

DETECTION OF MALWARE USING CNN

A PROJECT REPORT

Submitted by

E.V. PAVAN KALYAN (18113091)

A.P. ADARSH (18113112)

S.SAI LIKITH REDDY (18113125)

Under the guidance of

DR P N RENJITH

Associate Professor

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING



HINDUSTAN INSTITUTE OF TECHNOLOGY AND SCIENCE

CHENNAI - 603 103

MAY 2022



BONAFIDE CERTIFICATE

Certified that this project report **Detection of Malware Using CNN** is the bonafide work of **E.V. Pavankalyan (18113091), A.P. Adarsh 18113112), S. Sai Likith Reddy (18113125)** who carried out the project work under my supervision during the academic year **2021-2022**.

Dr. J. THANGAKUMAR,
HoD,
Department of CSE.

Supervisor name,
DR P N RENJITH
ASSOCIATE PROFESSOR

INTERNAL EXAMINER

Name: _____

Designation: _____

EXTERNAL EXAMINER

Name: _____

Designation: _____

Project viva-voce conducted on _____

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	Acknowledgement	v
	Abstract	vi
	List of Tables	vii
	List of Figures	vii
	List of Abbreviations	vii
1	INTRODUCTION	1
	1.1 Overview	1
	1.2 motivation of the Project	1
	1.3 Problem definition and scenario	1
	1.4 Organization of the Report	2
	1.5 Summary	2
2	LITERATURE REVIEW	3
	2.1 Introduction	3
	2.2 Paper 1	3
	2.3 Paper 2	3
	2.4 Paper 3	4
	2.5 Paper 4	4
	2.6 Paper 5	5
	2.7 Paper 6	5
3	PROJECT DESCRIPTION	6
	3.1 Objective of the Project work	6
	3.2 Existing System	6
	3.3 Shortcomings of Existing system	6
	3.4 Proposed system	6
	3.5 Benefits of Proposed system	6
4	SYSTEM DESIGN	7
	4.1 Architecture Diagram	7
5	PROJECT REQUIREMENTS	8
	5.1 Hardware and Software Specification	8
	5.2 Technologies Used	8

6	MODULE DESCRIPTION	9
	6.1 Modules	9
	6.2 Module 1	9
	6.3 Module 2	9
	6.4 Module 3	10
	6.5 Module 4	10
	6.6 Module 5	10
7	IMPLEMENTATION	11
8	RESULT ANALYSIS	13
	8.1 Results Obtained	13
9	CONCLUSION AND FUTURE WORK	16
	9.1 Conclusion	16
	9.2 Future Work	16
10	INDIVIDUAL TEAM MEMBER's REPORT	17
	10.1 Individual Objective	17
	10.2 Role of the Team Members	17
	10.3 Contribution of Team Members	17
	REFERENCES	18
	APPENDIX A: SAMPLE SCREEN	19
	APPENDIX B: SAMPLE CODE	21
	APPENDIX C: PLAGIARISM REPORT	25
	APPENDIX D: PUBLICATION DETAILS	26
	APPENDIX E: TEAM DETAILS	27

ACKNOWLEDGEMENT

First and foremost we would like to thank **ALMIGHTY** who has provided us the strength to do justice to our work and contribute our best to it.

We wish to express our deep sense of gratitude from the bottom of our heart to our guide **Dr P N Renjith(Associate Professor), Computer Science and Engineering**, for his motivating discussions, overwhelming suggestions, ingenious encouragement, invaluable supervision, and exemplary guidance throughout this project work.

We would like to extend our heartfelt gratitude to **Dr. J. Thangakumar, Ph.D., Professor & Head, Department of Computer Science and Engineering** for his valuable suggestions and support in successfully completing the project.

We thank the management of **HINDUSTAN INSTITUTE OF TECHNOLOGY AND SCIENCE** for providing us the necessary facilities and support required for the successful completion of the project.

As a final word, we would like to thank each and every individual who have been a source of support and encouragement and helped us to achieve our goal and complete our project work successfully.

ABSTRACT

As malware's hazards grow from home computers to industrial control systems, detection has become mission crucial. In nowadays, cyber security task became an important and competitive work to secure the networks and systems from different attacks that are happening digitally. So for this we proposed a project which can reduce some of the attacks. This project describes malware detection system predicated upon deep learning for detecting as well as categorizing malevolent software. Our article offers convolutional neural network-based malware detection method that is very accurate and efficient. The system proceeds with binary file as input and determines whether it's harmful or benign. The binaries are just lightly pre-processed, and feature discovery is left to the network during training. A key distinction from existing convolutional neural networks is used in this proposed method.

Unlike previous existing here we use ember dataset as it contains of thousands of malware and benign files. The existing methods are using maligma dataset which is very small, in the view of getting accurate results while training. This proposed system is very useful nowadays, because we are seeing many attacks and steal the information. We are using neural networks to get better accurate results while using large datasets also. As the part of research we noted the results of existing models that are upto 75-82% accuracy. As they are using small datasets and models like knn, decision trees and as well as SVM, we are using different method and algorithm called CNN and got the better results than previous existing methods. That is 85-90% accuracy. We did this project on VM, because we are testing the model with original malware so that system is not affected.

FIGURE NO.	LIST OF TABLES TITLE	PAGE NO.
1	List of Abberviations	vii

FIGURE NO.	LIST OF FIGURES TITLE	PAGE NO.
1	Architecture diagram	7

LIST OF ABBREVIATIONS
KNN K Nearest Neighbour
CNN Convolutional Neural Networks
DT Decision Trees
SVM Support Vector Machine
NB Naïve Bayes

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Malware is kind of malevolent application which is meant to harm or damage a computer and computer system. The term "malware" is short for "malicious software." Worms, viruses, Trojan horses, adware, spyware, and ransomware are examples of common malware. Malware can spread quickly by using simple methods to achieve its goal where consumer has no knowledge of it. Furthermore, contemporary malware is fortified by advanced elusion strategies also is capable of exploiting the host's weaknesses invisibly. As a result, with the growth of technology, it has become a severe menace. Because majority of PCs are connected via numerous networks, like Internet, information technology is used. Computer In the face of such threats, security becomes increasingly crucial. Malware that is more sophisticated. As a result, an increase in automatic malware detection and classification are extremely important. Signature-based and heuristic-based malware detection strategies are both ineffective in detecting malware. Detecting advanced malware that uses evasion techniques techniques.

1.2 MOTIVATION FOR THE PROJECT

As malware's hazards grow from home computers to industrial control systems, detection has become mission crucial. Nowadays we are going through many digital attacks called malware attacks that can steal our personal information like bank details, sensitive information etc. so to reduce these attacks we want to develop a method.

1.3 PROBLEM DEFINITION AND SCENARIO

There are many existing systems or methods for malware detection using different algorithms like SVM, DT as well as KNN and many more. The problem is they are using a very small dataset in order to gain better results. And we are seeing many people complaint about these kind of attacks nowadays.

1.4 ORGANIZATION OF THE REPORT

Starting with the abstract, we move on to introduction part. After that in chapter 2 we discussed about literature review. Futher in chapter 3 we

explained project description including existing system and its disadvantages, proposed system and its benefits. After that from chapter 4 to chapter 8 we implemented the architecture diagram of proposed work and modules as well as the implementation part. Then concludes with results, conclusion and future work.

1.5 SUMMARY

Malware have proven to be particularly beneficial for invasive attacks. Because of their everchanging nature This is the nature of evolution. The primary shortcoming of the popular anti-malware software is the prevalence of modern-day malware. Antimalware technologies based on signatures Furthermore, the vast array of existing and new software in this field Anti-malware technologies are required by the ecosystem. Because there is a significant indirect cost for even minor inaccuracies, it is important to have high accuracy. There's a good chance that attack which got through one of VMs will also get into a lot of other ones. Scaling possibilities are possible because of large number of VMs available., the attacker can inject Bot ware and utilize it to create a botnet. As a result, malware detection in virtual machines is crucial.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

[1] Malware detection has become increasingly crucial as the Internet has grown globally. Unlike in the past, modern malware uses a variety of protocols, port number as well as complicated frameworks to make it harder to detect. It is possible to detect malware using techniques such as port-based approaches with deep packet inspection (DPI), although this method is likely to provide incorrect estimates of traffic linked with a variety of different port numbers and protocols since it relies on packet headers to identify traffic. DPI is conventional well-known method of spotting viruses. [2] Malware is one of the most common risks to cloud computing. Injecting spyware in perpetrator's virtual machine is a kind of cloud malware injection (VM) for manipulation. Thousands of virtual machines have the identical setup as a result of cloud's design and automated provisioning.

2.2 PAPER 1

Flow-based Malware Detection Using Convolutional Neural Network (2016, M.Yeo, Y.Koo, Y.Yoon, T.Hwang)

Malware detection approaches have always been reliant on application packet fields such as port numbers and protocols, which is why Malware that uses unreliable ports is vulnerable to these tactics. Numbers and protocols are important. More information is provided by the proposed method. Because it employs 35 various methods to identify malware, it is both robust and accurate. Instead of port numbers, features taken from packet flow as well as protocols. From Stratosphere IPS project data, nine distinct public malicious packets have been evaluated and regular packets were changed in an uninfected environment. Netmate was used to send data, and the flow data was used to identify 35 characteristics.

Random forest (RF) achieved >85% precision, accuracy, and recall for all classes, whereas Multi-layer perceptron (MLP), support vector machine (SVM), and random forest (SVM) were all utilized for classification.

2.3 PAPER 2

Malware detection in Cloud Infrastructures Using Convolutional Neural Networks (2018, Mahmoud Abdelsalam, Ravi Sandhu, Ram Krishnan)

The Convolutional Neural Network(CNN), deep learning approach, is utilised into this study to introduce and discuss a efficient malware detecting method into cloud architecture. The malware we utilized in our tests was chosen at random. This makes it easier to detect the all-time most active malware by reducing the selection bias. We show as our 2D CNN model achieves less precision when compared to 3d CNN model.

2.4 PAPER 3

IRMD: Malware variant Detection Using Opcode Image Recognition (2016, Jixin Zhang, Zheng Qin, Hui Yin, Yupeng Hu)

Binary executables are proposed to be disassembled into opcode sequences in this article, and the opcodes are then converted into pictures. All of the photos follow a uniform set of conventions. Multiplying these probabilities and gains for every pixel in a picture yields its value as an integer. The first reason is to make it easy for others to identify and classify our photographs. Histogram erosion dilation and normalization are then used to improve contrast between benign and malicious pictures. Finally, we utilise convolutional neural network (CNN) to recognize malware.

2.5 PAPER 4

CNN based Android Malware Detection (2017, Priyanka Pednekar, Meenu Ganesh, Pooja Prabhuswamy, Divyashri Sreedharan Nair, Younghee Park)

Malicious applications can gain access to sensitive data by exploiting unsolicited permission controls. They proposed a method that uses static analysis on individual application permission settings. They examined 138 application permissions and classified them into four categories based on their level of protection. These are the levels used to train the model to identify the maliciousness of the application based on the pattern of permission settings.

2.6 PAPER 5

Malware Detection Using 1-Dimensional Convolutional Neural Networks (2019, Arindam sharma, Pasquale Malacaria, MHR Khouzani)

One of the most important components of using artificial neural networks to solve complicated computer vision problems has been CNNs. The

convolution procedure, which is performed as convolutional layers part, is the foundation of CNNs. Given that the network typically acts on 2-dimensional images, a 2-dimensional variant of this algorithm is employed for the majority of computer vision applications. An image may be scanned from right to left and bottom to top using a matrix of configurable parameters, and this process captures visual elements (such a straight line) no matter where they appear in the picture; this includes both vertical and diagonal as well as straight lines.

2.7 PAPER 6

Windows Malware Detector Using Convolutional Neural Networks (2019, Shiva darshan, Jaidhar c.d)

Kolter and Maloof proposed a machine learning-based classifier based detection technique for malware that occurred in the wild. After encoding all of PE files using the N-gram approach, they used Information Gain Feature Selection Approach for selecting superior N-grams and assess performance of different classifiers.

CHAPTER 3

PROJECT DESCRIPTION

3.1 OBJECTIVE OF THE PROJECT WORK

Today malware is the important topic that can be addressed to all. Due to these digital attacks many people lost their sensitive information and also money. We don't know what is underlying on the application we installed in our system. All the previous ones are not accurate in getting results, so we want to use deep learning methods to develop our proposed method.

3.2 EXISTING SYSTEM

As we all see there are many existing systems to detect malware. And they are using different methods or algorithms like SVM, KNN and decision trees etc. they are using a dataset called maligma where it contains only a hundreds of malware and benign images.

3.3 SHORT COMINGS OF EXISTING SYSTEM

As we researched about the previously existing methodologies their results are not accurate while testing the model. They use small datasets for getting the high accuracy while training. The conversion of files to images, the results of the image is also not proper.

3.4 PROPOSED SYSTEM

So here finally we came up with a method using deep learning algorithms CNN. Our idea is to classify or detect the malware using the images of malware and benign ones. Those images are the converted images of binary executables of the features extracted from the portable executable (PE) files. This includes image classification process, so as we all know neural networks are best for this.

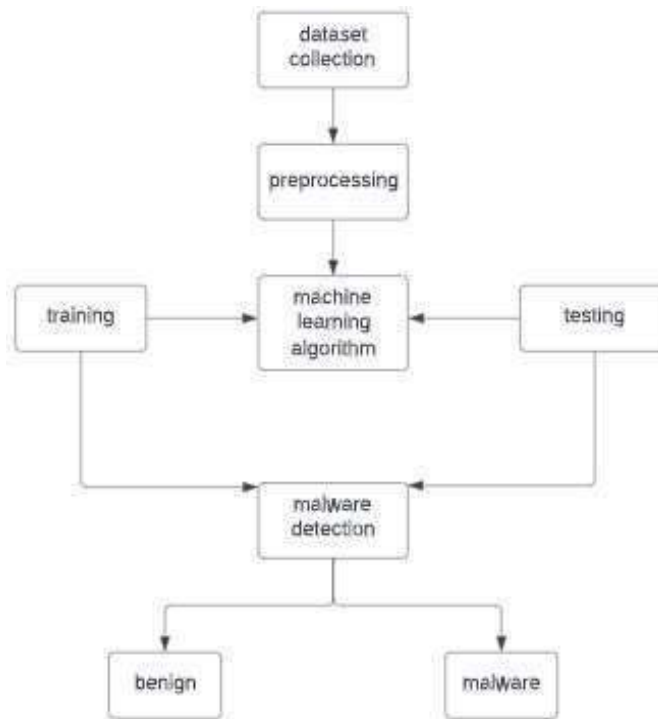
3.5 BENEFITS OF PROPOSED SYSTEM

We are using deep learning methods like neural networks which gives the required outcome accurately, though we are using large datasets. Another thing is the dataset we are using is called ember, which contains of thousands of malware and benign files. Using large datasets results in getting accurate results.

CHAPTER 4

SYSTEM DESIGN

4.1 Architecture Diagram



CHAPTER 5

PROJECT REQUIREMENTS

5.1 Hardware and Software Specifications

➤ **Hardware**

- Minimum of 8GB RAM
- 8th gen i5 Processor
- 256GB HD or SDD

➤ **Software**

- Python
- Tensorflow
- Flask
- Any Python IDE
- HTML
- CSS

5.2 TECHNOLOGIES USED

- **Python**

It is a programming language which will be used for the implementation of models with the help of other libraries

- **Tensorflow**

It is used to train the model and helps to save the models

- **Flask**

Flask is a micro backend framework based on python. In here we used this to build web application for our project.

- **HTML & CSS**

HTML is used to create web pages. In this project we create web pages to upload a file to get result. CSS is used for proper alignments and the buttons to click.

CHAPTER 6

MODULE DESCRIPTION

6.1 Modules

- Data collection
- Data preparation
- Model building
- User interface
- Integration

6.2 Module 1: Data collection

The data collection is the method of collecting, measuring & analysing the data is called data collection. We have collected the ember set for malware detection using CNN in this data set it consists of one lakh of data. Where its is a very huge combination of both benign and malicious files.

6.3 Module 2: Data preparation

The process of cleaning and transforming raw data prior to processing and analysis. It is an important step prior to processing and often involves reformatting data, making corrections to data and the combining of data sets to enrich data. From this data set we have removed the unwanted data and we took only the wanted data for the classification. We split the data into two. One is for data training (80) and testing (20).

6.4 Module 3: Model building

From this dataset we can build our model using keras. The following structure will be used, which contains a stack of conv2d and maxpooling 2d layers. These are some layers of our CNN model. Convolution Layer, Max Pooling Layer, Max Pooling Layer, DropOut Layer, Flatter Layer, Dense/Fully Connected Layer, DropOut Layer, Dense/Fully Layer.

6.5 Module 4: User interface

We have created a user interface where we can choose the file to check wheather it is malicious or safe. After uploading the file we can see a button to show the result of the file. We use html for creating web page and used CNN for alignment as well as buttons.

6.6 Module 5: Integration

At last integrate the model and the user interface for getting the required result

.

CHAPTER 7

IMPLEMENTATION

Normally, here we use some existed dataset that directly contains the images of malware and benign. Otherwise there is a method that convert a malware and benign into an image. Here, raw data with every file comprises hexadecimal forms of files binary content. Thought is changing those files in images so that they can be used as input to our CNN model.

Here is an example image of binary file conversion into a PNG. From this method we can develop our own malware image datasets.

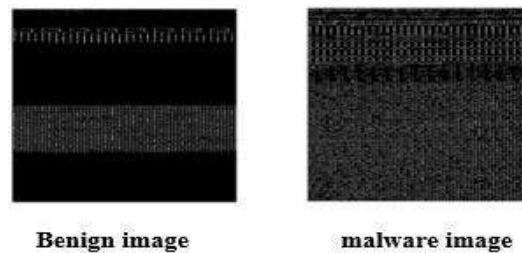


Fig.2 PNG of malware & benign

After the dataset is collected or created and preprocessed, we need to split the data between training and testing data at 70-30 ratio.

We can begin using keras to create our model now that dataset is available. Several factors must be taken into account while creating a CNN, such as how many layers there are in a network and how big the kernels are. CNN's organizational structure is as follows: Max pooling layer of (2*2) pool size. convolutional layer of 30 filters, (3*3) kernel size.

From the research and some experiments we selected finest considerations for developing our CNN: 2 hidden layers, 16 filters and 64*64 images. That is how structure of network would be. A convolutional layer could be made up of a pooling layer, numerous conv2d layers, , and sometimes a dropout layer. As a result, we have many conv2d layers clustered into 2 layers in our case. The flatten

layers combine the output into a single long vector, which is then processed by a classification layer.

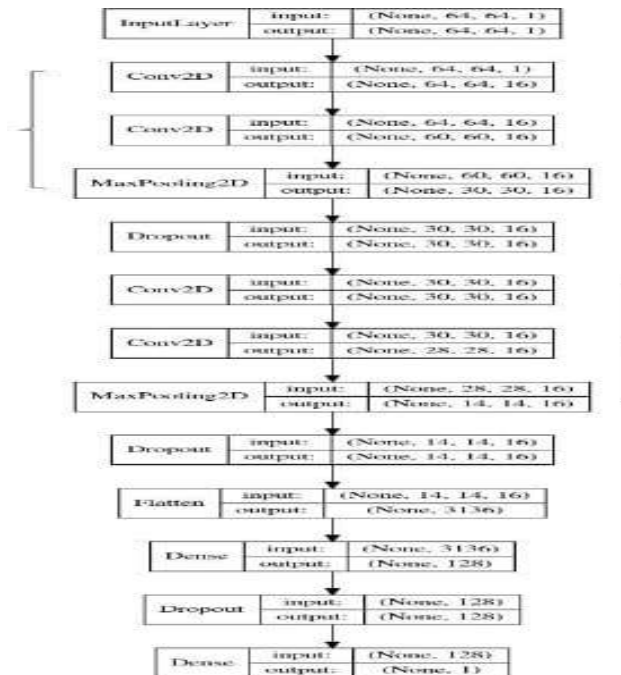


Fig.3 model architecture

After the training and testing we will print the accuracy and score of the model. The result of this experiment would be the prediction of a file is malware or benign.

Coming to the user interface part, we create a web page using html, where it has an option to upload a file and a button to test the file. As we already integrate the model it directly redirect to another web page of result.

CHAPTER 8

RESULT ANALYSIS

8.1 RESULT OBTAINED

At first we conducted an experiment that will decide which size of an image would do better classification. We took images of different sizes like 256*256, 128*128, 64*64 and 32*32. After the experiment was done, we came to know that both 64*64 and 32*32 got the high accuracy on classification. Here is the graph that shows the results.

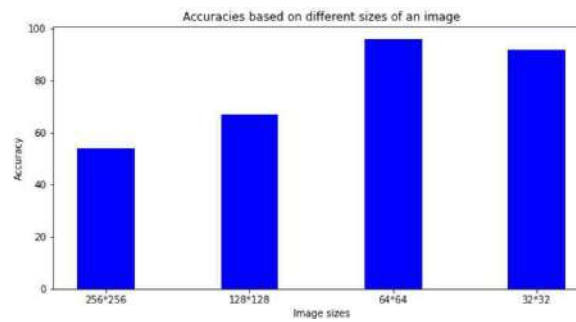


Fig accuracies on different images

As compared, 64*64 is the best option. So we proceed with this image size and did the model training and testing. The following metrics are used to evaluate the experimental outputs obtained in this work.

Accuracy: It is defined as the total no of correctly predicted samples divided with total no of samples. The formula is as follows:

$$\text{Accurateness} = \frac{\text{no of correct predictions}}{\text{total no of predictions}} = \frac{TP+TN}{TP+FP+TN+FN}$$

Where FP is false positive, TP is true positive, FN is false negative and TN is true negative.

Loss: It considers the prediction's uncertainty based on how far it deviates from the actual label. That is given by:

$$Loss = \frac{-1}{n} \sum_{i=1}^n \sum_{j=1}^m y_{ij} \log(P_{ij})$$

Here n is total no of samples, m is total no of classes.

Also, the training and testing time are used for measuring of the effectiveness of the model that are employed.

For this training we used 64*64 image size for the dataset. The following graph shows the accuracy that was got after testing the model with CNN on the dataset we loaded and got the accuracy of >95%.

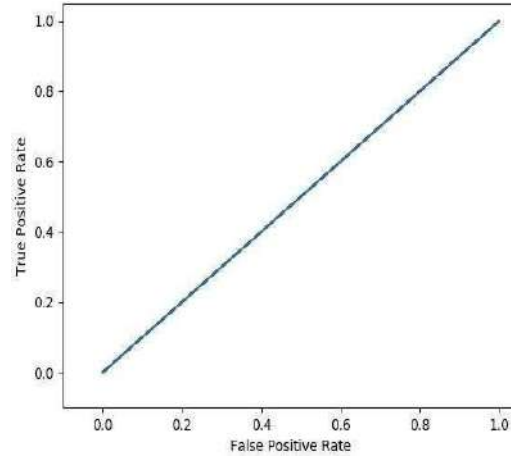


Fig true and false positives

As per the results we came to know that, training the model upto some level of epochs will get the better results. And then we calculated the loss of the model with the same epochs. And the following graph shows hoe the model got less loss after we increased the epoch size.

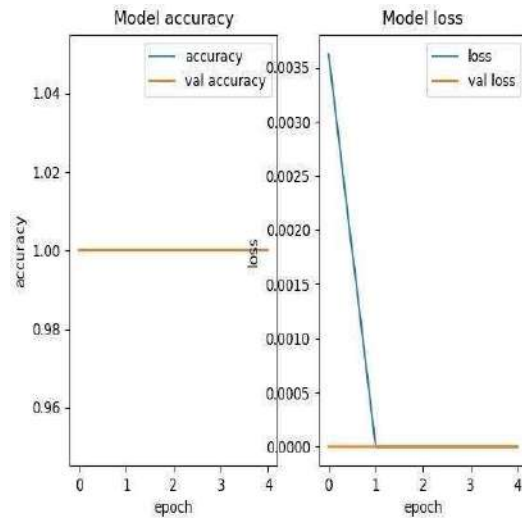


Fig accuracy and loss of the model

Here we got the model loss lesser than 0,15. That shows, our model did the best in detecting the malware and benign files accurately.

CHAPTER 9

CONCLUSION AND FUTURE WORK

6.1 Conclusion

Malware has become a widespread issue in computer security. As a result, an effective malware detection strategy that reliably identifies malware is critical. Traditional malware detection techniques exist, but they are ineffective in distinguish unusual malware. Therefore In this paper we have classified the data whether the data contains malware or not using machine learning algorithm. The CNN algorithm is used to classify and detect the malware from extracting the features from the PE files. CNN algorithm has the superior accuracy and efficient when compared to other algorithms. After performing the algorithm, we achieved an accuracy of 95%, which is not a bad one.

6.2 FUTURE WORK

In future we need to train the model with the newly arrived malware types as well as improve the web app design and add some more features to our application.

CHAPTER 10

INDIVIDUAL TEAM MEMBER's REPORT

10.1 Contribution of Team Members

E.V. PAVAN KALYAN (18113091) did the data collection, cleaning and helped in the paper work.

A.P. ADARSH (18113112) did the model training, testing and helped in the paper work.

S. SAI LIKITH REDDY (18113125) did the web app using flask and helped in paper work.

REFERENCES

- [1] IM. Yeo, 1Y. Koo, 1Y. Yoon, 1T. Hwang, 1J. Ryu, 2J. Song, *1C. Park. "Flow-based Malware Detection Using Convolutional Neural Network"
- [2] Mahmoud Abdelsalam, Ravi Sandhu, Yufei Huang and Ram Krishnan. "Malware Detection in Cloud Infrastructures using Convolutional Neural Networks ". IEEE 11th International Conference on Cloud Computing, 2018.
- [3] Jixin Zhang, Zheng Qin, Hui Yin, Lu Ou, Yupeng Hu. "IRMD: Malware variant Detection using opcodeImage Recognition". IEEE 22nd International Conference on Parallel and Distributed Systems, 2016.
- [4] Wei Yuan , Yuan Jiang, Heng Li , and Minghui Cai. "A Lightweight On-Device DetectionMethod for Android Malware ". IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS.
- [5] Priyanka Pednekar, Meenu Ganesh, , Pooja Prabhuswamy, Divyashri Sreedharan Nair, Hyeran Jeon , Younghee Park. "CNN-Based Android Malware Detection". 2017 International Conference on Software Security and Assurance
- [6] Arindam Sharma, Pasquale Malacaria, MHR Khouzani "Malware Detection Using 1-Dimensional Convolutional Neural Networks".2019 ieee european symposium on security and privacy work shops
- [7] shiva darshan s.l, Jaidhar c.d "Windows Malware Detector Using Convolutional Neural Network Based on Visualization Images". IEEE emerging topics in computing, 2019.
- [8] Sunita Choudhary, Anand Sharma "Malware Detection & Classification using Machine Learning". 2020 International Conference on Emerging Trends in Communication, Control and Computing (ICONC3) Mody University of Science and Technology, Lakshmangarh.
- [9] Pradosh priyadarshan, Prateek sarangi, Adyashrath, Ganapathi panda." Machine learning based improved malware detection schemes". 11th international conference on data science, cloud computing and engineering 2021.

APPENDIX A: SAMPLE SCREEN

Analyze suspicious files to detect malware

Upload your file here

Choose File No file chosen

SubmitReset

Analyze suspicious files to detect malware

Upload your file here

Choose File C:\a\...

SubmitReset

The model returned the status

1

Your file is safe

Analyze suspicious files to detect malware

Upload your file here

Choose File

The model returned the status

0

Your file is malicious

1

7/1

1 security vendor and no vendors flagged this file as malicious

17.82 KB

2025-11-07 14:19:31 UTC

1 year ago

72

SHA256: 94718781007032830266c4c19f84021a6b7840e0f5a42a1f1209a4224421

0x0

Community Score

0

DETECTION

DETAILS

COMMUNITY

Security Vendors Analysis

NANO Antivirus	⚠️ Virus Shield (Outgoing)	AV-Aware	🛡️ Undetected
Avira Lab	🛡️ Undetected	Avira-Lab-VE	🛡️ Undetected
Avast	🛡️ Undetected	Avast-AVL	🛡️ Undetected
Avast	🛡️ Undetected	Avast	🛡️ Undetected
Avast	🛡️ Undetected	Avast (in cloud)	🛡️ Undetected
Baidu	🛡️ Undetected	BitDefender	🛡️ Undetected
BitDefender Thawt	🛡️ Undetected	Blue-Box	🛡️ Undetected
Cipher	🛡️ Undetected	CMC	🛡️ Undetected
Comodo	🛡️ Undetected	Cyren	🛡️ Undetected
Cyren	🛡️ Undetected	Dynab	🛡️ Undetected
Emsisoft	🛡️ Undetected	ELC	🛡️ Undetected
ESET NOD32	🛡️ Undetected	E-Secure	🛡️ Undetected


```

File Edit Selection View Go Run Terminal Help
feature_sel_and_model.ipynb - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

model_brain5 data_extraction.ipynb feature_sel_and_model.ipynb X
C:\Users> cd > Downloads > codes > codes > feature_sel_and_model.ipynb > Import embedpath = "embed_data\DX_train,X_train,X_test,y_test = embed_read_vectorized_features(path\embeddata = embed_read_metadata(path\DX_train,meta

+ Code + Markdown ...

X = X_train[y_train != -1]
y = y_train[y_train != -1]

Python

labels = (y_train != -1)
X_train = X_train[labels]
y_train = y_train[labels]

Python

import h5py
h5f = h5py.File('X_train.h5', 'w')
h5f.create_dataset('X_train', data=X_train)
h5f.close()
h5f = h5py.File('y_train.h5', 'w')
h5f.create_dataset('y_train', data=y_train)
h5f.close()

Python

# MODEL MAKING
def make_model():
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers

    tf.compat.v1.disable_eager_execution()
    keras.backend.clear_session()
    # no. of extracted features from H5file = 2381
    feature_size = 2381

    Model: "sequential"
    Layer (type) Output Shape Param #
    -----
    dropout (Dropout) (None, 1, 2381) 0
    dense (Dense) (None, 1, 1500) 3573000
    dropout_1 (Dropout) (None, 1, 1500) 0
    dense_1 (Dense) (None, 1, 1) 1501
    -----
    Total params: 3574501
    Trainable params: 3574501
    Non-trainable params: 0
    None

    from sklearn.preprocessing import StandardScaler
    ssc = StandardScaler()
    for x in range(0,600000,100000):

```

```

File Edit Selection View Go Run Terminal Help
feature_sel_and_model.ipynb - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

model_brain5 data_extraction.ipynb feature_sel_and_model.ipynb X
C:\Users> cd > Downloads > codes > codes > feature_sel_and_model.ipynb > Import embedpath = "embed_data\DX_train,X_train,X_test,y_test = embed_read_vectorized_features(path\embeddata = embed_read_metadata(path\DX_train,meta

+ Code + Markdown ...

print(model.summary())
return model

Python

model = make_model()

Python

Model: "sequential"
Layer (type) Output Shape Param #
-----
dropout (Dropout) (None, 1, 2381) 0
dense (Dense) (None, 1, 1500) 3573000
dropout_1 (Dropout) (None, 1, 1500) 0
dense_1 (Dense) (None, 1, 1) 1501
-----
Total params: 3574501
Trainable params: 3574501
Non-trainable params: 0
None

from sklearn.preprocessing import StandardScaler
ssc = StandardScaler()
for x in range(0,600000,100000):

```

```

File Edit Selection View Go Run Terminal Help
feature_set_and_model.ipynb - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

model_final.h5 data_extraction.ipynb feature_set_and_model.ipynb
C:\Users> cd > Downloads > codes > feature_set_and_model.ipynb > import embedpath = "embed_dataset" <X_train, X_test, y_train, y_test = embed.read_vectorized_features(path)\metadatas = embed.read_metadata(path)\X_train, meta...

Train on 480000 samples, validate on 120000 samples
Epoch 1/30
480000/480000 [-----] - 132s 275us/sample - loss: 13.5765 - accuracy: 0.9025 - auc_2: 0.9358 - precision_2: 0.8806 - val_loss: 76.4952 - val_accuracy: 0.8815 - val_auc_2: 0.9162 -
val_precision_2: 0.9150
Epoch 2/30
480000/480000 [-----] - 131s 274us/sample - loss: 19.0699 - accuracy: 0.9034 - auc_2: 0.9352 - precision_2: 0.8798 - val_loss: 121.3589 - val_accuracy: 0.8809 - val_auc_2: 0.9194 -
val_precision_2: 0.9261
Epoch 3/30
480000/480000 [-----] - 130s 271us/sample - loss: 18.5595 - accuracy: 0.9022 - auc_2: 0.9372 - precision_2: 0.8803 - val_loss: 100.8005 - val_accuracy: 0.8816 - val_auc_2: 0.9194 -
val_precision_2: 0.9213
Epoch 4/30
480000/480000 [-----] - 130s 272us/sample - loss: 14.7903 - accuracy: 0.9013 - auc_2: 0.9394 - precision_2: 0.8757 - val_loss: 98.1234 - val_accuracy: 0.8828 - val_auc_2: 0.9214 -
val_precision_2: 0.9129
Epoch 5/30
480000/480000 [-----] - 132s 274us/sample - loss: 14.8438 - accuracy: 0.9010 - auc_2: 0.9398 - precision_2: 0.8837 - val_loss: 115.4747 - val_accuracy: 0.8764 - val_auc_2: 0.9184 -
val_precision_2: 0.9123
Epoch 6/30
480000/480000 [-----] - 131s 274us/sample - loss: 16.8825 - accuracy: 0.8977 - auc_2: 0.9373 - precision_2: 0.8738 - val_loss: 141.4731 - val_accuracy: 0.8746 - val_auc_2: 0.9169 -
val_precision_2: 0.9094
Epoch 7/30
480000/480000 [-----] - 132s 275us/sample - loss: 23.7689 - accuracy: 0.9011 - auc_2: 0.9396 - precision_2: 0.8734 - val_loss: 160.3509 - val_accuracy: 0.8761 - val_auc_2: 0.9190 -
val_precision_2: 0.9621
Epoch 8/30
480000/480000 [-----] - 133s 276us/sample - loss: 14.8833 - accuracy: 0.9011 - auc_2: 0.9414 - precision_2: 0.8700 - val_loss: 136.6552 - val_accuracy: 0.8816 - val_auc_2: 0.9207 -
val_precision_2: 0.9035
Epoch 9/30
480000/480000 [-----] - 132s 276us/sample - loss: 18.4307 - accuracy: 0.8955 - auc_2: 0.9380 - precision_2: 0.8770 - val_loss: 211.6149 - val_accuracy: 0.8436 - val_auc_2: 0.8956 -
val_precision_2: 0.9636
Epoch 10/30
480000/480000 [-----] - 133s 276us/sample - loss: 19.4518 - accuracy: 0.8977 - auc_2: 0.9407 - precision_2: 0.8626 - val_loss: 209.4780 - val_accuracy: 0.8693 - val_auc_2: 0.9173 -
val_precision_2: 0.9101

```

```

File Edit Selection View Go Run Terminal Help
feature_set_and_model.ipynb - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

model_final.h5 data_extraction.ipynb feature_set_and_model.ipynb
C:\Users> cd > Downloads > codes > feature_set_and_model.ipynb > import embedpath = "embed_dataset" <X_train, X_test, y_train, y_test = embed.read_vectorized_features(path)\metadatas = embed.read_metadata(path)\X_train, meta...

# TRAINING THE MODEL
import tensorflow
tensorflow.compat.v1.disable_eager_execution()
base_dir = ""

history = model.fit(X_scaled, y,
                    batch_size=136,
                    epochs=31,
                    shuffle=True,
                    validation_data=((X_scaled[600000:1200000], y[600000:1200000])))

name="model_final.h5"
weights=weights_final.h5"
model.save(base_dir+name)
model.save_weights(base_dir+weights)
print(name, weights, "are saved.")

Train on 600000 samples, validate on 120000 samples
600000/600000 [-----] - ETA: 0s - loss: 0.2198 - accuracy: 0.9107 - auc: 0.9726 - precision: 0.9134

C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training_v1.py:2057: UserWarning: 'Model.state_updates' will be removed in a future version. This property should not be used in TensorFlow 2.0, as 'updates' are
applied automatically.
updates = self.state_updates

600000/600000 [-----] - 203s 339us/sample - loss: 0.2198 - accuracy: 0.9107 - auc: 0.9726 - precision: 0.9134 - val_loss: 0.1306 - val_accuracy: 0.9436 - val_auc: 0.9890 - val_precision:
0.9583
model_final.h5 weights_final.h5 are saved.

```

```
File Edit Selection View Go Run Terminal Help
feature_sel_and_model.pyrb - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
C:\Users> cd > Downloads > codes > feature_sel_and_model.pyrb > import embedpath = "embed_dataset" <X_train, y_train, X_test, y_test = embed.read_vectorized_features(path)/metdata = embed.read_metadata(path)/X_train, meta...

Total params: 5,826,001
Trainable params: 5,826,001
Non-trainable params: 0

# TESTING
res = trained_model.evaluate(x_test_scaled, y_test_scaled)
print("ACCURACY: [res[1]] and LOSS: [res[0]]")

ACCURACY 0.9050149917602539 and LOSS 0.2308523683208227

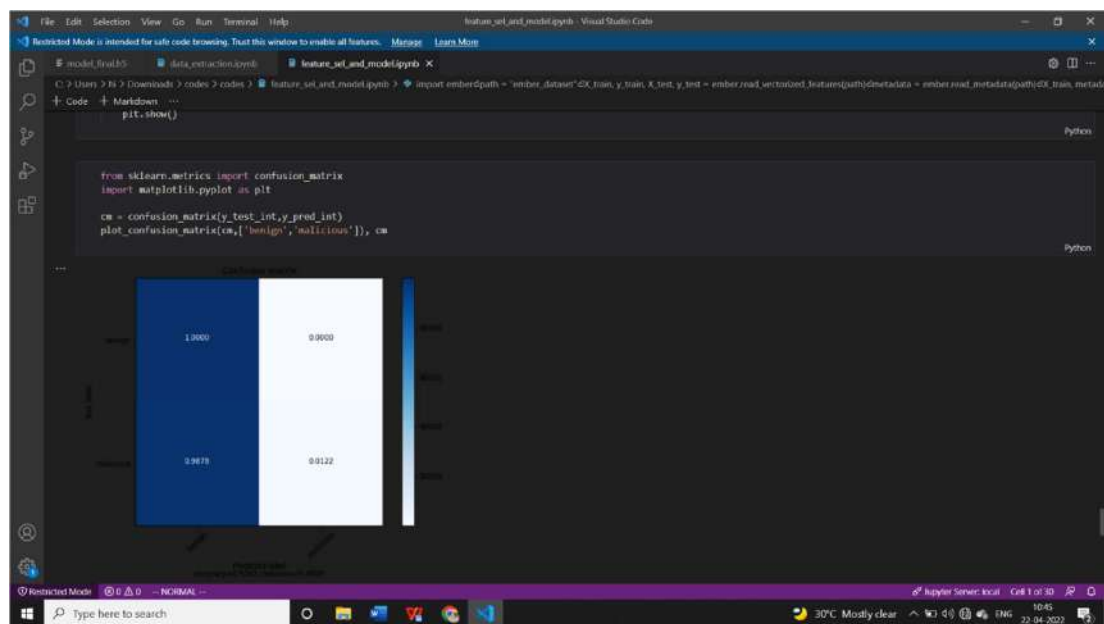
predicted = trained_model.predict(X_test_scaled)

C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training_v1.py:2079: UserWarning: 'Model.state_updates' will be removed in a future version. This property should not be used in Tensorflow 2.0, as 'updates' are
applied automatically.
updates=self.state_updates,

y_test_scaled = np.reshape(y_test_scaled, (-1))
y_pred = np.reshape(predicted, (-1))

y_test_scaled, y_pred, len(y_test_scaled), len(y_pred)

(mnemonic[1, 0, 1, ..., 0, 0, 1], dtype=float32)
array([9.9975538e-01, 1.4440119e-01, 9.9999678e-01, ..., 7.1203322e-06,
2.4233449e-01, 9.9961364e-01], dtype=float32)
```



APPENDIX C: PLAGIARISM REPORT

36

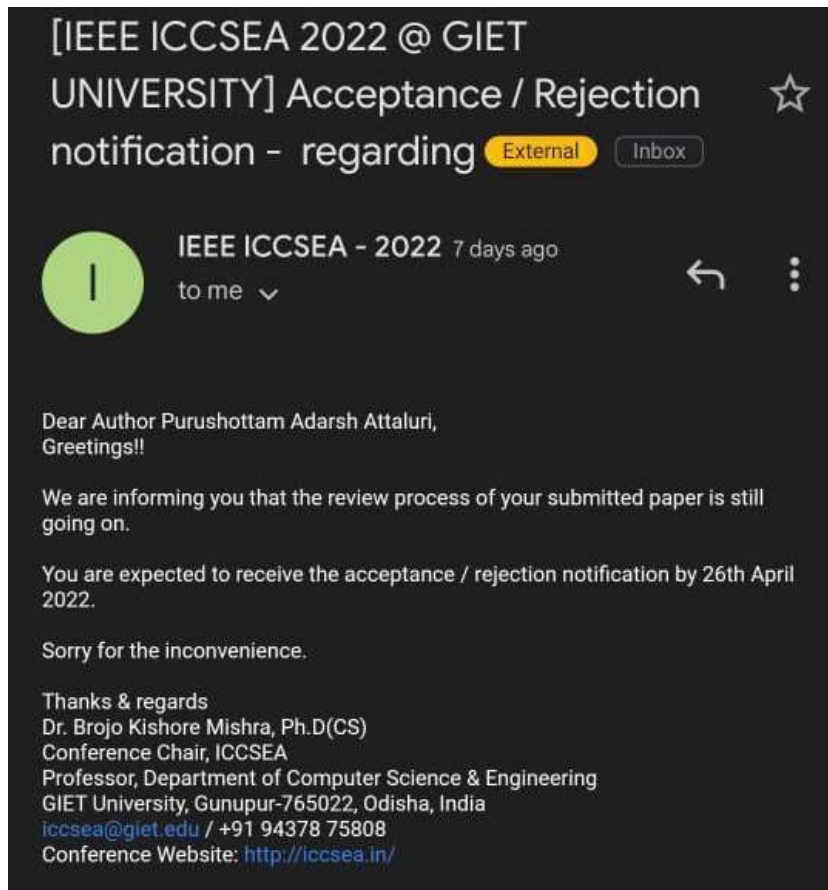
ORIGINALITY REPORT

13%	12%	4%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.coursehero.com Internet Source	6%
2	Jixin Zhang, Zheng Qin, Hui Yin, Lu Ou, Yupeng Hu. "IRMD: Malware Variant Detection Using Opcode Image Recognition", 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), 2016 Publication	1%
3	Submitted to North South University Student Paper	1%
4	koreascience.or.kr Internet Source	1%
5	www.msc-gima.nl Internet Source	1%
6	www.ijrte.org Internet Source	1%
7	Submitted to B.S. Abdur Rahman University Student Paper	<1%
8	www.scilit.net Internet Source	

APPENDIX D: PUBLICATION DETAILS



APPENDIX E: TEAM DETAILS

MEMBER 1: Emmela. Venkata Pavan Kalyan

ROLL NO : 18113091

DEPT : CSE

MEMBER 2: Attaluri. Purushottam Adarsh

ROLL NO : 18113112

DEPT : CSE

MEMBER 3: Simhadri. Sai Likith Reddy

ROLL NO : 18113125

DEPT : CSE