NETWORK DATA ANALYSIS WITH NEURAL NETWORKS TO IMPROVE SECURITY

Lingisetty Lakshmi Charan,<name>,<name >,<name>

Masters in Computer Science, University of Central Missouri
Missouri, United States

LxL69020@ucmo.edu

@ucmo.edu

@ucmo.edu
@ucmo.edu

***Abstract*— Surveillance implies the sensation of being watched, which can result from the assortment, conglomeration, or potentially utilization of one's data. The sensation of being surveyed may be an issue, like close to home pain. It could likewise be an issue in light of the fact that such an inclination can influence how individuals act, on the off chance that individuals begin to mull over the things they do, peruse, or look for. Disclosure of data beyond the setting in which it was gathered. One exposure danger may be the interfering representative who looks into individuals he knows in a corporate data set. Another may be a character hoodlum who effectively hacks into a data set. Issues of frailty are in this sense issues of revelation. Less noxiously, data may be uncovered to individuals who end up being close by and see the advertisements on another individual's PC. Discrimination, or at least, treating individuals distinctively on the premise of data gathered about them. There are various sorts of separation dangers. The clearest may be attempting to anticipate participation in some safeguarded class, for example, race or religion, and afterward separating on that premise. Some could additionally have a problem with any separation that is related with a safeguarded trademark, whether it frames the express reason for the focusing on. Malware is a pervasive and severe cyber threat that continues to infect millions of devices worldwide. It can perform a range of malicious activities, from stealing sensitive data to disrupting system performance. Consequently, detecting and preventing malware attacks is of utmost importance for ensuring our devices remain secure. One emerging and promising technology for detecting malware is Deep Learning. The algorithms are powerful and scalable approaches that can handle large datasets, making them effective for detecting the high number of malware variants affecting desktop and mobile platforms today. This study explores current learning technologies for detecting malware attacks on different type of platforms like Android, Linux, and Windows platforms. We present various categories of learning algorithms, network optimizer, and regularization methods, along with different loss functions, activation functions, and frameworks for implementing Deep Learning models.**

***Keywords*— Malware, Intrusion, Deep Learning, Platform, Surveillance, Attacks.**

I. INTRODUCTION

Malicious software, or malware, has been a problem for as long as computers have existed. Malware includes computer viruses, worms, Trojan horses, and other types of harmful software. These programs can cause a lot of damage, such as stealing sensitive information, disrupting online services, and damaging computer systems.

One of the most well-known groups that use malware to carry out attacks is called "Anonymous." They often engage in large-scale attacks that aim to make servers or other infrastructure resources unavailable.

Detecting malware is important because it can prevent financial losses, protect personal information, and keep computer systems working properly. There are different ways to detect malware, but researchers have proposed using deep learning algorithms to identify it more accurately [1].

By detecting and mitigating malware, organizations can protect their reputation, comply with regulations, and safeguard computer systems, data, privacy, finances, and reputation. To do this, a combination of signature-based and machine-learning approaches is the most effective way to detect malware and prevent cyber-attacks. Deep learning is a powerful technique that falls under the umbrella of artificial neural network algorithms. By analysing massive amounts of data, deep learning models are able to perform tasks such as classification and prediction with incredible accuracy. This technology serves as the foundation for a wide range of cutting-edge applications including computer vision, natural language processing, autonomous vehicles, fraud detection, malware detection, and many others. Deep learning models are known for their complex network architectures, which include multiple hidden layers and are often referred to as deep neural network. These networks are capable of classifying records, sound, and text data with remarkable precision, and can even extract features from input datasets without requiring manual intervention. A Recurrent Neural Network is a Deep Learning algorithm specifically designed for sequential or time series data. This sets it apart from traditional feed-forward neural networks, which only work with independent data features. When dealing with sequential

Github Link : https://github.com/chadivevarshithareddy/neural-final-project

data, Recurrent Neural Network must be modified to capture dependencies between feature values or data points. To accomplish this, Recurrent Neural Network algorithms use the concept of "memory" to store previous inputs/states and produce the next output in a sequence. This allows Recurrent Neural Network to handle sequential data and inputs of varying lengths while also memorizing historical information. As a result, Recurrent Neural Network has been successfully used in malware detection. One example is a model that learns from sequences of byte information using Recurrent Neural Network and different feature factorization techniques such as one-hot encoding and random feature vectors. However, Recurrent Neural Network computation can be slow, and the classical RNN algorithm does not consider future inputs when making decisions. Additionally, Recurrent Neural Network is prone to the vanishing gradient problem, which can prevent the model from learning new samples. This occurs when the gradients used to compute weight updates approach zero [1].

The Internet has undergone rapid development, leading to an increase in the prevalence of malware, which is now considered one of the major cyber threats. Malware is any software designed to perform malicious actions, including information stealing and espionage. While the diversity of malware is on the rise, anti-virus scanners are unable to provide adequate protection, resulting in millions of hosts being attacked. According to Kaspersky Labs, there were 6,563,145 different hosts attacked, and 4,000,000 unique malware objects detected in 2015.

In addition to this, there has been a decrease in the level of skill required for malware development, owing to the ready availability of attacking tools on the internet. The high availability of anti-detection techniques, as well as the ability to buy malware on the black market, results in the opportunity to become an attacker for anyone, regardless of their skill level. Current studies indicate that more and more attacks are being issued by script-kiddies or are automated. Therefore, malware protection of computer systems is one of the most important cyber security tasks for single users and businesses, as even a single attack can result in compromised data and significant losses [2].

Given the magnitude of losses and the frequency of attacks, accurate and timely detection methods are imperative. However, current static and dynamic methods do not provide efficient detection, particularly when dealing with zero-day attacks. Therefore, deep learning-based techniques can be utilized. This work discusses the main points and concerns of machine learning-based malware detection, as well as explores the best feature representation and classification methods.

The sandbox will be employed to extract the behaviour of malware samples, which will serve as input to the machine learning algorithms. The goal is to ascertain the most accurate algorithm that can distinguish malware families with the lowest error rate and to determine the best feature representation method and how the features should be extracted. Accuracy will be measured for both detection of whether the file is malicious and classification of the file to the malware family.

Malware detection techniques can be classified into signature-based and behaviour-based methods. However, it's crucial to understand the fundamentals of two malware analysis approaches - static and dynamic malware analysis - before diving into these methods. Static analysis involves examining the malware "statically", i.e., without executing the file. In contrast, dynamic analysis is performed on the file while it is being executed, for instance, in a virtual machine. Static analysis can be thought of as "reading" the source code of the malware and deducing its behavioural properties. Various techniques can be employed under static analysis, such as file format inspection, string extraction, fingerprinting, and scanning [6].

II. MOTIVATION

The primary motivation of Malware Identification analysis with Deep Learning on Neural Networks is to detect the find out the Malware in the network dataset. In this work, the dataset containing the malware dataset will be taken into consideration. The pre-processing will be applied in to the dataset and the noisy and null value data will be removed from the dataset. After the data will be analysed and visualized for further processing. The Convolutional Neural Networks algorithm will be chosen to implementation process. The project evaluation can be tested with the deep learning algorithm prediction results. Since the Convolutional Neural Networks algorithm will be used to predict the Malware, the accuracy of the algorithm result will be helpful to evaluate the results. The accuracy score of the algorithm in the Malware Identification helps to evaluate the dataset.

The Deep learning will be the python based application which contributes to find out the Malware early stage. It will be helpful for the human to detect at early and to take necessary treatments in the correct time. The progression of profound learning influences is generally applied to classification assignments and portrayals learning. These profound frameworks with numerous layers have been displayed to yield promising execution in removing serious areas of strength for more of information. The streamlining of the goal capability becomes curved in the event that we adjust one variable and fix the others [2].

Since cross-approval on review information presumably prompts overoptimistic results, cross-approval is improper for this issue. To assess the capacity of the models to anticipate newfound affiliations, we train and test the dataset.

Thus, via consistently consolidating the model for helper side data and the cooperative filter for the quality sickness affiliations grid, our model learns a significantly more

significant portrayal for every quality and illness and gives more exact expectation. The project evaluation can be tested with the deep learning algorithm prediction results. Since the Deep learning algorithm will be used to predict the Malware, the accuracy of the algorithm result will be helpful to evaluate the results. The accuracy score of the algorithm in the Malware identification helps to evaluate the dataset.

III. CONTRIBUTION & OBJECTIVE

- The objective of Malware identification with deep learning is to detect the Malware in the network with the available attributes.
- In this work, the dataset containing the malware dataset will be taken into consideration.
- The primary contribution is to apply the deep learning to detect the Malware.
- The pre-processing will be applied in to the dataset and the noisy and null value data will be removed from the dataset.
- After the data will be analysed and visualized for further processing.
- The Deep Learning neural network algorithm will be chosen to make the good accuracy prediction.
- It will be helpful in all the distributed network records to detect the Malware.
- The aspect of correlation coefficient data is less sensitive to malware compared to the intrusion dataset.
- Distinct attributes of Malware occurrence can be removed on varying scales to achieve greater accuracy in performance.

IV. RELATEDWORK

In recent years, computer malware has become a significant threat to our digital security. To combat this problem, researchers have developed machine learning-based systems that can analyze malicious software and detect new threats.

One such system was proposed by Liu et al. in 2017. This system has three components: data processing, decision making, and new malicious software detection. The data processing part extracts the characteristics of malicious software using various techniques, such as gay-scale records and import functions. The decision-making part uses these features to classify malicious software and identify suspicious samples. Finally, the detection module employs an algorithm to discover new families of malicious software [1].

This approach was evaluated using a dataset of over 20,000 malicious software samples, and the results showed that the system was effective in classifying unknown malicious software with an accuracy of up to 98.9%. Furthermore, the system could detect 86.7% of new malicious software, indicating its capability to identify previously unseen threats [1].

Another research study by Kediri et al. in 2019 focused on identifying malware in mobile applications using machine learning methods and reverse engineering of Android Java code. They found the algorithms that provide the highest malware detection rate and identified the application features with the highest utility in classifying malware. The results showed that two classification algorithms, Random Forest and K Nearest Neighbors, performed best in terms of correctly classifying instances of malware. Overall, these studies show the effectiveness of deep learning in detecting and classifying malicious software and can help enhance our digital security.

Malware attacks are becoming more common every day, and it's important to have effective systems in place to detect and prevent them. Machine learning algorithms are being used to develop such systems, but there are some challenges. For example, biased data can limit the performance of these algorithms in real-world scenarios [2].

Several research studies have compared traditional machine learning algorithms with more advanced deep learning algorithms for malware detection, classification, and categorization using various datasets. These studies also propose new methods that combine static, dynamic, and record processing techniques to create an efficient model for zero-day malware detection.

One study suggests the use of Convolutional Neural Networks (CNNs) for malware detection, which can classify malware files quickly and accurately while being resistant to redundant API injections. Another study proposes a unique framework for Android malware detection that incorporates multiple features to reflect the characteristics of Android apps from different angles.

In recent years, there has been a rise in the number of harmful software programs affecting Android phones. The traditional methods of detecting malware are becoming less effective as malware creators are finding new ways to hide their malicious activities. To address this issue, researchers have proposed a new approach called DL-Droid, which uses deep learning techniques to identify potentially harmful Android applications. This method has been tested on a dataset of over 30,000 applications, both good and bad, and showed that it can detect up to 99.6% of harmful apps. However, more research is needed to improve the system further. Another study explores the use of deep learning in detecting Android malware and discusses the challenges researchers face in this field. These findings could help us detect malware more effectively and keep our phones safe [3].

In a recent study by Singh (2021), analytic approaches were used to determine the effectiveness of malware detection systems. Research environments were constructed using both static and dynamic tools, and malware classifiers were trained in the second step. While traditional methods were used in the

past, machine learning algorithms are now employed to classify malware due to their ability to keep up with the complexity and speed of virus creation. This paper provides a comprehensive analysis of machine learning-based malware detection methods, including the challenges involved in creating virus classifiers. The development of an efficient malware detection system is the ultimate goal after overcoming several malware detection challenges [4].

According to Osman et al. (2021), the increasing interconnectedness of devices in the near future may lead to the collection of private information and an increase in security risks and cybercrimes. To prevent cybercrimes, innovative cyber security methods are necessary, such as those which can detect harmful Internet Protocol (IP) addresses before communication takes place. One such method is the IP reputation system, which profiles security risks to the cyber-physical system and is considered one of the best approaches. Big data forensics is used to anticipate IP reputation at the pre-acceptance stage, and behavioural analysis and the Decision Tree (DT) approach are used to classify related zero-day attacks [5].

In a recent study, researchers evaluated different methods for reducing the number of features used to detect malware processes. They found that a combination of a feature reduction method called VT with a classification model called RF achieved the best accuracy of 99.78%. They also found that RF performed the best among different classification models, with an accuracy of 99.7%. The researchers proposed a two-stage method for detecting malware processes using a combination of CNN and RNN models. They tested their approach on 150 process behaviour logs and achieved an AUC of 0.96. However, due to limited data availability, they suggest that future work should focus on increasing the amount of data to improve the approach's effectiveness [6].

V. PROPOSED SYSTEM

The proposed methods aim to find the Malware with higher standard. The accuracy levels of the identification of the Malware will be improved with the proposed system. The deep learning on neural network will provide the better solution to solve the problem of identification of the Malware in the real world intrusion data. The Convolutional Neural Network algorithm will check the data in more compact with training and testing the data. It will provide more accuracy as compared with the other type of techniques. The malware dataset will be taken as the input to the application and the dataset will be passed into the Convolutional Neural Network algorithm and the data will be analysed with the different visual graphs.

The proposed approach applies a biased neural network function with effectiveness, enabling reliable recognition of Malware. The Malware dataset is given as input to the application and the pre-processing is applied , next the data cleaning is performed after the training and test data are split down and will be passed into Deep Learning algorithm and the Malware will be predicted. The Malware identification with deep learning on neural networks will be the python based application which contributes to find out the Malware. It will be helpful in finding of the Malware based on the attributes of the network records.

Fig 1 Proposed Architecture diagram

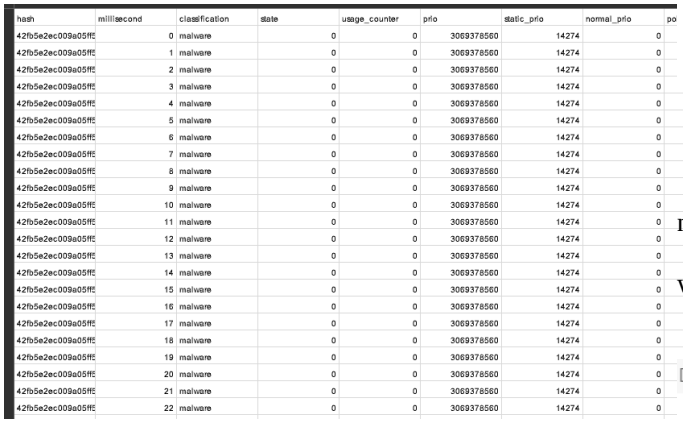
The testing and training variables are split and passed into the algorithm for the Malware prediction. In this algorithm will provide a comprehensive and intelligent solution for discovering high utility item sets, enabling users to access important information and streamline their search processes.

The application will be developed with Google Colab Python Tool as the project can be directly executed in any type computer systems with internet connection. There is no need of any specific software to be installed in the user system. The Colab Tool helps to develop and run the application directly inside the cloud server where the Python library files are installed. The deep learning algorithm libraries are built inside the Colab.

VI. DATA DESCRIPTION

The dataset for Malware identification is taken from the source of kaggle dataset. This dataset contain the fields needed for the analysing of the network dataset. Exploratory examination is a cycle to investigate and comprehend the information and information relationship in a total profundity with the goal that it makes highlight designing and deep

learning demonstrating steps smooth and smoothed out for expectation. Exploratory examination assists with validating our presumptions or misleading. Most of the record in a dataset are noisy and contain lots of information. But with feature engineering do, will get more good results. The first step is to import the libraries and load data. After that will take a basic understanding of data like its shape, sample, is there are any NULL values present in the dataset. Understanding the data is an important step for prediction or any deep learning project. It is good that there are no NULL values.



hash	millisecond	classification	state	usage_counter	prio	static_prio	normal_prio	po
42fb5e2ec009a05ff5	0	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	1	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	2	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	3	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	4	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	5	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	6	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	7	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	8	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	9	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	10	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	11	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	12	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	13	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	14	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	15	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	16	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	17	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	18	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	19	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	20	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	21	malware	0	0	3069378560	14274	0	0
42fb5e2ec009a05ff5	22	malware	0	0	3069378560	14274	0	0

Fig. 2 Malware Dataset

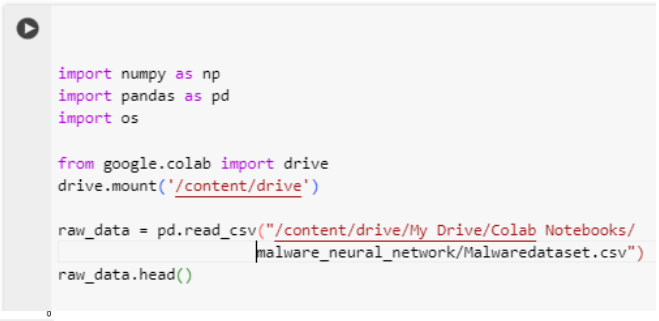
The information about the malware network records with different types of attributes are collected from kaggle data. The dataset total contains of record dataset with training and testing malware affected records of network. It will begin from the principal segment and investigate every section and comprehend what influence it makes on the objective segment.

The aim of this dataset samples was to compile a comprehensive compilation of network traffic, encompassing both lawful and botnet traffic, in addition to background traffic. For the purposes of this investigation, the dataset was utilized for feature extraction and model training. The malware dataset, comprised of 34 unique columns of identified malware, was recorded in bidirectional intrusion identified files. The characteristics of this dataset can be found in Figure 2, and it is comprised of a total of 10000 records.

At the necessary step, we will likewise perform pre-processing and include designing undertakings. The point in acting top to bottom exploratory examination is to get ready and clean information for better Deep Learning demonstrating to accomplish elite execution and summed up models. So it should begin with breaking down and setting up the dataset for expectation.

VII. EXPERIMENTAL ANALYSIS

The Initial process of loading the dataset into the Google Colab into the drive is the first step in execution process. The record data containing the information of the record with respect to the path and the description of the record location and the record related style are linked.

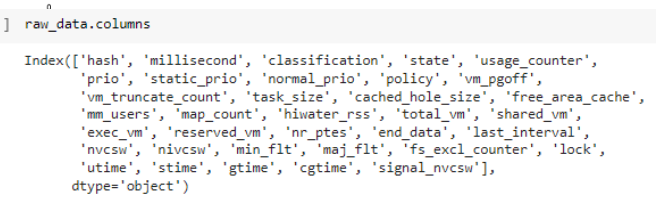


```
import numpy as np
import pandas as pd
import os

from google.colab import drive
drive.mount('/content/drive')

raw_data = pd.read_csv("/content/drive/My Drive/Colab Notebooks/
                       malware_neural_network/Malwaredataset.csv")
raw_data.head()
```

Fig. 3 Load dataset

The pre-processing is applied to the dataset where all the noisy data are removed.

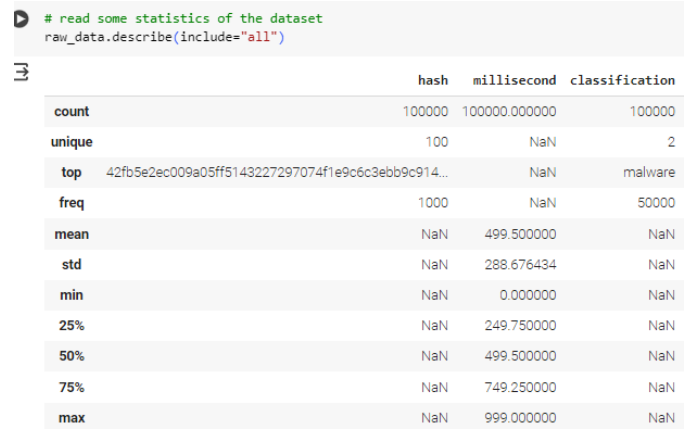
The information has an extremely straightforward design with elements.



```
[ ] raw_data.columns

Index(['hash', 'millisecond', 'classification', 'state', 'usage_counter',
       'prio', 'static_prio', 'normal_prio', 'policy', 'vm_pgoff',
       'vm_truncate_count', 'task_size', 'cached_hole_size', 'free_area_cache',
       'mm_users', 'map_count', 'hiwater_rss', 'total_vm', 'shared_vm',
       'exec_vm', 'reserved_vm', 'nr_ptes', 'end_data', 'last_interval',
       'nvcsw', 'nivcsw', 'min_flt', 'maj_flt', 'fs_excl_counter', 'lock',
       'utime', 'stime', 'gtime', 'cgtime', 'signal_nvcsw'],
      dtype='object')
```

Fig. 4 Malware Attributes

The Malware attributes contains 34 unique columns with each column denoting the attribute which identifies the malware in the network. The noisy data present inside the record is also removed and improves the record quality which will be more helpful in the application of the prediction of the Malware in the dataset.



```
# read some statistics of the dataset
raw_data.describe(include="all")
```

	hash	millisecond	classification
count	100000	100000.000000	100000
unique	100	NaN	2
top	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	NaN	malware
freq	1000	NaN	50000
mean	NaN	499.500000	NaN
std	NaN	288.676434	NaN
min	NaN	0.000000	NaN
25%	NaN	249.750000	NaN
50%	NaN	499.500000	NaN
75%	NaN	749.250000	NaN
max	NaN	999.000000	NaN

Fig. 5 Malware Metrics

The numerical columns are considered for the metric calculations, which display the total count records, mean,

standard values and also the minimum and maximum values of the records. The figure 5 displays these record details in more elaborated structure. Next the classification part of the malware dataset is analysed to find out number of malwares types in the dataset.

```
[ ] data["classification"].value_counts()

    classification
    malware    50000
    benign     50000
    Name: count, dtype: int64
```

```
[ ] data['classification'] = data.classification.map({'benign':0, 'malware':1})
    data.head()
```

	hash	millisecond	classification
0	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	0	1
1	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	1	1
2	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	2	1
3	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	3	1
4	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	4	1

Fig. 6 Classification records

Based on the execution part the classification denotes there are 50,000 malware records and 50,000 benign records present in the dataset.

The record dataset is divided into testing and training to pass in to the neural network model.

```
[ ] X = data.drop(["hash","classification",'vm_truncate_count','shared_vm','exec_vm','
    Y = data["classification"]
```

```
[ ] from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=1)
```

Fig. 7 Training and Test data

The data before passing into the neural network algorithm it is normalized.

Nnormalize the data - Neural Network

```
[ ] # Data normalization

    from sklearn.preprocessing import StandardScaler

    scaler = StandardScaler()
```

```
    x_train = scaler.fit_transform(x_train)
    x_test = scaler.transform(x_test)
```

Fig. 8 Normalize data

Google's Tensor Flow is a popular library and platform widely used for deep learning applications. Although it wasn't specifically designed for creating neural networks, it is frequently utilized for this purpose. Before proceeding, it's necessary to install and import the Tensor Flow module. With its open-source framework, Tensor Flow enables the creation of highly adaptable Convolutional Neural Network architectures ideal for various computer vision tasks.

The Tensor Flow Models Programming interface provides a versatile platform for constructing intricate neural networks by adding and removing layers. The Application Programming Interface supports both sequential and functional models with a single input and output or multiple inputs and outputs, respectively. The training module encompasses various methods, including generating the model, optimizer, and loss function, fitting the model and evaluating and predicting input data. Furthermore, the Application Programming Interface includes methods for batch data processing, testing, and prediction. The Models Programming interface in Tensor Flow also enables users to save and pre-process their models for future use. Therefore, this Application Programming Interface offers an efficient and comprehensive solution for building and training neural networks. The Tensor Flow library files are applied into the execution process.

```
import tensorflow as tf
```

```
#Number of attributes
input_size = 27

#Number of Outputs
output_size = 2

# Use same hidden layer size for both hidden layers. Not a necessity.
hidden_layer_size = 50

# define how the model will look like
model = tf.keras.Sequential([
    tf.keras.layers.Dense(hidden_layer_size, input_shape=(input_size,), activation=
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'),
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'),
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'),
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'),
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'),
    tf.keras.layers.Dense(output_size, activation='softmax') # output layer
])

model.summary()
```

Fig. 9 Sequential model with TensofFlow

The sequential flow of the data model is depicted:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 50)                1400
_________________________________________________________________
dense_1 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_2 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_3 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_4 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_5 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 102
=================================================================
Total params: 14,252
Trainable params: 14,252
Non-trainable params: 0
```

Fig. 10 Sequential Dense Layers

The results of the Keras Sequential with tensor flow are produced with integrating with the library files.

```
# set the batch size
batch_size = 100

# set a maximum number of training epochs
max_epochs = 20

early_stopping = tf.keras.callbacks.EarlyStopping(patience=2)
```

```
result = model.fit(x=x_train,
                   y=y_train,
                   batch_size=batch_size,
                   epochs=max_epochs,
                   verbose=1,
                   #callbacks=[early_stopping],
                   validation_split=0.2)
```

Fig. 11 Model Execution

The training data are passed into the model and it is fit to execution process with finalizing the mac_epochs.

The accuracy and loss percentage of the neural network is given below:

```
Epoch 1/20
640/640 [==============================] - 2s 2ms/step - loss: 0.0890 - accuracy: 0.9630 - val_loss: 0.0113
Epoch 2/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0052 - accuracy: 0.9986 - val_loss: 0.0039
Epoch 3/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0042 - accuracy: 0.9988 - val_loss: 0.0017
Epoch 4/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0115 - accuracy: 0.9970 - val_loss: 0.0050
Epoch 5/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0016 - accuracy: 0.9995 - val_loss: 0.0034
Epoch 6/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0018 - accuracy: 0.9996 - val_loss: 0.0020
Epoch 7/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0029 - accuracy: 0.9991 - val_loss: 0.0021
Epoch 8/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0023 - accuracy: 0.9993 - val_loss: 0.0335
Epoch 9/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0035 - accuracy: 0.9992 - val_loss: 0.0019
Epoch 10/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0019 - accuracy: 0.9993 - val_loss: 0.0026
Epoch 11/20
640/640 [==============================] - 2s 3ms/step - loss: 8.9101e-04 - accuracy: 0.9998 - val_loss: 0.0
Epoch 12/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0020 - accuracy: 0.9994 - val_loss: 0.0035
Epoch 13/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0050 - accuracy: 0.9987 - val_loss: 0.0023
```

Fig. 12 Accuracy and Loss

The attributes of input data is been trained and tested then the model is been build, trained and tested and the predicted output is displayed.

```
# Visualize the result
acc = result.history['accuracy']
val_acc = result.history['val_accuracy']
loss = result.history['loss']
val_loss = result.history['val_loss']

epochs = range(1, len(acc) + 1)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
sns.set_style("white")
plt.suptitle('Train history', size = 15)

ax1.plot(epochs, acc, "bo", label = "Training acc")
ax1.plot(epochs, val_acc, "b", label = "Validation acc")
ax1.set_title("Training and validation acc")
ax1.legend()

ax2.plot(epochs, loss, "bo", label = "Training loss", color = 'red')
ax2.plot(epochs, val_loss, "b", label = "Validation loss", color = 'red')
ax2.set_title("Training and validation loss")
ax2.legend()

plt.show()
```

Fig. 13 Finding of Training and Validation accuracy

The training and validation of the model were evaluated and the accuracy is calculated.

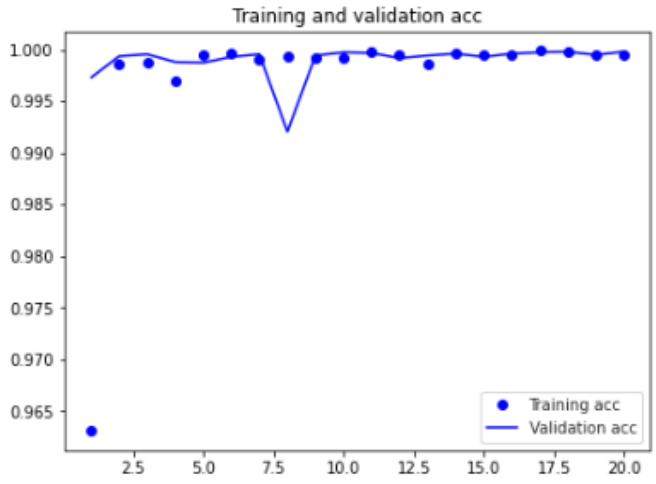


Fig. 14 Training and Validation accuracy

The training and the validation accuracy graph shows the results with graphical format. The trainig accuracy is getting in the increase ratio and it reaches the good saturation point.

The results of the CNN with sequential data analysis provides the accuracy. The validation and testing accyracy were identified while executing the convolutional neural network algorithm.The Convolutional Neural Networks algorithm is applied with creating the sequential model. The output of the sequential model with layers is displayed

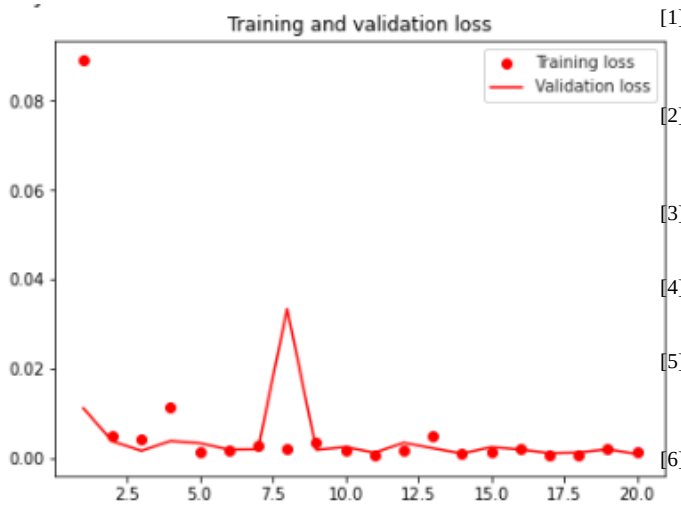


Fig. 15 Testing and accuracy loss

The training loss is getting in the decrease ratio and it reaches the good saturation point. Thus the Malware identification accuracy is calculated to make the prediction quality good.

VIII. CONCLUSION

A cutting-edge framework for detecting Malware has been developed using deep neural networks and diverse distributed network data. The framework employs all records with malware information for model training and data classification. By constructing functional intellectual networks based on the correlation, the neural network formation is optimized using correlation coefficient information. This methodology greatly enhances diagnostic accuracy compared to traditional approaches, demonstrating that integrating advanced deep learning with the expertise is an effective way to diagnose malwares in their early stages. The same or similar methodologies can be applied to diagnose other Malwares, providing a foundation for ongoing scan of malwares in this field. Assessing the effectiveness of deep learning techniques and algorithms for forecasting malware and their subcategories can enhance the model's precision. Additionally, scrutinizing the dataset has enabled us to identify the most suitable feature sets for model fitting. Enhancing accessibility and usability can be achieved through the addition of a Graphical Interface, such as website applications. The application of Malware predictive models in varied networks can enhance the performance and limitations of the proposed models, thus refining network security practice.

REFERENCES

[1] Dong, G., Liao, G., Liu, H., Kiang, G., 2018. Are view of the auto-encoder and its variants: A comparative perspective from target recognition in synthetic caperture radar images. IEEE Geoscience and Remote Sensing Magazine 6, 44–68.

[2] Kedziora, M., Gawin, P., Szczepanik, M., & Jozwiak, I. (2019). Malware detection using machine learning algorithms and reverse engineering of android java code. International Journal of Network Security & Its Applications (IJNSA) Vol, 11.

[3] Alzaylaee, M. K., Yerima, S. Y., & Sezer, S. (2020). DL-Droid: Deep learning based android malware detection using real devices. Computers & Security, 89, 101663.

[4] Singh, J., & Singh, J. (2021). A survey on machine learning-based malware detection in executable files. Journal of Systems Architecture, 112, 101861.

[5] Usman, N., Usman, S., Khan, F., Jan, M. A., Sajid, A., Alazab, M., & Watters, P. (2021). Intelligent dynamic malware detection using machine learning in IP reputation for forensics data analytics. Future Generation Computer Systems, 118, 124-141.

[6] Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., & Zheng, Q. (2020). IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. Computer Networks, 171, 107138.

[7] Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., & Venkatraman, S. (2019). Robust intelligent malware detection using deep learning. IEEE Access, 7, 46717-46738.

[8] Wang, Z. J., Turko, R., Shaikh, O., Park, H., Das, N., Hohman, F., ... & Chau, D. H. P. (2020). CNN explainer: learning convolutional neural networks with interactive visualization. IEEE Transactions on Visualization and Computer Graphics, 27(2), 1396-1406.

[9] Soofi, A. A., & Awan, A. (2017). Classification techniques in machine learning: applications and issues. J. Basic Appl. Sci, 13, 459-465.

[10] Talan, T. ve Aktürk, C. (2021) Bilgisayar Biliminde teorik ve uygulamalı araştırmalar, Efe akademi yayıncılık, Istanbul, ISBN: 978-625-8065-42-8.

[11] Jha,S.,Prashar,D.,Long,H.V.,Taniar,D.,2020. Recurrent neural network for detecting malware. Computers & Security 99, 102037.

[12] Kakisim, A.G., Gulmez, S., Sogukpinar, I., 2022. Sequential opcode embedding-based malware detection method. Computers & Electrical Engineering 98, 10770

[13] Khan, M., Baig, D., Khan, U.S., Karim, A., 2020. Malware classification framework using convolutional neural network, in: 2020 International Conference on Cyber Warfare and Security (ICCWS), pp. 1–7.

[14] Kim, J., Ban, Y., Ko, E., Cho, H., Yi, J.H., 2022b. Mapas: a practical deep learning-based android malware detection system. International Journal of Information Security , 1–1

[15] Mane, D., Kumbharkar, P., Javheri, S.B., Moorthy, R., 2022. An adaptable ensemble architecture for malware detection, in: International Conference on Innovative Computing and Communications, Springer. pp. 647–659.

[16] Maniriho, P., Mahmood, A.N., Chowdhury, M.J.M., 2022. A study on malicious software behaviour analysis and detection techniques: Taxonomy, current trends and challenges. Future Generation Computer Systems 130, 1–18.

[17] Marsh, K., Haddadpajouh, H., 2022. Ransomware threat detection: A deep learning approach, in: Handbook of Big Data Analytics and Forensics. Springer, pp. 253–269.

[18] Massarelli, L., Aniello, L., Ciccotelli, C., Querzoni, L., Ucci, D., Baldoni, R., 2017. Android malware family classification based on resource consumption over time, in: 2017 12th International Conference on Malicious and Unwanted Software (MALWARE), IEEE. pp. 31–38

[19] Millar,S.,McLaughlin,N.,delRincon,J.M.,Miller,P.,2021. Multi-viewdeep learning for zero-day android malware detection. Journal of Information Security and Applications 58, 102718

[20] Nwankpa, C., Ijomah, W., Gachagan, A., Marshall, S., 2018. Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint arXiv:1811.03378 ..