (Hinton et al., 2014).

E. **Batch Normalization**: accelerates the training of DL models by reducing the internal covariate shift. It is used to control exploding numeric values of computations in the training phase (Ioffe et al., 2015).

F. **Loss Functions:** these are cost functions used for minimizing the overall error of a model.

4.4.2. Modeling and Problem Formulation

For a function $f(x) = a1 * x1 + a2 * x2 - a3 * x3$, the coefficients a_1, a_2 and a_3 can be taken as the degree of importance to x_1, x_2 and x_3 respectively. The magnitude of the coefficient quantifies the importance measurement while the sign indicates whether the variable is directly or inversely proportional to the output. In other words, the contribution of variable x_1 for the output values is equal to a_1 and the relative contribution is the ratio of a_1 from the total sum of the coefficients, i.e.,

$$\frac{a_1}{a_1 + a_2 + a_3} \tag{4.8}$$

Any ANN can be simplified to a perceptron. This simplification can be helpful to quantify the contribution of each feature to an output. Similar to a Logistic Regression, in the perceptron simplification of ANNs, the contribution of each input variable is directly proportional to their respective assigned final weight variable of the output layer.

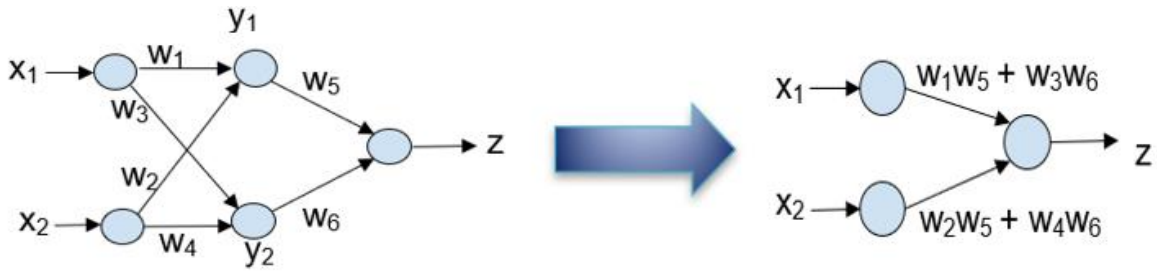


Figure 4. 8: ANN and Perceptron Simplification of ANN

For input feature vector space $X = (x1 + x2 + \ldots + xn)$, and weight vector $W = (w1 + w2 + \ldots + wn)$ and for linear activation function σ, the output z is

$$y1 = \sigma(w1 * x1 + w2 * x2)\ y2 = \sigma(w3 * x1 + w4 * x2)$$

$$f(X) = z = \sigma(w5 * y1 + w6 * y2)$$

$$f(X) = z = \sigma\big(w5 * \sigma(w1 * x1 + w2 * x2) + w6 * \sigma(w3 * x1 + w4 * x2)\big)$$

$$f(X) = z = w5 * w1 * x1 + w5 * w2 * x2 + w6 * w3 * x1 + w6 * w4 * x2$$

$$f(X) = z = (w5 * w1 + w6 * w3) * x1 + (w5 * w2 + w6 * w4) * x2$$

$$f(X) = z = (w1w5 + w3w6) * x1 + (w2w5 + w4w6) * x2$$

The relative contribution of x_1 will also be (this also works for x2 in a similar fashion),

$$\frac{W_1W_5 + W_3W_6}{W_1W_5 + W_3W_6 + W_2W_5 + W_4W_6} \tag{4.9}$$

As shown above, this attribution can be done using the dot-product of the weight variables starting from the output layer back to the first hidden layer.

As of the feature attribution, the attention layer is designed to have two kernels (basic FC layers). The first accepts the input sample and maps it to a hidden representation whereas the second kernel inverts the hidden representation back to its original feature dimension. This vector is then used as the criterion for feature selection. This can be shown as follows.

Let X ∈ R be a feature vector space representing a sample given to a normal FC layer,

$$F(X) = tanh(w_1 F(X) + b_1) \tag{4.10}$$

where w_1 and b_3 are the weight and bias of the first kernel layer respectively.

Then, another normal FC layer of which the output vector has the same dimension as X is applied:

$$G(X) = tanh(w_2 F(X) + b_2) \tag{4.11}$$

where w_3 and b_3 are the weight and bias of the second kernel layer respectively.

G(X) can be used to serve as a relative attention weight for each feature in the vector space X. The final output of the classifier is calculated as follows:

For Binary Classification

$$Z(X) = sigmoid(w_3 G(X) + b_3) \tag{4.12}$$

For Multiclass Classification

$$Z(X) = softmax(w_3 G(X) + b_3) \tag{4.13}$$

where w_3 and b_3 are the weight and bias of the final classifier layer respectively.

This is applicable to DNN and other deep learning models. The main attribution criterion in this proposed model focuses on the computation of this final simplified form of deep learning models in terms of perceptron and its weight parameters. In the proposed model, this can be implemented by embedding the code directly into the model or simply using callback functions of the Keras API library. So long as the input sample is presented with a predetermined dimension, the proposed model can be used for any classification task.

4.4.3. Working Principle of the Proposed Network

The interpreter component accepts batch normalized input at the first step and computes the alignment score or mask of the input. This score vector represents the quantitative importance measure of each feature. Then, the Selector block masks the normalized input using the alignment score vector, thus achieving feature selection. The selected features in the form of masked input are fed to the corresponding classifier in the detector component at each step.

In the detector component, each of these masked inputs is fed to their respective Classifier blocks. The classifiers then transform the given input vectors into different feature representations. These feature representations are then aggregated, and fed to the final FC layer. The final decision layer makes the classification tasks based on the representations. The alignment scores (masks) at each decision step are also traced so as to explain every decision path taken. They are weighted by the step importance which is the aggregated sum of step output. The values of the masks can be used to determine the degree of importance of each feature, and are averaged to give global interpretation as well.

- **Prior scale** is a way of keeping track of the prevalence factor of features; to which extent they have been used in previous steps. It is useful to regulate the dominancy of features at subsequent steps. Initially given the values of identity, it is updated at the end of each step.
- **Softmax** and **Sparsemax** yield a vector score that always has a probabilistic distribution summation of one. This vector is used to quantify the relative contribution of each future.

The feature selection is done by the Selector block using matrix multiplication between the input and the alignment scores from the Attentive Block to give a masked feature vector. This attention score vector makes sure the network attends to the selected features. This feature selection is essential as it enables the generalization of decision boundaries to the combination of linear features where coefficients specify the relative weights of each feature, which in the end leads to the network's interpretability.

The presentation subcomponent uses the aggregated mask values to make interpretations at the local and global levels using rendering techniques, such as NLP and/or visualizations. This subcomponent is not discussed in detail in this study and will be implemented as a simple client-server visualizing tool.

4.4.4.1. Interpretation of Concept Learning Process

The learning process of the network can be tracked using a layer-wise backward relevance propagation. It can trace the grasped concepts of the network at each batch and epoch to provide a way of surveilling the learning process of the network, thus achieving interpretability at the training phase. Here, Keras's callback class can be used to implement the discussed approach.

A Callback is a tool provided by Keras for monitoring the behavior of a Keras-based model during training or evaluation. It can be used to perform certain operations at various stages of training. The callback class object is passed to the "*.fit()"* method of the network, thus the relevant methods of the callbacks are then called at the specified stage of the training.

1. **Flowchart for General Purpose Layer-wise Backward Relevance Propagation**

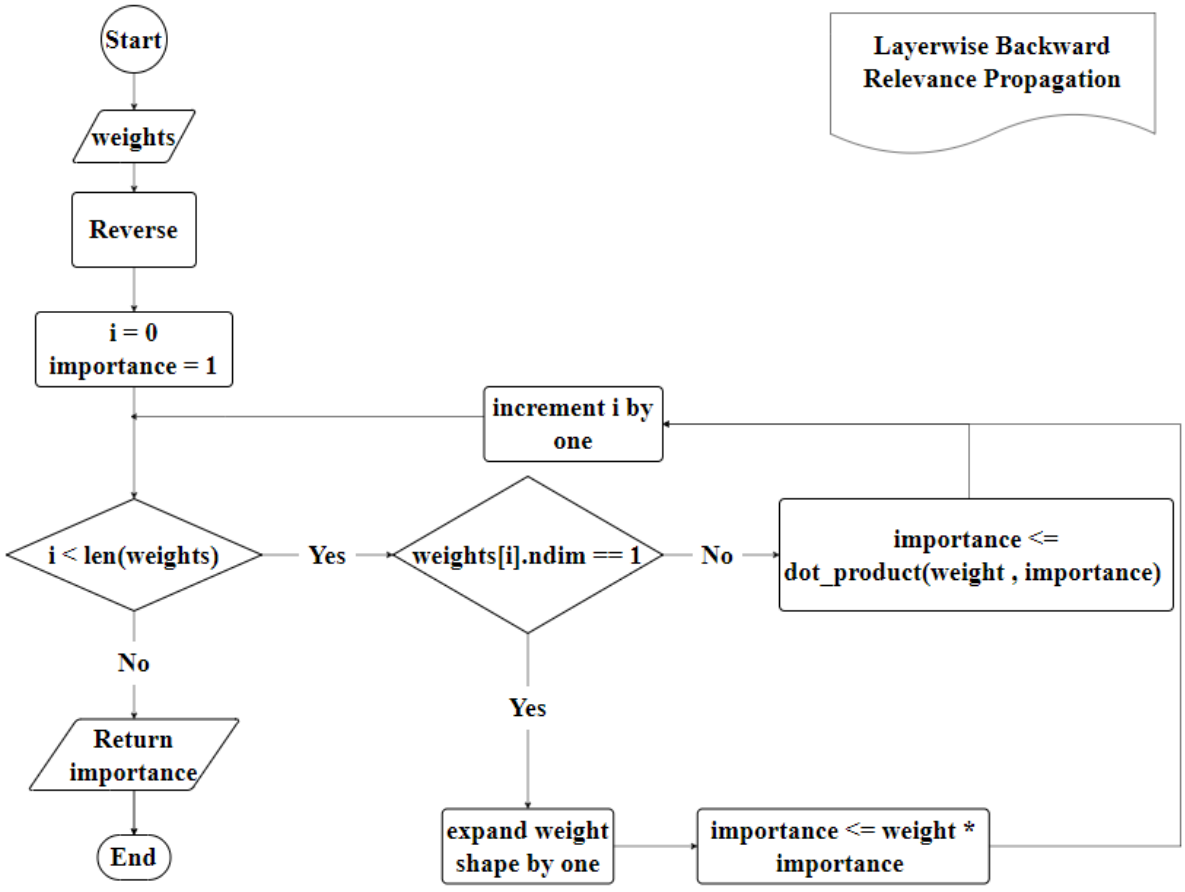


Figure 4. 9: Flowchart of General-Purpose Layer-wise Backward Relevance Propagation

2. Pseudocode for Interpretation of IXNet Concept Learning Process

Input: *model classifier and Transformer weights*
Output: *relative feature importance*
Start:
Initialize empty list to store feature ***importance***
Reverse *stored* ***weights***
Classifier weight ⇐ ***weights[0]*** *// first kernel weight*
Common BN weight ⇐ ***weights[-1]*** *// last gamma weight*
Expand ***Common BN weight*** *shape by one*
For ***step*** *in model* ***steps*** *do:*
Transformer BN weight ⇐ ***weights[step]*** *// gamma*
Expand ***Transformer BN weight*** *shape by one*
Transformer weight ⇐ ***weights[step+1]*** *// kernel*
temp ⇐ *Element-wise product of* ***Transformer BN weight*** *and* ***Classifier weight***
temp ⇐ *Dot product of* ***Transformer weight*** *and* ***temp***
importance ⇐ *Element-wise product of* ***BN weight*** *and* ***temp***
Store ***importance***
End For
End For
importance ⇐ *Reduced mean average of* ***importance***
Remove all size one dimension of ***importance***
Normalize ***importance***
Compute ***relative importance***
Return feature importance
End At
End

Code Snippet 4. 1: Pseudo Code for Interpretation of IXNet Learning Process

❖ Here, an object of Callback class is used to track the learning process of IXNet network at the end of each epoch. The weights learned are computed in a reverse-layer-wise relevance propagation manner to trace the saliency of each feature at the given interval. Thus, giving the network the capability to provide the learned concept in terms of the data features provided in the training dataset.

CHAPTER FIVE

IMPLEMENTATION DETAILS

5.1. Chapter Overview

This chapter implements the design discussed in the previous chapter. It describes the details of the environmental setup, the data transformation implementation, the implementation of the proposed network, and the experiment conducted.

5.2. Environmental Setup

This environmental setup of hardware and software tools used to implement the proposed network is discussed as follows.

5.2.1. Hardware Tools

Table 5. 1: Hardware Tools

Laptop	Desktop Computer
Lenevo E-51 Core-i7 7th Generation	HP 290 G4 Microtower P
Intel™ i7-6500U CPU@ 20GHz 3.1 GHz	Intel® Core™ i7-10700 CPU@ 2.90GHz × 16
16 GB RAM	8 GB RAM
Window 11 64-bit OS, x64-based processor	Ubuntu 64-bit OS, x64-based processor
Intel ® HD Graphics4600 Graphics card	Mesa Intel® UHD Graphics 630 (CML GT2)
No GPU	GPU: GeForce RTX2070 Super 6GB RAM

5.2.2. Software Tools

- **Python3**: is used as the primary programming language.
- **Google Drive**: used as online storage for datasets, models, and results.
- **Google Colaboratory (Colab)**: used as an online IDE for implementation and evaluation. The Nvidia K80 GPU with internal RAM Memory of 12.68GB, memory clock of 0.82GHz, and Disk space of 78.19GB is used.
- **TensorFlow 2.1**: intensively used as a framework for the implementation of the proposed network and some other baseline works.

5.3. Implementation of Data Transformation

5.3.1. Data Preprocessing

The datasets are loaded in panda's DataFrame format using the Pandas library. They are cleaned using different preprocessing techniques.

- **Drebin Preprocessing**

Some invalid data values were found in the dataset, ['0', '1', 'nan', '?']

- Fill 'nan' values with '0' best approach to remove a small percentage of 'nan' values in the dataset
- Some numeric values were found as strings. So, they are casted into integer format.
- The remaining invalid data values ('?') are replaced with '0'.

```
df = pd.read_csv(dataset_path)
dataframe.fillna(value=0, inplace= True)
df = df.replace(value, int(value))
df = df.replace('?', 0)
```

Code snippet 5. 1: Drebin Preprocessing

- **MalMem2022 Preprocessing**

This dataset is perfectly cleaned at its state of publication. The dataset is balanced and contains no invalid, missing, or 'nan' values in independent variables. The data type of all of the independent variable's data is nominal. The dependent variable 'Category' in the dataset contains unnecessary file names/hash values appended to it in the format of 'category-family-filename_hash_values'. Since the scope of this work is limited to binary classification (detection), the column is dropped.

```
df = df.drop(columns={'category'})
```

Code snippet 5. 2: MalMem Preprocessing

❖ The rest data preprocessing techniques are common to both datasets.

- ***Shuffling***: the datasets are shuffled before partitioning
- ***Partitioning***: the datasets are splitted into features (independent variables) and targets (dependent variables).
    - The name of the features is also recorded for interpretability purpose.
- ***Standardization***: the independent variables are standardized using StandardScaler.
- **Encoding:** the dependent (target) variables are encoded using One-Hot Encoding.

```
df = df.sample(frac=1, axis=0, random_state=3, ignore_index=True)
features = df.drop(Target, axis=1)
target = df[Target]
Featurelist = features.columns
standardScaler = StandardScaler()
features_values = np.asarray(features).astype('float32')
standardized_features = standardScaler.fit_transform(features_values)
oneHotEncoder = OneHotEncoder()
target_values = target.values.reshape(-1,  1)
encoded_target = onehot.fit_transform(target_values).toarray()
```

Code snippet 5. 3: Common Data Processing

Artificial Dataset Generation

For validating the work of the proposed network, an Artificial dataset has been generated as a dataset of controlled variables. The number of features and total instances are chosen from a range randomly. The number of features is set to be between 15 and 50, while the total instance is in the range of 10,000 to 100000 at 10000 intervals.

```
def return_coefficients(num):
    limiter = np.random.randint(num/2 + 1)
    coefficients = np.round(np.random.randn(num-limiter), 1)
    coefficients = np.hstack((coefficients, np.zeros(limiter)))
    np.random.shuffle(coefficients)
    return coefficients
num_of_instances = np.random.randint(1, 11) * 10000
num_of_features = np.random.randint(15, 51)
coefficients = return_coefficients(num_of_features)
fake_features = np.random.randn(num_of_instances, num_of_features)
fake_label = [np.sum(coefficients*np.array(args)) for args in fake_features]
fake_label = np.round(np.array(fake_label)).astype(int)
```

Code Snippet 5. 4: Artificial Dataset Generation Implementation

5.4. Implementation of Proposed Network

This section explains how the proposed network and a model based on the proposed network is implemented as described in the previous chapter. It has three subsections; Implementation of IXNet, Explainable Implementation, and Interpretable Implementation.

5.4.1. Implementation of IXNet

5.4.1.1. Implementation of Parts of Interpreter Components

The Interpreter component has two blocks mainly; the Attentive block and the Selector block. The implementation of each block is presented.

A. **Selector Block**

The Selector block has a batch normalization layer. It is implemented as follows.

```
class SelectorBlock(Layer):
   def __init__(self, momentum = 0.9, **kwargs):
       super(SelectorBlock, self).__init__(**kwargs)
       self.momentum = momentum
       self.bn = BatchNormalization(momentum=self.momentum)
   def call(self, x, y, training=None):
       return self.bn(tf.multiply(x, y))
```

Code Snippet 5. 5: Selector Block Implementation

B. **Attentive Block**

The Attentive block, incorporating multi-head self-attention, is implemented as follows.

```
class AttentiveBlock(Layer):
    def __init__(self, units=1, num_heads=2, act_type='softmax', loss_regularizor=False,
                 sparsity=None, momentum = 0.9, epsilon = 1e-5, **kwargs):
        super(AttentiveBlock, self).__init__(**kwargs)

        self.epsilon, self.units, self.num_heads, self.momentum = epsilon, units, num_heads, momentum
        self.act_type, self.sparsity, self.loss_regularizor = act_type, sparsity, loss_regularizor
        self.bn = BatchNormalization(momentum=self.momentum)
        self.multi_head_attention = [Attention(momentum = self.momentum, act_type=self.act_type,
                                               units=self.units, loss_regularizor=self.loss_regularizor
                                               sparsity=self.sparsity) for _ in range(self.num_heads)]
    def call(self, x, prior=None, training=None):
        attention_output_space = []
        for head in self.multi_head_attention:
            mask = head(x, prior=prior, training=training)
            attention_output_space.append(mask)
        pooled_mask = tf.reduce_mean(attention_output_space, axis=0)
        return self.bn(tf.convert_to_tensor(pooled_mask))
```

Code Snippet 5. 6: Attentive Block Implementation

C. **Attention Layer**

The attention heads in the multi-head self-attention of the Attentive are implemented as standalone Attention layers. Each head is implemented as follows.

```
class Attention(Layer):
   def __init__(self, units=1, act_type='softmax', momentum=0.9,
                loss_regularizor=False, sparsity=None, **kwargs):

       super(Attention, self).__init__(**kwargs)
       self.units, self.act_type, self.sparsity = units, act_type, sparsity
       self.loss_regularizor, self.momentum = loss_regularizor, momentum
       self.kernel1=self.add_weight(name='kernel1', initializer='glorot_uniform',
                                    shape=(input_shape[-1], self.units), trainable=True)
       self.kernel2=self.add_weight(name='kernel2', initializer='glorot_uniform',
                                    shape=(self.units, input_shape[-1]), trainable=True)
       self.bn = BatchNormalization(momentum=self.momentum)
       if self.loss_regularizor:
           self.regularization = MaskLossRegularizer(sparsity=self.sparsity)

   def call(self, x, prior=None, training=None):
       score = K.relu(K.dot(x, self.kernel1))
       score = K.tanh(K.dot(score, self.kernel2))
       score = self.bn(score)
       if self.act_type == 'softmax':
           mask = K.softmax(score * prior)
       else:
           mask = self.sparsemax(score * prior)
       if self.loss_regularizor:
           mask = self.regularization(mask)
       context = K.sum(x * mask, axis=0)
       return mask, context
```

Code Snippet 5. 7: Attention Layer Implementation

D. **Mask Loss Regularizer**

Each Attention layer has a built-in Custom Loss Regularizer layer. It is implemented as follows.

```
class MaskLossRegularizer(Layer):

    def __init__(self, sparsity=1e-2, epsilon = 1e-8, **kwargs):
        super(MaskLossRegularizer, self).__init__(**kwargs)
        self.sparsity = sparsity
        self.epsilon = epsilon

    def call(self, x, training=None):
        entropy = -x * tf.math.log(x + self.epsilon)
        mask_loss = tf.reduce_mean(tf.reduce_sum(entropy, axis=-1))
        self.add_loss(self.sparsity * mask_loss)

        return x
```

Code Snippet 5. 8: Mask Loss Regularizer Implementation

5.4.1.1.1. Implementation of Parts of Detector Components

The Detector component in turn has two blocks as well; Transformer and Classifier blocks.

A. Transformer Block

The Transformer block contains two Dense layers, a Dropout layer, a Batch Normalization, and a GLU layer. Its implementation is presented as follows.

```
class TransformerBlock(Layer):
    def __init__(self, units, momentum = 0.9, epsilon = 1e-5, dout_rate=0.2, **kwargs):
        super(TransformerBlock, self).__init__(**kwargs)
        self.units, self.momentum, self.epsilon, self.dout_rate = units, momentum, epsilon, dout_rate
        self.dense1 = Dense(self.units, activation='relu', kernel_regularizer=l2(0.001))
        self.dense2 = Dense(self.units, activation='relu', kernel_regularizer=l2(0.001))
        self.dropout = Dropout(self.dout_rate)
        self.bn = BatchNormalization(momentum=self.momentum)
        self.glu = GLU()
    def call(self, x, training=None):
        return  tf.nn.relu(self.glu(self.bn(self.dense2(self.dropout(self.dense1(x))))))
```

Code Snippet 5. 9: Transformer Block Implementation

B. GLU Layer

The GLU block basically computes the multiplication of input with its sigmoid value. It is implemented as follows.

```
class GLU(Layer):
    def __init__(self, sparsity=1e-2, epsilon = 1e-8, **kwargs):
        super(GLU, self).__init__(**kwargs)
    def call(self, x, training=None):
        return x * tf.nn.sigmoid(x)
```

Code Snippet 5. 10: GLU Layer Implementation

C. Classifier Block

Classifier is basically made up of a single Dense layer only. It selects the classification type based on the number of output classes. It is implemented as follows.

```
class ClassifierBlock(Layer):
    def __init__(self, num_classes, **kwargs):
        super(ClassifierBlock, self).__init__(**kwargs)
        self.num_classes = num_classes
        self.final_act_fun = 'sigmoid' if self.num_classes==1 else 'softmax'
        self.layer = Dense(self.num_classes, activation=self.final_act_fun,
                           use_bias=False, kernel_regularizer=l2(0.001))
    def call(self, x, training=None):
        return self.layer(x)
```

Code Snippet 5. 11: Classifier Block Implementation Code

5.4.1.3. Implementation of IXNet-based Model

A model based on the proposed network, IXNet, is implemented to test the genuineness of the architectural design. The number of steps of the IXNet-based model (IXNet model from here on) determines the number of Attentive, Selector, and Transformer blocks in which the models consist of. The implementation is as follows.

```
class IXNet(Model, BaseEstimator):

    def __init__(self, num_features, num_classes, Featurelist, num_heads=2, steps=4, output_dim=32,
                 att_units=1, act_type='softmax', loss_regularizor=False, sparsity=1e-2,
                 gamma=1.1, epsilon = 1e-8, dropout_rate=0.2, logdir='/', **kwargs):

        super(IXNet, self).__init__(**kwargs)
        self.num_features, self.steps, self.num_classes = num_features, steps, num_classes
        self.num_heads, self.sparsity, self.gamma, self.epsilon = num_heads, sparsity, gamma, epsilon
        self.dropout_rate, self.act_type, self.loss_regularizor = dropout_rate, act_type, loss_regularizor
        self.output_dim, self.att_units, self.Featurelist = att_units, Featurelist, output_dim
        self.feature_importance =  self.global_feature_importance = None
        self.writer = tf.summary.create_file_writer(self.logdir)
        self.bn = BatchNormalization()
        self.attention_blocks = [
                            AttentiveBlock(num_heads=self.num_heads, act_type=self.act_type, units=self.att_units,
                                           sparsity=self.sparsity, loss_regularizor=self.loss_regularizor)
                            for _ in range(self.steps)]
        self.selector_blocks = [SelectorBlock() for _ in range(self.steps)]
        self.transformer_blocks = [TransformerBlock(self.output_dim, self.dropout_rate) for _ in range(self.steps)]
        self.classifier = ClassifierBlock(self.num_classes)
```

```
    def call(self, x, training=None):
        masks = {}
        batch_size = tf.shape(x)[0]
        mask = tf.zeros([batch_size, self.num_features])
        prior = tf.ones([batch_size, self.num_features])
        output_aggregated = tf.zeros([batch_size, self.output_dim])
        feature_importance = tf.zeros([batch_size, self.num_features])
        with self.writer.as_default():
            x = masked_input = self.bn(x)
            for step in range(self.steps):
                mask = self.attention_blocks[step](x, prior)
                masked_input = self.selector_blocks[step](masked_input, mask)
                output = self.transformer_blocks[step](masked_input)
                output_aggregated += tf.nn.relu(output)
                prior *= (self.gamma - mask)
                masks[step] = mask
                step_importance = tf.reduce_sum(output, axis=1, keepdims=True)
                feature_importance += (mask * step_importance)
                tf.summary.image("Mask for step" + str(step), tf.expand_dims(
                        tf.expand_dims(mask, 0), 3), max_outputs=1, step=step)
            tf.summary.image("Feature Importance context", tf.expand_dims(
                    tf.expand_dims(feature_importance, 0), 3), max_outputs=1, step=0)
        y_pred = self.classifier(output_aggregated)
        return y_pred, masks, feature_importance
```

Code Snippet 5. 12: IXNet Model Call Method Implementation

5.4.2. Explainable Implementation

The Interpretable component of the proposed model performs a series of operations to trace the decision process. Then, the result of these operations is recorded and displayed using the following mentioned functionalities of the model.

I. ***Step:*** Implementing explainability of decisions at each step

```
def explain_step(self, x):

    _, masks, _, _ = self(x, training=False)
    fig, axs = plt.subplots(1, len(masks.keys()), figsize=(15, 20))
    for step, mask in masks.items():
        axs[step].imshow(mask[:feature_threshold]) // Step Local
        axs[step].set_title(f"{title}_{step}")

        mask_mean = tf.reduce_mean(mask, axis=0)
        mask_mean_norm = mask_mean /np.max(mask_mean)
        pd.DataFrame(mask_mean_norm,  columns=['Degree of Importance']).plot(kind='bar')
        plt.show()      // Step Global
```

Code Snippet 5. 13: Step Explainability Implementation

II. ***Local:*** Implementing explainability of decisions of sample/ group of samples

```
def explain_local(self, x, feature_threshold=None):

    _, _, feature_importance, _ = self(x, training=False)
    feature_importance /= np.sum(feature_importance, axis=1)[:, None]
    plt.matshow(feature_importance[: feature_threshold], cmap='viridis')
    plt.title('Local Feature Importances')
```

Code Snippet 5. 14: Local Explainability Implementation

III. ***Global:*** Implementing explainability of aggregated decisions of samples

```
def explain_global(self, x, feature_threshold=None):

    _, _, feature_importance, _ = self(x, training=False)
    feature_importances = np.sum(feature_importance, axis=0)
    global_feature_importances = feature_importances / np.sum(feature_importances)
    pd.DataFrame(global_feature_importances[: feature_threshold],
                 columns=['Degree of Importance']).plot(kind='bar', title=title)
    plt.show()
```

Code Snippet 5. 15: Global Explainability Implementation

5.4.3. Interpretable Implementation

The proposed model also has an interpretability feature both built-in and using a Callback class. This feature is implemented to track the learning process of IXNet on each epoch, batch, and step end; locally and globally.

A. Using Callback class

- Computing and displaying the relative importance learned at the training phase at the end of each epoch/batch.

```
def on_epoch_end(self, epoch, logs=None):
    weights = [weight for weight in self.model.trainable_variables
               if 'bias' not in weight.name and 'beta' not in weight.name
               and 'selector_block' not in weight.name and 'attentive_block' not in weight.name]
    weights, class_weight = weights[::-1], weights[0]
    bn_weight = K.expand_dims(weights[-1], axis=-1)
    importances = []
    for i in range(self.model.steps):
        trsf_bn_weight = K.expand_dims(weights[i+1], axis=-1)
        trsf_weight = weights[i+2]
        c_t_b = tf.multiply(trsf_bn_weight, class_weight)
        c_t = np.dot(trsf_weight, c_t_b)
        importances.append(tf.multiply(bn_weight, c_t))
    importance = np.array(tf.squeeze(tf.reduce_mean(importances, axis=0)))
    feature_importance = importance / np.sum(np.abs(importance))
    self.visualize(feature_importance, epoch)
```

Code Snippet 5. 16: Epoch-End Layer-wise Relevance Propagation Implementation

B. Using inherent methods

- The feature importance learned when fitting the model can be retrieved at the train step.

```
self.masks  = masks
self.feature_importance = feature_importance
sum_feature_importance = tf.reduce_sum(feature_importance, axis=0)
global_feature_importance = sum_feature_importance/tf.reduce_sum(sum_feature_importance)
self.global_feature_importance = global_feature_importance
```

Code Snippet 5. 17: Train Step Interpretability Implementation

- The instance-specific and global feature importance can also be estimated from the Attention layer weights, taken as alignment scores, at the Attention layer.

```
def get_attention_weight(self):
    return K.softmax(tf.linalg.diag(self.kernel2))
```

Code Snippet 5. 18: Attention Layer Interpretability Implementation

- Getting the Attention layer weights as alignment scores at the Attention layer to estimate the feature importance at Attentive block

```
def get_attention_weights(self):
    activated_weight_matrices = []
    for head in self.multi_head_attention:
        activated_weight_matrices.append(head.get_attention_weight())
    return np.stack(activated_weight_matrices, axis = 0)
def get_mean_attention_weights(self):
    return K.reduce_mean(self.get_attention_weight(), axis = 0)
```

Code Snippet 5. 19: Attentive block Interpretability Implementation

5.4.4. Presentation Subcomponent Implementation

The Presentation subcomponent is implemented using the Flask app server**.** It accepts feature importance and a list of corresponding features to visualize the learning progress of the model.

- The model-side (client) code of the Presentation subcomponent is implemented as follows.

```
def visualize(self, feature_importance, epoch):
    headers = {"content-type": "application/json"}
    data = json.dumps({"feature_importance": feature_importance.tolist(),
                       "Featurelist": self.Featurelist, 'epoch': epoch})
    json_response = requests.post(self.url+'/visualize',data=data,headers=headers)
```

Code Snippet 5. 20: Model-side Presentation Subcomponent Implementation

- The server-side code of the Presentation subcomponent is implemented as follows.

```
app = Flask(__name__)
run_with_ngrok(app)
@app.route("/visualize",methods=['POST'])
def home():
    data = request.json
    plot_importance(data['feature_importance'], data['Featurelist'], data['epoch'], data['batch'])
    return json.dumps("Response": "OK")
app.run()
def plot_importance(feature_importances, Featurelist, epoch=0, batch=0, title='Feature Importance'):
    feature_importances = np.array(feature_importances)
    if feature_importances.ndim == 1:
        feature_importances = tf.expand_dims(feature_importances, axis=-1)
    for iter, feature_importance in enumerate(tf.transpose(feature_importances)):
        plt.bar(np.arange(feature_importance.shape[-1]), feature_importance, label='Class ' + str(iter+1))
    plt.ylabel('Degree of Importance')
    plt.xlabel('Features')
    plt.title(f'Feature Importance at Epoch {epoch+1} Batch {batch+1}')
    plt.xticks(np.arange(0, len(Featurelist)), Featurelist, rotation=90)
    plt.legend(loc="upper left")
    plt.show()
```

Code Snippet 5. 21: Server-side Presentation Subcomponent Implementation

5.5. Hyperparameter Selection and Tuning

5.5.1. Hyperparameter Selection

Hyperparameters are adjustable factors of a machine learning model that have unmediated repercussions on its learning process. Often, they are set before the start of the training phase and remain unchanged. In some cases, hyperparameters can be gradually changed depending on the training performance of the model. Some common hyperparameters include epochs, batch size, optimizer, learning rate, metrics, and loss functions.

- **Learning rate**: the scale learnable parameters are updated during backpropagation.
- **Epoch**: the number of times the algorithm runs on the whole training dataset.
- **Batch**: the size of a subset of input data sample fed to a model at once.
- **Cost /Loss Function**: calculate the difference between predicted and actual value.
- **Optimizers**: modify the neural network's properties to reduce overall losses and improve accuracy.

5.5.2. Hyperparameter Tuning

The hyperparameters used are tuned using the **GridSearchCV** algorithm. Sklearn's model_selection module provides this algorithm mainly for employing a brute force search over predetermined sets of parameter values. The "param_grid" variable used for this algorithm includes learning rate, sparsity, gamma, and epsilon.

```
param_grid = {
    'gamma': [1.2, 1.3],
    'eps': [1e-4, 1e-5, 1e-6, 1e-8],
    'sparsity': [1e-2, 1e-3, 1e-4, 1e-5],
    'learning_rate': [0.02, 0.05, 0.002, 0.005],
    'dropout_rate': [0.1, 0.2, 0.3, 0.4, 0.5],
}
grid_search = GridSearchCV(estimator = model, param_grid=param_grid,
                           scoring='roc_auc', cv=10, verbose=0)
grid_search.fit(x_train, y_train)
best_params = grid_search.best_params_
```

Code Snippet 5. 22: Hyperparameter Tuning using Grid Search

5.6. Experimentation Setup

This part of the experiment uses two datasets that are composed of features from different analysis techniques. The classification type is binary(detection) This diversity is useful to create a model capable of performing in a variety of possible scenarios.

Experiment 1: Searching for Optimal *IXNet* model

- ❖ Using the proposed model and two datasets

Table 5. 2: List of Experiment of Class-1

Experiment	Hyperparameter	Experimentation Values
Exp 1.1	Hyperparameter Tuning	Hyperparameters as specified in (5.5.2)
Exp 1.2	Batch size	100, 500, 1000
	Optimizer	Adam, Nadam, SGD, RMSProp
	Transformer Neuron Units	8, 16, 32, 64
	Attention Neuron Units	1, 8, 16, 32, 64
	Number of Attention Heads	2, 3, 4, 5
	Number of steps	2, 3, 4, 5
	Activation Function	Softmax, Sparsemax
	Train-Test Split Ratio	60:10:30, 70:10:20, 70:15:15

Experiment 2: Evaluation of *IXNet* model on two Datasets

This experiment tests the performance of IXNet on two datasets using K-Fold cross-validation.

Experiment 3: Comparison of *IXNet* model with Different Models and Datasets

This experiment was done to compare the performance of the model with other models.

Experiment 4: Evaluation of Transfer Learning on *IXNet* model

This experiment evaluated the impact of using transfer learning on the proposed network. A shared layer is employed to achieve transfer learning in IXNet.

Experiment 5: Evaluation and Comparison of Interpretation of *IXNet* model

This experiment evaluates the interpretability of IXNet, and also compares it with other models.

CHAPTER SIX

RESULTS AND DISCUSSIONS

6.1. Chapter Overview

In this chapter, the results of the data transformation and other different experiments carried out with the guideline stated in the previous chapter are discussed. The results are also analyzed in detail to highlight findings and how these findings are used to answer the research questions.

6.2. Results of the Study

The results from the data transformation and the implementation of the proposed model are discussed in detail in the subsequent subsections. These subsections include data transformation, experimentation, and evaluation results.

6.2.1. Data Transformation Results

Compared to datasets with a larger number of columns, the process of loading, cleaning, and training datasets with a larger number of instances (rows), is easier and less time and resources consuming. Some datasets used in this study appeared to have a large number of columns increasing the challenge.

- **Drebin:** the samples with invalid values were cleaned. The final dataset contains 15036 rows and 216 columns, 215 dependent variables, and one target variable with two classes.
- **MalMem:** the dataset is balanced and clean. The data type of all of the independent variable's data is nominal. The resulting dataset contains 58596 rows and 56 columns; 55 dependent variables and one target variable with two classes.

The rest of the data preprocessing techniques have been applied similarly for both datasets. The datasets first are shuffled and then partitioned into Features and Label (Target). The Features are scaled using StandardScaler standardization, while the Targets were encoded using One-Hot encoding as discussed in section 5.3.2.

6.2.2. Experimentation Results

For this study, two classes of quantitative experiments were carried out to evaluate the proposed model and compared it with other models. It is done in two parts; classification and interpretation with two datasets of Binary classification (Detection). The time taken in most of the experiments was a bit tricky to be taken as a criterion to make a comparison. Sometimes the Colab-Notebook took its own time before starting the training. Therefore, less emphasis was given to the time it took to finish experiments. The experiments were done on each compatible dataset while keeping the environmental setup similar for comparison. The results from the different experimental classes are discussed as follows.

6.2.2.1. Experiment 1: Searching for Optimal IXNet model

This part of the experiment was conducted to find the best combination of hyperparameters for the proposed model. Here some of the hyperparameters were tuned using the Grid Search algorithm as discussed in the previous chapter. The best approaches for setting the hyperparameters were done in the following experiments. All of these experiments were conducted with an epoch of 50 on the Drebin dataset.

Exp 1.1: Hyperparameter Tuning

This experiment was done using a Grid Search algorithm for tuning some of the hyperparameters of the IXNet-based model. From the result, it is clear that the model is optimal when trained with values of gamma value of 0.3, a learning rate of 0.02, a dropout rate of 0.3, and epsilon and sparsity values of e^{-8} and e^{-2}.

```
best_params = grid_search.best_params_
print('Gamma =', best_params['gamma'])
print('Epsilon =', best_params['epsilon'])
print('Sparsity =', best_params['sparsity'])
print('Learning Rate =', best_params['dropout_rate'])
print('Dropout Rate =', best_params['learning_rate'])
```

```
Gamma = 1.1
Epsilon = 1e-08
Sparsity = 0.01
Learning Rate = 0.3
Dropout Rate = 0.02
```

Hyperparameter	Value
Learning rate	0.02
Dropout rate	0.3
Gamma	1.1
Epsilon	1e-8
Sparsity	1e-2

Figure 6. 1: Hyperparameter Tuning Result

Exp1.2: Optimal Hyperparameters

In this experiment class, the search for an optimal model based on the proposed network was conducted, and hyperparameters needed to optimize the proposed model were found. From the result analysis, it can be deduced that increasing the batch size has been shown to increase the accuracy and the training time while decreasing the loss. With a batch size of one thousand, the model performed very well achieving the highest accuracy and lower loss almost on all occasions. As for the candidate optimizers, Adam, Nadam, and RMSProp have shown to perform very well while SGD seems the loser of this race as it performs badly almost in all criteria. RMSProp is incurred to be the best option of the four.

An experiment was also done to find the optimal number of neurons of the fully connected layer in the Transformer block. From the result, sixteen (16) neuron units have given better results while using twice as much time as taken by eight (8) units. From the curve graph, it can be concluded that increasing the number of neurons introduces a sudden spike in the loss curve. This is a sign that overfitting is imminent and sixteen (16) is the go-to number for this model. The number of neuron units in the attention layer is used for the hidden representation of the feature values for feature masking and attribution. This experiment shows that increasing these units decreases the loss significantly while maintaining good accuracy. From the result, it is hard to choose the best option. But with the help of the Accuracy-Loss graph, eight (16) units of a neuron is found to be a suitable choice as it produces a more stable curve.

The number of attention heads in the Attentive layer is a crucial part of the proposed model. It regulates the attentive process by conducting a fair voting mechanism. The result shows that increasing the number of attention heads increases the training and validation accuracy of the model. But the respective losses also increase, indicating a possible overfitting problem. Moreover, the training time increases radically. From the results of the test accuracies, a safe option is choosing 3-5 heads for the model as it achieved a good accuracy while maintaining a small loss in less computation time. The choice of the activation function to be used in the Attentive block can have profound implications as the nature of Sparsemax and Softmax differs. The usage of Sparsemax is appropriate for harder attention, which is masking the much smaller values while giving higher values more attention. The Softmax function gives some value however insignificant their relevance. The curve, figure B.6 in Appendix B.2, shows that both

perform great; although Sparsemax presents a great accuracy, Softmax gives the model smoother training.

The step of the model is one of the necessary hyperparameters that must be analyzed carefully. They are bound to be different for different purposes. Although, for this particular case, IXNet with two (2) steps performed very well compared to the rest. IXNet model with three (3) steps is a close second and a comparable alternative for higher accuracy. An experiment was done to find the best ratio to split the datasets into training and testing datasets. According to the results obtained, splitting the dataset into 70% training, 15% validation and 15% testing dataset produces the best accuracy while minimizing the training and validation losses decently compared to the rest.

From the experiment, it was observed that the use of a custom loss regularizer increases the initial error value. This is due to the custom loss adding its own loss value to the model's computed loss increasing the total loss. It takes time to reduce the total loss to the level equivalent to the normal loss value. When the result Loss curve is observed, the slope of the IXNet loss curve is very steep compared to the others. This shows that the model can lower the loss by increasing the epoch.

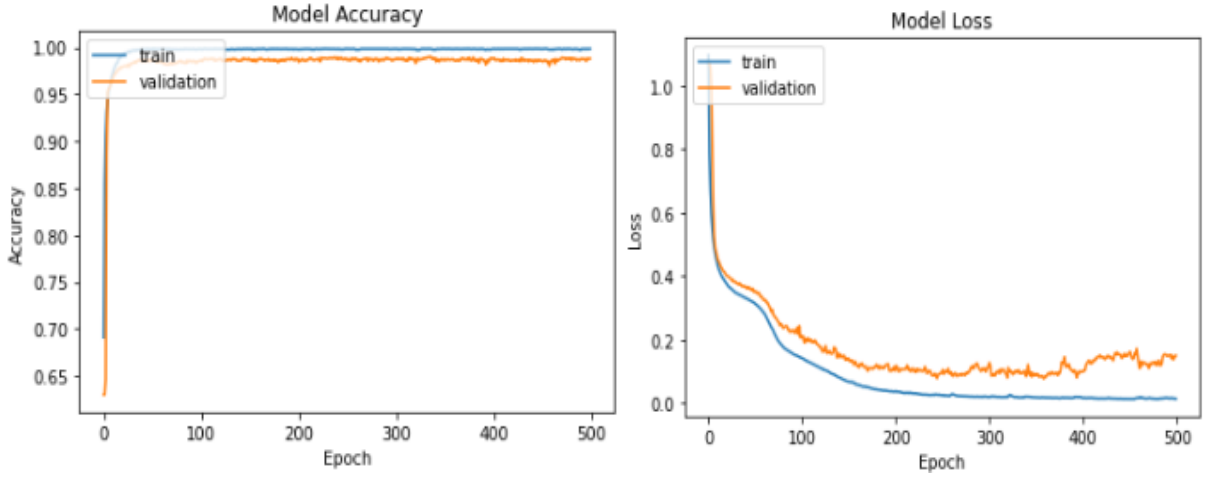


Figure 6. 2: Loss and Accuracy curve at higher epoch

The results from this experiment class are summarized in the table 6.1.

Table 6. 1: Result of Optimal Hyperparameters search

Hyperparameters		**Validation**		**Test**	**Hyperparameters**		**Validation**		**Test**
Criterion	**Options**	**Accuracy**	**Loss**	**Accuracy**	**Criterion**	**Options**	**Accuracy**	**Loss**	**Accuracy**
Batch Size	100	0.9818	0.4285	0.9832	Optimizer	Adam	0.9840	0.3685	**0.9898**
	500	**0.9863**	0.3974	0.9827		Nadam	0.9845	0.3058	0.9885
	750	0.9837	0.3770	0.9840		SGD	0.9570	0.6481	0.9592
	1000	0.9840	**0.3685**	**0.9898**		RMSProp	**0.9858**	**0.2840**	0.9849
Transformer Neuron Units	8	0.9831	0.3309	0.9854	Attention Neuron Units	8	0.9823	0.3655	**0.9885**
	16	**0.9867**	**0.2398**	**0.9880**		16	0.9805	0.1992	0.9849
	32	0.9809	0.2603	0.9863		32	**0.9827**	0.3113	0.9854
	64	0.9858	0.2840	0.9849		64	0.9814	**0.1619**	0.9849
Number of Attention Heads	2	0.9792	**0.2854**	0.9845	Steps	2	**0.9854**	**0.2586**	0.9871
	3	0.9823	0.3655	**0.9885**		3	0.9845	0.3203	**0.9889**
	4	0.9854	0.3282	**0.9885**		4	0.9818	0.3638	0.9863
	5	0.9849	0.3597	0.9854		5	0.9849	0.3740	0.9858
	10	**0.9858**	0.5837	0.9867	Train-Test Split Ratio	60:10:30	0.9758	0.4591	0.9816
Activation	Softmax	0.9849	0.2728	**0.9894**		70:10:20	0.9805	0.3573	0.9837
	Sparsemax	**0.9858**	**0.2521**	0.9889		70:15:15	**0.9881**	**0.3037**	**0.9894**

6.2.2.2. Experiment 2: Evaluation of IXNet model on Two datasets

The proposed model, IXAM, was tested using two datasets with the parameters set up as described in table 6.2.

Table 6. 2: IXNet Hyperparameter Setup

Hyperparameter	Value	Hyperparameter	Value	Hyperparameter	Value
Learning rate	0.02	Batch Size	1000	Split Ratio	80:5:15
Dropout rate	0.3	Transformer Unit	16	Optimizer	RMSProp
Gamma	1.1	Attention Unit	8	Activation	Softmax
Epsilon	1e-8	Attention Head	5	Loss Regularizer	Custom
Sparsity	1e-2	Steps	3		

Exp 2.1: IXNet Model on Drebin

In this experiment, the performance of the proposed model was evaluated using the Drebin dataset. The result shows that the model performed well with a 98.9 % F1-score and 98.89% accuracy. It also showed impressive results with a 1.56% false positive rate and a 0.84% false negative rate in just 50 iterations. This ensures the reliability of the model in high-stake sectors.

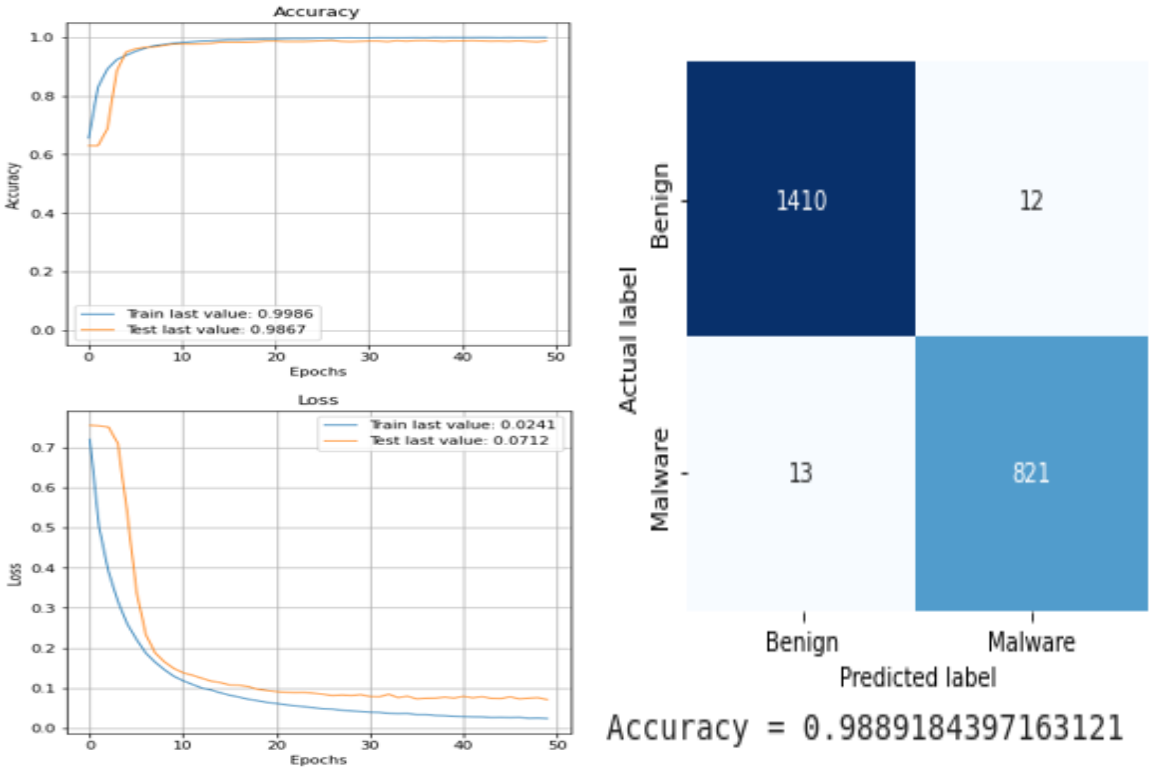


Figure 6. 3: Result of IXNet on Drebin

Exp 2.2: IXNet Model on MalMem

In this part experiment, the performance of the proposed model was tested using the MalMem dataset. The result shows that the model performed better than it did in the Drebin dataset. Here, it scored an accuracy of 99.98% with a 99.97% F1-score. The model also classified the test sample with an insignificant number of false alerts (both false positive and false negative). This comes with no surprise as the dataset was developed with high quality.

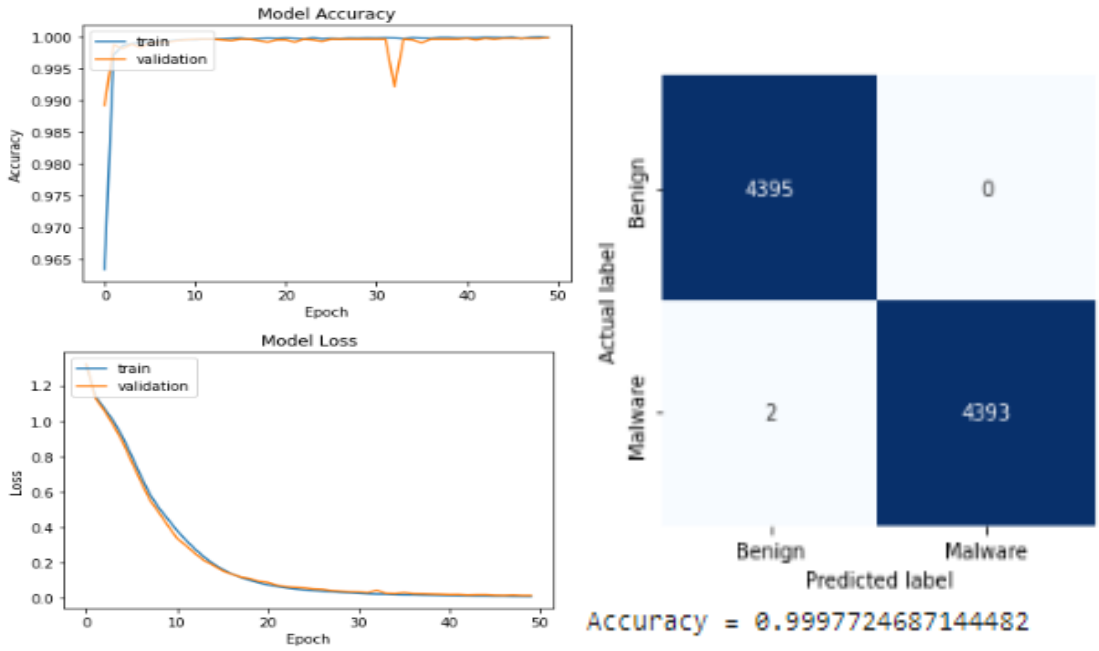


Figure 6. 4: Result of IXNet on MalMem

From the results, it is fair to say that IXNet performs well on both Drebin and MalMem datasets. It reaches optimal accuracy within the first 10 iterations for both datasets. Moreover, it doesn't show any sign of overfitting. The result showed how accurately the proposed model can detect android malware. The result is summarized in the table below.

Table 6. 3: Result of IXNet on two Datasets

Datasets	Validation		Test Acc	Precision	Recall	F1-Score	FPR	FNR	Time
	Acc	Loss							
Drebin	0.9867	0.0712	0.9889	0.9910	0.9916	0.9890	0.0156	0.0084	129.448
MalMem	0.9999	0.0154	0.9998	0.9995	1.0	0.9997	0.0005	0	91.675

6.2.2.3. Experiment 3: Comparison of IXNet with Different Models

This part of the experiment was conducted to find the comparative performance of the proposed model relative to other models. The number of epochs used was 50.

Exp 3.1: Different Models on Drebin

This experiment was done on the Drebin dataset, the benchmark dataset for Android malware detection. IXNet's performance outweighs the rest almost in every criterion with the usage of custom loss seeming to reduce the accuracy and increase the loss. But the slope of the loss graph of these models is very steep, an indication that it can significantly reduce the loss with some more iterations. The result from the evaluation metrics also shows the superior performance of the proposed model compared to the three models. It has a very small false positive and false negative rate. This is very important in Android malware detection. The confusion matrix of each experiment is found in table B.1 in Appendix B.3 section.

- *Drebin Loss Accuracy Train-Validation Curve*

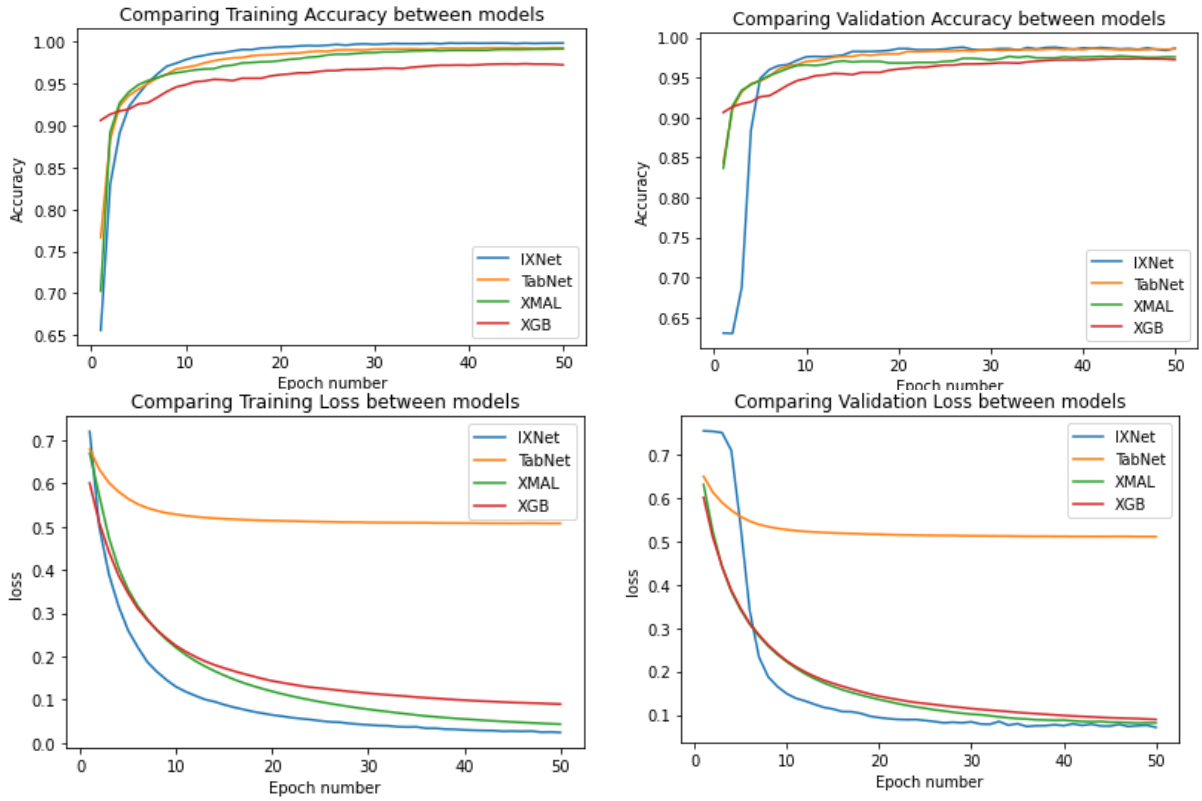


Figure 6. 5: Accuracy(upper)-Loss(lower) Train(left)-Validation(right) curves on Drebin

Exp 3.2: Different Models on MalMem

As observed from experiment 2.1, the result analyzed from the experiment on the MalMem dataset is almost identical to that of the Drebin. The proposed model performed slightly better than XGBoost and significantly better than the TabNet model. TabNet model appears to be less stable, while IXNet keeps its composure at higher epochs. From the objective evaluation as well, IXNet is observed to perform better than the rest, Although XGBoost is a close second. The confusion matrix of each experiment is found in table B.1 in Appendix B.3 section.

Table 6. 4: Result of Exp 2.3 Model Comparison on Drebin

- *MalMem Accuracy Train-Validation Curve*

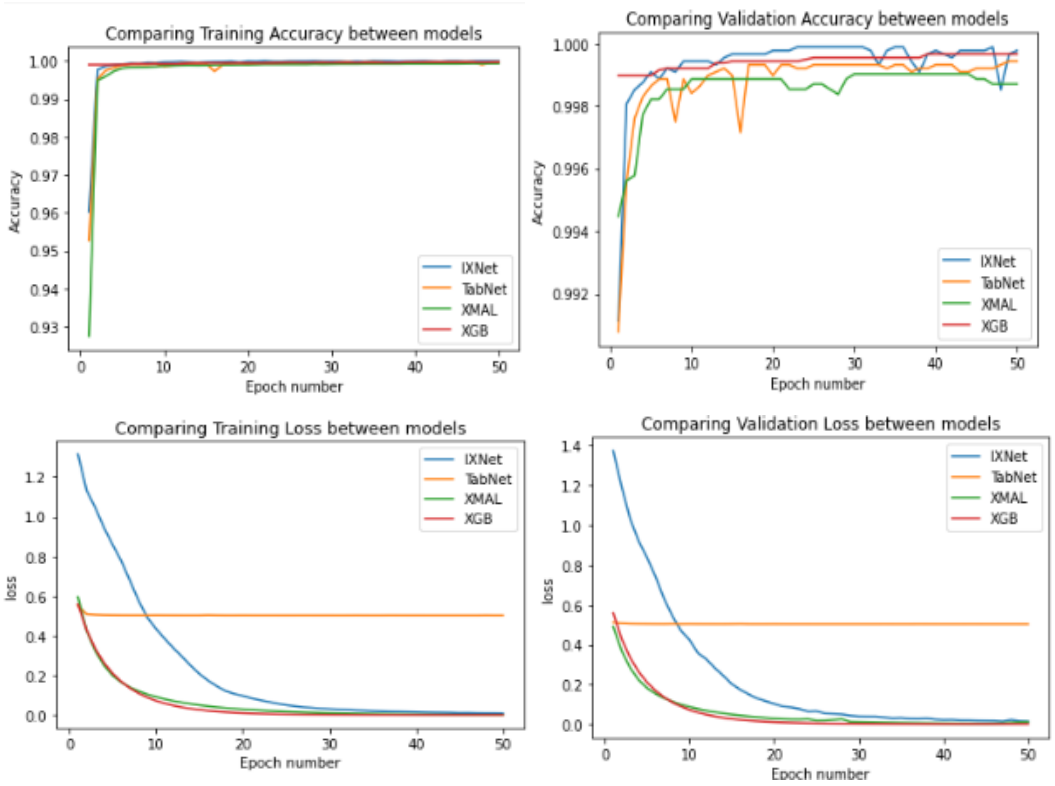


Figure 6. 6: Accuracy(upper)-Loss(lower) Train(left)-Validation(right) curves on MalMem

This part of the experiment was done to compare IXNet with XGBoost, Xmal, and TabNet. The result shows that the proposed model performs comparably with XGBoost, even superior in accuracy and loss in some cases, although IXNet appears to take more time compared to its counterparts in most cases. Here, the baseline network TabNet seems to perform worst compared to the rest.

❖ The overall comparative result is summarized and presented in the table below.

Table 6. 5: Comparison of IXNet with other models

Dataset	**Model**	**Validation Accuracy**	**Validation Loss**	**Test Accuracy**	**Precision**	**Recall**	**F1-Score**	**FPR**	**FNR**	**Time**
Drebin	XGBoost	0.9721	0.0894	0.9774	0.9910	0.9737	0.9823	0.0167	0.0263	**15.636**
	XMAL	0.9759	0.0815	0.9774	0.9784	0.9859	0.9821	0.0372	0.0141	21.754
	TabNet	0.9858	0.5109	0.9734	0.9703	0.9880	0.9791	0.0515	**0.0120**	149.117
	IXNet	**0.9867**	**0.0712**	**0.9889**	**0.9910**	**0.9916**	**0.9890**	**0.0156**	0.0084	129.448
MalMem	XGBoost	0.9997	**0.0034**	0.9997	**0.9998**	0.9995	0.9996	**0.0002**	0.0005	**11.259**
	Xmal	0.9987	0.0123	0.9981	0.9980	0.9982	0.9981	0.0020	0.0018	26.015
	TabNet	0.9994	0.5035	0.9982	0.9980	0.9984	0.9982	0.0020	0.0016	78.444
	IXNet	**0.9999**	0.0154	**0.9998**	0.9995	**1.0**	**0.9997**	0.0005	**0**	91.675

6.2.2.4. Result of Experiment of Transfer Learning

An experiment was done to test the effect of transfer learning with a shared layer. The result showed that the use of a shared layer to transfer knowledge in IXNet degrades the performance.

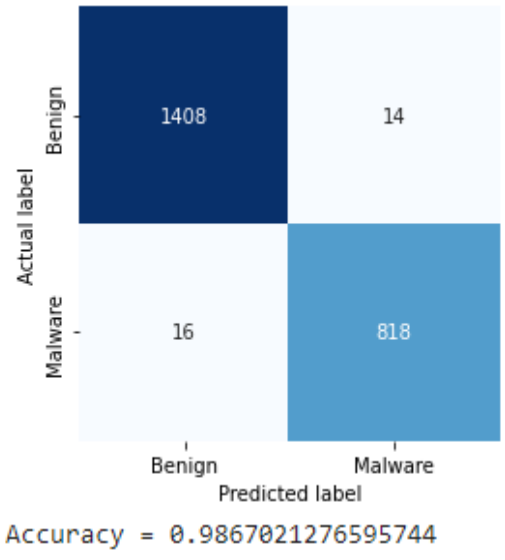
Table 6. 6: Impact of Transfer Learning on IXNet Performance

Mode	Train		Validation		Test	Time
Transfer Learning	Accuracy	Loss	Accuracy	Loss	Accuracy	
No	**0.9984**	**0.1822**	**0.9878**	**0.2266**	**0.9867**	130.134
Yes	0.9921	0.3083	0.9849	0.3555	0.9823	**127.927**

Table 6. 7: Evaluation of Impact of Transfer Learning on IXNet Performance

Transfer Learning	Accuracy	Precision	Recall	F1-Score	FPR	FNR
No	**0.9867**	**0.9888**	**0.9902**	**0.9878**	**0.0192**	**0.0098**
Yes	0.9823	0.9832	0.9887	0.9859	0.0288	0.0113

- With Transfer Learning (Shared Layer)



- Without Transfer Learning (Shared Layer)

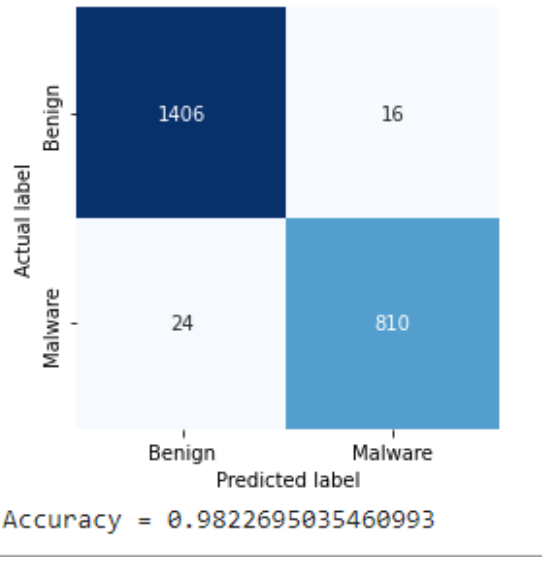


Figure 6. 7: Confusion matrics of Transfer Learning Experiment

6.2.2.5. Result of Interpretation Experiment

This study mainly focuses on the interpretation of deep learning networks. In this part of the experiment, the interpretability and explainability of the proposed IXNet model are presented.

6.2.2.4.1. Explainability of IXNet

The model, as indicated in the design, provides step, local and global explainability. 50 samples were given for the explainability experiments. The result images show that the masking process is more instance-based at every step. The local explanation presents the prominent features that contribute to the detection decision for each sample individually, while the global shows the overall prominence of features across the sample batch.

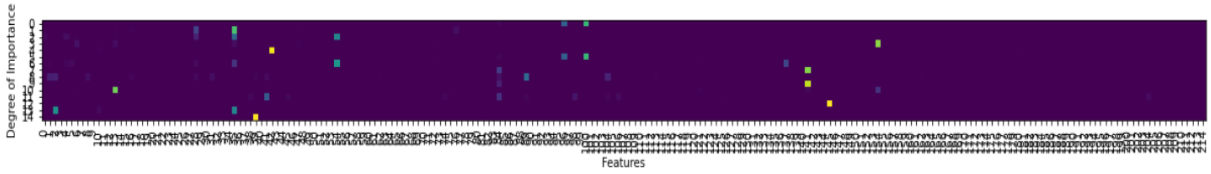


Figure 6. 8: IXNet Local Explainability result

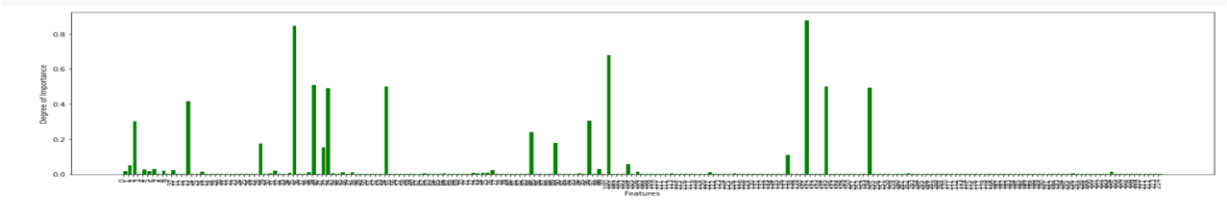


Figure 6. 9: IXNet Global Explainability result

The proposed model also provides step-wise mask explainability to the tested sample. This can be used to determine which features are given importance and which are deprived when the model makes its decision at each step.

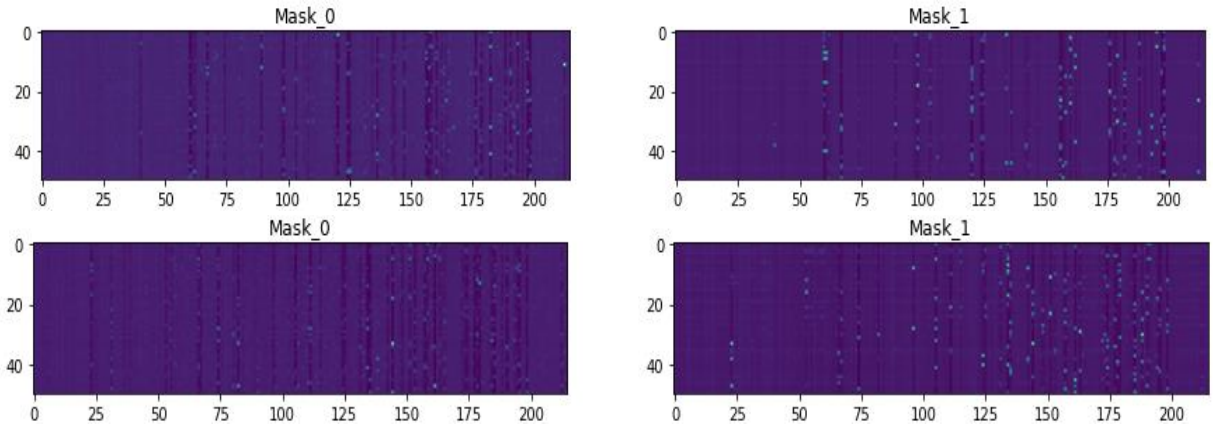
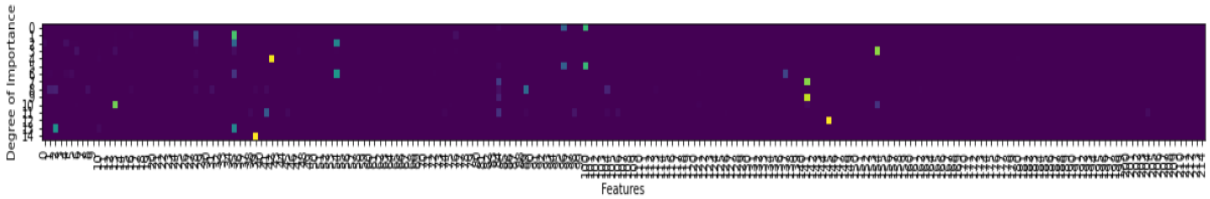


Figure 6. 10: IXNet Step Explainability result

6.2.2.4.2. Explainability Comparison

This part of the experiment was conducted to find the relative explainability of the proposed model IXNet in comparison with TabNet and Xmal. Here also, for comparison purposes, 50 samples were given for the explainability experiments. The results clearly show how well the proposed approach provides explanation. Compared to TabNet, the local explanation is instance-specific. For this particular case, TabNet seems not to employ the instance-based feature selection, whereas IXNet appears to do so.

- *IXNet*



- *TabNet*

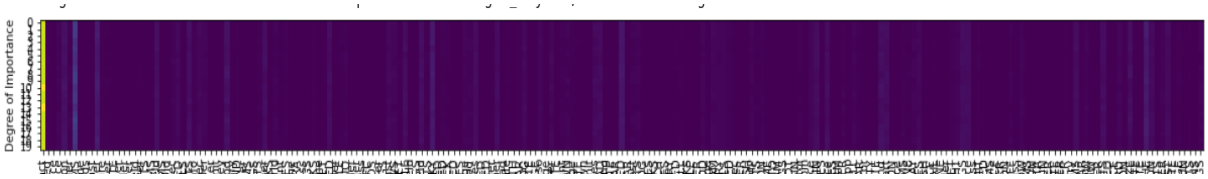
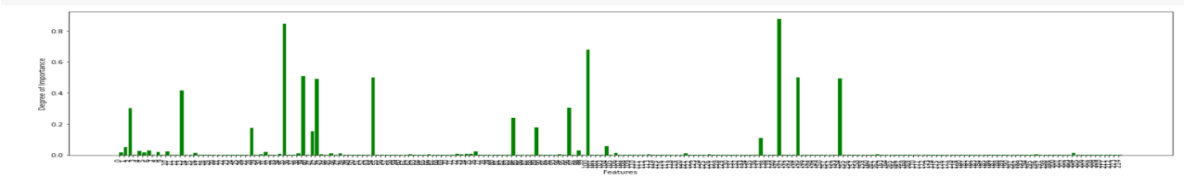


Figure 6. 11: Local Explainability Comparison

For the case of Global explainability, the importance of other features is greatly deprived in the TabNet explanation. The explanation given by the IXNet is more standardized. They are scaled in relative to each other showing a more relevant abstraction of reality. The below figures show the sample output of each model.

- *IXNet*



- *TabNet*

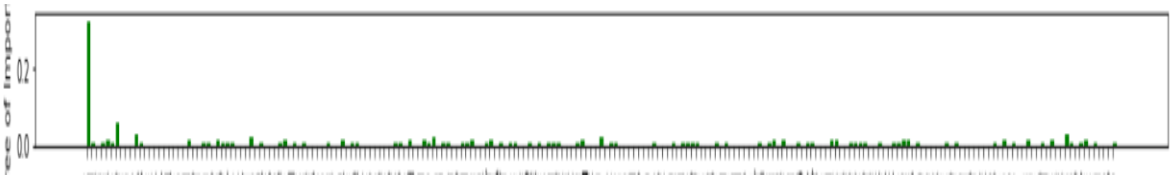


Figure 6. 12: Global Explainability Comparison between TabNet and IXNet

As for the feature importance, the explanation given by the Xmal model is unclear and ambiguous. More importantly, the direct or inverse relationship between the features and the target class is not presented in Xmal. IXNet, on the other hand, presents the relative magnitude of each feature in relation to the target. The relative magnitude is also given excitatory or inhibitory signed values to indicate the relationship of features as seen from the prediction. The below figures show the comparison between the output of the two models.

- **IXNet**
- **Xmal**

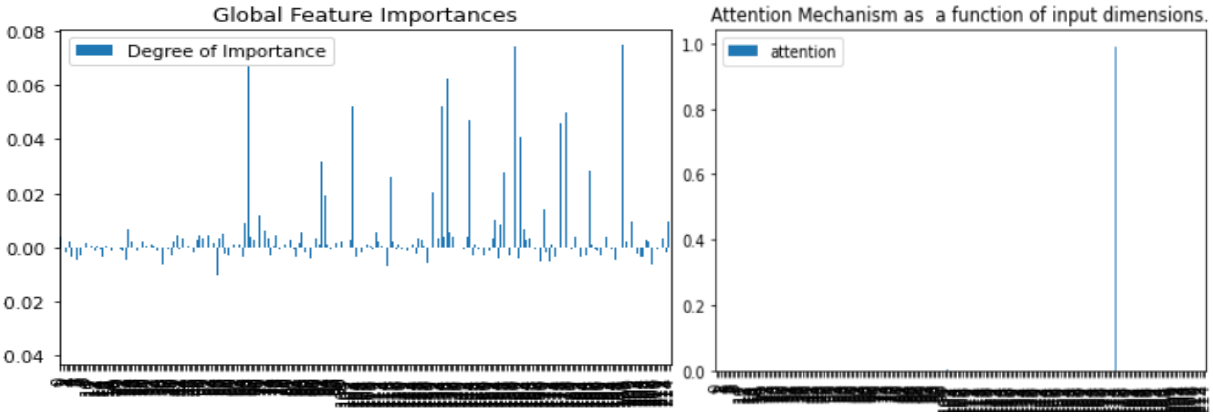


Figure 6. 13: Global Feature Importance Comparison between Xmal and IXNet

6.2.2.4.3. Interpretability of IXNet

The result from the images of Tensorboard, on the other hand, also displays the result of the masking process at every step iteration. It is also compatible with the step explainability.

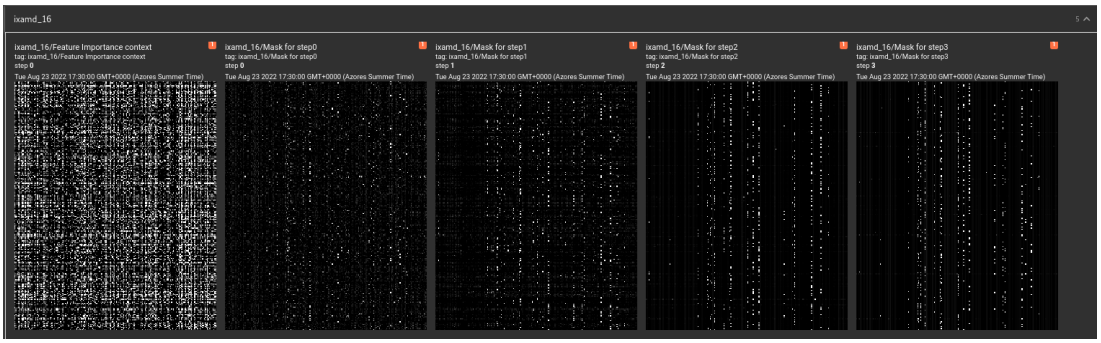


Figure 6. 14: IXNet Training Step-wise Interpretability

The inherent interpretability provided by the model presented the step-wise learning process of the model during the training phase. It records the progress of the model's feature selection phenomena at each step in Tensorboard. This is useful to debug and monitor the decision process of the model as it appears to be quantified at each stage to be validated and verified.

6.2.2.4.4. Interpretability of Learning process of IXNet

As for the experiment of interpretability of the learning process, the first, experiment was done on an artificial dataset to provide a controlled environment for cross-checking the result and validating the algorithm mentioned in section 4.3. The artificial dataset was generated using NumPy's random library to provide random coefficients (both positive and negative) and the input values to compute the corresponding output values. The output was then binarized with respect to the mean value across the output vector space to give an output of either 0 or 1. Then, the artificially generated dataset was fed to the proposed model, and its output was monitored using the coefficient values as a reference. This process was repeated until the proposed algorithm to compute the feature importance (coefficients in this case) was verified to be accurate and efficient.

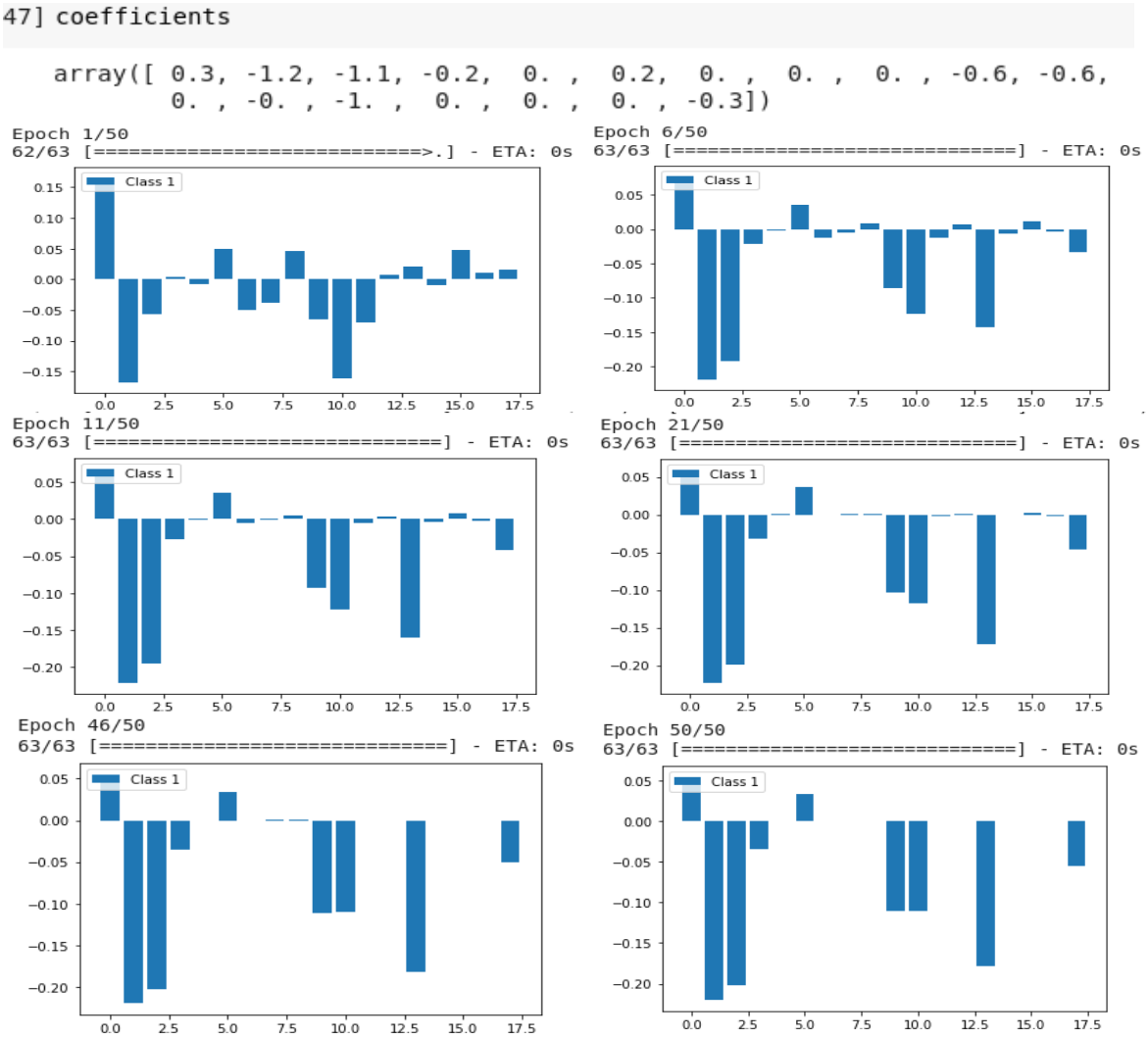


Figure 6. 15: IXNet Training Interpretability with Artificial Dataset

After the tuning procedure was finished, another experiment was done on the Drebin dataset with label encoding and one-hot encoding. The label encoding was done to present the learned importance of the features directly related to the inputs themselves, whereas the one-hot encoding was done to present the learned importance of the features on either side of positive and negative detection.

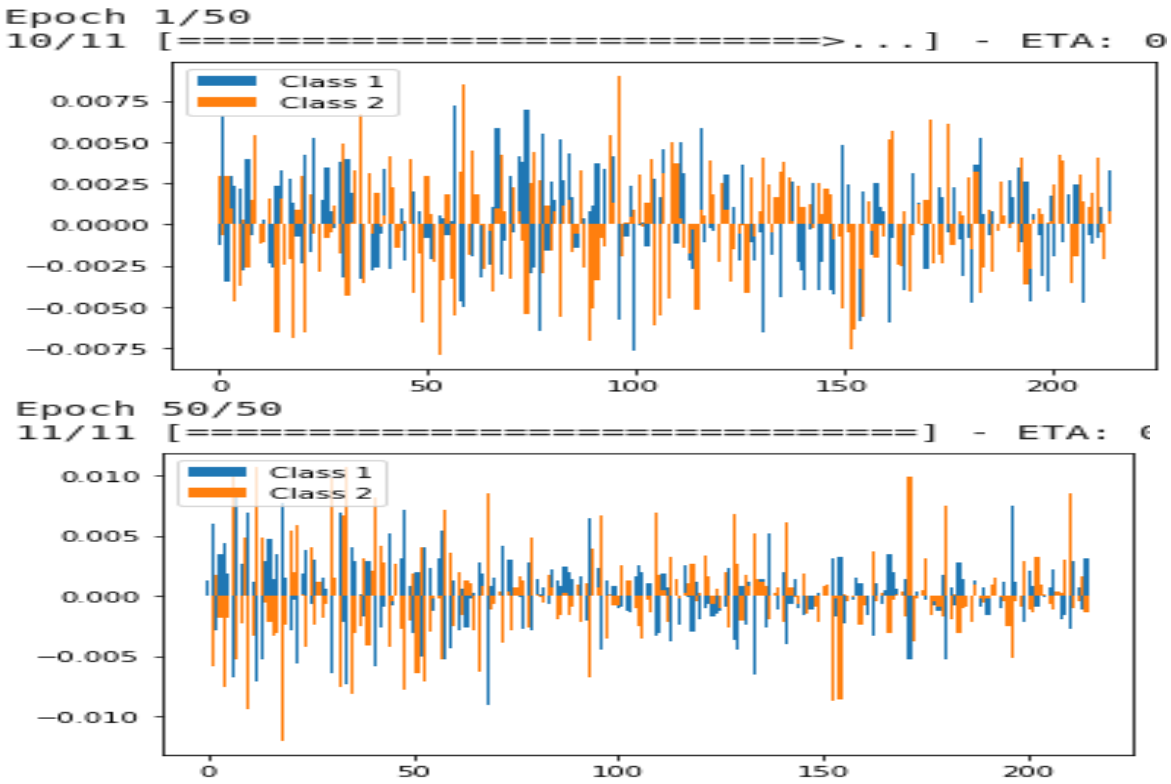


Figure 6. 16: IXNet Training Interpretability with Drebin (One-Hot encoding)

The result shows that the callback implementation presents a learning curve compatible with the global explainability of the model, which is a good sign as it actually displays the concepts grasped during the training phase. The proposed models were evaluated with and without the learning process interpretation (use of callback class method). The obtained comparative result is stated in the table below.

Table 6. 8: Impact of Callback implementation of Interpretability on IXNet Performance

Mode	**Train**		**Validation**		**Test**	Time
Interpretability	**Accuracy**	**Loss**	**Accuracy**	**Loss**	**Accuracy**	
No	0.9935	0.1177	0.9890	0.1263	0.9858	**98.211**
Yes	**0.9940**	**0.1138**	**0.9905**	**0.1202**	**0.9889**	102.634

Furthermore, from the result, as seen in table 6.8, it is clear that the interpretation of the learning process using the Callback function has little impact on the training time, while still maintaining the accuracy level of the model, perhaps, even improving it significantly.

6.2.2.3.5. Subjective Evaluation on Interpretation

The interpretability of the proposed IXNet model has been evaluated by Malware security experts working at INSA. For comparison purposes, the result from the other models has been included in the evaluation. The subjective evaluation has been carried out using a questionnaire that uses certain criteria in which the proposed model and the rest are compared. The questionnaire provides the interpretation of 10 random samples from test data to the experts and the criteria to evaluate them. Only the top 15 salient features are included in the questionnaire to provide evaluators with more grasp of the scenarios. The criteria include:

- ***Soundness***: the validity of detection and explanation.
- ***Sensibility***: the coherency of explanation with the result.
- ***Comprehensibility***: the degree to which the explanations are understood by experts.
- ***Description***: the clearness of the delivery of interpretation.

The measuring scale was from 1 up to 10. The informativeness of IXNet and its acceptance relative to the other interpretability models is evaluated as follows.

Table 6. 9: The Subjective Evaluation of IXNet Model

Testers	Soundness (%)	Sensibility (%)	Comprehension (%)	Descriptiveness (%)	Average (%)
Evaluator-1	74	71.5	53	49	61.875
Evaluator-2	78.5	70	66	57.5	68
Evaluator-3	83.5	82.5	75.5	45	71.625
Evaluator-4	83	76	71.5	61	72.875
Evaluator-5	79	75.5	78	71.5	76
Evaluator-6	89	86.5	80.5	47.5	75.875
Average	81.5	77	70.75	55.25	71.042

The subjective evaluation of the IXNet model shows positive feedback from the experts. The evaluators rated the Soundness and Sensibility of the detection result and explanations click more often than they did not, but they feel like there should be a more presentable way to address the interpretation, like NLG and animations. They also rated the explainability of the proposed model with the result from TabNet and Xmal. The result here also shows that the proposed model is superior to the rest in the case of explainability. Adding to the effect, they even gave the proposed model a better rating than the previous standalone evaluation. Xmal on other hand has received a significantly lower rate.

Table 6. 10: The Relative Subjective Evaluation of IXNet Model

Model	Soundness (%)	Sensibility (%)	Comprehension (%)	Descriptiveness (%)	Average (%)
IXNet	90	85	85	65	81.25
TabNet	75	80	65	40	65
XMAL	60	40	25	5	32.5

- Other experimental results are also found in the Appendix B section of this document.

6.3. Discussion

As discussed in the first chapter, the primary objective of this study is to develop an interpretable and explainable deep attentive network-based model for accurate android malware detection. Apart from providing accurate detection, this work mainly hypothesizes that the transparency of DL models can be achieved using attention mechanisms and backward layer-wise relevance propagation from layer weights. The proposed network was subjected to a chain of experiments to implement and verify it properly. The results of the experiments were analyzed to highlight the key findings of this study in accordance with the prior set objective. Here, how this study achieved each specific objective will be discussed in detail.

Before the start of the experimentations, the dataset preparation stage was handled. Two datasets were collected from reliable public online repositories that are popular in the cybersecurity arena. The datasets are properly preprocessed and made ready for experimentation. The design for the proposed solution was made using the idea from TabNet and Xmal. To achieve an accurate detection model, a sequential attentive deep network was developed. The network has two components; Detector and Interpreter. The Interpreter acts as the mind and heart of the network, handling the feature selection and interpretation tasks. It consists of an Attentive block, containing a multi-head self-attention layer, and a Selector block that masks the less relevant features at each stage using the alignment scores approximated by the attentive block. This block gives the network the focus it needs for accurate detection. This component also contains the Presentation subcomponent that accepts the aggregated alignments scores and renders them to give an interpretation. On the other hand, the Detector component, consisting of the Transformer and Classifier blocks, accepts the masked input data and performs classification. The Transformer block transforms the masked input into different hidden representation subspaces suitable for the classifier. The classifier block then makes the final prediction, thus detection.

The proposed network is used to develop a model for android malware detection. Before evaluating the model, a series of experiments were performed in search of optimal hyperparameters. From the result of the hyperparameters tuning and the rest of experiment Exp 1, as summarized in Table 6.2, it is found that IXNet trained using the optimal hyperparameters deduced from the aforementioned experiments is optimal.

After that, the model was trained on the two datasets prepared and evaluated using different metrics. The result evaluation of the model shows that the model performs very well on both datasets. The experiment with the custom loss regularizer was tricky in the sense that it seems to be increasing the loss significantly. This rise is the outcome of additional loss value calculated by the custom layer, not the result of incompetency from the model. But in time, the loss decreases rapidly and levels with the others. Furthermore, the slope of the loss graph is steep indicating the possibility of further improvement with time. The experimental result shows that the usage of the custom regularizer improves the overall performance of the model. Another experiment was also conducted to find the comparative performance of IXNet relative to the other three models. Experimental results show that IXNet outperforms its counterpart models. This fulfills the need for accurate android malware detection.

The experiment of testing the impact of transfer learning through the use of a share layer across the model steps on sequential attentive deep networks shows a decline in the overall performance. This erodes the fundamental ground TabNet architecture stands on. This can be further strengthened by experimenting with classification problems. As this is out of the scope of this study, it will not be discussed here.

From the results of the experiments, it can be observed an improvement of accuracy when the attention mechanism is employed. This is consistent with the analysis of attention mechanisms in different domains and applications. The usage of multi-head self-attention in the attentive block also enabled the network to provide inherent interpretability and explainability. The alignment scores outputted from the block are used to record the feature selection process performed at the instance level, and then quantify the degree of importance given to the features at each step. This comes in handy when an explanation of the decision process of the trained model is required and makes the network transparent. The recording process is applied both at the training and testing phase, unlike most works which only explain the result of prediction at the testing/deployment stage. According to the results, the model was able to present transparency at different levels of abstraction; step-wise, locally, and globally.

As enlightening as it is, the explainability provided by the proposed model presented the theoretical foundation it stands on, which is masking (selection) at the instance level. The explainability of IXNet was also compared and contrasted with its counterparts TabNet and

Xmal models and shown to be more informative. The subjective evaluation also points out the tendency of acceptance by experts towards more transparent automated models, hence IXNet received more credit than the rest. This process also creates an opportunity to intel the learning process and working principle of the model inherently. This notion of the model was designed to give the said learned concept during the training, but the tensors used in the computation of matrix in TensorFlow are symbolic, which apparently means they hold no real value during training. This presented a challenge to record the outputted mask and context vectors. This forced the use of Tensorboard. to note down the masks and record the whole process

The other concept employed in this work is the use of Layer-wise backward relevance propagation at the detector component. An algorithm, as described in section 4.4.4, was designed to track the change in the weights of the layers in the detector component and propagate the relevance of the features to the final prediction back to them. It is implemented using the Keras Callback class to trace the said knowledge epoch-wise and batch-wise. This enabled the network to present its learning process in real time as observed by the detector component. This is useful for model debugging and analysis at runtime of the training phase. After the implementation was done, the validity of the approach was verified in a controlled environment using a randomly generated artificial dataset. The coefficients used to generate the datasets are used to crosscheck the progress of the model and the returned final feature importance. After the validation was finalized, the ability of IXNet to explain itself as it progresses with the learning process was tested using real datasets. The result was consistent with the explainability and interpretability provided at the end of training and during the testing phase.

From the results on the two datasets, it is evident that the network learned the importance of features often in both direct and inverse relations, some even in the same direction to both benign and malware classes. This is a classical problem in malware analysis, where the identification of malware from common features found in both malware and benign applications is a very complex task. These features are the ones holding the key information, unlike the scarce features found only in either application. Their magnitude directly/inversely relates to the amount of information they can provide. Therefore, the identified features and their respective importance can provide the baseline criteria to identify which is which and the reason why.

❖ This study has set out mainly to answer four research questions.

RQ1: How to develop a transparent android malware detection model without compromising detection accuracy?

The attention mechanism implemented by IXNet is based on self-attention multi-head attentive blocks. This block performs self-attention in the input data and approximates an attention score to provide a mask for instance-wise feature selection. Moreover, these instance-wise feature selection enables the model to only focus on the relevant part of the data input, greatly improving the accuracy. The feature selection criteria recorded as relevance score embedded within the attention masks can be used to showcase the process of feature selection for each individual sample. The analysis evaluation results from the experiments have shown that the proposed network provides comparable accuracy while integrating transparency.

RQ2: What is the effect of attention mechanisms-based feature selection and interpretation on the overall performance and transparency of a deep learning model?

Although some works accuse interpretation mechanisms to erode the accuracy and performance of a model, this work has shown model accuracy is improved with the implementation of inherent interpretation through the use of attention mechanisms. The attention mechanism enables attending to key features quickly, thus decreasing the computation involved in the decision-making. As for the layer-wise relevance propagation, although there is some computational complexity, it is insignificant compared to the overall computation done by the model. Thus, the advantages provided by the propagation surely outweigh the extra cost.

RQ3: What is the effect of transfer learning in ensemble-like deep learning networks?

As in the TabNet architecture, a shared layer is used to assess the impact of Transfer learning on the proposed network. The result showed that the proposed approach enhanced the detection performance relative to the performance score in the use of a shared layer. The fact that thw IXNet-based model outperforms the TabNet-based model infers that the transfer learning employed in the shared layer of TabNet diverts the principal concept of ensemble learning by ensembled sequential attentive networks.

In conclusion, the objective of the study has been achieved and the research questions have been answered. The experiments show the possibility of achieving the ultimate method to make DL models explainable and interpretable. The mechanisms used are also shown not to have a significant negative impact on the accuracy and performance of the model. Attention masks at each step can be used to provide the local element-wise feature importance and to aggregate the global importance of features as well as the main attribution criteria. Although the inefficiency of DL models on tabular data has been improved significantly by the TabNet model, IXNet can also be used as a worthy competitor. It also provides extra functionalities with a more expendable approach. The subjective evaluation indicates the possibility of acceptance by domain experts for transparent AI-based models. Although far more work is expected before attaining full trust in automated systems, this works is a step forward in that direction.

❖ To conclude, the key findings and contributions of this study are summarized as follows.

➢ **Key findings of the study**

- Attention mechanisms can tune the performance of DL models on tabular data, while still sustaining interpretation comparable to logistic regression models.
- The concept of Transfer learning negatively impacts the performance of ensembled sequential attentive deep network-based models.
- Inherent interpretability and explainability give a more precise and complementary view of the internal working principle and decision-making process.
- Layer-wise relevance propagation can be applied to quantify the learned concepts of a deep learning model.
- The transparency of deep learning models can enhance their acceptance of deep learning models in high-stake sectors.

➢ **Contribution of the study**

- Integration of Interpretability and Explainability for the transparency of DL models.
- Proposed an explainable and interpretable deep attentive network, based on the ideas adopted from TabNet, with a multi-head self-attention-based Attention mechanism for accurate and transparent android malware detection.
- Implementation of layer-wise relevance propagation for surveilling the learning process of deep learning models.

CONCLUSION AND FUTURE DIRECTION

Conclusion

The acceptance of deep learning applications in high-stake sectors, such as cybersecurity, is highly affected by the absence of a transparent deep learning network. This study set out to develop an attentive deep-learning model with an interpretable and explainable capability to detect android malware accurately. The proposed network is made up of two parallel components; Interpreter and Detector. The interpreter implemented a multi-head self-attention mechanism and layer-wise relevance propagation to provide comprehensibility to deep learning models.

The attention mechanism serves two purposes; the first one is element-wise feature selection, in which the attentive block yields an aggregated score vector from different attention heads. This score vector is used to mask less relevant features and promote more salient features, thus achieving selection. The second is the mask vectors (alignment scores) used for feature selection provide means of quantifying the learning process and the internal working principle of the model at different steps and iterations. Moreover, the attention score values are used to explain the selection decision process that is put into making the final classification decision The proposed network also implemented layer-wise relevance propagation using the Callback class of Keras. As the Detector handles the feature transformation and decision tasks, the constituent blocks of this component provide the concepts grasped during the training session in relation to the final decision made. This bestows the capability of surveilling the learning process of the model, as perceived particularly by the Detector component.

The use of attention and layer-wise relevance propagation can surely be used to achieve the intended objective of this study. As shown in the experiments, the proposed model was able to portray its learning process during the training phase. The design of the model presents a capability to quantify the learning and decision-making process across each step. The relative feature relevance grasped at each iteration was also illustrated to give a real-time view of learning progress. This was made possible by using the layer-wise relevance propagation of learned weights of the model. The sign and magnitude of the weights are traced during the

training stage to show the progress of the model and approximate the relative importance of each feature as perceived by the model at the time.

Various metrics were used to assess the overall performance of the proposed model as well as other contrasting models for a better understanding of the results. Experimental analysis was made on the proposed model using two datasets, Drebin and MalMem. The result illustrates that the proposed model performed outstandingly on both datasets. It also outperforms its counterpart models in performance while sustaining an excellent and superior-level interpretation. With the right rendering approach, this information can be provided to the end user for a better understanding of the model. The subjective evaluations also showed a promising level of acceptance from domain experts.

Future Direction

As a future direction, the next obvious step is the optimization of the proposed network for minimal computational complexity and resource usage. The use of a dataset with a more diverse feature set to train the model is preferable to attain a more robust android malware detection tool and is intended to be conducted in the future. The model should also be trained and tested with multilabel and multiclass classifications such as Android malware family and category classifications. The network can also be generalized to embed more types of datasets other than tabular. The presentation subcomponent can also be improved using more visualization and natural language generation (NLG) applications.

As a recommendation, a different approach can be taken to interpret deep learning models. Perhaps, a more inherent method can also be used to implement the layer-wise propagation functionality into the network. This can be done using more flexible frameworks like PyTorch. An alternative gradient-based approach to layer-wise propagation can also be investigated.

The fund for this research work was granted by Adama Science and Technology University.

ASTU/SM-R/505/22

REFERENCES

Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., & Giacinto, G. (2016, March 9). Novel feature extraction, selection and fusion for effective malware family classification. *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. http://dx.doi.org/10.1145/2857705.2857713

Ahuja, V. (2021). Explainable artificial intelligence: Guardian for cancer care. In *Explainable Artificial Intelligence for Smart Cities* (pp. 65–81). CRC Press. http://dx.doi.org/10.1201/9781003172772-5

Alani, M. M., & Awad, A. I. (2022). PAIRED: An explainable lightweight android malware detection system. *IEEE Access*, *10*, 73214–73228. https://doi.org/10.1109/access.2022.3189645

Arik, S. O., & Pfister, T. (2019, August 20). *TabNet: Attentive interpretable tabular learning*. ArXiv.Org. https://arxiv.org/abs/1908.07442

Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., & Rieck, K. (2014). Drebin: Effective and explainable detection of android malware in your pocket. *Proceedings 2014 Network and Distributed System Security Symposium*. http://dx.doi.org/10.14722/ndss.2014.23247

Bahdanau, D., Cho, K., & Bengio, Y. (2014, September 1). *Neural machine translation by jointly learning to align and translate*. ArXiv.Org. https://arxiv.org/abs/1409.0473

Bourebaa, F., & Benmohammed, M. (2020, November 28). Android Malware Detection using Convolutional Deep Neural Networks. *2020 International Conference on Advanced Aspects of Software Engineering (ICAASE)*. http://dx.doi.org/10.1109/icaase51408.2020.9380104

Carrier, T., Victor, P., Tekeoglu, A., & Lashkari, A. (2022). Detecting Obfuscated Malware using Memory Feature Engineering. *Proceedings of the 8th International Conference on Information Systems Security and Privacy*. http://dx.doi.org/10.5220/0010908200003120

Choi E, Bahadori MT, Sun J, Kulas J, Schuetz A, Stewart W. Retain: an interpretable predictive model for healthcare using reverse time attention mechanism. In: Advances in Neural Information Processing Systems; 2016. p. 3504–12.

Dauphin, Y. N., Fan, A., Auli, M., & Grangier, D. (2016, December 23). *Language modeling with gated convolutional networks*. ArXiv.Org. https://arxiv.org/abs/1612.08083

Elayan, O. N., & Mustafa, A. M. (2021). Android malware detection using deep learning. *Procedia Computer Science*, *184*, 847–852. https://doi.org/10.1016/j.procs.2021.03.106

Fan, M., Wei, W., Xie, X., Liu, Y., Guan, X., & Liu, T. (2021). Can we trust your explanations? Sanity checks for interpreters in android malware analysis. *IEEE Transactions on Information Forensics and Security*, *16*, 838–853. https://doi.org/10.1109/tifs.2020.3021924

Gera, T., Singh, J., Mehbodniya, A., Webber, J. L., Shabaz, M., & Thakur, D. (2021). Dominant feature selection and machine learning-based hybrid approach to analyze android ransomware. *Security and Communication Networks*, *2021*, 1–22. https://doi.org/10.1155/2021/7035233

Gibert, D., Mateu, C., & Planes, J. (2019, July). A hierarchical convolutional neural network for malware classification. *2019 International Joint Conference on Neural Networks (IJCNN)*. http://dx.doi.org/10.1109/ijcnn.2019.8852469

Gibert, D., Mateu, C., & Planes, J. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, *153*, 102526. https://doi.org/10.1016/j.jnca.2019.102526

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., & Kagal, L. (2018, October). Explaining explanations: An overview of interpretability of machine learning. *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. http://dx.doi.org/10.1109/dsaa.2018.00018

Guo, W., Ge, W., Cui, L., Li, H., & Kong, L. (2019). An interpretable disease onset predictive model using crossover attention mechanism from electronic health records. *IEEE Access*, *7*, 134236–134244. https://doi.org/10.1109/access.2019.2928579

Hinton, Geoffrey & Srivastava, Nitish & Krizhevsky, Alex & Sutskever, Ilya & Salakhutdinov, Ruslan. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 15. 1929-1958.

Iadarola, G., Martinelli, F., Mercaldo, F., & Santone, A. (2021). Towards an interpretable deep learning model for mobile malware detection and family identification. *Computers & Security*, *105*, 102198. https://doi.org/10.1016/j.cose.2021.102198

Ioffe, S. & Szegedy, C.. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. <i>Proceedings of the 32nd International Conference on Machine Learning, in Proceedings of Machine Learning Research 37:448-456 Available from https://proceedings.mlr.press/v37/ioffe15.html.

Islam, R., Tian, R., Batten, L. M., & Versteeg, S. (2013). Classification of malware based on integrated static and dynamic features. *Journal of Network and Computer Applications*, *36*(2), 646–656. https://doi.org/10.1016/j.jnca.2012.10.004

Karbab, E. B., Debbabi, M., Derhab, A., & Mouheb, D. (2018). MalDozer: Automatic framework for android malware detection using deep learning. *Digital Investigation*, *24*, S48–S59. https://doi.org/10.1016/j.diin.2018.01.007

Khoulimi, H., Lahby, M., & Benammar, O. (2022). An overview of explainable artificial intelligence for cyber security. In *Studies in Computational Intelligence* (pp. 31–58). Springer International Publishing. http://dx.doi.org/10.1007/978-3-030-96630-0_2

Kinkead, M., Millar, S., McLaughlin, N., & O'Kane, P. (2021). Towards explainable cnns for android malware detection. *Procedia Computer Science, 184*, 959–965. https://doi.org/10.1016/j.procs.2021.03.118

Kumar, R., Xiaosong, Z., Khan, R. U., Kumar, J., & Ahad, I. (2018). Effective and explainable detection of android malware based on machine learning algorithms.

Proceedings of the 2018 International Conference on Computing and Artificial Intelligence - ICCAI 2018. http://dx.doi.org/10.1145/3194452.3194465

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444. https://doi.org/10.1038/nature14539

Li, M. Q., Fung, B. C. M., Charland, P., & Ding, S. H. H. (2021). I-MAD: Interpretable malware detector using Galaxy Transformer. *Computers & Security*, *108*, 102371. https://doi.org/10.1016/j.cose.2021.102371

Liu, Y., Tantithamthavorn, C., Li, L., & Liu, Y. (2022). Deep learning for android malware defenses: A systematic literature review. *ACM Computing Surveys*. https://doi.org/10.1145/3544968

Lu, T., Du, Y., Ouyang, L., Chen, Q., & Wang, X. (2020). Android malware detection based on a hybrid deep learning model. *Security and Communication Networks*, *2020*, 1–11. https://doi.org/10.1155/2020/8863617

Luong, M.-T., Pham, H., & Manning, C. D. (2015, August 17). *Effective approaches to attention-based neural machine translation*. ArXiv.Org. https://arxiv.org/abs/1508.04025

Mahdavifar, S., Abdul Kadir, A. F., Fatemi, R., Alhadidi, D., & Ghorbani, A. A. (2020, August). Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. http://dx.doi.org/10.1109/dasc-picom-cbdcom-cyberscitech49142.2020.00094

Marais, B., Quertier, T., & Chesneau, C. (2021). Malware Analysis with Artificial Intelligence and a Particular Attention on Results Interpretability. In *Distributed Computing and Artificial Intelligence, Volume 1: 18th International Conference* (pp. 43–55). Springer International Publishing. http://dx.doi.org/10.1007/978-3-030-86261-9_5

Martins, A. F. T., & Astudillo, R. F. (2016, February 5). *From softmax to sparsemax: A sparse model of attention and multi-label classification*. ArXiv.Org. https://arxiv.org/abs/1602.02068

Mathews, S. M. (2019). Explainable artificial intelligence applications in NLP, biomedical, and malware classification: A literature review. In *Advances in Intelligent Systems and Computing* (pp. 1269–1292). Springer International Publishing. http://dx.doi.org/10.1007/978-3-030-22868-2_90

Melis, M., Maiorca, D., Biggio, B., Giacinto, G., & Roli, F. (2018, September). Explaining black-box android malware detection. *2018 26th European Signal Processing Conference (EUSIPCO)*. http://dx.doi.org/10.23919/eusipco.2018.8553598

Montavon, G., Binder, A., Lapuschkin, S., Samek, W., & Müller, K.-R. (2019). Layer-Wise relevance propagation: An overview. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning* (pp. 193–209). Springer International Publishing. http://dx.doi.org/10.1007/978-3-030-28954-6_10

Niu, Z., Zhong, G., & Yu, H. (2021). A review on the attention mechanism of deep learning. *Neurocomputing*, *452*, 48–62. https://doi.org/10.1016/j.neucom.2021.03.091

Owens, E., Sheehan, B., Mullins, M., Cunneen, M., & Ressel, J. (2022). Explainable artificial intelligence (xai) in insurance: A systematic review. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4088029

Pei, X., Yu, L., & Tian, S. (2020). AMalNet: A deep learning framework based on graph convolutional networks for malware detection. *Computers & Security*, *93*, 101792. https://doi.org/10.1016/j.cose.2020.101792

Pei, X., Yu, L., Tian, S., Wang, H., & Peng, Y. (2020). Combining multi-features with a neural joint model for Android malware detection. *Journal of Intelligent & Fuzzy Systems*, *38*(2), 2151–2163. https://doi.org/10.3233/jifs-190888

Rahali, A., Lashkari, A. H., Kaur, G., Taheri, L., Gagnon, F., & Massicotte, F. (2020, November 27). DIDroid: Android malware classification and characterization using

deep image learning. *2020 the 10th International Conference on Communication and Network Security*. http://dx.doi.org/10.1145/3442520.3442522

Razgallah, A., Khoury, R., Hallé, S., & Khanmohammadi, K. (2021). A survey of malware detection in Android apps: Recommendations and perspectives for future research. *Computer Science Review*, *39*, 100358. https://doi.org/10.1016/j.cosrev.2020.100358

Ribeiro, M., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. http://dx.doi.org/10.18653/v1/n16-3020

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, *1*(5), 206–215. https://doi.org/10.1038/s42256-019-0048-x

Saraf, A. P., Chan, K., Popish, M., Browder, J., & Schade, J. (2020, June 8). Explainable artificial intelligence for aviation safety applications. *AIAA AVIATION 2020 FORUM*. http://dx.doi.org/10.2514/6.2020-2881

Shanshan Wang, Chen, Z., Zhang, L., Yan, Q., Yang, B., Peng, L., & Zhongtian Jia. (2016, June). TrafficAV: An effective and explainable detection of mobile malware behavior using network traffic. *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. http://dx.doi.org/10.1109/iwqos.2016.7590446

Shickel, B., Tighe, P. J., Bihorac, A., & Rashidi, P. (2018). Deep EHR: A survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. *IEEE Journal of Biomedical and Health Informatics*, *22*(5), 1589–1604. https://doi.org/10.1109/jbhi.2017.2767063

Stamp, M., Alazab, M., & Shalaginov, A. (Eds.) (2021). *Malware analysis using artificial intelligence and deep learning*. (2021c). (1st. ed.) Springer International Publishing. http://dx.doi.org/10.1007/978-3-030-62582-5

Stuart Millar, Niall McLaughlin, Jesus Martinez del Rincon, Paul Miller, and Ziming Zhao. 2020. DAndroid: A multi-view discriminative adversarial network for obfuscated

Android malware detection. In Proceedings of the 10th ACM Conference on Data and Application Security and Privacy. 353—364

Tian, K., Yao, D., Ryder, B. G., Tan, G., & Peng, G. (2020). Detection of repackaged android malware with code-heterogeneity features. *IEEE Transactions on Dependable and Secure Computing*, *17*(1), 64–77. https://doi.org/10.1109/tdsc.2017.2745575

Vilone, G., & Longo, L. (2020, May 29). *Explainable artificial intelligence: A systematic review*. ArXiv.Org. https://arxiv.org/abs/2006.00093

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, June 12). *Attention is all you need*. ArXiv.Org. https://arxiv.org/abs/1706.03762

Wang, W., Zhao, M., & Wang, J. (2018). Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. *Journal of Ambient Intelligence and Humanized Computing*, *10*(8), 3035–3043. https://doi.org/10.1007/s12652-018-0803-6

Wen, L., & Yu, H. (2017). An Android malware detection system based on machine learning. *AIP Conference Proceedings*. http://dx.doi.org/10.1063/1.4992953

Wu, B., Chen, S., Gao, C., Fan, L., Liu, Y., Wen, W., & Lyu, M. R. (2021). Why an android app is classified as malware. *ACM Transactions on Software Engineering and Methodology*, *30*(2), 1–29. https://doi.org/10.1145/3423096

Yan, A., Chen, Z., Zhang, H., Peng, L., Yan, Q., Hassan, M. U., Zhao, C., & Yang, B. (2021). Effective detection of mobile malware behavior based on explainable deep neural network. *Neurocomputing*, *453*, 482–492. https://doi.org/10.1016/j.neucom.2020.09.082

Yuxin, D., & Siyi, Z. (2017). Malware detection based on deep learning algorithm. *Neural Computing and Applications*, *31*(2), 461–472. https://doi.org/10.1007/s00521-017-3077-6

APPENDICES

Appendix A: Preview

A.1. Profile of Evaluators

Table A. 1: Profile of Evaluators

Evaluator	Expertise Area	Profession	Level	Year Experience
Evaluator-1	Cybersecurity	Malware Analyst	Junior	1
Evaluator-2	Cybersecurity	Malware Analyst	Junior	1
Evaluator-3	Cybersecurity	Malware Analyst	Senior	4
Evaluator-4	Cybersecurity	Malware Analyst	Senior	4
Evaluator-5	Cybersecurity	Malware Analyst	Senior	4
Evaluator-6	Cybersecurity	Malware Analyst	Expert	6

A.2. 2-Step IXNet Model Summary

```
Model: "ix_net"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization_104 (Ba  multiple                 860
 tchNormalization)

 attentive_block_20 (Attenti  multiple                 13760
 veBlock)

 attentive_block_21 (Attenti  multiple                 13760
 veBlock)

 selector_block_20 (Selector  multiple                 860
 Block)

 selector_block_21 (Selector  multiple                 860
 Block)

 transformer_block_30 (Trans  multiple                 3520
 formerBlock)

 transformer_block_31 (Trans  multiple                 3520
 formerBlock)

 classifier_block_12 (Classi  multiple                 32
 fierBlock)

=================================================================
Total params: 37,172
Trainable params: 32,378
Non-trainable params: 4,794
```

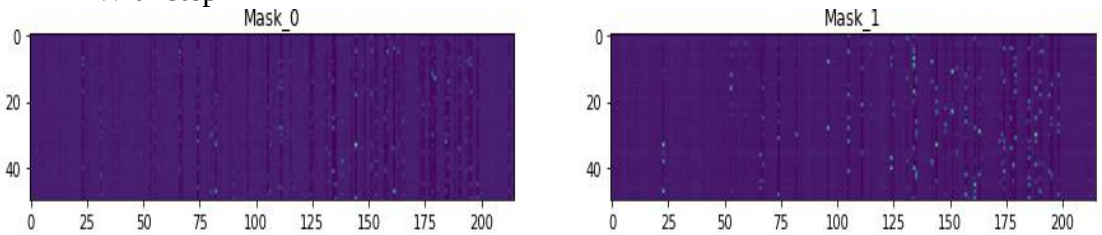
Figure A. 1: Model Summary of IXNet with Step=2

Appendix B: Ablation Study

B.1. Explainability Experimentation Results

I. Step Mask Explainability

- With step = 2



- With Step = 4

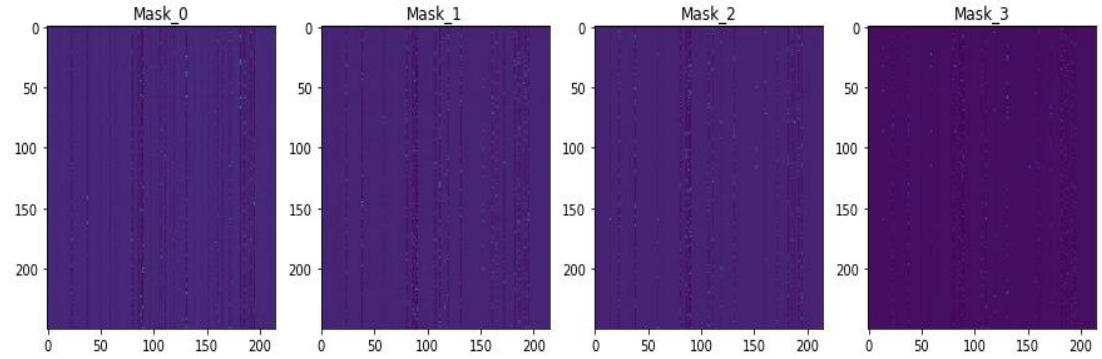


Figure B. 1: Sample Result Step Mask Explainability

II. Local Mask Explainability

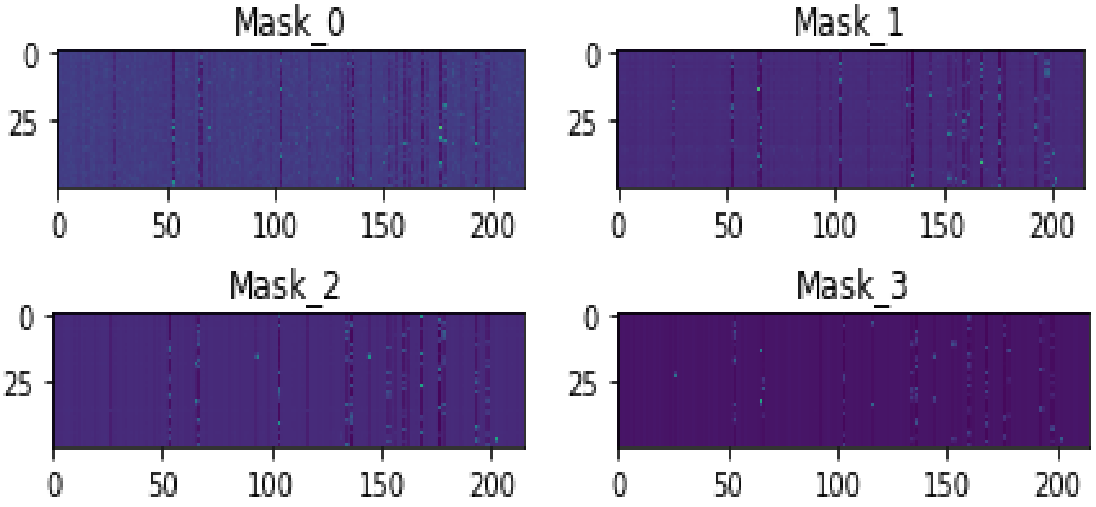


Figure B. 2: Sample Result Local Mask Explainability

B.2. Interpretability Experimentation Results

- **Epoch Wise**
  - **Result for Artificial Dataset (Last sample is the Average)**

```
47] coefficients

    array([ 0.3, -1.2, -1.1, -0.2,  0. ,  0.2,  0. ,  0. ,  0. , -0.6, -0.6,
            0. , -0. , -1. ,  0. ,  0. ,  0. , -0.3])
```

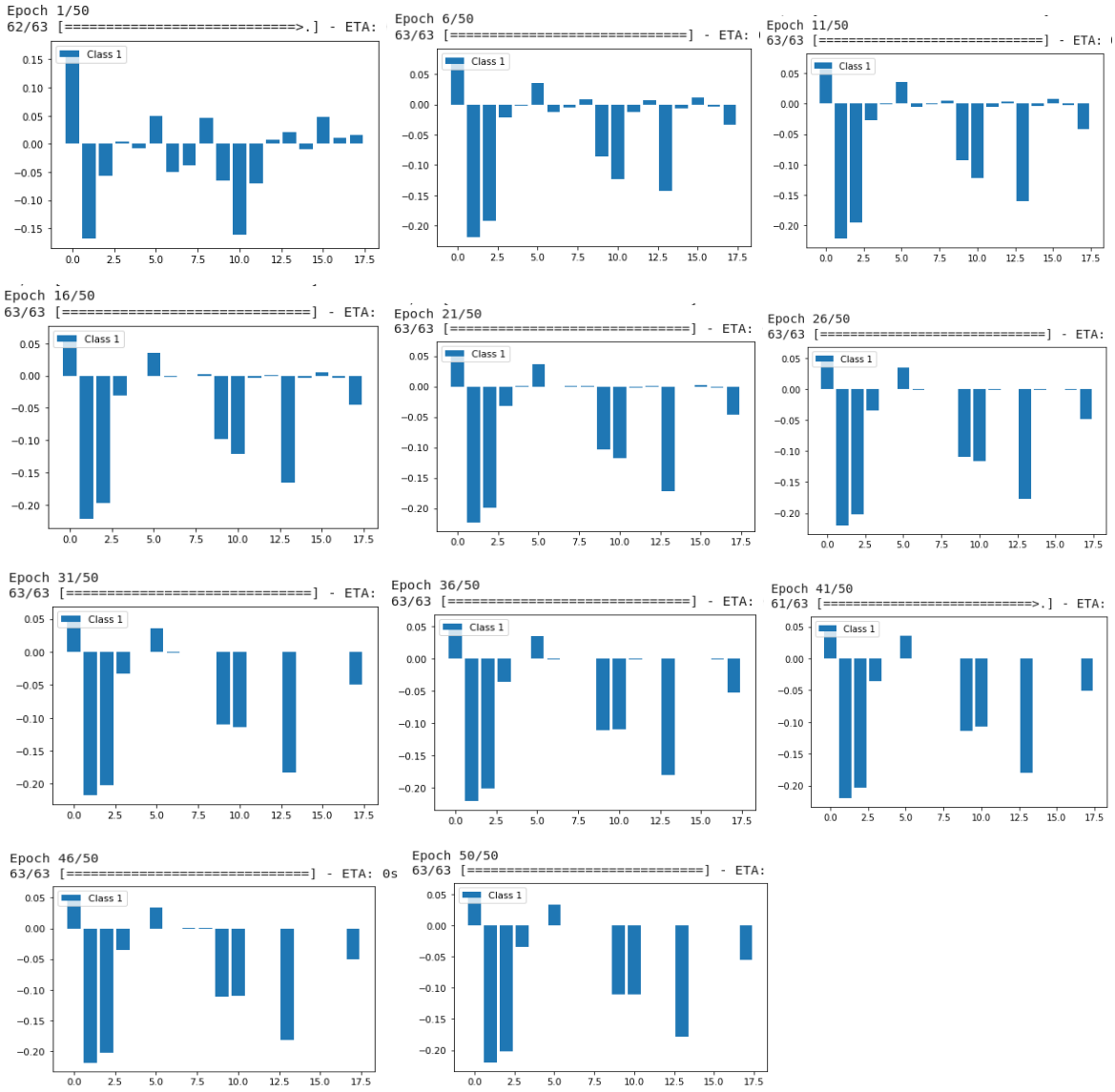


Figure B. 3: Sample Result for Epoch Wise Interpretability on Artificial Dataset

- **Result of Learning Process of IXNet on Drebin**

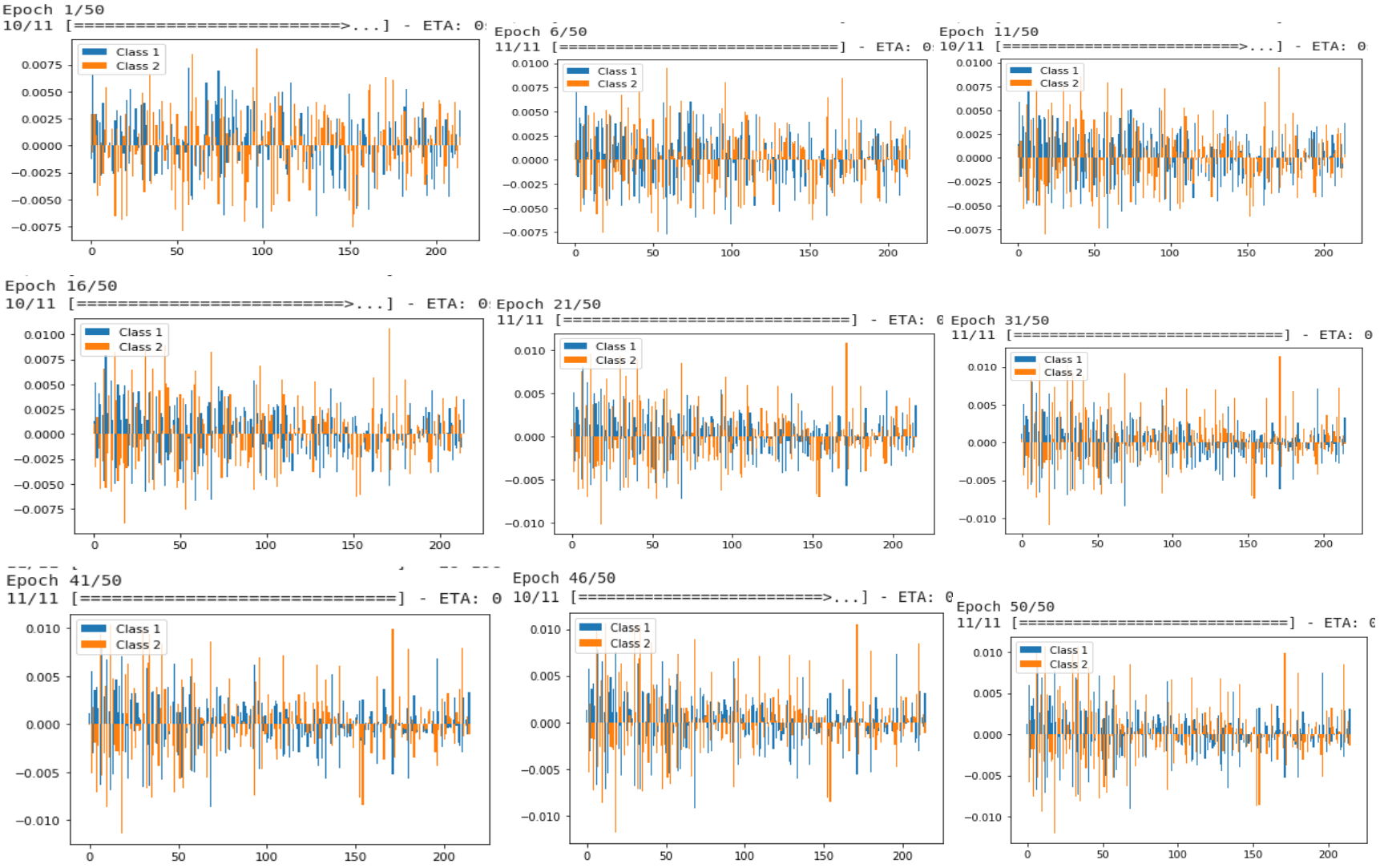


Figure B. 4: Sample Result for Epoch Wise Interpretability on Drebin

- **Result of Learning Process of IXNet on MalMem**

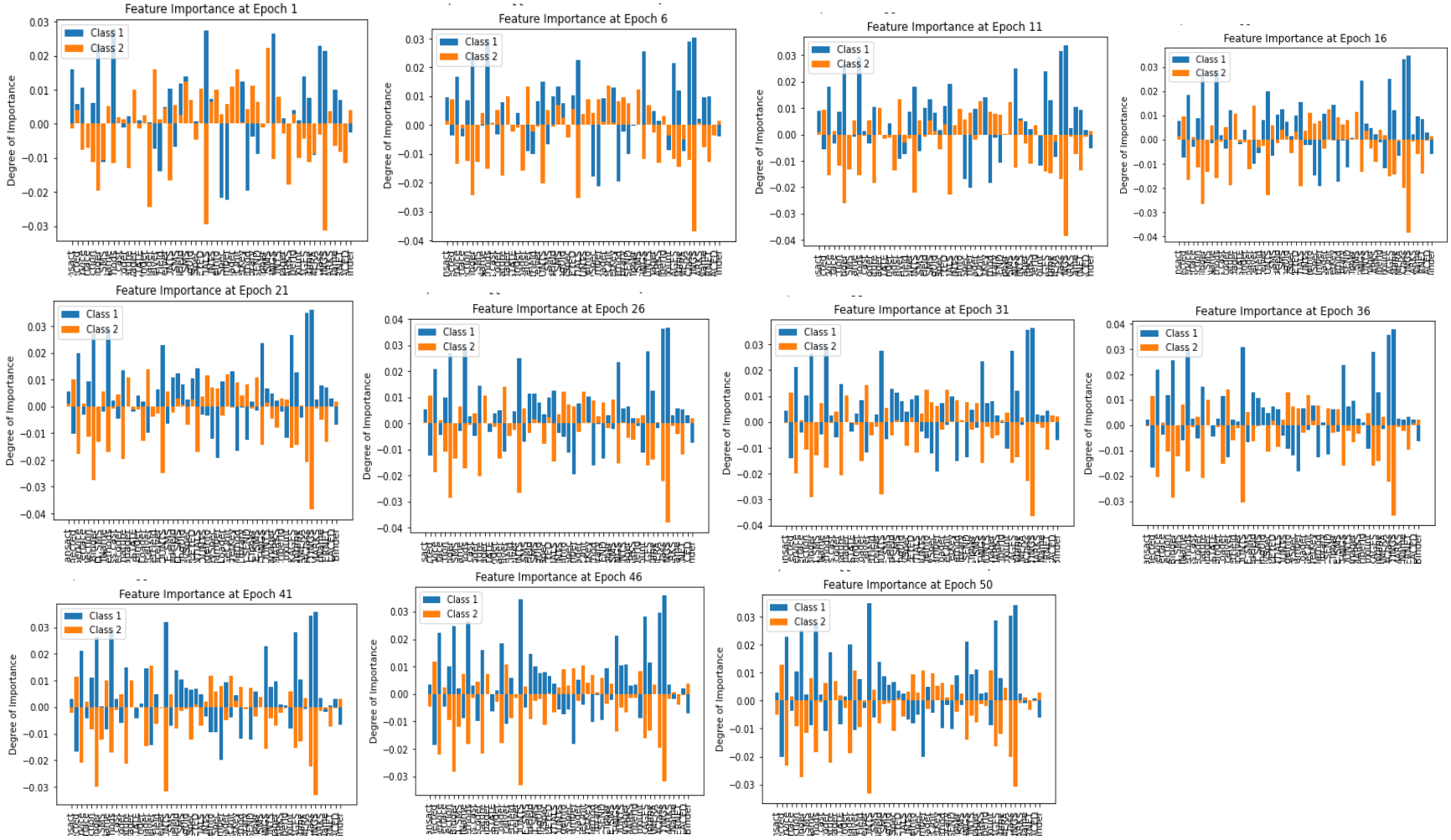


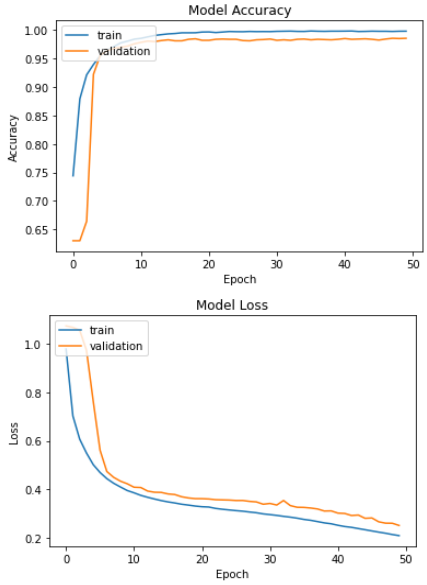
Figure B. 5: Sample Result for Epoch Wise Interpretability on MalMem

B.3. Result Curve and Confusion Matrix

Some notable results of the experiment on different basis grounds are listed below.

➢ **Sparsemax vs Softmax**

- *IXNet with Softmax*



- *IXNet with Sparsemax*

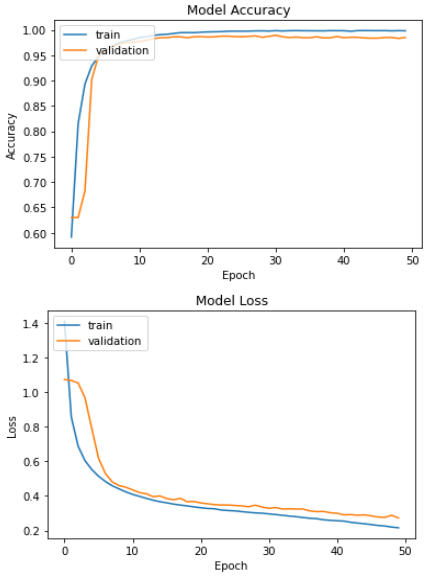
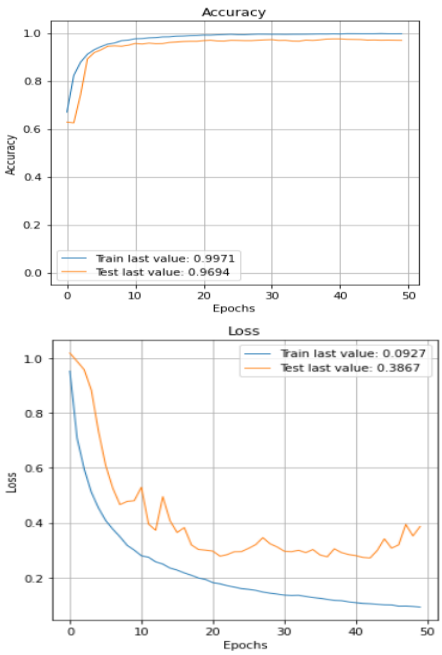


Figure B. 6: Comparison of Softmax and Sparsemax

➢ ***Loss Regularizer***

- *Without Custom Loss Regularizer*



- *With Custom Loss Regularizer*

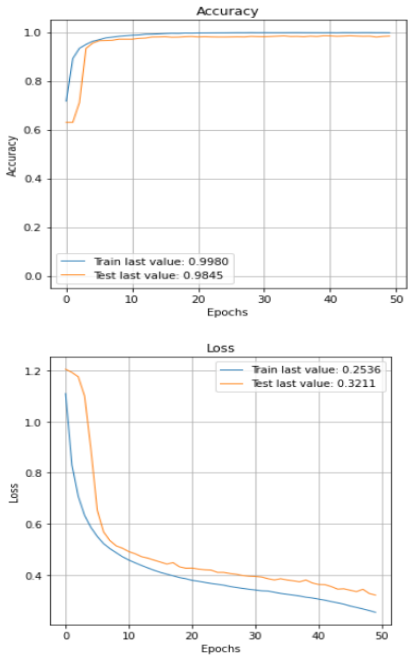
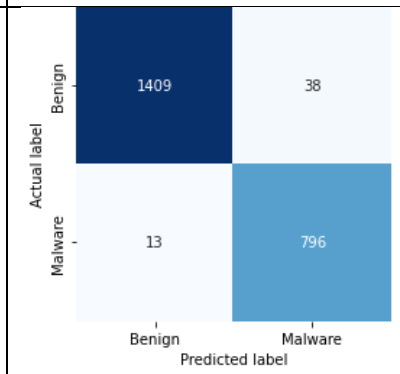
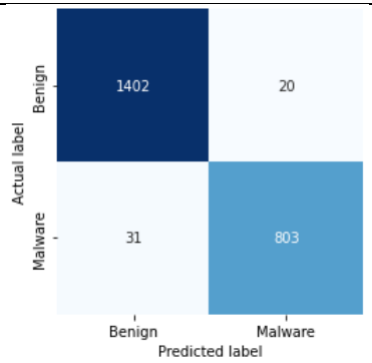
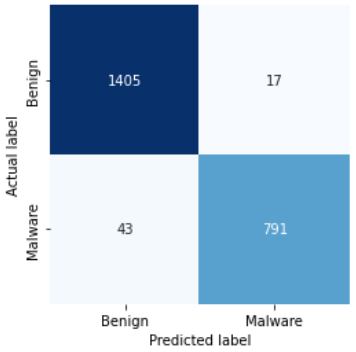
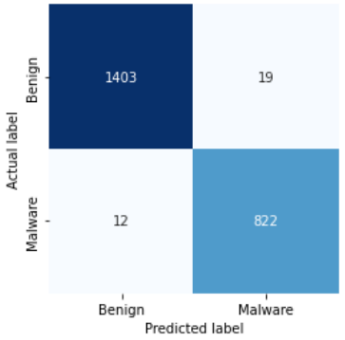


Figure B. 7: Comparison of Softmax and Sparsemax

- **Confusion Matrix**

Table B. 1: Confusion Matrix of Models on Drebin and MalMem

Datasets	XGBoost	Xmal	TabNet	IXNet
Drebin				
MalMem