**JOURNAL OF SOFT COMPUTING AND DATA MINING**
e-ISSN: 2716-621X

**JSCDM**

Vol. 6 No. 1 (2025) 378-393
https://publisher.uthm.edu.my/ojs/index.php/jscdm

# Android Ransomware Detection by Deep Learning

**Nuraini Syahirah Binti Abd Rais[1], Cik Feresa Mohd Foozy[1], Andi Maslan[2]**

[1] *Institut Kejuruteraan Integrasi, Pusat Kecemerlangan Industri-Rail (ICoE-Rail), Information Security Interest Group (ISIG), Fakulti Sains Komputer dan Teknology Maklumat, Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA*
[2] *Faculty of Engineering and Computer Science, Putera Batam University, Batam, INDONESIA*

*Corresponding Author: feresa@uthm.edu.my
DOI: https://doi.org/10.30880/jscdm.2025.06.01.025

**Article Info**

Received: 24 August 2024
Accepted: 5 May 2025
Available online: 30 June 2025

**Keywords**

Android, ransomware, deep learning, CNN, LSTM

**Abstract**

This research proposes a novel deep learning-based detection model to combat the growing menace of Android ransomware. Deep learning models can learn complex features and training models with many convolutional layers and millions of parameters, leading to overfitting in a few numbers of epochs. As shown by previous works, current methods for Android malware detection are constrained by insufficient feature sets and preprocessing methods. Combining static and dynamic information for a more thorough analysis is essential to improve detection accuracy. While Recurrent Neural Network (RNN) has effectively solved temporal problems, it has limitations such as gradient dispersion and high calculation costs. The goal is to develop Android ransomware detection by deep learning with optimal epochs and test the model using parameter evaluation of accuracy, precision, recall and F-1 score. The methodology comprises of five phases: dataset, data preprocessing, Deep Learning model (CNN and LSTM), 10-fold cross-validation and result. The model is tested on an Android dataset, CICAndMal2019, which includes Permission and Intent as static features and API calls as dynamic features. For this research, only the sample of ransomware from Jisut, RansomBO, Charger, Lockerpin, Koler, Pletor, PornDroid, Simplocker, SVpeng, WannaLocker family and other benign samples will be used. The CNN model obtained its best performance at 40 epochs, with the result of 97.93% accuracy, 98.00% precision, 99.93% recall and 98.95% F-1 score. The LSTM model performed best at 10 epochs, with a result of 97.74% accuracy, 97.74 % precision, 100% recall, and 98.86% F-1 score. This research highlights that the CNN model obtains accuracy slightly higher than the LSTM model, improving the Android ransomware detection by deep learning.

## 1. Introduction

Android ransomware is a virus that encrypts essential data saved on user's device, prevents them from accessing the operating system, and demands a fee to recover unlock the data(Rahima Manzil & Naik, 2023). Numerous studies have suggested ransomware detection solutions mainly through Convolutional Neural Networks (CNN). Deep learning models have the capacity to learn complex features, training models with many convolutional layers and millions of parameters. As a result, the model performs poorly and overfits in a few numbers of epochs(Hemalatha et al., 2021). According to review deep-learning based Android malware detection by Z Wang et al. (Wang et al., 2020), current methods for Android malware detection by previous works are constrained by

insufficient feature sets and preprocessing methods. Many temporal problems, including machine translation, speech recognition and language processing have been effectively solved by using Recurrent Neutral Network (RNN). However, the RNN has issues with gradient dispersion and calculation cost (Wang et al., 2020).

This goal of this research is to design a deep learning framework for Android ransomware detection together with various epoch settings to determine the optimal number that maximize the model performance. This research also aims to reduce overfitting and training inefficiencies while increasing detection accuracy by experimenting with different epoch values. Additionally, the efficacy of the suggested methodology in differentiating ransomware from benign applications will be tested using frameworks such as accuracy, precision, recall and F-1 score.

This research involves the development of a framework for detecting Android ransomware using deep learning, incorporating two key algorithms: Convolutional Neural Network (CNN) and Long Short-Term Memory Recurrent Neural Network (LSTM-RNN). The framework will encompass features designed to identify precision, accuracy, recall, and F-1 score. Utilizing the CICInvesAndMal2019 dataset from the University of New Brunswick (UNB), consisting of 426 Android malware and 5065 benign instances(Investigation on Android Malware 2019 | Datasets | Research | Canadian Institute for Cybersecurity | UNB, n.d.). TensorFlow on the Google Colab platform is used in training the deep learning model, with Phyton serving as the programming language for ransomware detection.

Using the CICInvesAndMal2019 dataset, a thorough comparison of the CNN and LSTM algorithms with different epochs demonstrated good performance, with CNN obtaining accuracy slightly higher than LSTM. The CNN model performed best at 40 epochs, with 97.93% accuracy, 98.00%precision, 99.93% recall and 98.95% F-1 score. The LSTM model performed best at 10 epochs, with 97.74% accuracy, 97.74 % precision, 100% recall and 98.86% F-1 score.

The report's remaining sections are organized as follows: Chapter 2 will discuss the related works done that involve in-depth analysis of current methodologies and techniques. Next, Chapter 3 provides a description of the methodology used to assess and compare the research's performance. Chapter 4 will explain the experiment's acquired results on both models. Section 5 provides conclusions drawn from this work and suggestion for future works within this topic.

## 2. Related Works

This section will provide an overview of ransomware and other related works on the same dataset, methodologies and approaches.

### 2.1 Android Ransomware

Android ransomware can function in two ways: it can encrypt all the infected device's files or lock it. Following infection, the attacker demands payment from the victim of the ransomware before providing the key needed to restore file access or regain control of the Android device (Rahima Manzil & Naik, 2023). To avoid being discovered, fraudsters typically ask for payments in digital currencies like bitcoins (Alsoghyer & Almomani, 2019).

### 2.2 Android Architecture

Android is an open-source operating system built on Linux. On September 23, 2008, Android version 1.0 was released while Android version 12 was released in October 2021 as the most recent. Android applications are stored as packaged APK files (.apk) and are written in Kotlin and Java (Rahima Manzil & Naik, 2023). Each application is composed of an AndroidManifest.xml, an XML descriptor document that is necessary and runs continuously in a separate process. AndroidManifest.xml provides details about the packages, libraries, APIs, and other resources needed by the application, as well as the permissions it requests and imposes. Applications were divided into four sections: provider of content, broadcast receiver, activity and service. These components use messages known as intents to convey information(Naway & Li, n.d.).

Since the Linux kernel forms the basis of Android OS, it has Android permissions and a sandboxing security feature that separates resources from many applications. An Android application can set permissions in the AndroidManifest.xml file to limit access to private resources. The user may grant, accept or deny permissions throughout the installation process. Linux gives every application a unique identity (UID) to implement application sandboxing. Application sandboxing helps improve Android OS security by preventing the interaction of dangerous and benign apps(Rahima Manzil & Naik, 2023).

### 2.3 Forms of Ransomware

Ransomware has evolved into numerous variants, each with its own set of traits and methods of operation(Florida et al., 2022). Below are the primary types of ransomwares:

i. Locker Ransomware (Lock Screen Ransomware)

Locker ransomware aims to obstruct essential device functionality. It stops victims from accessing their devices by displaying a lock screen, frequently posing as law enforcement, and requesting payment to access the devices. Locker ransomware can encrypt specific files, locking the keyboard or screen of the device. Nevertheless, it is easy to remove by using an on-demand virus scanner or rebooting the device in safe mode(Aslan, 2020).

ii. Crypto Ransomware (File Encryptor Ransomware)
Important data on user's devices is encrypted by crypto-ransomware using advanced encryption algorithms and demand payment, typically in cryptocurrency, to unlock the victim's content (Aslan, 2020). It is capable of using hybrid, asymmetric, or symmetric encryption. Based on the steps involved, the encryption process can be classified into three groups: Class A files are not relocated or renamed, but they are encrypted; Class B files are encrypted and moved but not relocated; and Class C files are encrypted, transferred and renamed, making it more challenging to locate and retrieved the file (Sgandurra et al., n.d.).

iii. Scareware
Scareware frequently uses pop-up advertisements to fool people into thinking they must download a particular software. Cybercriminals use scareware to capitalize on people's concerns rather than restricting device access or encrypting data. However, there is no actual harm done to the victim's device (Al-rimy et al., 2018).

## 2.4 Ransomware Detection Approach

Ransomware is a threat that is too great as it keeps getting more sophisticated and secretive. Thus, many methods are being used to overcome the difficulties in this field (Florida et al., 2022). Below are the three types of detection approach:

i. Static Approach
A portion of the application is screened by the without really being executed. Therefore, the application's harmful behaviour has no effect on the mobile device, making it a secure method of analysing malware(Taheri et al., 2019). This approach retrieves the APK source code through reverse engineering methods. This method can be used to extract several static features including missions and API calls features, which can be used in the malware detection process(Alzahrani & Alghazzawi, 2019).

If this ransomware is successfully detected, it can also be prevented from ever getting a chance to be executed. However, this approach is vulnerable to obfuscation of code. A straightforward insertion of regular operation codes may cause a discrepancy with harmful codes that have already been found. This approach is also not effective towards multi-phase attack. Instead of carrying out a comparable malevolent operation, the first code might only be a straightforward procedure to unlock a backdoor for the download of additional codes(Wang et al., 2020).

ii. Dynamic Approach
In order to observe application behavior, dynamic analysis necessitates running the program in a separate environment. Since code execution processes are present, dynamic analysis has a benefit over static analysis in that it can disclose the malware's natural behavior(Taheri et al., 2019). This approach declares rights to filter apps in the manifest file. This approach involves running the APK and setting it up to generate executions, evaluate the application's activity and create data flows for the application. The malware must be executed in a virtual environment to lessen the impact of any outside influences that can alter the virus 's behavior (Alzahrani & Alghazzawi, 2019).

This kind of analysis can analyze encrypted code and is less susceptible to obfuscation. To accomplish its goal, malicious conduct must be a component of the procedure. The drawback to this approach is the investigation requires a time-consuming and expensive setup. To capture the ransomware behaviour accurately, the environment configuration must be as near to an actual setting as possible. Since one of the ransomware's setup activities is environment mapping, it may discover that the analysis is taking place on a virtual computer, which can save costs and resources, and cease exhibiting all of its behavior (Wang et al., 2020).

iii. Hybrid Approach
Static and dynamic approaches are combined to create the hybrid approach. Both differ in some positive and negative ways. For instance, malware can bypass static analysis approaches by using code encryption techniques, yet it can be more readily identified during execution(Taheri et al., 2019). As the hybrid approach integrates a range of run-time and application features, it can be used to increase the efficiency of malware detection(Alzahrani & Alghazzawi, 2019).

As suggested by Kok, S H Abdullah, Azween Jhanjhi et al.(Wang et al., 2020), the solution needs to be two-fold. They suggest creating a ransomware attack model incorporating information from both dynamic and static analyses to guarantee a thorough understanding of ransomware infection and attack patterns.

## 2.5 Deep Learning Model

Artificial Neural Networks (ANN) serves as the model for deep learning, a subfield of machine learning that picks up knowledge from examples. It is a novel method that is widely applied to voice control, driverless cars, image processing and malware detection(Bayazit et al., 2022).

### 2.5.1 Convolutional Neural Network (CNN)

Convolutional structures in data allow the Convolutional Neural Network (CNN), a type of feedforward neutral network, to extract features from the data. CNN does not need the features to be extracted manually, in contrast to traditional feature extraction methods. Visual perceptions influence the design of CNN. Artificial neurons behave similarly to biological neurons where CNN kernels are receptors with the ability to respond to various aspects. Activation functions resemble the mechanisms that permits neural electric signals beyond a threshold to go to the next neuron. Optimizers and loss functions were created to teach the CNN system what was expected of it, as a whole.

In particular, four components are needed to build a CNN model. Convolution is an essential step of the feature extraction process. Feature maps are the results of a convolution. When a convolution kernel is set to a specific size, we will lose information about the boundary. By indirectly changing the size, padding is the process of increasing the input with a zero value. The stride also regulates the convolving density. As stride length increases, the density falls. After convolution, feature maps are composed of numerous features, which raises the possibility of an overfitting problem. Therefore, pooling (also known as down sampling) is suggested to eliminate redundancy. This includes average and max pooling. Fig. 1 shows the procedure of 2D CNN (Li et al., 2022).

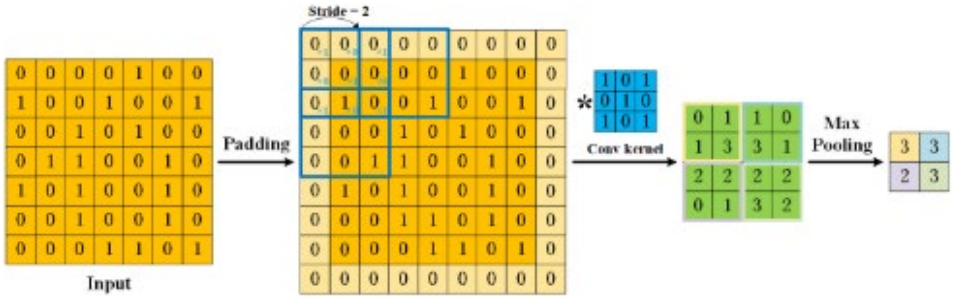


**Fig. 1** *Procedure of 2D CNN*

### 2.5.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of Recurrent Neutral Network (RNN) that is capable of learning long-term dependencies. The LSTM's four neural network layers communicate with one another to decide whether to add or remove information from a cell state(Xu et al., 2020).

The information flow via the network is controlled by a several components that make up LSTM. Each gate that controls these components has a distinct function in creating the hidden state and updating the cell state. LSTM equations mostly use the following terms:

- *ht*: Hidden state at time step t, which serves as the output of the LSTM cell.
- *xt*: Input at the current time step.
- *Ct*: Cell state at time step t, representing the memory of the LSTM.
- *ft*: Forget gate activation, which determines the proportion of previous memory to retain.
- *it*: Input gate activation, which regulates how much new information is added to the memory.
- *ot*: Output gate activation. Which controls how much memory is passed to the hidden state.
- *Wf, Wi, WC, Wo*: Weight matrices associated with the respective gates.
- *bf, bi, bC, bo*: Bias terms for the respective gates.
- $\sigma$: Sigmoid activation function, which outputs values between 0 and 1.
- *tanh*: Hyperbolic tangent function, which outputs values between -1 and 1.

LSTM will first decide how much data a cell must hold from the proceeding layer. A sigmoid layer that functions as a forgetting mechanism determines how long information is retained. Given $x_t$ and $h_{t-1}$, the sigmoid activation function Ø produces an output value between 0 and 1, which is then multiplied by the value of $c_t1$. The prior value will be entirely retained if the sigmoid output is set to 1, and it will be removed if it is set to 0as shown in Equation (1)(Aslan, 2020).

$$ft = \sigma(Wf[ht-1, xt] + bf) \quad (1)$$

The next step is updating the memory cell, followed by the hyperbolic tangent function (tanh layer) that returns values between 1 and -1, and the sigmoid layer that determines which values to update as in Equation (2). Equation (3) shows a vector of fresh candidate values called $c_t$ is created by the tanh layer.

$$it = \sigma(W[ht-1, xt] + bi) \quad (2)$$

$$Ct = tanh(WC \cdot [ht-1, xt] + bC) \quad (3)$$

The old state $c_{t-1}$ is multiplied with ft, which forgets the values decided in the previous step. Next, the findings are added to $i_t \cdot c_t$. After that, the candidate value produced will be kept in the cell. In this stage, the cell gathers fresh data. A filtered representation of the values in the cell state is the output $h_t$ as shown in Equation (4) and (5). The sigmoid layer will determine the cell's output region. The tanh function receives the cell value, multiplies it by the sigmoid layer output, and then forwards the result to the next layer (Aslan, 2020).

$$ht = ot \cdot tanh(Ct) \quad (5)$$

$$ot = \sigma(W[ht-1, xt] + bo) \quad (4)$$

The LSTM architecture will be shown in Fig. 2(Gohari et al., 2021).


**Fig. 2** *LSTM architecture*

### 2.5.3 Epoch Usage in Deep Learning Model

Y. Liu and Y. Wang (Liu & Wang, 2019) used a dataset of 13518 harmful samples and 7680 non-malicious sample annotated with antivirus software findings from VirusTotal to assess the efficacy of the Bidirectional Long -Short Term Memory (BiLSTM) network in malware detection. The BiLSTM neural network was created using Phyton's Keras framework, and the dataset was divided into training, validation, and testing sets. The model was trained with a dropout rate of 0.5, a batch size of 64 and a loss function of log across 30 epochs, taking advantage of the computational capacity of NVIDIA GeFOorce GTX 1080 GPU for faster processing. The result showed that, beginning with the 10 epoch, the accuracy of both LSTM and BiLSTM models outperformed another neural network such as GRU, BGRU and SimpleRNN, with a proportionally reduced loss rate, showing their superior effectiveness in malware detection.

## 2.6 Study of Existing Approach

E. C. Bayazit, O. K. Sahingoz, and B. Dogan(Bayazit et al., 2022) did a study with the aim to create a reliable malware detection system for Android operating systems utilizing deep learning methods, specifically Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (Bi-LSTM), and Gated Recurrent Unit (GRU). The study applied static analysis approaches to assess these models on the CICInvesAndMal2019 dataset. The BiLSTM model outperformed the other algorithms, with an accuracy rate of 98.85%. This demonstrates the effectiveness of BiLSTM in improving malware detection accuracy in the increasingly susceptible Android market.

In this research by M. AlJamal, R. Alquran, I. AL-Aiash, A. Mughaid, S. AlZu'bi, and A. A. Abutabanjeh(AlJamal et al., 2023), they employ a wide range of machine learning (ML) algorithms and methodologies to handle the changing threat landscape confronting Android devices. Using deep learning, ensemble classifiers, KNN, Random Forest, and LSTM algorithms, as well as feature extraction, reduction, and optimization techniques, the study shows considerable success in detecting and classifying important malware families, with a particular emphasis on ransomware. The results demonstrate high accuracy rates, with the Bayesian network classifier detecting

Wannalocker ransomware with 99.1% accuracy and the LSTM-based model detecting ransomware with 97.08% accuracy in Android. The study emphasizes the necessity of efficient machine learning methodologies for improving detection accuracy while reducing processing overhead and continual advances in algorithms and techniques to combat emerging threats.

A.S. Shatnawi, Q. Yassen, and A. Yateem(Shatnawi et al., 2022) proposed an Android malware detection using static feature analysis and Machine Learning (ML) algorithms such as Support Vector Machines (SVM), K-nearest neighbors (KNN), and Naive Bayes (NB), with Android permissions and API calls. The study attempts to improve malware detection efforts by measuring these algorithms' accuracy, recall, precision, and F1 scores, emphasizing the importance of security in mobile development. Principal Component Analysis (PCA) and Recursive Feature Elimination (RFE) are used to pick features from the CICInvesAndMal2019 dataset, which includes permissions, intents, API requests, and dynamic features. The results indicate SVM with 94.36% accuracy, KNN with 93.42% accuracy, and NB with 84.33%.

Table 1 shows the comparison between previous research with the same dataset; CICInvesAndMal2019.

**Table 1** *Comparison between previous research with the same dataset*

| Work by | Dataset | Algorithm | Approach | Accuracy Result |
|---|---|---|---|---|
| E. C. Bayazit, O. K. Sahingoz, and B. Dogan(Bayazit et al., 2022) | CICInvesAndMal2019 | Bi-LSTM | Deep Learning | 98.85% |
| M. AlJamal, R. Alquran, I. AL-Aiash, A. Mughaid, S. AlZu'bi, and A. A. Abutabanjeh(AlJamal et al., 2023) | CICInvesAndMal2019 | LSTM | Deep Learning | 97.08% |
| A. S. Shatnawi, Q. Yassen, and A. Yateem(Shatnawi et al., 2022) | CICInvesAndMal2019 | Support Vector Machine (SVM) | Machine learning | 94.36% |
| | | K-nearest neighbors (KNN) | | 93.42% |
| | | Naïve Bayes (NB) | | 84.33% |

## 3. Methodology

This chapter will cover deep learning techniques, which consist of two algorithms: Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) that will be used in this research.

### 3.1 Methodology Phases

The methodology comprises of five phases: dataset, data preprocessing, Deep Learning model (CNN and LSTM), 10-fold cross-validation and result as shown in Fig. 4.

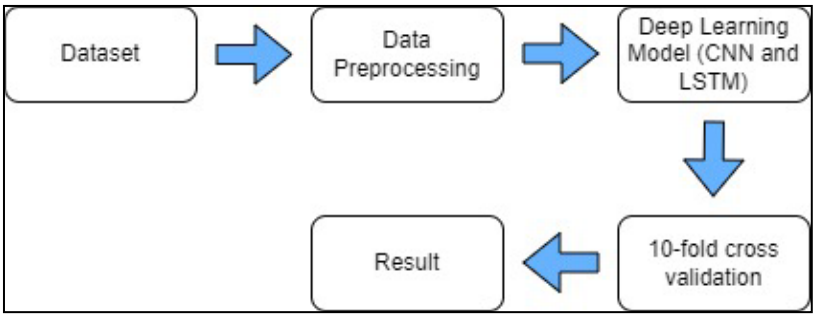


**Fig. 4** *Methodology used for Android ransomware detection*

i. Dataset

The model is tested on an Android dataset, CICAndMal2019, which contains static features: Permission and Intent and dynamic features: API calls. Initial data exploration has been conducted to get familiar with the data's structure, quality and characteristic. From the dataset, dynamic features, static features and labels are differentiated to determine the features that will be used in the data preprocessing. After removing the irrelevant and static features from the dataset, only 41 of 80 features including 'Label' will be used from this dataset.

ii. Data Preprocessing

Data preprocessing is important in this phase. Data cleaning, transformation, and reduction will be done to prepare it for analysis. Firstly, data preprocessing will be done as the dataset can have many irrelevant and missing parts. It involves checking and handling missing values.

After handling missing values, outliers in the dataset will be removed to improve the dataset's quality by removing these anomalies that could bias the model. Next, categorical data will be converted into numerical data. Normalization is done to scale the data in a specific range (0.0 - 1.0). The number of instances in which the model accurately predicted the negative class.

iii. Deep Learning Model (CNN and LSTM)

This phase can be divided into building and training the models. Through the application of convolutional layers in CNN, they are responsible for capturing spatial hierarchies in data. CNN can be used to process binary files or other malware representations. For LSTM, the model can learn long-term dependencies from the data. LSTM can also be used to analyze sequences of API calls, system events, or other temporal patterns in ransomware behavior as it is suited for sequenced data.

The training process involves feeding the preprocessed data to the models. This step's objective is to build models that can learn to recognize the distinctive characteristics and patterns of ransomware from the dataset. The models can be generalized to detect previously unseen ransomware samples by training them.

iv. 10-Fold Cross Validation

In the 10-fold cross validation phase, the dataset is divided into 10 equal parts, or "folds". Nine folds are used in the training process while the remaining one-fold is used as a testing set. This process is carried out 10 times, one test set is used for each fold. To provide final assessment for the model, the performance measured from each of the ten iterations is averaged.

v. Result

The confusion matrix is used to determine the results, including recall, accuracy, precision and F-1 score.

## 3.2 Deep Learning Model

The experiment will consist of two classifiers: Convolutional Neural Network (CNN) and Long Short-Term Memory Recurrent Neural Network (LSTM-RNN). The deep learning model is as shown in Fig. 5.

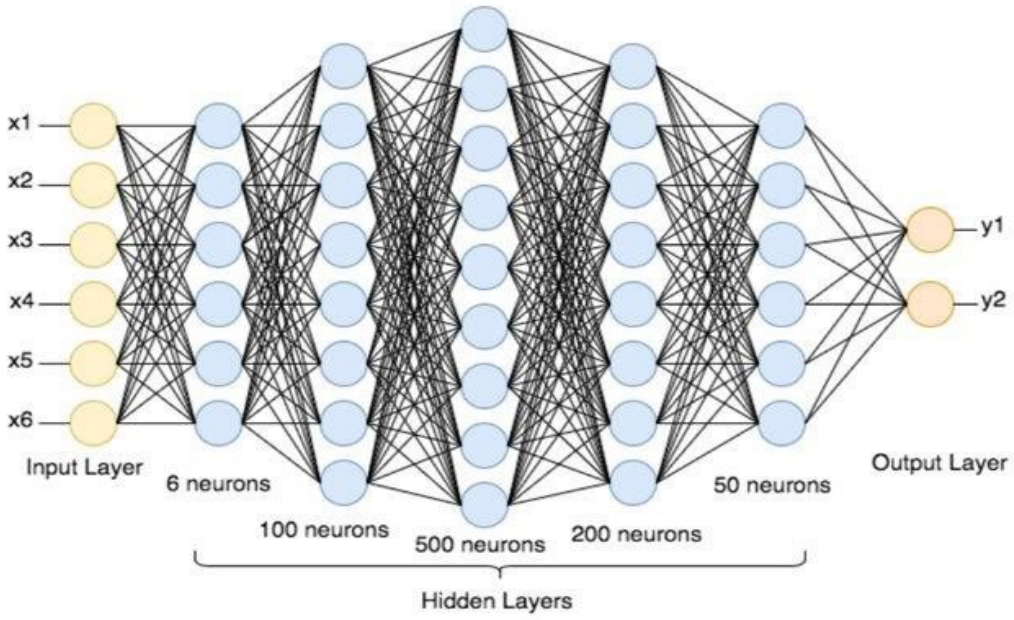


**Fig. 5** *Deep learning model*

### 3.2.1 Convolutional Neural Network (CNN)

CNN's layers architecture may be divided into three types: Convolutional Layer, Pooling Layer, and Fully Connected Layer. The Fully Connected Layer is simply operating as a neural network, which has been discussed. The CNN algorithm has two primary processes: convolution and sampling, which occur on convolutional and max-pooling layers. Every neuron receives information from a rectangle xnx portion of the preceding layer, which is referred to as the local receptive field. Equation (6) shows the mathematical formula for CNN architecture(Li et al., 2022).

$$x_{i,j}^{(l)} = \sigma(b + \sum_{r=0}^{n} \sum_{c=0}^{n} w_r, c^{x_{i+r,j+c}^{(j-1)}} \quad (6)$$

### 3.2.2 Long Short-Term Memory Recurrent Neural Network (LSTM-RNN)

Recurrent Neural Networks (RNNs) have a layer called Long Short-Term Memory (LSTM), which is applied to solve the exploding gradient problem. It moves through the entire chain using a state cell. As a result, the data related to them does not change. In addition to using input, output and forget gates, the LSTM has the ability to add and remove data from the state cell. Equations (7), (8) and (9) may give a vivid understanding of LSTM gates mention above mathematically(Fernando et al., 2020).

i. Input gate

$$it = \sigma(W[ht-1, xt] + bi) \quad (7)$$

ii. Forget gate

$$ft = \sigma(Wf[ht-1, xt] + bf) \quad (8)$$

iii. Output gate

$$ot = \sigma(W[ht-1, xt] + bo) \quad (9)$$

Where $\sigma$ represents the sigmoid function, $w_x$ represents the wight for "x" gate, $X_t$ represents input at the current timestamp, $h_{t-1}$ is for the output of the previous LSTM block and, $b_x$ is the biases for the gate "x"(Alzahrani & Alghazzawi, 2019).

## 3.3 Evaluation Metrics

The confusion matrix will be deployed to evaluate the model's performance. The classifiers are assessed using a confusion matrix for test data. The confusion matrix is a table that helps visualize this comparison. It consists of four key components:

i. True Positives (TP)
The number of instances in which the model accurately predicted the positive class.
ii. True Negatives (TN)
The number of instances in which the model accurately predicted the negative class.
iii. False Positives (FP)
The number of instances in which the model inaccurately predicted the positive class (false alarm).
iv. False Negatives (FN)
The number of instances in which the model inaccurately predicted the negative class (false alarm).

Four metrics were selected to evaluate the model, which are as follows:

i. Accuracy
Accuracy gives us an idea of the percentage of test data that the model accurately predicted out of all test data. It is calculated with Equation (10).

$$accuracy = \frac{TP + TN}{TP + TN + FN + FP} \quad (10)$$

ii. Precision
Precision, p is in charge of measuring the quality of the models and the proportion of predicted positive cases. It is calculated with Equation (11).

$$recall = \frac{TP}{TP + FN} \quad (11)$$

iii. Recall
Recall shows how much relevant data the model has been able to extract relative to the total amount of relevant data. It is calculated with Equation (12).

J. of Soft Computing and Data Mining Vol. 6 No. 1 (2025) p. 378-393

$$precision, p = \frac{TP}{TP + FP} \tag{12}$$

iv. F-1 score

F-1 score is utilized for multi-class. Once the precision and recall have been computed, the two scores will allow us to use the Equation (13) to get the f-measure.

$$f - measure = \frac{2 \times precision \times recall}{precision + recall} \tag{13}$$

## 4. Results and Discussion

This chapter discuss a detailed analysis of the research design, focusing on the implementation and evaluation results of CNN and LSTM models.

### 4.1 Dataset

CICInvesAndMal2019 dataset is retrieve from University of New Brunswick, Canadian Institute for Cybersecurity(*Investigation on Android Malware 2019 | Datasets | Research | Canadian Institute for Cybersecurity | UNB*, n.d.). This dataset includes network-flow features, comprising network traffic, logs, API/SYS calls, phone and memory statistics of benign and four malware categories. These features were extracted using CICFlowMeter. For this research, only the sample of ransomware from Jisut, Koler, Charger, Lockerpin, PornDroid, Pletor, RansomBO, Simplocker, WannaLocker and SVpeng family will be used. The data set used consisted of network traffic characteristics, packet details, and attributes related to network communication.

### 4.2 Experiment Design

This work studies Android ransomware detection using two deep learning models: Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) with different epochs (10, 20, 30, 40 and 50). Both experiments use Conv1D layers, followed by architectural components particular to each model type, such as Batch Normalization, ReLU activation, Dropout layers, and a Dense layer for classification. In this experiment, four factors stayed constant: dataset, percentage of training and testing, program code and learning rate. For information, the training percentage is 90 percent while another 10 percent is for testing. The percentage ratio has been determined by previous related work. Evaluation criteria such as accuracy, precision, recall, and F1 score will be used to compare the performance of CNN and LSTM architectures in ransomware detection. The tests are carried out using TensorFlow and Keras, to identify the most efficient model. Besides, evaluation parameters and implementation will also be discussed in this section.

### 4.3 Preprocessing

Some dataset features have been removed, such as Flow ID, Source/Destination IP and Port, Protocol, Time Stamp, various TCP flags, and aggregate statistics like Average Forward/Backward Segment Size. These elements often remain constant or provide set summaries throughout a network flow. While useful for identifying flows and providing control information, they fail to capture the dynamic character of network traffic required by deep learning models such as CNN and LSTM.

The retained features are selected because they provide valuable insights into network traffic behaviour and are more likely to capture the patterns indicative of ransomware activity. In contrast, the deleted features are either too specific, redundant, or not directly related to the detection of ransomware, which helps reduce noise and avoid overfitting. Table 3 shows the 41 out of 80 features used in this research.

**Table 3** *Features used in this research*

| No. | Category | Column No. | Features |
|---|---|---|---|
| 1 | Packet Count Features | 1-2 | Total Fwd Packets, Total Backward Packets |
| 2 | Packet Length Features | 3-12 | Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std |
| 3 | Traffic Flow Features | 13-14 | Flow Bytes/s, Flow Packets/s |
| 4 | Flow Inter-Arrival Time (IAT) Features | 15-18 | Flow IAT Mean, Flow IAT Std, Flow IAT Max, Flow IAT Min |
| 5 | Forward Inter-Arrival Time (IAT) Features | 19-23 | Fwd IAT Total, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Max, Fwd IAT Min |
| 6 | Backward Inter-Arrival Time (IAT) Features | 24-28 | Bwd IAT Total, Bwd IAT Mean, Bwd IAT Std, Bwd IAT Max, Bwd IAT Min |
| 7 | Packet Rate | 29-30 | Fwd Packets/s, Bwd Packets/s |
| 8 | Packet Length | 31-36 | Min Packet Length, Max Packet Length, Packet Length Mean, Packet Length Std, Packet Length Variance, Average Packet Size |
| 9 | Packet Header Length | 37 | Fwd Header Length |
| 10 | Flow Control | 38-40 | Init_Win_bytes_forward, Init_Win_bytes_backward, min_seg_size_forward |
| 11 | Classification Label | 41 | Label |

Preprocessing the dataset started by loading the data and then verifying for missing values to ensure data completeness. If missing values are discovered, they are handled accordingly, either by filling them in or eliminating the corresponding rows. This step is critical for maintaining the dataset's integrity and ensuring that incomplete data do not influence future analyses.

The next step is to handle any category labels in the cleaned dataset. These labels are recorded in a numerical format that machine learning algorithms normally demand. This encoding procedure converts categorical data into a format that is easily integrated into the model training process. After label encoding, any outliers in the dataset are detected and deleted by using z-score. The result is shown in Fig. 6 Outliers can have a major impact on model performance and robustness, resulting in erroneous predictions. By reducing these anomalies, we ensure that the data is more reflective of typical situations, enhancing the model's ability to generalize to new, untested data.


**(a)** **(b)**

**Fig. 6** *Outlier removal (a) Original data shape; (b) Data shape after outlier removal*

Next, Min-Max normalization is applied to scale the features within a specified range, typically 0 to 1. This normalization process helps in facilitating quicker convergence and improved accuracy during model training by making certain that makes an equal contribution to the learning process while avoiding the dominance of any one feature because of its size. Table 4 shows the first five rows and columns of the dataset features before and after preprocessing.

J. of Soft Computing and Data Mining Vol. 6 No. 1 (2025) p. 378-393

**Table 4** *Dataset contents before and after preprocessing*

| Dataset Before Preprocessing | | | | | Dataset After Preprocessing | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Total Fwd Packets: | Total Backward Packets: | Total Length of Fwd Packets: | Total Length of Bwd Packets: | Fwd Packet Length Max: | Total Fwd Packets: | Total Backward Packets: | Total Length of Fwd Packets: | Total Length of Bwd Packets: | Fwd Packet Length Max: |
| 2 | 0 | 31 | 0 | 31 | 0.0112 | 0 | 0.0024 | 0 | 0.0158 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0.0109 | 0 | 0 | 0 |
| 2 | 0 | 31 | 0 | 31 | 0.0112 | 0 | 0.0024 | 0 | 0.0158 |
| 5 | 3 | 194 | 985 | 194 | 0.0449 | 0.0326 | 0.015 | 0.0103 | 0.0989 |
| 5 | 3 | 194 | 1253 | 194 | 0.0449 | 0.0326 | 0.015 | 0.0131 | 0.0989 |

## 4.4 Model Building Structure

To have fair comparison, the models are somehow built using the same sequential API with slightly different parameters. Using sequential model allows us to have more layers, as it is formally defined as stack of layers. The shared parameters for both models include a batch size of 64 and an input shape defined by the training data dimensions, specifically (X_train_cnn.shape[1], 1) for the CNN and (X_train_lstm.shape[1], 1) for the LSTM. Both models use the Adam optimizer with a learning rate specified as 'adam' and employ the binary cross-entropy loss function. The activation function for both models is 'relu', and a dropout rate of 0.5 is applied to avoid overfitting. The models are trained for 10 to 50 epochs, with the verbosity set to the default value of 1.

## 4.5 Model Implementation

The experiment set up for the two algorithms in this deep learning model, namely Convolution Neutral Network (CNN) and Long Short-Term Memory network (LSTM), are described in depth in this part. Both methods begin by importing libraries and mounting Google Drive to retrieve the cleaned dataset.

### 4.5.1 CNN Model

The first model, CNN, was created using the 'tf.keras.Sequential' API, to allow the building of linear stack of layers. This model consists of 7 layers:

i. Input layer
The input layer consists of a conv1D layer with 64 filters, a kernel size of 3 and ReLU activation. This layer processes the input data by applying convolutional operations along one spatial dimension. Additionally, the input shape is set as '(X_train.shape[1], X_train.shape[2])', where 'X_train.shape[1]' is the number of features and 'X_train.shape[2]' is 1 (which is added for compatibility with conv1D)

ii. Batch normalization layer
This layer is used to enhance training efficiency and stability by normalizing the activations of the previous layer.

iii. Additional conv1D layer and batch normalization layer
These layers will continue to apply convolutional operations as in layer i and ii, allowing the model to learn more complex features.

iv. Flatten layer
This layer flattens the output of the convolutional layers into a 1D vector, which then can be fed into the dense (fully connected) layer.

v. Dense (fully connected) layer
The fully connected layer is set with 100 neurons and ReLU activation. This layer learns high-level representations from the flattened features.

vi. Dropout layer
In this layer, 0.5 dropout regularization is done to avoid overfitting. 50% of the neurons will be randomly dropped.

vii. Output layer
The output layer is set with a single neuron and sigmoid activation, which is suitable for binary classification tasks.

Lastly, the Adam optimizer and binary cross-entropy loss function are used to compile the model.

### 4.5.2 LSTM Model

Next, the 'tf.keras.Sequential' API is used to generate the LSTM model. The are three layers in this model:

i. Input layer and LSTM layer
The LSTM layer is set with 100 units, ReLU activation and L2 regularization. This layer is used as it is well suited for processing sequences of data and they can learn long-term dependencies. The input shape is specified as '(X_train_lstm.shape[1], 1)', where 'X_train_lstm.shape[1]' is the number of features and 1 is the number of time steps (reshaped from the original features).

ii. Dropout layer
In this layer, 0.5 dropout regularization is done to avoid overfitting. 50% of the neurons will be randomly dropped.

iii. Output layer
The output layer is set with a single neuron and sigmoid activation, which is suitable for binary classification tasks.

Lastly, the Adam optimizer and binary cross-entropy loss function are used to compile the model

## 4.6 10-Fold Cross-Validation

10-fold cross-validation is used to evaluate the model performance. By splitting data into ten equal pieces, this method will train and verified the model for ten times. The first fold is used as the training set while the remaining nine folds is used as training set. Different epochs (10, 20, 30, 40 and 50) will be applied with this method. The epochs will allow the model to learn from every data point in the training dataset and help it perform better over time. There are six steps involved in 10-fold cross-validation:

i. Setup StratifiedKFold
StratifiedKFold guaranteed each fold contains the same proportion of samples from each target class as in The dataset. The data is shuffled before splitting into 10 folds in order to enhance the validation process. 'random_state=42' is applied in this step to ensure reproducibility by controlling the number generator.

ii. Initialize metric lists
The metric lists: accuracy, precision, recall and F-1 score is initialized for training and testing to store the evaluation for each fold.

iii. Define learning rate
To prevent overfitting and minimize the loss function, the learning rate is set to 0.001. This step can balance convergence and stability during training to achieve optimal performance in deep learning model.

iv. Iterate over each fold
The data is divided into sets of training and testing for each fold using indices provided by 'StratifiedKFold'. The data is also reshaped to be compatible with the CNN and LSTM models. In this step, the epochs are set to 10 for each model. After all the steps (i to vi) are completed, the epochs are then changed to 20, 30, 40 and 50.

v. Evaluate the model
Predictions are made on the training and testing sets for model evaluation. The predictions are then converted to binary values (0 to 1) using 0.5 threshold. Average metrics across ten folds are calculated.

vi. Print metrics for each fold
These aggregated metrics provide the summary of the model's performance with its standard deviations, which indicate how the model's efficiency varies among folds.

These implementations demonstrate how to create, train, and evaluate CNN and LSTM models for network traffic classification, from loading the data and evaluating the model inside a cross-validation framework. Dropout regularization and different epochs are used to increase model performance while avoiding overfitting.

## 4.7 Results and Discussion

This research initially was to compare two deep learning algorithms: CNN and LSTM with different epochs: 10, 20, 30, 40 and 50 in detecting Android ransomware using the CICInvesAndMal2019 dataset. Table 5 shows the result of CNN model after applying different epochs.

J. of Soft Computing and Data Mining Vol. 6 No. 1 (2025) p. 378-393

**Table 5** *Results for CNN after applying different epochs*

| Epochs | Accuracy | Precision | Recall | F-1 Score |
|---|---|---|---|---|
| 10 | 97.73% | 97.77% | 99.95% | 98.85% |
| 20 | 97.76% | 97.81% | 99.95% | 98.87% |
| 30 | 97.85% | 97.91% | 99.93% | 98.91% |
| 40 | 97.93% | 98.00% | 99.93% | 98.95% |
| 50 | 97.84% | 98.04% | 99.80% | 98.91% |

Based on Table 4, the results from the CNN model were evaluated across several epochs. The result shows a high-performance level and proves the model's efficiency in detecting data patterns. The model's accuracy remains consistently high across epochs (10, 20, 30, 40 and 50), beginning with 97.73% for 10 epochs slightly increasing to 97.93% at 40 epochs until a little drop to 97.84% at 50 epochs. This shows that the model is well-tuned and benefits from additional epochs until a point is reached at which accuracy improvements plateau or slightly decline, indicating potential overfitting.

Precision remains robust across all epochs, starting at 97.77% and proceeding to increase to 98.04% after 50 epochs. This constant high precision suggests that the model successfully minimizes false positives. Similarly, recall values begin at 99.95% for 10 epochs, 99.93% by 30 and 40 epochs, before dropping slightly to 99.80% at 50 epochs. This high recall illustrates the model's effective limitation of false negatives while capturing all relevant instances.

Precision and recall are combined into a single number, F-1 score demonstrates consistently strong performance. It starts at 98.85% after 10 epochs and rises to 98.95% after 40 epochs, closely matching the trends seen in both precision and recall. The results for LSTM are shown in Table 6.

**Table 6** *Results for LSTM after applying different epochs*

| Epochs | Accuracy | Precision | Recall | F-1 Score |
|---|---|---|---|---|
| 10 | 97.74% | 97.74% | 100% | 98.86% |
| 20 | 97.74% | 97.74% | 100% | 98.86% |
| 30 | 97.74% | 97.74% | 100% | 98.86% |
| 40 | 97.74% | 97.74% | 100% | 98.86% |
| 50 | 97.74% | 97.74% | 100% | 98.86% |

Based on Table 5, as training goes through 10 epochs, performance shows high results across all metrics. The model results in 97.74% accuracy, 97.74% precision, 100% recall, and 98.86% F-1 score. This rapid improvement suggests that the LSTM model effectively learns patterns in the training data, improving its capacity to distinguish between the dataset's classes.

From epoch 10 onwards, the performance metrics remain stable until epoch 50. The accuracy, precision, recall and F-1 score remain at high levels of 97.74%, 100% and 98.86% respectively. This consistency indicates that the model has reached its peak performance and will no longer improve with subsequent training epoch. The 100% recall rate suggests the model can detect all positive classes.

Overall, these findings for LSTM model are quite effective for the CICInvesAndMal2019 dataset, obtaining high performance early in the training phase and sustaining high level of accuracy, precision, recall and F-1 score throughout consecutive epochs. This efficiency can be due to the model's ability to detect temporal connections and patterns in sequential data, which is the strength of LSTM model when dealing with sequence data. Through experiments with different epochs, CNN prove higher accuracy at 97.84% with 40 epoch.

The comparison between the results of this research and other existing work is shown in Table 6.

**Table 7** *Comparison between this research and other existing work*

| Work by | Dataset | Algorithm | Approach | Accuracy Result |
|---|---|---|---|---|
| E. C. Bayazit, O. K. Sahingoz, and B. Dogan(Bayazit et al., 2022) | CICInvesAndMal2019 | Bi-LSTM | Deep Learning | 98.85% |
| M. AlJamal, R. Alquran, I. AL-Aiash, A. Mughaid, S. AlZu'bi, and A. A. Abutabanjeh(AlJamal et al., 2023) | CICInvesAndMal2019 | LSTM | Deep Learning | 97.08% |
| A. S. Shatnawi, Q. Yassen, and A. Yateem(Shatnawi et al., 2022) | CICInvesAndMal2019 | Support Vector Machine (SVM)<br>K-nearest neighbors (KNN)<br>Naïve Bayes (NB) | Machine learning | 94.36%<br>93.42%<br>84.33% |
| Android Ransomware Detection by Deep Learning | CICInvesAndMal2019 | CNN<br>LSTM | Deep Learning | 97.93%<br>97.74 |

## 5. Conclusion

This chapter will cover three primary topics: the research's strengths and shortcomings and future studies. This study considered two deep learning algorithms: CNN and LSTM. The two algorithms were to be compared with different epochs (10, 20, 30, 40 and 50) using four assessment metrics: accuracy, precision, recall, and F-1 score.

This research has achieved all the objectives that were set at the beginning. It created a deep learning framework for Android ransomware detection, finding the optimal epochs and evaluating the model using accuracy, precision, recall and F-1 score as parameter evaluations. Using the CICInvesAndMal2019 dataset, a thorough comparison of the CNN and LSTM algorithms with different epochs demonstrated good performance, with CNN obtaining accuracy slightly higher than LSTM. The CNN model performed best at 40 epochs, with 97.93% accuracy, 98.00%precision, 99.93% recall and 98.95% F-1 score. The LSTM model performed best at 10 epochs, with 97.74% accuracy, 97.74 % precision, 100% recall and 98.86% F-1 score.

The CNN model's ability to capture spatial hierarchies in data may have contributed to its better performance over epochs than the LSTM model, which excels in sequential data processing but achieve optimal results at fewer epochs. The findings show that, while CNN model can improve with additional training, LSTM model may achieve optimum performance sooner, thus providing a faster yet more effective alternative for ransomware detection. To summarize, this research found that advanced deep learning approaches, notably CNN and LSTM are extremely good at detecting Android ransomware.

### 5.1 Future Work

The results of this study can be expanded upon in several ways for future research. One possible approach is to improve the dataset by adding more recent and varied samples of ransomware and other malware variants. To offer a more thorough comprehension of ransomware activity, an alternative strategy would be combining dynamic analysis features with static ones by using sophisticated feature extraction methods to record real-time data exchanges. Furthermore, investigating hybrid approach that integrate the benefits of CNNs and LSTMs may increase detection rates.

## Acknowledgements

This work was supported by the Universiti Tun Hussein Onn Malaysia (UTHM) through Tier1 (vot Q508).

## Conflict of Interest

The authors declare that there is no conflict of interest regarding the publication of the paper.

## Author Contribution

*The authors confirm contribution to the paper as follows:* ***experiments process****: Nuraini Syahirah Binti Abd Rais, Cik Feresa Binti Mohd Foozy;* ***experiments planning****: Cik Feresa Binti Mohd Foozy;* ***analysis the existing studies*** *Nuraini Syahirah Binti Abd Rais;* ***result and validation****: Cik Feresa Mohd Foozy, Andi Maslan. All authors reviewed the results and approved the final version of the manuscript.*

## References

[1] Al-rimy, B. A. S., Maarof, M. A., Zainudeen, S., & Shaid, M. (2018). Ransomware threat success factors , taxonomy , and countermeasures : A survey and research directions. *Computers & Security*, *74*, 144–166. https://doi.org/10.1016/j.cose.2018.01.001

[2] AlJamal, M., Alquran, R., AL-Aiash, I., Mughaid, A., AlZu'bi, S., & Abutabanjeh, A. A. (2023). A Novel Machine Learning Cyber Approach for Detecting WannaLocker Ransomware Attack on Android Devices. *2023 International Conference on Information Technology (ICIT)*, 135–142. https://doi.org/10.1109/icit58056.2023.10226130

[3] Alsoghyer, S., & Almomani, I. (2019). Ransomware detection system for android applications. *Electronics (Switzerland)*, *8*(8), 1–36. https://doi.org/10.3390/electronics8080868

[4] Alzahrani, N., & Alghazzawi, D. (2019). A review on android ransomware detection using deep learning techniques. *11th Internationa[1] N. Alzahrani and D. Alghazzawi, "A Review on Android Ransomware Detection Using Deep Learning Techniques," 11th Int. Conf. Manag. Digit. Ecosyst. MEDES 2019, No. November 2019, Pp. 330–335, 2019, Doi: 10.1145/3297662.3365785.l Confere*, *November 2019*, 330–335. https://doi.org/10.1145/3297662.3365785

[5] Aslan, Ö. (2020). A Comprehensive Review on Malware Detection Approaches. *IEEE Access*, *8*, 6249–6271. https://doi.org/10.1109/ACCESS.2019.2963724.

[6] Bayazit, E. C., Sahingoz, O. K., & Dogan, B. (2022). A Deep Learning Based Android Malware Detection System with Static Analysis. *HORA 2022 - 4th International Congress on Human-Computer Interaction, Optimization and Robotic Applications, Proceedings*, 1–6. https://doi.org/10.1109/HORA55278.2022.9800057

[7] Fernando, D. W., Komninos, N., & Chen, T. (2020). A Study on the Evolution of Ransomware Detection Using Machine Learning and Deep Learning Techniques. *Internet of Things*, *1*(2), 551–604. https://doi.org/10.3390/iot1020030

[8] Florida, S., San, T., & Carolina, S. (2022). *Ransomware Detection and Classification Strategies*. 316–324. https://doi.org/10.1109/BlackSeaCom54372.2022.9858296

[9] Gohari, M., Hashemi, S., & Abdi, L. (2021). Android Malware Detection and Classification Based on Network Traffic Using Deep Learning. *2021 7th International Conference on Web Research, ICWR 2021*, 71–77. https://doi.org/10.1109/ICWR51868.2021.9443025

[10] Hemalatha, J., Roseline, S. A., Geetha, S., Kadry, S., & Damaševičius, R. (2021). An efficient densenet-based deep learning model for Malware detection. *Entropy*, *23*(3), 1–23. https://doi.org/10.3390/e23030344

[11] *Investigation on Android Malware 2019 | Datasets | Research | Canadian Institute for Cybersecurity | UNB*. (n.d.). https://www.unb.ca/cic/datasets/invesandmal2019.html

[12] Li, Z., Liu, F., Yang, W., Peng, S., Zhou, J., & Member, S. (2022). A Survey of Convolutional Neural Networks : Analysis , Applications , and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(12), 6999–7019. https://doi.org/10.1109/TNNLS.2021.3084827

[13] Liu, Y., & Wang, Y. (2019). A robust malware detection system using deep learning on API calls. *Proceedings of 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2019*, *Itnec*, 1456–1460. https://doi.org/10.1109/ITNEC.2019.8728992

[14] Naway, A., & Li, Y. (n.d.). *A Review on The Use of Deep Learning in Android Malware Detection*. 1–15.

[15] Rahima Manzil, H. H., & Naik, S. M. (2023). Android ransomware detection using a novel hamming distance based feature selection. *Journal of Computer Virology and Hacking Techniques*. https://doi.org/10.1007/s11416-023-00495-w

[16] Sgandurra, D., Muñoz-González, L., Mohsen, R., & Lupu, E. C. (n.d.). *Automated Dynamic Analysis of Ransomware : Benefits , Limitations and use for Detection*.

[17] Shatnawi, A. S., Yassen, Q., & Yateem, A. (2022). An Android Malware Detection Approach Based on Static Feature Analysis Using Machine Learning Algorithms. *Procedia Computer Science*, *201*(C), 653–658. https://doi.org/10.1016/j.procs.2022.03.086

[18] Taheri, L., Fitriah, A., Kadir, A., Lashkari, A. H., Fitriah, A., & Unb, A. H. L. (2019). *Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls*. *Cic*.

[19] Wang, Z., Liu, Q., & Chi, Y. (2020). Review of Android Malware Detection Based on Deep Learning. *IEEE Access*, *8*. https://doi.org/10.1109/ACCESS.2020.3028370

[20] Xu, A., Chen, L., Kuang, X., Lv, H., Yang, H., Jiang, Y., & Li, B. (2020). A Hybrid Deep Learning Model for Malicious Behavior Detection. *Proceedings - 2020 IEEE 6th Intl Conference on Big Data Security on Cloud, BigDataSecurity 2020, 2020 IEEE Intl Conference on High Performance and Smart Computing, HPSC 2020 and 2020 IEEE Intl Conference on Intelligent Data and Security, IDS 2020*, 55–59. https://doi.org/10.1109/BigDataSecurity-HPSC-IDS49724.2020.00021