
**King Khalid University**

**Deep Learning based Malware Detection System**

**By**

**Leen Yahya Alsharafi 441801171**

**Maha Nasser Asiri 441801191**

**Razan Nazieh Khalaf 441813462**

**Waad Abdulah Mohammed 441801189**

**Supervised by:**

**Dr. Ali Mohammed Alqahtani**

# TABLE OF CONTENTS

# TABLE OF TABLES



# TABLE OF FIGURES

# ABSTRACT

Malicious software (malware) continues escalating and poses a significant security concern in the digital age. Exponential growth in malware attacks has influenced many computer users, corporations, and governments. Malware detection continues to be an open research topic. Current malware detection approaches are adopting static and conventional methods, which are time-consuming and have proven to be ineffective in identifying unknown malware in real-time. The use of deep learning has grown increasingly in recent years, thereby becoming a much-discussed topic across a diverse range of fields, especially in pattern recognition. This work will first introduce a new novel deep learning approach to classify malicious software. Empirically, the efficiency of the proposed methods are demonstrated on several public datasets compared to the previous state-of-the-art methods. Later, the proposed method will be deployed into a deep learning web-based system, which could provide an effective, general, and scalable mechanism for the detection of existing and unknown malware in an executable file.

# CHAPTER 1. INTRODUCTION

## 1.1 INTRODUCTION

Although the internet is an essential part of our life, its security is seriously threatened by malicious software (malware). This malware is increasing at an alarming rate and malware detection through standard signature-based methods is getting more and more difficult and inaccurate since all the current malware applications tend to avoid detection because this modern-day malware comes with intelligent approaches such as code encryption, obfuscation, and polymorphism. Furthermore, anyone with bad intent can modify malware code that is made public and add more features to create new malware even without knowing much programming. This allows the attackers to recreate more sophisticated versions of pre-existing malware. This project is concerned with the development of a deep learning web-based system for malware detection in Portable Executable files (PE file) which is run on Windows operating systems.

## 1.2 MOTIVATION

The majority of attacks target Windows, and PE files are a form of file that can be executed on Windows; for this reason, the PE files were selected as the target of the project.

Beyond that, Windows is the operating system of choice for more than 70% of users.

In addition, as malware evolved from simple to sophisticated code that can professionally evade detection, and as malware authors evolved too, there is a necessity need for a method that can efficiently use deep learning to detect malware on PE files.

## 1.3 PROBLEM STATEMENT

With the growth of technology, the number of malware is also increasing. Malware now is designed with mutation characteristics that cause enormous growth. Traditional signature-based malware detection is proven to be ineffective against the vast variation.

to understand the traditional approach, Signature is a short sequence of bytes, often distinctive to a malware. A database of predefined signatures is used by signature-based detection techniques. This kind of approach is effective with known malware but fails with new malware.

## 1.4 PROJECT OBJECTIVES

### 1.4.1 Aim

The project aims to develop a system that detects and alarms whether the portable executable file is malicious or benign using an effective approach to detection.

### 1.4.2 Objectives

- To provide a User Interface that enables the users to upload .exe files.
- To extract important features from the header of the PE files using DL.
- To use Deep Learning techniques for classification.
- To detect new malware that is unknown before like: zero-day malware.
- To make the malware detection process easier and more effective without any manual process.
- To achieve high accuracy and minimize the number of False Positive (FP) and False Negative (FN) rates.

## 1.5 PROJECT SCOPE

This project is a graduation project, as a part of the graduation requirements of King Khalid University, at Computer Science College. This project is planned to use Neural Networks and DL which allow computer programs to recognize patterns and solve common problems. The project is focused on the field of Conventional Neural Networks (CNN) as its high capability of achieving difficult classification tasks. According to the datatype that will be used in this project, Portable Executable files will be examined and the system will focus more on the header part of these files while extracting the features

## 1.6 PROPOSED SOLUTION

The proposed solution is basically to create a technique using a deep learning approach to classify malware executable files, based on the information in the Portable Executable header of these files. Figure1, present the diagram of the proposed deep learning based malware detection system.

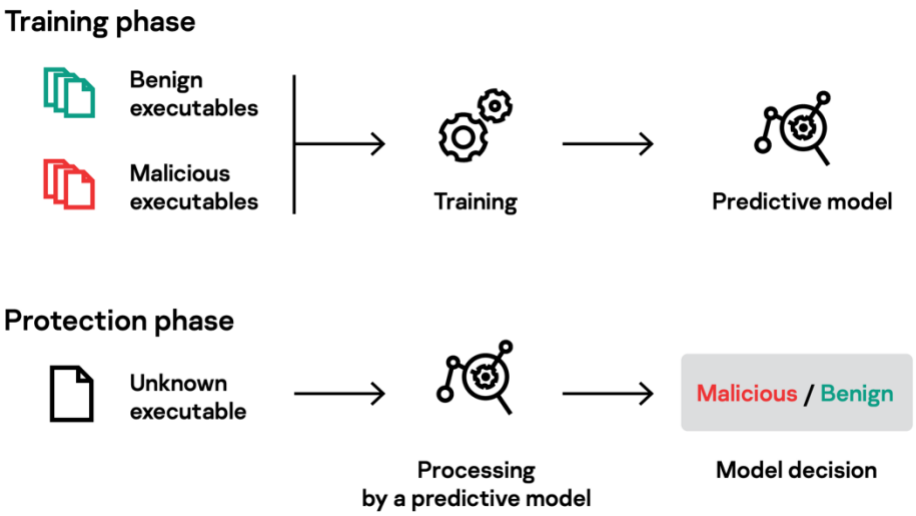


*Figure 1 : General structure of the malware detection model [8].*

The following points have been noted from the above figure (Figure1) :

- The model takes benign and malicious executable files as input.
- The deep learning model is trained on a training dataset based on the features selected.
- The above process will generate a system that can predict whether the unknown executable is benign or malicious and produce the result.

## 1.7 PROJECT CONTRIBUTION

After the project is completed, its capabilities will be focused in detecting unknown malware, such as polymorphic viruses and zero-day malware, which are viruses that can evolve and change their signature in order to avoid detection by particular systems. The system will also have the ability to classify the portable executable file as harmful or benign and effectively detect several malware. The project will be carried out in several disciplines by either individuals or groups, including universities or businesses, etc.

## 1.8 SUMMARY

To sum up, the aim of this project is to build a deep learning model for detecting malware in the portable executable files that exist on windows operating systems and specifically the PE header part will be examined. The model will be trained on a dataset that contains several features extracted from the PE header and these files can be benign or malicious. So the model will learn

both the features of these two types of files. Then, the model will be tested by providing test data and then calculating the accuracy. After that, building a website to enable the users to upload the executable file that is to be checked through User Interface so the user can get the result easily from the website.

# CHAPTER 2. BACKGROUND AND LITERATURE REVIEW

## 2.1 OVERVIEW

This chapter describes the literature review and basic information according to the project.

Section 2.2 describes background information related to the work of the project. Section 2.3 discusses the literature review. Section 2.4 describes existing malware detection systems. Section 2.5 discusses the malware detection algorithms.

## 2.2 BACKGROUND

In this section, basic information related to the work of this project will be explained.

### 2.2.1 Malware:

Malware is a combination of two words “malicious" and "software”. It refers to any parasitical and destructive software developed by hackers with the goal of stealing data, damaging or destroying computers and computer systems. Examples of common malware such as: viruses, worms, Trojan viruses, spyware, adware, ransomware, and zero-day malware which is a virus that is not recognized before in computer systems, and which this project is trying to detect.

### 2.2.2 Executable File:

An executable file (EXE file) is a type of file of a computer system that contains an encoded sequence of instructions that the system will execute directly when the user clicks on the file icon. Executable files commonly have an .exe file extension, but there are hundreds of other executable file formats.

### 2.2.3 Executable File Header (PE header ) :

Executable File Header is part of the Executable file which contains metadata and all of the important and necessary information required by the OS (Windows) to execute the executable file such as the architecture of the file's code, the signature, the time stamp related to file itself, and various flags. It specifies where the execution begins and these all are important for the detection.

### 2.2.4 Deep Learning Approach:

Deep Learning is part of Artificial intelligence and a subset of machine learning approach, that facilitates the extraction of features. Deep Learning is using Neural Networks. These neural networks simulate the human brain which consists of a number of cells or neurons that process information by sending and receiving signals. Deep neural network learning consists of a set of layers which communicate with each other and process information and Perform arithmetic operations between layers. The word "Deep" in Deep Learning points out to the number of layers in the network; more the number of layers means deeper is the network.

### 2.2.5 Deep Learning Model Overview:

Figure 2 shows the schema of a deep learning based malware detection system that summarizes the basic operations the system will perform to achieve the goal.

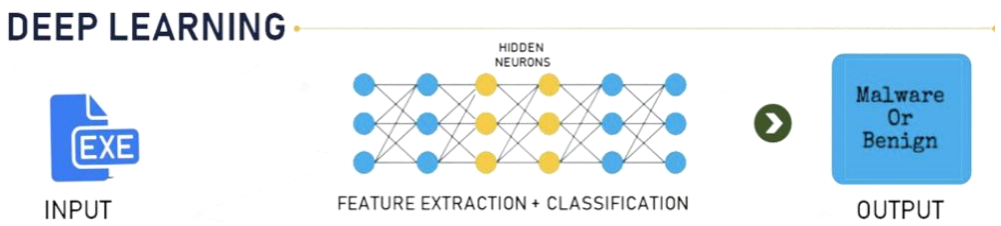


*Figure 2 : Schema of malware detection system .*

The scheme can be divided into three main phases: (input data phase, feature extraction phase, and classification phase ).

- **Input data phase:** at this phase , the data from which the deep learning model learns will be entered in the form of a comma-separated values (csv) file.
- **Feature extraction phase**: that is the process of extracting meaningful attributes from a large number of columns of the dataset and taking a subset of relevant features. Also, label the columns and drop unimportant ones.
- **Classification phase:** build and train the model. Train the built model based on a specific algorithm to be able to distinguish between malicious and benign files according to the features extracted in the previous phase.

### 2.2.6 Convolutional Neural Networks algorithm :

Convolutional Neural Networks also known as (CNNs) come under the umbrella of Deep Learning, it is a subset of machine learning that uses multi-layered artificial neural networks to deliver state-of-the-art accuracy in tasks. which developed for processing data with a grid-structured topology and perform difficult classification tasks without the need for feature extraction.

## 2.3 LITERATURE REVIEW

Table 1 below shows a description of some previous papers that shows malware detection systems based on a deep learning approach:

---

**I. Windows PE Malware Detection Using Ensemble Learning [1].**

This system was released in Aug 2021, the model used in this system is Convolutional Neural Network (CNN), the input data type is PE headers and the number of samples is 19,611 samples.

A hybrid ensemble learning framework consisting of fully connected and convolutional neural networks (CNNs) with the Extra Trees classifier as a meta learner for malware recognition.

---

**II. Malware Detection using Machine Learning and Deep Learning [2].**

This system was released in April 2019, the model used in this system is CNN, input data type is .asm data, and the number of samples used is 11,688 samples.

Modeling malware analysis and detection as machine learning and deep learning problems and comparing the models built using Random Forest (RF) and Deep Neural Network (DNN) (DNN-2L, DNN-4L, DNN-7L).

---

**III. An investigation of a deep learning based malware detection system [3] .**

This system was released in Sep 2018, the model used in this system is CNN, the input data type is .asm data, and the number of samples used is 13800 samples.

Using deep learning for detecting the malware file using Neural Network with 2-4-7 hidden layers and 1-layer & 3-layer (Stacked) Auto-Encoder.

---

| | |
|---|---|
| **IV.** | **DL4MD: A Deep Learning Framework for Intelligent Malware Detection [4].** This system was released in 2016, the model used in this system is Stacked Auto Encoders (SAEs), input data type used is Windows Application Programming Interface (API) called data, the number of samples is 50,000 samples . A deep learning architecture using the stacked Auto Encoders (SAEs) model for malware detection with the input resting on Windows Application programing Interface (API) calls extracted from the (PE) files**.** |

*Table 1 : Literature review of some previous papers for malware detection systems .*

## 2.4 EXISTING MALWARE DETECTION SYSTEMS

### 2.4.1 Technologies involved in malware detection systems.

- **Signature-Based Detection** : Signature-based detection uses a text called signature, which is a distinct digital trace that programs put in. The programs are scanned by antivirus applications, which then analyze their signatures and compare them to those of known malware.
- **Behavior-Based Detection:** Behavior-Based Detection evaluates software based on intended actions. The behavior of the program is analyzed by running it to look for suspicious activities.
- **Static-Based Detection:** The idea of Static-Based Detection is to observe the features of the file without running it, try to study these features, analyze them and draw conclusions without doing anything to the executable file.

### 2.4.2 Algorithms involved for malware detection systems.

Different technologies are used to build malware detection systems such as :

- **SAEs** (Stacked AutoEncoders): Auto-Encoders: In deep learning auto-encoders are unsupervised learning methods which require only feature vector (opcode frequency), and not class labels for dimensionality reduction.
    a. A single layer auto-encoder (Non Deep Auto-Encoder), also referred to as AE-1 L which contains one encoder layer and a decoder layer.
    b. A 3-layer stacked auto-encoder(Deep Auto-Encoder), also referred to as AE-3L which contains three encoders followed by three decoders.

- **CNN** (Conventional Neural Network): A hybrid ensemble learning framework consisting of fully connected and Convolutional Neural Networks (CNNs) with the Extra Trees classifier as a meta learner for malware recognition in addition to a comprehensive study of the performance of classifiers for selecting the components of the framework.
- **DNN** (Deep Neural Network): They used different architectures of Auto-Encoders (AE) for feature extraction (along with scaling the data) and Deep Neural Network (DNN) for classification with fully connected 3-layer for feature extraction.
- **Random Forest**: Random Forest is a Supervised Machine Learning Algorithm that is used widely in classification and Regression problems .On various samples, it generates decision trees and takes their average for classification and the majority vote for regression.

### 2.4.3 Datatype used for malware detection systems.

- **PE headers** (Portable Executable headers): Files contain all of the important and necessary information required by the operating system, as exist in section 2.2 .
- **.asm file** (Assembly source code): Low-level source code that can be converted straight into machine code that is contained in an assembly source code file.
- **.bytes file** (Unity Binary Text): It is a digital way to represent executable files with important and necessary information.

## 2.5 MALWARE DETECTION SYSTEM

### 2.5.1 System model specification:

By reviewing previous research and evaluating previous systems for detecting malware, the specifications of this project can be determined as follows:

- **Approach**: Deep Learning approach will be used in this project. The main reason is that DL does not require manual feature extraction and it is based on learning, so it is the suitable method when dealing with datasets that have a large number of features.
- **Algorithm**: One Dimensional Convolutional Neural Network (1D CNN) algorithm will be used. The reason for choosing this algorithm is that CNN is highly capable of achieving challenging classification tasks, with its effectiveness at learning local features. Also, it is suitable for the dataset that we are using.

- **Datatype**: PE header data type is used as they contain all the important and necessary information required by the OS to execute the file, and the PE file is a type of files which can be executed on windows.

### 2.5.2 Algorithm used in the system :

**Algorithm used in the system: CNN algorithm**

Convolutional Neural Networks (CNNs) come under the umbrella of Deep Learning, a subset of machine learning that uses multi-layered artificial neural networks to deliver state-of-the-art accuracy in tasks. CNN has different types of layers, including the **1D-Convolutional layer** which extracts features by applying dot products between transformed waves and a 1D learnable kernel (filter), computing the output of neurons that are connected to local temporal regions in the input. CNN also has a layer called **Fully connected layer** in which the neuron applies a linear transformation to the input vector through a weights matrix. A non-linear transformation is then applied to the product through a non-linear activation function. CNN also has a number of important features, including:

- **Filter sizes:** it is a part of network architecture **that** moves locally on 1D data and contains trainable weights.
- **Nodes:** set of layers which communicate with each other, process information, and perform arithmetic operations between layers.
- **Dropout :** to refers for dropping out the nodes (input and hidden layer) in a neural network.

Figure 3 illustrates how the filter will move on CNN models . Each row represents a series for some axis. The filter can only move in one dimension along the axis of time.

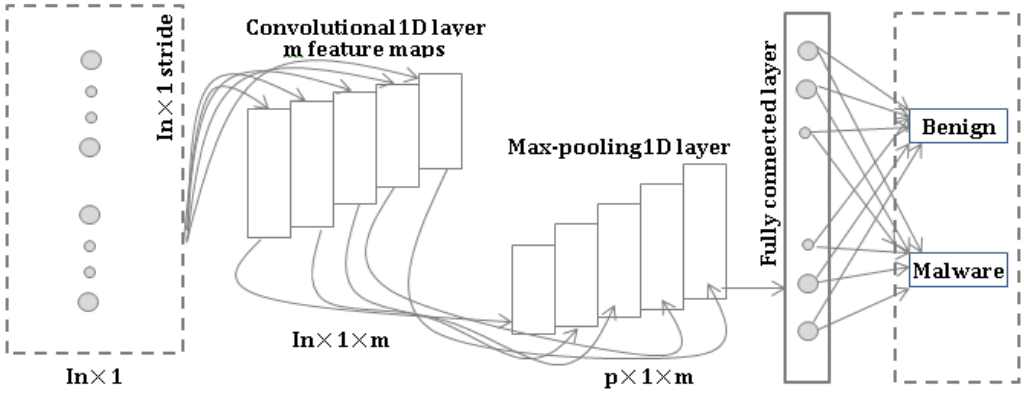


*Figure 3 : Convolutional Neural Networks (CNNs) architecture [9].*

In figure 3, m is the number of filters, In is the input of features and p denotes the reduced feature dimension. This network consists of one convolution 1D layer, one pooling 1D layer and one fully connected layer. In the first layer, the filters slide over the sequence data which is in one dimension and extract the features. Then, the dimension of the optimal features are reduced using the pooling layer. Lastly, the fully connected layer is responsible for the classification. In short , 1D CNN will be used in the project with fully connected layers and number of filter and size, The layers are arranged in such a way so that they detect pattern of linear data with the help of the dropout method , and going to flatten the feature map into a column.

## 2.6 SUMMARY

In summary, the system will be a deep learning based that uses the 1D CNN algorithm on PE header extracted files in order to operate like a warning system, and as everyone knows the evolution of viruses is increasing day by day, so there is a need for a system that detects and alerts these viruses efficiently.

# CHAPTER 3. SYSTEM ANALYSIS

## 3.1 OVERVIEW

This chapter includes the analysis of the proposed system, Sections 3.2 and 3.3 will explain the system requirements. Sections 3.4 will illustrate the architecture system schema and scenarios in the form of use case diagram, class diagram, activity diagram and sequence diagram . Section 3.5 will show a prototype interface for the system.

## 3.2 SYSTEM REQUIREMENTS

This section focuses on the functional and non-functional requirements of malware detection systems.

### 3.2.1 Functional Requirements

The features that the system will need in order to deliver or function are known as functional requirements. In the case of this project, it was crucial to collect specific needs, which are listed here, in order to meet the previously stated goals.

I. The user must be able to upload a .exe file.
II. The system must be able to extract features from the PE header of the file.
III. The system must be able to detect malware files
IV. The system must be able to detect Zero-Day malware.

### 3.2.2 Non-Functional Requirements

Non-functional requirements outline the system constraints which affect how the system should work. They include the extra anticipated aspects of the system, such as its usability, security, and speed. The following are the Non-functional requirements that the system must meet.

I. The system must show the result within a few seconds.
II. The system must be easy to use.
III. The system must be safe and reliable.
IV. The system must be able to achieve high accuracy and minimize the number of False Positives

## 3.3 SOFTWARE AND HARDWARE REQUIREMENT:

### 3.3.1 Hardware Requirements:

The system can run on a computer with specifications not less than

I. CPU Intel Core i5.
II. RAM 3 GB.
III. HDD 500 GB.
IV. OS Win 7.

### I.3.2 Software requirements:

I. Platform
   a. Google Colaboratory ( Google Colab ) [11] .
   b. Python Programming Language with version 3.4.3 or higher [12] .

II. Packages:
   a. TensorFlow : end-to-end machine learning platform [13].
   b. Keras : high-level API of TensorFlow [14]
   c. NumPy : library used to work with arrays [15].
   d. Pandas : library used for data analysis and manipulation tool [16].
   e. Pefile : Portable Executable reader library .[17]

III. Dataset
   a. Classification of Malwares (CLaMP) dataset [5] from MendeleyData source, contains 5210 file samples of PE header data as .csv file .
   b. Benign & Malicious PE Files dataset [6] from Kaggle platform contains 19612 samples of PE header data as .csv file .
   c. MalwareDataSet dataset [7] from GitHub platform contains a total of 137,444 samples of PE header data as .csv file .

IV. HTML, CSS, JavaScript.

## 3.4 DEVELOPMENT METHODOLOGY:

### 3.4.1. Project schema

The schema below in figure 4, shows the system architecture of a malware detection project using a deep learning algorithm.

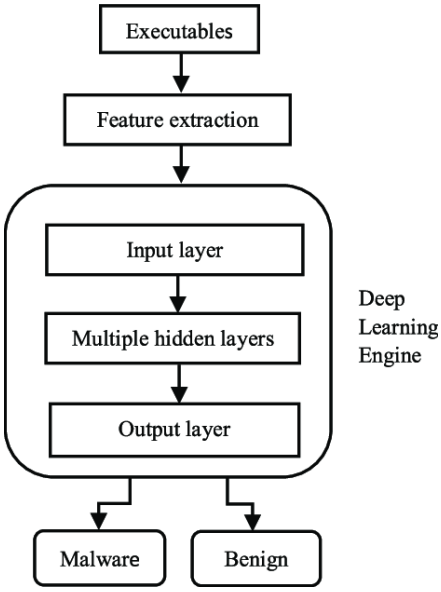


*Figure 4 : Deep Learning Based Malware Detection Schema [10].*

### 3.4.2 Unified Modeling Language (UML) diagrams

1. Use case diagram

The use case diagram in figure 5 represents the user's interaction with the system.

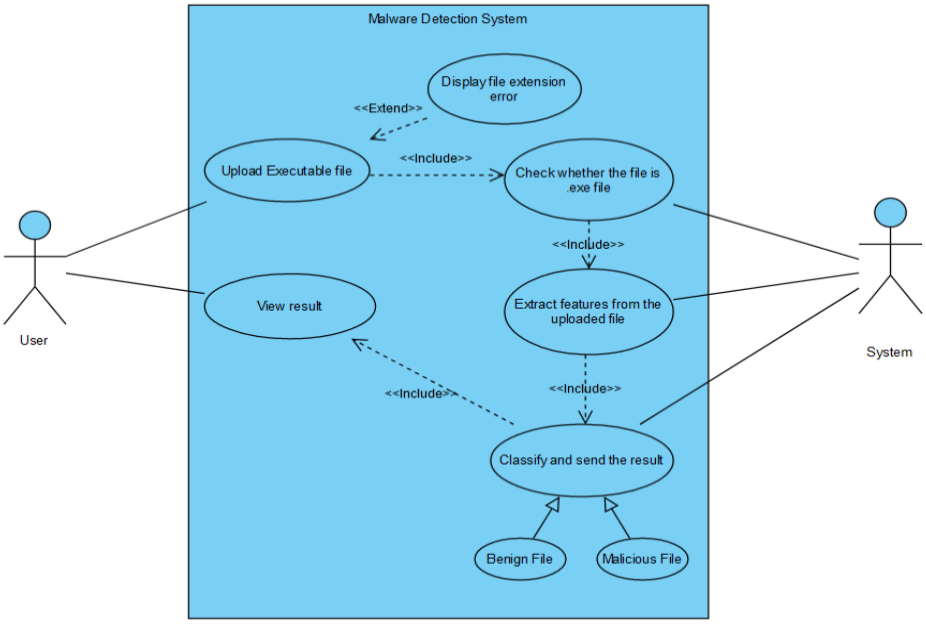


*Figure 5 : Use case diagram of Malware Detection system.*

2. Class diagram

The class diagram in figure 6, displays the relationships between the user and their system , and the static structure of them.

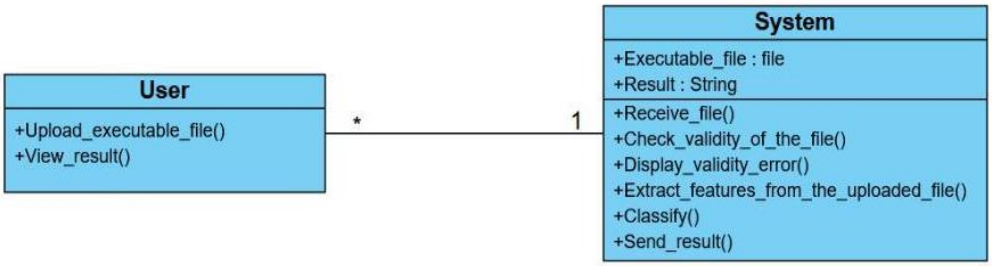


*Figure 6 : Class diagram of Malware Detection system.*

3. Activity diagram

The activity diagram in figure 7 shows the activities going in the system regarding the malware detection process and what are the steps taken to detect the exe file.

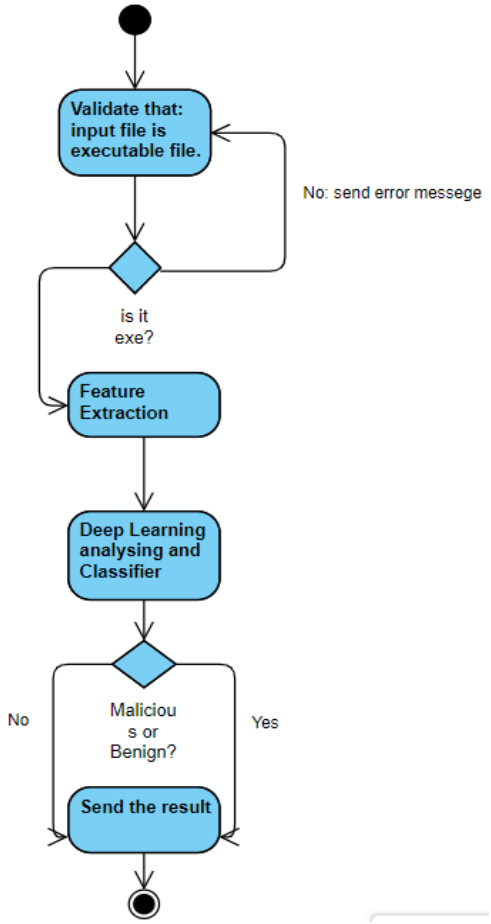


*Figure 7 : Activity diagram of Malware Detection system.*

4. Sequence diagram

The sequence diagram in figure 8 shows the sequence of events that this system is based on starting from uploading the executable file to the last event which is to show the user the result of the executable file, whether malicious or benign.

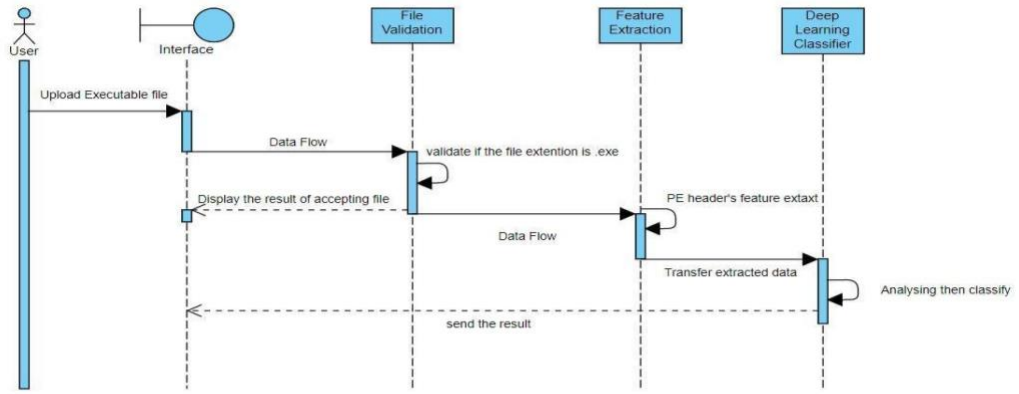


*Figure 8 : Sequence diagram of Malware Detection system.*

## 3.5 INTERFACE :

As shown in figure 9 , the user can upload an executable file on the user's home page as input.

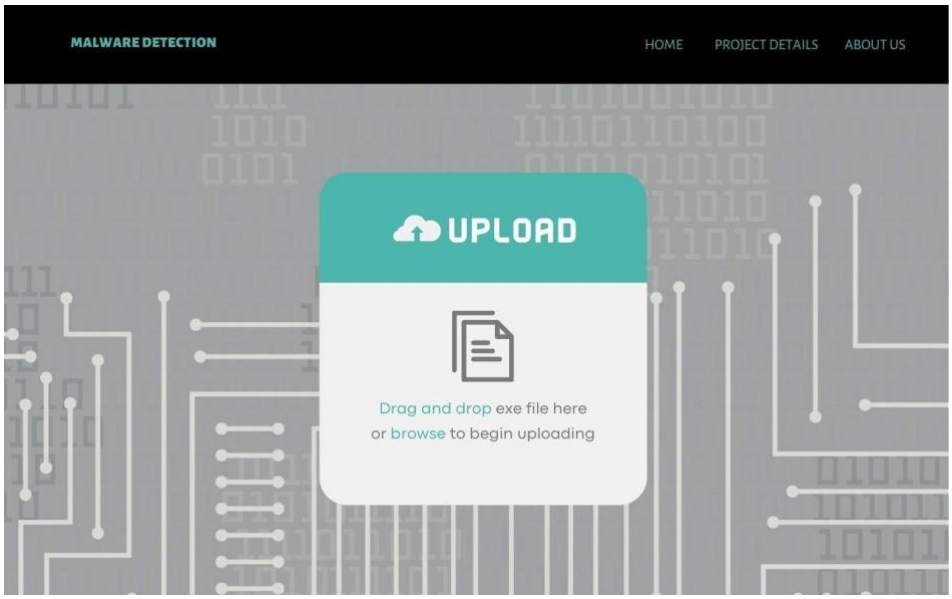


*Figure 9 : User Interface Home Page before uploading .exe file.*

After the scan, the result will appear either that the file is benign as shown in figure 10 or suspicious as shown in figure 11.

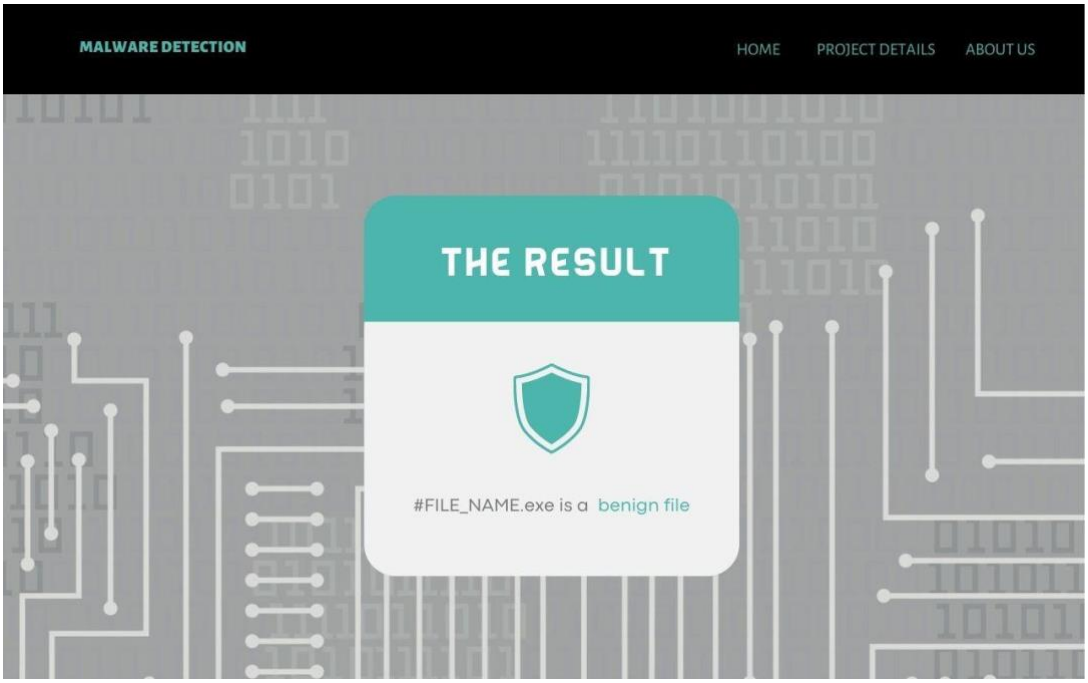


*Figure 10 : User Interface with the result of checking the suspicious file.*

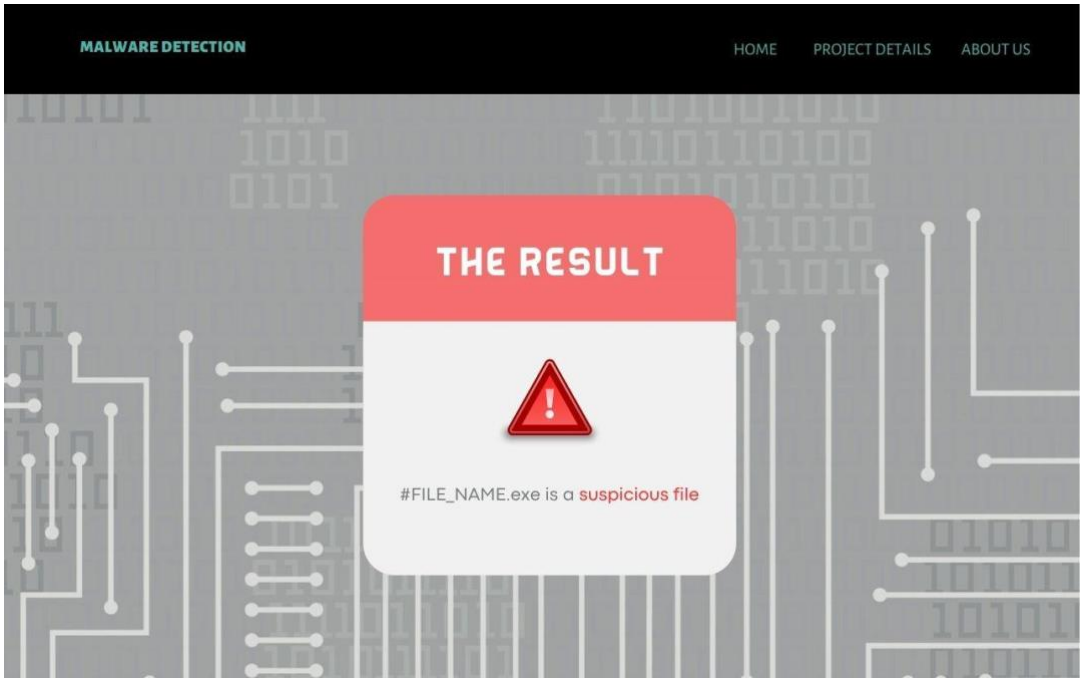


*Figure 11 : User Interface with the result of checking the benign file.*

# CHAPTER4: EXPERIMENTS AND DISCUSSION

## 4.1 INTRODUCTION

Malware detection problem type has been identified as a binary classification problem where malware is one class and benign is another class. The proposed approach consists of different stages that perform several tasks: collection of the datasets, feature extraction, dimension reduction, building classification models, and empirical analysis of the results based on different metrics.

## 4.2 EXPERIMENTAL SETUP

### 4.2.1 Dataset

For experiments, different PE header datasets for malware and benign files were collected from different sources. These datasets were downloaded from an open source repository:

I. Classification of Malwares (CLaMP) dataset [5] which was obtained from MendeleyData source, contains 5210 file samples, where 2722 are malware 2488 are benign files.

II. Benign & Malicious PE Files dataset [6] from Kaggle community platform with 19612 samples , 14600 samples are malware and 5010 are benign .

III. MalwareDataSet dataset [7] from GitHub platform contains a total of 137,444 samples , 40,918 of them are malware and 96,526 are benign . Through the experiments

Later, the dataset that shows the best results will be chosen .

### 4.2.2 Features extraction

Feature extraction is the process of turning a huge collection of inputs into a set of features, These features differ according to the different dataset. So the features in Classification of Malwares (CLaMP) dataset [5] is 69 features as shown in table 2. And the Benign & Malicious PE Files dataset [6] contains 79 features as it appears in table 3 , and the MalwareDataSet dataset [7] contains 9 features as shown in table 4.

| Classification of Malwares (CLaMP) dataset features |
|---|
| e_cblp', 'e_cp', 'e_cparhdr', 'e_maxalloc', 'e_sp', 'e_lfanew', 'NumberOfSections', 'CreationYear', 'FH_char0', 'FH_char1', 'FH_char2', 'FH_char3', 'FH_char4', 'FH_char5', 'FH_char6', 'FH_char7', 'FH_char8', 'FH_char9', 'FH_char10', 'FH_char11', 'FH_char12', 'FH_char13', 'FH_char14', 'MajorLinkerVersion', 'MinorLinkerVersion', 'SizeOfCode', SizeOfInitializedData', 'SizeOfUninitializedData', 'AddressOfEntryPoint', 'BaseOfCode', 'BaseOfData', 'ImageBase', 'SectionAlignment', 'FileAlignment', 'MajorOperatingSystemVersion', 'MinorOperatingSystemVersion', 'MajorImageVersion', 'MinorImageVersion', 'MajorSubsystemVersion', 'MinorSubsystemVersion', 'SizeOfImage', 'SizeOfHeaders', 'CheckSum', 'Subsystem', 'OH_DLLchar0', 'OH_DLLchar1', 'OH_DLLchar2', 'OH_DLLchar3', 'OH_DLLchar4', 'OH_DLLchar5', 'OH_DLLchar6', 'OH_DLLchar7', 'OH_DLLchar8', 'OH_DLLchar9', 'OH_DLLchar10', 'SizeOfStackReserve', 'SizeOfStackCommit', 'SizeOfHeapReserve', 'SizeOfHeapCommit', 'LoaderFlags', 'sus_sections', 'non_sus_sections', 'packer', 'packer_type', 'E_text', 'E_data', 'filesize', 'E_file', 'fileinfo', 'class' |

*Table 2 : Classification of Malwares (CLaMP) dataset features.*

| Benign & Malicious PE Files dataset features |
|---|
| 'Name', 'e_magic', 'e_cblp', 'e_cp', 'e_crlc', 'e_cparhdr', 'e_minalloc', 'e_maxalloc', 'e_ss', 'e_sp', 'e_csum', 'e_ip', 'e_cs', 'e_lfarlc', 'e_ovno', 'e_oemid', 'e_oeminfo', 'e_lfanew', 'Machine', 'NumberOfSections', 'TimeDateStamp', 'PointerToSymbolTable', 'NumberOfSymbols', 'SizeOfOptionalHeader', 'Characteristics', 'Magic', 'MajorLinkerVersion', 'MinorLinkerVersion', 'SizeOfCode', 'SizeOfInitializedData', 'SizeOfUninitializedData', 'AddressOfEntryPoint', aseOfCode', 'ImageBase', 'SectionAlignment', 'FileAlignment', 'MajorOperatingSystemVersion', 'MinorOperatingSystemVersion', 'MajorImageVersion', 'MinorImageVersion', 'MajorSubsystemVersion', 'MinorSubsystemVersion', 'SizeOfHeaders', 'CheckSum', 'SizeOfImage', 'Subsystem', 'DllCharacteristics', 'SizeOfStackReserve', 'SizeOfStackCommit', 'SizeOfHeapReserve', 'SizeOfHeapCommit', 'LoaderFlags', 'NumberOfRvaAndSizes', 'Malware', 'SuspiciousImportFunctions', 'SuspiciousNameSection', 'SectionsLength', 'SectionMinEntropy', 'SectionMaxEntropy', 'SectionMinRawsize', 'SectionMaxRawsize', 'SectionMinVirtualsize', 'SectionMaxVirtualsize', 'SectionMaxPhysical', 'SectionMinPhysical', 'SectionMaxVirtual', 'SectionMinVirtual', 'SectionMaxPointerData', 'SectionMinPointerData', 'SectionMaxChar', 'SectionMainChar', 'DirectoryEntryImport', 'DirectoryEntryImportSize', 'DirectoryEntryExport', 'ImageDirectoryEntryExport', 'ImageDirectoryEntryImport', 'ImageDirectoryEntryResource', ImageDirectoryEntryException', 'ImageDirectoryEntrySecurity' |

*Table 3 : Benign & Malicious PE Files dataset features.*

| MalwareDataSet dataset features |
|---|
| AddressOfEntryPoint', 'MajorLinkerVersion', 'MajorImageVersion', 'MajorOperatingSystemVersion', 'DllCharacteristics', 'SizeOfStackReserve', 'NumberOfSections', 'ResourceSize', 'legitimate' |

*Table 4 : MalwareDataSet dataset features.*

## 4.3 EVALUATIONS OF DIFFERENT DEEP NETWORKS :

In this section, based on the datasets described in Section 4.1, and on previous network models from Section 2.2 An investigation of a deep learning based malware detection system [3] paper , In which they built three fully-connected networks as shown in table 5 , in brief:

- A 2-hidden layer Deep Neural Network contains 1024, and 32 nodes , named as 2L-DNN.
- A 4-hidden layer Deep Neural Network contains (1024, 256, 64, 16 nodes) , named as 4L-DNN.
- A 7-hidden layer Deep Neural Network contains (1024, 512, ..., 16 nodes) , named as 7L-DNN.

Detailed configuration of the fully-connected networks architecture

| 2L-DNN | |
|---|---|
| Layer | Nodes |
| Fully-connected 1 | 1024 |
| Fully-connected 2 | 32 |

| 4L-DNN | |
|---|---|
| Layer | Nodes |
| Fully-connected 1 | 1024 |
| Fully-connected 2 | 256 |
| Fully-connected 3 | 64 |
| Fully-connected 4 | 16 |

| 7L-DNN | |
|---|---|
| Layer | Nodes |
| Fully-connected 1 | 1024 |
| Fully-connected 2 | 512 |
| Fully-connected 3 | 256 |
| Fully-connected 4 | 128 |
| Fully-connected 5 | 64 |
| Fully-connected 6 | 32 |
| Fully-connected 7 | 16 |

*Table 5 : Detailed configuration of the fully-connected networks architecture*

By the following standard:

I. Data Standard
   a. Randomly shuffle: Change the position of each sample in the data set randomly.
   b. The dataset is divided into the training data with 80% and the test data with 20% of the total dataset.
   c. Data Normalization: to eliminate redundant data by changing the values of numeric columns in the dataset to a common scale .

II. Model Architecture Standard
   a. Model: using Sequential Model for a plain stack of layers .

b. Activation: function to guarantee that positive inputs have a slope greater than one
   - The hidden layer: use Exponential Linear Unit (ELU) activation.
   - The output layer: use Sigmoid activation.

c. Dropout : set 0.1 dropout in each hidden layer to reduce the overfitting of the model.

III. Compile Model Standard

a. Optimizer: use Adam optimizer for stochastic gradient descent of training model.

b. Metrics: use Accuracy metrics for evaluating classification models.

IV. Model Training Standard

a. The Batch size: use 64 samples processed before the model is updated.

b. Epochs: use 120 epochs of passes of the entire training dataset the model has completed.

Applying the same created deep learning network models of that paper and fitting the selected datasets on them have been experimented to get the initial accuracy. The results in figure 12 demonstrate the initial accuracy values.

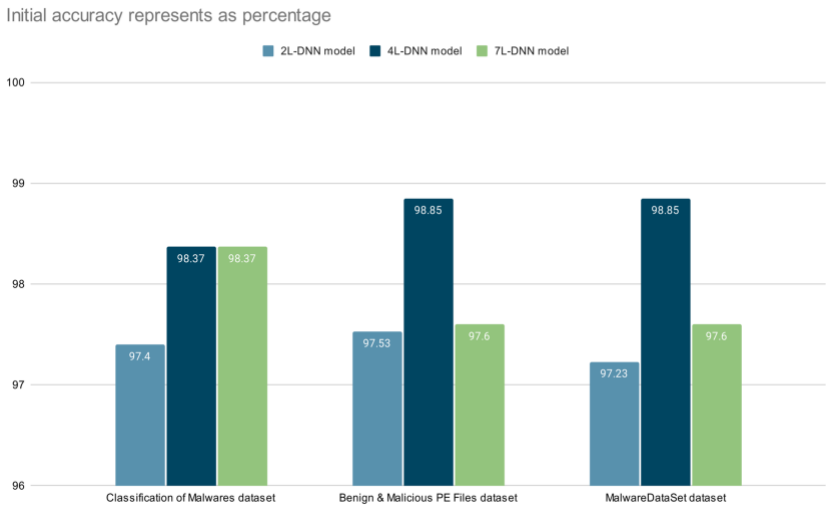


*Figure 12 : Initial accuracy values for datasets with three models.*

## 4.4 COMPARISONS BETWEEN DEEP LEARNING MODELS

From initial accuracy values in Section 4.3, Previous models will be modified to be more suitable for the selected datasets to show higher results while maintaining the standard of models. To improve the models, some changes will be done:

I. Reshaped the dataset from 2D to 1D form.

II. Add Conv1D layer in the Model Architecture.

III. Add Flatten layer in the Model Architecture.

In order to try to improve the results according to the mentioned changes, 4 Networks was created using 1D-convolutional layers and fully-connected layers as shown in table 6 :

- MODEL1 : 1D CNN network model with 4 hidden layers, 3 of them are 1D-convolutional layers containing ( 16 , 32 , 64 nodes ) with filter size 3, and one fully-connected layer with 128 nodes .
- MODEL2 : 1D CNN network model with 4 hidden layers, 3 of them are 1D-convolutional layers containing ( 16 , 16 , 16 nodes ) with filter size 3, and one fully-connected layer with 128 nodes .
- MODEL3 : 1D CNN network model with 4 hidden layers, 3 of them are 1D-convolutional layers containing ( 32 , 32 , 32 nodes ) with filter size 3, and one fully-connected layer with 128 nodes .
- MODEL4 : 1D CNN network model with 4 hidden layers, 3 of them are 1D-convolutional layers containing ( 64 , 64 , 64 nodes ) with filter size 3, and one fully-connected layer with 128 nodes .

Detailed configuration of the 1D CNN models architecture

| MODEL1 | |
|---|---|
| Layer | Biological Dataset |
| Convolutional | 3 × 1 × 16 |
| Convolutional | 3 × 1 × 32 |
| Convolutional | 3 × 1 × 64 |
| Fully-connected | 128 |

| MODEL2 | |
|---|---|
| Layer | Biological Dataset |
| Convolutional | 3 × 1 × 16 |
| Convolutional | 3 × 1 × 16 |
| Convolutional | 3 × 1 × 16 |
| Fully-connected | 128 |

| MODEL3 | |
|---|---|
| Layer | Biological Dataset |
| Convolutional | 3 × 1 × 32 |
| Convolutional | 3 × 1 × 32 |
| Convolutional | 3 × 1 × 32 |
| Fully-connected | 128 |

| MODEL4 | |
|---|---|
| Layer | Biological Dataset |
| Convolutional | 3 × 1 × 64 |
| Convolutional | 3 × 1 × 64 |
| Convolutional | 3 × 1 × 64 |
| Fully-connected | 128 |

*Table 6 : Detailed configuration of the 1D CNN models architecture .*

All four models have been experimented by fitting all the datasets. The results in table 7 and figure13 demonstrate the effectiveness of all the suggested models for all datasets.

| COMPARISONS BETWEEN DEEP LEARNING MODELS | | | | | |
|---|---|---|---|---|---|
| Model Name | | MODEL1 | MODEL2 | MODEL3 | MODEL4 |
| Classification of Malwares dataset | ACY | 0.9779 | **0.9837** | 0.9779 | 0.9750 |
| | LOSS | 0.1202 | **0.0833** | 0.1213 | 0.1744 |
| Benign & Malicious PE Files dataset | ACY | **0.9901** | 0.9873 | 0.9878 | 0.9890 |
| | LOSS | **0.1119** | 0.1015 | 0.2357 | 0.2913 |
| MalwareDataSet dataset | ACY | 0.9705 | 0.9686 | 0.9722 | **0.9725** |
| | LOSS | 0.0713 | 0.0739 | 0.0704 | **0.0702** |

*Table 7 : Comparisons of results for 1D CNN models.*

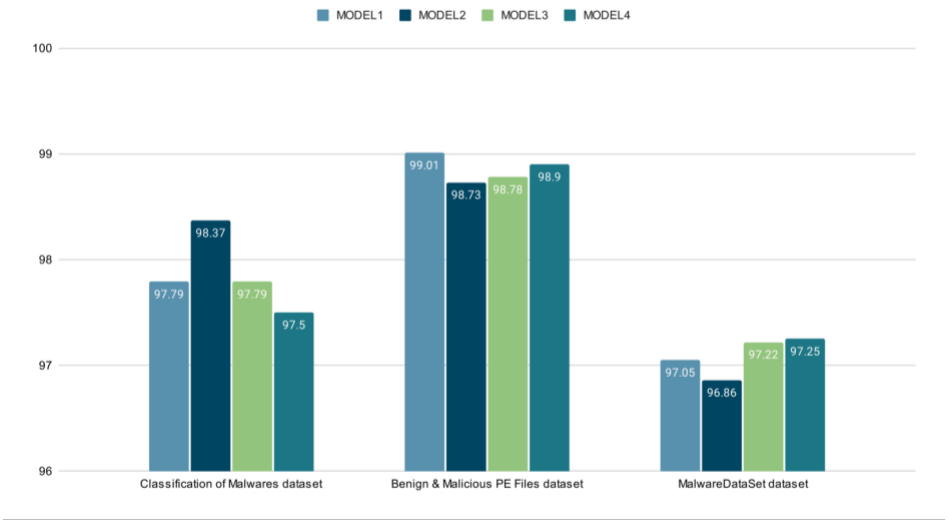


*Figure 13 : Comparisons chart of results for 1D CNN models.*

## 4.5 RESULTS AND DISCUSSION

Table 7 in Section 4.4 shows the models were used and selected in this experiment which give the different accuracy detection compared to the initial accuracy values .

The conclusion from the results is that the Classification of Malwares (CLaMP) dataset [5] is more suitable with model2 . Benign & Malicious PE Files dataset [6] is more suitable with model1. MalwareDataSet dataset [7] is more suitable with model4 . The best results appear in the Benign & Malicious PE Files dataset [6] and in the model2. From that point the model will be further developed.

## REFERENCES


**[1] Azeez, N. A., Odufuwa, O. E., Misra, S., Oluranti, J., & Damaševičius, R. (2021, February). Windows PE malware detection using ensemble learning. In Informatics (Vol. 8, No. 1, p. 10). MDPI.**

**[2] Rathore, H., Agarwal, S., Sahay, S. K., & Sewak, M. (2018, December). Malware detection using machine learning and deep learning. In International Conference on Big Data Analytics (pp. 402-411). Springer, Cham.**

**[3] Sewak, M., Sahay, S. K., & Rathore, H. (2018, August). An investigation of a deep learning based malware detection system. In Proceedings of the 13th International Conference on Availability, Reliability and Security (pp. 1-5).**

**[4] Hardy, W., Chen, L., Hou, S., Ye, Y., & Li, X. (2016). DL4MD: A deep learning framework for intelligent malware detection. In Proceedings of the International Conference on Data Science (ICDATA) (p. 61). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).**

**[5] Kumar, Ajit (2020), "ClaMP (Classification of Malware with PE headers)", Mendeley Data, V1, doi: 10.17632/xvyv59vwvz.1**

**[6] Mauricio, (2018), "Benign & Malicious PE Files", Kaggle Data ,V1 , from https://www.kaggle.com/datasets/amauricio/pe-files-malwares .**

**[7] Emrah Yıldırım , (2022), "MalwareDataSet" , GitHub , V1 , from https://github.com/emr4h/Malware-Detection-Using-Machine-Learning**

**[8] Cybersecurity, K. E. (2017). Machine learning for malware detection.**

**[9] POORNACHANDRAN, P., & VENKATRAMAN, S. Robust Intelligent Malware Detection Using Deep Learning.**

**[10] Zhu, D., Jin, H., Yang, Y., Wu, D., & Chen, W. (2017, July). DeepFlow: Deep learning-based malware detection by mining Android application for abnormal usage of sensitive data. In 2017 IEEE symposium on computers and communications (ISCC) (pp. 438-443). IEEE.**

**[11] https://colab.research.google.com**

**[12] https://www.python.org/**

**[13] https://www.tensorflow.org/**

**[14] https://keras.io/**

**[15] https://numpy.org/**

**[16] https://pandas.pydata.org/**

**[17] https://pypi.org/project/pefile/**