A

Major Project

On

MALWARE DETECTION USING DEEP LEARNING WITH IMAGE PROCESSING

(Submitted in partial fulfillment of the requirements for the award of degree)

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINERRING

By

B.VIVEKANANDA (197R1A05C7)

Under the guidance of

Dr. K. SRUJAN RAJU

(Professor)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi)

Recognized Under Section 2(f) &12(B) of the UGC Act.1956, Kandlakoya (V), Medchal Road,

Hyderabad-501401

2019-2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled "**MALWARE DETECTION USING DEEP LEARNING WITH IMAGE PROCESSING** " being submitted by **B.VIVEKANANDA (197R1A05C7)** in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science and Engineering to the Jawaharlal Nehru Technological University Hyderabad, is a record of bonafide work carried out by them under our guidance and supervision during the year 2022-23.

The results embodied in this thesis have not been submitted to any other. University or Institute for the award of any degree or diploma.

Dr. K. SRUJAN RAJU	**Dr. A. RAJI REDDY**
(Professor)	DIRECTOR
INTERNAL GUIDE	
Dr. K. SRUJAN RAJU	**EXTERNAL EXAMINER**
HOD	

Submitted for viva voice Examination held on ___________________

ACKNOWLWDGEMENT

Apart from the efforts of mine, the success of any project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project.

I take this opportunity to express my profound gratitude and deep regard to my guide **Dr. K. SRUJAN RAJU,** Professor for his exemplary guidance, monitoring, and constant encouragement throughout the project work. The blessing, help, and advice given by him shall carry us a long way in the journey of life on which we are about to embark.

I also take this opportunity to express a deep sense of gratitude to the Project Review Committee (PRC) **Dr. Punyaban Patel, Ms. K. Shilpa, Dr. T . Subha Mastan Rao & J. Narasimharao** for their cordial support, valuable information and guidance, which helped me in completing this task through various stages.

I'm also thankful to **Dr. K. Srujan Raju**, Head of the Department of Computer Science and Engineering, **Dr. Ashuthosh Saxena**, Dean R&D, and **Dr. D T V Dharmajee Rao**, Dean Academics for providing encouragement and support for completing this project successfully.

I'm obliged to **Dr. A. Raji Reddy**, Director for being cooperative throughout thecourse of this project. I also express my sincere gratitude to Sri. **Ch. Gopal Reddy**, Chairman for providing excellent infrastructure and a nice atmosphere throughout the course of this project.

The guidance and support received from all the members of CMR Technical Campus contributed to the completion of the project. I am grateful for their constant support and help.

Finally, I would like to take this opportunity to thank my family for their constant encouragement, without which this assignment would not be completed. I sincerely acknowledge and thank all those who gave support directly and indirectly in the completion of this project.

B.VIVEKANANDA (197R1A05C7)

ABSTRACT

The increasing frequency of protection breaches caused by malicious software, commonly known as malware, is a significant security concern in the digital age. With a growing number of computer users, organizations, and governments falling victim to malware attacks, detecting and mitigating malware has become a critical area of research. Traditional malware detection methods, which rely on static and dynamic analysis of malware signatures and behavior patterns, are time-consuming and have proven to be ineffective in identifying unknown malwares in real-time.

Modern malwares are designed to evade detection by using techniques such as polymorphism and metamorphism, which rapidly change the behavior of the malware and generate numerous variants of the same malware. Machine learning algorithms (MLAs) have been increasingly employed to conduct effective malware analysis, particularly using feature engineering, feature learning, and feature representation. However, these methods also require significant time and resources.

To address this challenge, advanced MLAs, such as deep learning, can be utilized to eliminate the need for extensive feature engineering. Deep learning algorithms are capable of automatically extracting relevant features from the data, allowing for more efficient and effective malware analysis. By leveraging the power of deep learning, the time-consuming feature engineering phase in traditional MLAs can be circumvented, leading to faster and more accurate detection of unknown malwares in real-time.

In conclusion, the increasing sophistication and evasive techniques employed by modern malwares pose a significant security challenge. Traditional malware detection methods are ineffective and time-consuming. However, advanced MLAs, such as deep learning, offer promising solutions by automating feature extraction and enabling real-time detection of unknown malwares, thereby improving the effectiveness and efficiency of malware analysis in the virtual age.

LIST OF FIGURES / TABLES

FIGURE / TABLE NO	FIGURE/TABLE NAME	PAGE NO
Table 2.1	Malware Family and its Malware Kind	11
Figure 3.1	Project Architecture of Malware Detection using Deep Learning with Image Processing	12
Figure 3.2	Use Case Diagram of Malware Detection using Deep Learning with Image Processing	14
Figure 3.3	Sequence Diagram of Malware Detection using Deep Learning with Image Processing.	15
Figure 3.4	Activity Diagram of Malware Detection using Deep Learning with Image Processing.	16
Table 6.1	Test Cases	40

LIST OF SCREENSHOTS

SCREENSHOT NO	SCREENSHOT NAME	PAGE NO
SCREENSHOT 5.1	Training Data set	25
SCREENSHOT 5.2	Validation Data Set	26
SCREENSHOT 5.3	Types of Malware as Png Images	27
SCREENSHOT 5.4	Initial Epochs for Machine when not re-scaled	28
SCREENSHOT 5.5	Final Epochs for Machine when not re-scaled	28
SCREENSHOT 5.6	Accuracy and Loss for the Given Dataset when not re-scaled	29
SCREENSHOT 5.7	Initial Epochs for the Machine when re-scaled	30
SCREENSHOT 5.8	Final Epochs for Machine when re-scaled	30
SCREENSHOT 5.9	Accuracy and Loss for the Given Dataset when re-scaled	31
SCREENSHOT 5.10	Summary of the Machine	32

TABLE OF CONTENTS

ABSTRACT i
LIST OF FIGURES ii
LIST OF SCREENSHOTS iii
1. INTRODUCTION 1
1.1 PROJECT SCOPE 1
1.2 PROJECT PURPOSE 1
1.3 PROJECT FEATURES 2
2. SYSTEM ANALYSIS 3
2.1 PROBLEM DEFINITION 3
2.2 EXISTING SYSTEM 3
2.2.1 DISADVANTAGES OF THE EXISTING SYSTEM 3
2.3 PROPOSED SYSTEM 4
2.3.1 ADVANTAGES OF THE PROPOSED SYSTEM 4
2.4 FEASIBILITY STUDY 4
3.4.1 ECONOMIC FEASIBILITY 4
3.4.2 TECHNICAL FEASIBILITY 5
3.4.3 BEHAVIORAL FEASIBILITY 5
2.5 HARDWARE & SOFTWARE REQUIREMENTS 5
2.5.1 HARDWARE REQUIREMENTS 5
2.5.2 SOFTWARE REQUIREMENTS 6
3. ARCHITECTURE 12
3.1 PROJECT ARCHITECTURE 12
3.2 DESCRIPTION 12
3.3 USE CASE DIAGRAM 14
3.4 SEQUENCE DIAGRAM 15
3.5 ACTIVITY DIAGRAM 16
4. IMPLEMENTATION 17
5. RESULTS 25
6. TESTING 33
6.1 INTRODUCTION TO TESTING 33
6.2 TYPES OF TESTING 33
6.2.1 UNIT TESTING 33
6.2.2 INTEGRATION TESTING 34
6.2.3 FUNCTIONAL TESTING 34
6.3 TEST CASES 35
7. CONCLUSION & FUTURE SCOPE 44
8. BIBLIOGRAPHY 46
GITHUB LINK 46
9. MANUSCRIPT 47

1. INTRODUCTION

1. INTRODUCTION

1.1 PROJECT SCOPE

The project titled "Malware Detection Using Deep Learning with Image Processing" aims to leverage the power of deep learning to improve the efficiency of detecting malware in devices. Traditional methods of malware detection can be costly, inefficient, and time-consuming. However, with advancements in technologies like deep learning, there is an opportunity to upgrade these methods for better performance. By utilizing deep learning techniques, the project aims to enhance the accuracy and efficiency of malware detection. This project seeks to explore the potential of image processing in combination with deep learning for detecting malware. By leveraging these cutting-edge technologies, this project aims to provide a more effective solution for malware detection, addressing the limitations of traditional methods and improving the overall detection capabilities.

1.2 PROJECT PURPOSE

The main purpose of this project is to develop a malware detection system using deep learning and image processing techniques. The process involves converting a file into a grayscale image. The machine is then trained with a dataset to determine whether the converted image represents a malware or not. This approach leverages the power of deep learning algorithms to automatically extract relevant features from the grayscale images, allowing for efficient detection of malware. The system aims to improve the accuracy and efficiency of malware detection compared to traditional methods. The grayscale image conversion and deep learning-based detection process enable real-time analysis of files for malware presence. The project combines the fields of deep learning and image processing to create an effective and efficient malware detection system. This approach eliminates the need for extensive feature engineering, making the detection process faster and more accurate. The project has the potential to contribute to the advancement of malware detection techniques using cutting-edge technologies.

1.3 PROJECT FEATURES

This project employs Convolutional Neural Networks (CNN), a type of deep learning approach, for the categorization of malware families. CNNs are widely used for image/object recognition and classification tasks due to their ability to automatically learn relevant features from input data. The project utilizes visualization techniques in combination with deep learning to categorize malware families based on their visual characteristics.

The CNN architecture consists of convolutional layers, which extract features from the input data by applying filters to capture local patterns. The output of convolutional layers is then passed through pooling layers, which down-sample the feature maps, reducing spatial dimensions while retaining important information. This helps in reducing computational complexity and improving efficiency.

Finally, the output of pooling layers is passed through one or more fully connected layers, which are responsible for making predictions or classifying the input data into different malware families. The fully connected layers learn complex patterns and relationships between features extracted from the input data.

The combination of CNN architecture, visualization techniques, and deep learning enables effective and efficient categorization of malware families based on their visual characteristics, contributing to the advancement of malware detection and analysis techniques..

2. SYSTEM ANALYSIS

2. SYSTEM ANALYSIS

SYSTEM ANALYSIS

System Analysis is the important phase in the system development process. The System is studied to the minute details and analyzed. The system analyst plays an important role of an interrogator and dwells deep into the working of the present system. In analysis, a detailed study of these operations performed by the system and their relationships within and outside the system is done. A key question considered here is, "what must be done to solve the problem?" The system is viewed as a whole and the inputs to the system are identified. Once analysis is completed the analyst has a firm understanding of what is to be done.

2.1 PROBLEM DEFINITION

A real time deep learning algorithm to detect and classify malware with image processing techniques available such as Convolutional Neural Network.

2.2 EXISTING SYSTEM

A traditional malware detection system does pattern matching among the file, to consider whether a particular file is malware or not. It uses ready made algorithm for detecting malware. While detecting the malware a little decryption of the file is done.

2.2.1 DISADVANTAGES OF EXISTING SYSTEM

Following are the disadvantages of the existing system:

- The malware uses polymorphic, metamorphic or evasive techniques.
- The algorithms are static.
- Only partial decryption is done on the files.
- The algorithm that supports one Operating System may not support another operating system.

2.3 PROPOSED SYSTEM

A Real time Malware Detection System which uses deep learning with image processing techniques.

2.3.1 ADVANTAGES OF PROPOSED SYSTEM

Following are the advantages of the proposed system:

- This method is fastest as compared to static and dynamic approaches.
- The proposed method is agnostic to packed malware. In other words, packed malware variants from the unpacked malware can contain visual similarity.
- The proposed system has the capability to work on malwares from different OS such as Windows, Android Linux etc.

2.4 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and a business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. Three key considerations involved in the feasibility analysis:

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

2.4.1 ECONOMIC FEASIBILITY

The developing system must be justified by cost and benefit. Criteria to ensure that effort is concentrated on a project, which will give best, return at the earliest. One of the factors, which affect the development of a new system, is the cost it would require.

The following are some of the important financial questions asked during preliminary investigation :

The cost conducts a full investigation system. The cost of hardware and software.

The benefits are in the form of reduced costs of fewer costly errors.

Since the system is developed as part of project work, there is no manual cost to spend for the proposed system. Also, all the resources are already available, it gives an indication that the system is economically possible for development.

2.4.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

2.4.3 SOCIAL FEASIBILITY

This includes the following questions:

- Is there sufficient support for the users?
- Will the proposed system cause harm?

The project would be beneficial because it satisfies the objectives when developed and installed. All behavioral aspects are considered carefully and conclude that the project is behaviorally feasible.

2.5 HARDWARE & SOFTWARE REQUIREMENTS

2.5.1 HARDWARE REQUIREMENTS:

Hardware interfaces specify the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements.

•Processor: x86_64 CPU architecture; 5th generation Intel Core or newer.

•Hard Disk: 8GB or more.

•RAM: 8GB or more.

•Screen: 1280 x 800 px.

2.5.2 SOFTWARE REQUIREMENTS:

Software Requirements specifies the logical characteristics of each interface and software components of the system. The following are some software requirements.

•Operating System: 64-bit Microsoft® Windows® 10 or more.

•Language: Python.

•Framework: Keras, Tensorflow

•Tools: Google Collaborator, Kaggle

What is Malware ?

Malware, which refers to any software intentionally designed to cause disruption or harm to computer systems, has become a significant threat to individuals and businesses on the Internet. It can take various forms, including computer viruses, worms, Trojan horses, ransomware, spyware, adware, rogue software, wipers, and keyloggers, among others. These types of malware can have serious consequences, such as gaining unauthorized access to information or systems, leaking private information, interfering with computer security and privacy, and depriving access to information.

The proliferation of malware is a growing concern, as evidenced by the increasing number of malware variants. According to Symantec's 2018 Internet Security Threat Report (ISTR), the number of malware variants reached 669,947,865 in 2017, which is twice as many as in 2016. Cybercrime, including malware attacks and other computer-related crimes, has been predicted to cost the world economy $6 trillion USD in 2021, with a yearly increase rate of 15%. Moreover, there has been a worrisome trend of malware being designed to specifically target critical infrastructure systems, such as the electricity distribution network, posing significant risks to the operation of essential services.

Defense strategies against malware vary depending on the type of malware, but commonly involve installing antivirus software, firewalls, and applying regular patches to reduce the risk of zero-day attacks. Securing networks from intrusion, implementing regular backups, and isolating infected systems are also important measures. However, despite these efforts, malware creators are continuously evolving their techniques to evade detection by antivirus software and other security measures. This highlights the need for ongoing research, development, and implementation of advanced defense strategies to combat the ever-evolving threat of malware.

In conclusion, malware presents serious challenges to individuals and businesses, and its impact on the global economy is substantial. The increasing number of malware variants and their potential to target critical infrastructure systems emphasize the need for robust defense strategies and continuous advancements in cybersecurity measures. It is crucial for organizations and individuals to stay vigilant, implement best practices, and stay updated with the latest developments in the field to effectively protect against the constantly evolving threat of malware.

The Purpose of Malware:

Since the rise of widespread broadband Internet access, malicious software has more frequently been designed for profit. Since 2003, the majority of widespread viruses and worms have been designed to take control of users' computers for illicit purposes. Infected "zombie computers" can be used to send email spam, to host contraband data such as child pornography, or to engage in distributed denial-of-service attacks as a form of extortion. Malware is used broadly against government or corporate websites to gather sensitive information, or to disrupt their operation in general. Further, malware can be used against individuals to gain information such as personal identification numbers or details, bank or credit card numbers, and passwords.In addition to criminal money-making, malware can be used for sabotage, often for political motives.

Types of Software:

There are many possible ways of categorizing malware and some malicious software may overlap into two or more categories. Broadly, software can categorised into three types:[33] (i) goodware; (ii) greyware and (iii) malware.

i) goodware : Obtained from trustworthy sources

ii) greyware : Insufficient consensus and/or metrics

iii) malware : Broad consensus among antivirus software that program is malicious or obtained from flagged sources.

Types of Malware:

1) Virus

2) Worm

3) Rootkits

4) Back doors

5) Trojan House

6) Droppers

7) Ransomware

8) Grayware

9) Potentially Unwanted Program (PUP)

10) Adware

11) Spyware

Detection of Malware:

Antivirus software typically uses two techniques to detect malware: (i) static analysis and (ii) dynamic analysis. Static analysis involves studying the software code of a potentially malicious program and producing a signature of that program. This information is then used to compare scanned files by an antivirus program. Because this approach is not useful for malware that has not yet been studied, antivirus software can use dynamic analysis to monitor how the program runs on a computer and block it if it performs unexpected activity.The aim of any malware is to conceal itself from detection by users or antivirus software. Detecting potential malware is difficult for two reasons. The first is that it is difficult to determine if software is malicious. The second is that malware uses technical measures to make it more difficult to detect it. An estimated 33% of malware is not detected by antivirus software.

Risks that bring in Malware:

1) Vulnerable Software

2) Excessive Privileges

3) Weak Passwords

4) Use of the Same Operating System

Mitigation of Malware:

1) Anti-malicious Software
    - Real Time Protection
    - Removal
    - Sandboxing
2) Website Security Scans
3) Network Segregation
4) “Air gap” Isolation or “Parallel Network”.

Kaggle Dataset : Malimg

This dataset includes byteplot images of 25 malware families.
They’re as follows.

1. Adialer.C :
2. Agent.FYI
3. Allaple.A
4. Allaple.L
5. Alueron.gen!J
6. Autorun.K
7. C2LOP.gen!g
8. C2LOP.P
9. DialPlatform.B
10. Dontovo.A
11. Fakerean
12. Instantaccess
13. Lolyda.AA1
14. Lolyda.AA2
15. Lolyda.AA3
16. Lolyda.AT

17. Malex.gen!J
18. Obfuscator.AD
19. Rbotigen
20. Skintrim.N
21. Swizzor.gen!E
22. Swizzor.gen!I
23. VB.AT
24. Wintrim.BX
25. Yuner.A

Conversion of PE to Greyscale images

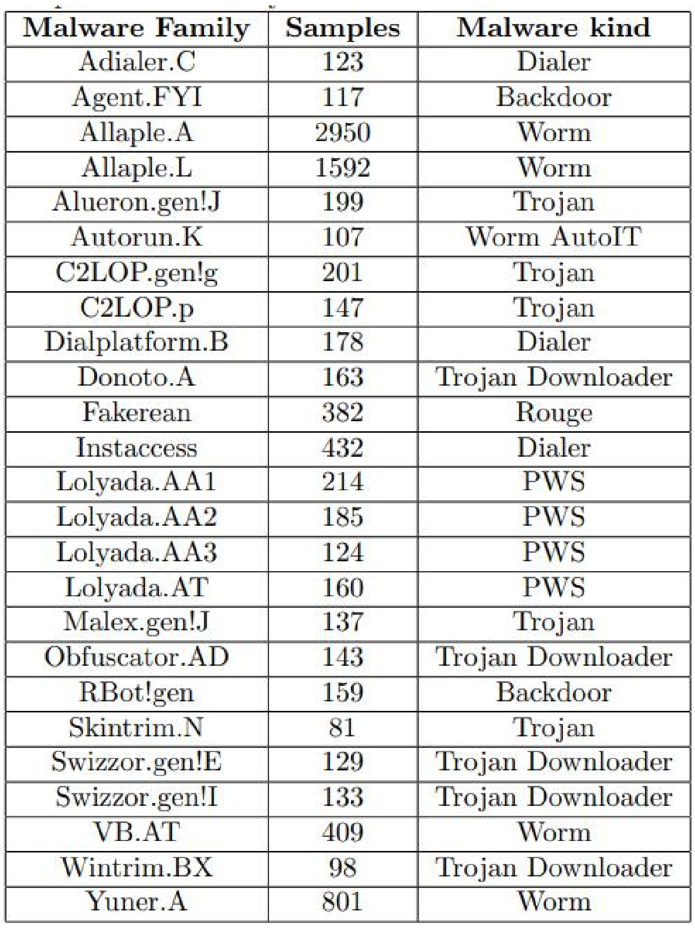
A given malware binary is read as a vector of 8 bit unsigned integers and then organized into a 2D array. This can be visualized as a gray scale image in the range [0,255] (0: black, 255: white). The width of the image is fixed and the height is allowed to vary depending on the file size. The malware binaries we use are in Portable Executable(PE) form. Generally, PE files are programs that have filename extensions such as .bin, .dll and .exe.

PE files areusually recognized through their components, which arecalled .tex, .rdata, .data and .rsrc. The first component,called .text, is the code section, containing the program'sinstructions. .rdata is the part that contains read onlydata, and .data is the part that contains data that can be modified, and .rsrc is the final component that stands forresources used by the malware.Malicious data binaries can be converted 8 bits at atime to pixels in a grayscale image, consisting of textural patterns. Based on these patterns, we can classify malware.

Researchers and practitioners can understand malwarebetter by visualizing malware binaries as images since thepatterns within such images become clearly visible. Finding patterns within images can be performed well by deeplearning.

The most important patterns of features in the malware images can be used to identify the malware families also. Images for a specific malware family have similar patterns, allowing a deep learning model to recognize important patterns using automatic extraction of features. In particular, CNN models are good at classifying images because they can extract relevant features within an image by subsampling through convolutions, pooling and other computations.

In this case, CNNs look for the most relevant features within an image from a specific malware family for the purpose of classification. Malware binaries can be translated into images using an algorithm that converts a binary PE file into a sequence of 8 bit vectors or hexadecimal values. An 8 bit vector can be represented in the range 00000000 (0) to 11111111 (255). Each 8 bit vector represents a number, and can be converted into pixel in a malware image.



Malware Family	Samples	Malware kind
Adialer.C	123	Dialer
Agent.FYI	117	Backdoor
Allaple.A	2950	Worm
Allaple.L	1592	Worm
Alueron.gen!J	199	Trojan
Autorun.K	107	Worm AutoIT
C2LOP.gen!g	201	Trojan
C2LOP.p	147	Trojan
Dialplatform.B	178	Dialer
Donoto.A	163	Trojan Downloader
Fakerean	382	Rouge
Instaccess	432	Dialer
Lolyada.AA1	214	PWS
Lolyada.AA2	185	PWS
Lolyada.AA3	124	PWS
Lolyada.AT	160	PWS
Malex.gen!J	137	Trojan
Obfuscator.AD	143	Trojan Downloader
RBot!gen	159	Backdoor
Skintrim.N	81	Trojan
Swizzor.gen!E	129	Trojan Downloader
Swizzor.gen!I	133	Trojan Downloader
VB.AT	409	Worm
Wintrim.BX	98	Trojan Downloader
Yuner.A	801	Worm

Table 2.1 Malware Family is related to wMalware Kind.

3. ARCHITECTURE

3.ARCHITECTURE

3.1 PROJECT ARCHITECTURE

This project architecture shows how the program works using the dateset provided and the algorithm
used.

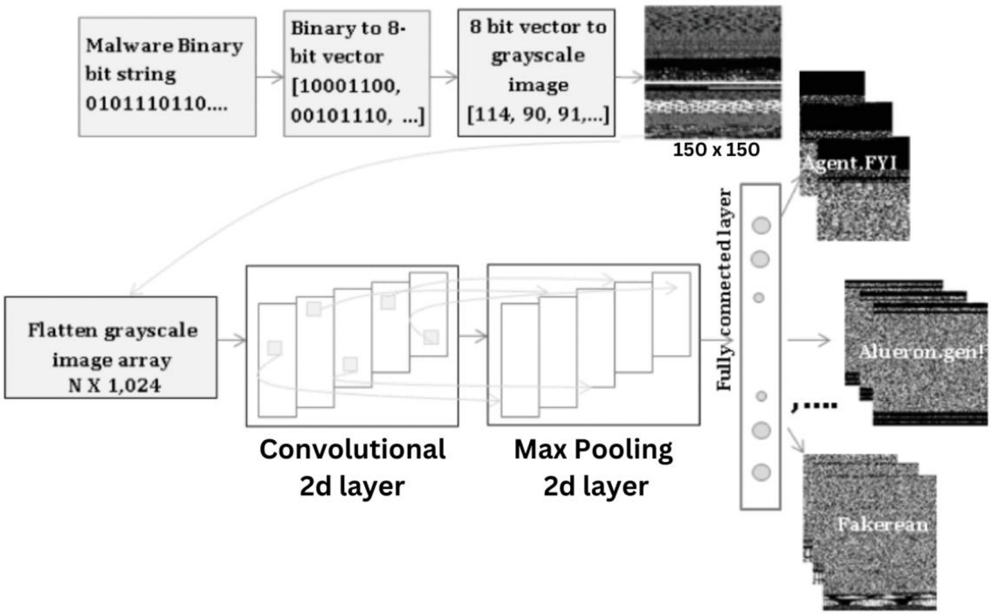

Malware Binary bit string 0101110110....
Binary to 8-bit vector [10001100, 00101110, ...]
8 bit vector to grayscale image [114, 90, 91,...]
150 x 150
Agent.FYI
Fully connected layer
Flatten grayscale image array N X 1,024
Convolutional 2d layer
Max Pooling 2d layer
Alueron.gen!
,....
Fakerean

Figure 3.1: Project Architecture of Malware Detection using Deep Learning with Image Processing.

3.2 DESCRIPTION

The project titled "Malware Detection Using Deep Learning with Image Classification" is centered around using deep learning techniques, specifically convolutional neural networks (CNNs), to detect malware. The goal of the project is to categorize malware into different families using visualization techniques with deep learning.

The project starts by converting a file, which is suspected to be malware, into a grayscale image. This image serves as the input to the CNN model. Grayscale images have only one channel (gray intensity), which simplifies the input data for the neural network.

The CNN model is trained on a dataset of labeled malware samples. The training process involves feeding the grayscale images into the CNN and adjusting the model's weights and biases to learn the patterns and features that distinguish malware from benign files. The CNN learns to extract relevant features from the images, such as pixel intensities, textures, and shapes, to make accurate predictions.

Once the CNN model is trained, it can be used for malware detection. The grayscale image of a suspected file is fed into the trained CNN, and the model predicts whether the file is malware or not based on the patterns it has learned during training.

The project also explores two different approaches for detecting malware: "when rescaled" and "when not rescaled." The "when rescaled" approach likely refers to rescaling the grayscale image to a different size before feeding it into the CNN, while the "when not rescaled" approach may involve using the original grayscale image without any resizing. This suggests that the project may investigate how the size of the input image affects the accuracy of the malware detection model.

In summary, the project utilizes deep learning techniques, particularly CNNs, to detect malware by converting files into grayscale images and using visualization techniques for malware family categorization. It explores different approaches for image preprocessing and detection, and aims to provide an effective solution for malware detection using deep learning.

3.3 USE CASE DIAGRAM

In the use case diagram, we basically have actors who are users of this application.

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has. The users here are stick figures.

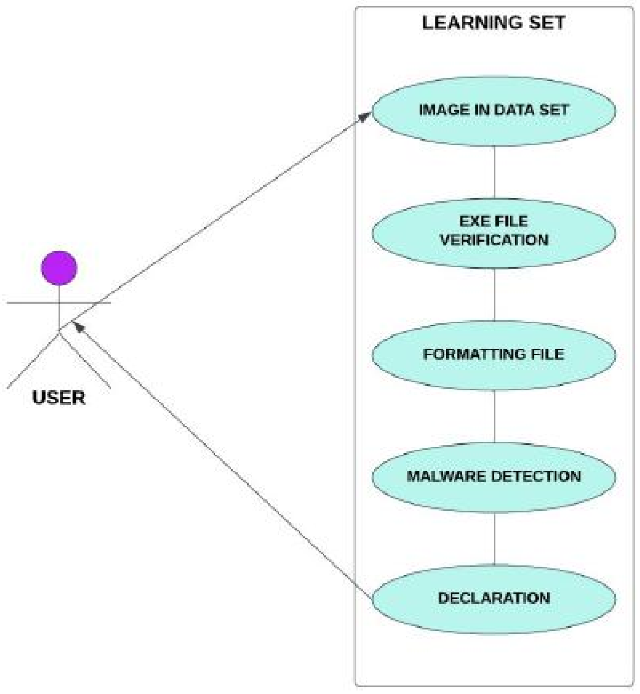


Figure 3.2: Use Case Diagram of Malware Detection using Deep Learning with Image Processing.

3.4 SEQUENCE DIAGRAM

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development.

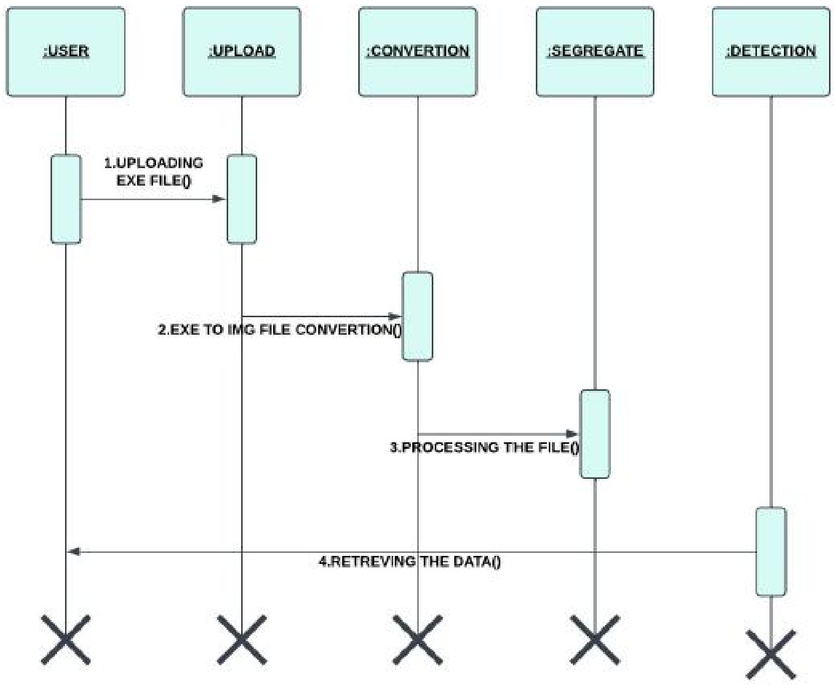


Figure 3.3: Sequence Diagram of Malware Detection using Deep Learning with Image Processing.

3.5 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. They can also include elements showing the flow of data between activities through one or more data stores.

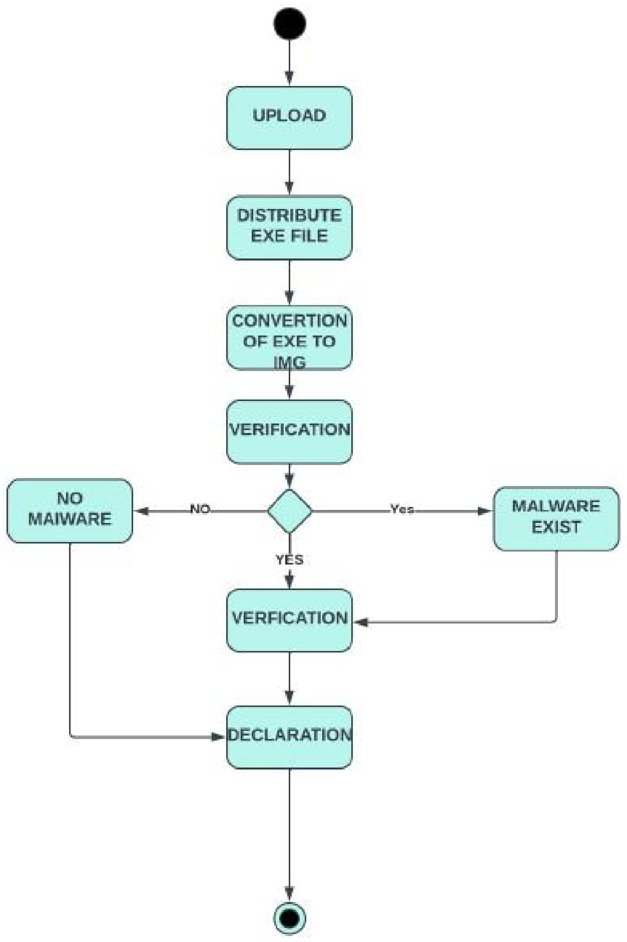


Figure 3.4: Activity Diagram of Malware Detection using Deep Learning with Image Processing.

4. IMPLEMENTATION

4.IMPLEMENTATION

4.1 SAMPLE CODE

```
!pip install -q kaggle
from google.colab import files
files.upload()
#A json file should be made which contains credentials of kaggle account
!mkdir ~/.kaggle
! cp kaggle.json  ~/.kaggle
! chmod 600 ~/.kaggle/kaggle.json

#dataset downloading and unzipping

!kaggle datasets download -d vivekanandabharupati/malimg
!unzip malimg.zip

#THE PROGRAM

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import keras
from keras.models import Sequential, Model
from keras.layers import Input, Dense, Dropout, Flatten , BatchNormalization
from keras.layers import Conv2D, MaxPooling2D , AveragePooling2D,GlobalAveragePooling2D
from keras import models, layers
from keras.layers import LSTM,TimeDistributed
from keras.callbacks import ReduceLROnPlateau,EarlyStopping
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
```

```
train="/content/archive/train"
val='/content/archive/validation'
batch_size=32
IMAGE_SIZE = [150, 150]

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    train,
    seed=123,
    shuffle=True,
    image_size=(150,150),
    batch_size=batch_size
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    val,
    seed=123,
    shuffle=True,
    image_size=(150,150),
    batch_size=batch_size
)
class_names=dataset.class_names

plt.figure(figsize=(10, 10))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")

def get_dataset_partitions_tf(ds, train_split=0.7, test_split=0.3, shuffle=True,
shuffle_size=10000):
    assert (train_split + test_split) == 1
    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    test_size = int(test_split * ds_size)

    train_ds = ds.take(train_size)
    test_ds = ds.skip(train_size).take(test_size)



    return train_ds, test_ds

train_ds, test_ds = get_dataset_partitions_tf(dataset)

train_ds                                                                =
train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds                                                                  =
val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds                                                                 =
test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)

def malware_model(width , height):
    Malware_model = Sequential()
    Malware_model.add(Conv2D(30,    kernel_size=(3,    3),    activation='relu',
input_shape=(width, height, 3)))
    Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    Malware_model.add(Conv2D(15, (3, 3), activation='relu'))
    Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    Malware_model.add(Flatten())
    Malware_model.add(Dropout(0.25))
    Malware_model.add(Dense(128, activation='relu'))
    Malware_model.add(Dropout(0.5))
    Malware_model.add(Dense(50, activation='relu'))
    Malware_model.add(Dense(25, activation='softmax'))

Malware_model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(
from_logits=False), optimizer = 'adam', metrics=['accuracy'])
    return Malware_model

model=malware_model(150 , 150)

model_fit = model.fit(train_ds, epochs=100 ,batch_size =
batch_size ,validation_data=val_ds, verbose =1)

# plot the loss
plt.plot(model_fit.history['loss'], label='train loss')
plt.plot(model_fit.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss.jpg',format='jpg')
plt.close()

# plot the accuracy
plt.plot(model_fit.history['accuracy'], label='train acc')
plt.plot(model_fit.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc.jpg',format="jpg")
plt.close()

model.evaluate(test_ds)

model.save("./ProjMaj_model.h5")

from tensorflow.keras.preprocessing import image
```

```
img_path="/content/archive/validation/Instantaccess/05b2e918c436eb86f8661
55e024964ed.png"
img = image.load_img(img_path, target_size=(150, 150))
img_tensor = image.img_to_array(img)                    # (height, width, channels)
img_tensor = np.expand_dims(img_tensor, axis=0)         # (1, height, width,
channels), add a dimension because the model expects this shape: (batch_size,
height, width, channels)
                                # imshow expects values in the range [0, 1]

pred=model.predict(img_tensor)
print(class_names[np.argmax(pred)])

#WITH SHAPING MODEL

resize_and_rescale = tf.keras.Sequential([
  tf.keras.layers.experimental.preprocessing.Resizing(150, 150),
  tf.keras.layers.experimental.preprocessing.Rescaling(1./255)
])

def malware_model(width , height):
    Malware_model = Sequential()
    Malware_model.add(resize_and_rescale)
    Malware_model.add(Conv2D(30, kernel_size=(3, 3), activation='relu',
input_shape=(width, height, 3)))
    Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    Malware_model.add(Conv2D(15, (3, 3), activation='relu'))
    Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    Malware_model.add(Flatten())
    Malware_model.add(Dropout(0.25))
    Malware_model.add(Dense(128, activation='relu'))
    Malware_model.add(Dropout(0.5))
    Malware_model.add(Dense(50, activation='relu'))
    Malware_model.add(Dense(25, activation='softmax'))
Malware_model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(
from_logits=False), optimizer = 'adam', metrics=['accuracy'])
   return Malware_model

EPOCH = 100
lr_reduction = ReduceLROnPlateau(monitor='val_accuracy',patience=4,
verbose=1, factor=0.4, min_lr=0.0001)
early_stop = EarlyStopping(monitor='val_accuracy', min_delta=0.00001,
patience=8, mode='auto', restore_best_weights=True)

model=malware_model(150 , 150)
model_fit = model.fit(train_ds, epochs=EPOCH ,batch_size =
batch_size ,validation_data=val_ds, verbose
=1,callbacks=[early_stop,lr_reduction])

model.save("./Best_Model.h5")

model.summary()
model.evaluate(test_ds)

# plot the loss
plt.plot(model_fit.history['loss'], label='train loss')
plt.plot(model_fit.history['val_loss'], label='val loss')
plt.legend()
plt.show()

plt.close()
# plot the accuracy
plt.plot(model_fit.history['accuracy'], label='train acc')
plt.plot(model_fit.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.close()

for images_batch, labels_batch in test_ds.take(1):

    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction = model.predict(images_batch)
    print("predicted label:",class_names[np.argmax(batch_prediction[0])])

from tensorflow.keras.preprocessing import image
img_path="/content/00d1e43e3179ddce3b489b3ce5fa65cb.png"
img = image.load_img(img_path, target_size=(150, 150))
img_tensor = image.img_to_array(img)                    # (height, width, channels)
img_tensor = np.expand_dims(img_tensor, axis=0)         # (1, height, width,
channels), add a dimension because the model expects this shape: (batch_size,
height, width, channels)
                                # imshow expects values in the range [0, 1]

pred=model.predict(img_tensor)
print(class_names[np.argmax(pred)])
```

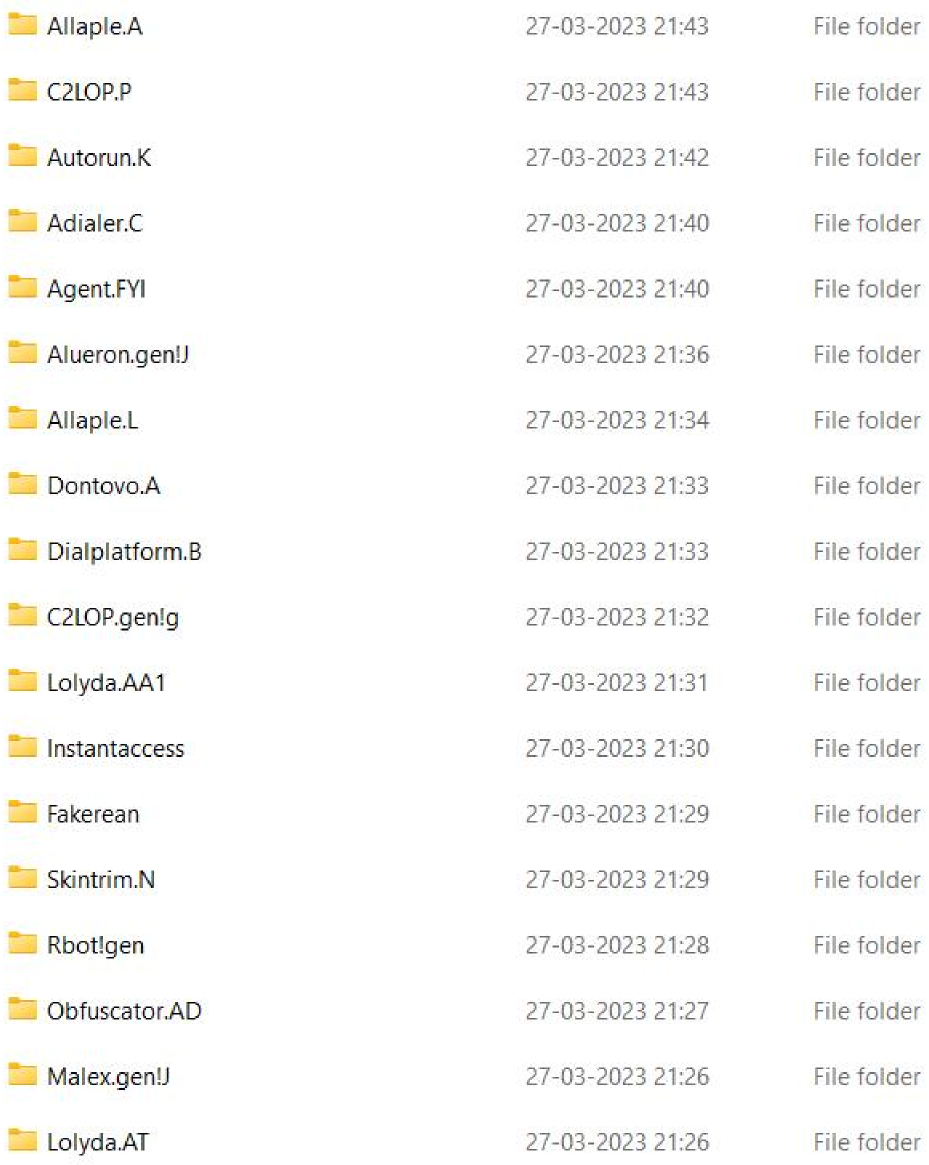
CODE EXPLANATION

Lines 1-12	As this code is compiled on Google Collaborator, few directories are made to store the data, the datasets are imported from Kaggle. The datasets are custom made and altered from the original. The actual Malimg Dataset contains 9,339 malware byteplot images from 25 different families. Then they are downloaded and as the dataset is compressed to a zip, it is unzipped.
Lines 16-29	Here are the statements to import all the libraries, it includes open source libraries like keras, tensorflow. NumPy to add support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Pandas for data manipulation and analysis. Matplotlib is a plotting library. Keras is a interface for Artificial Neural Networks. Tensorflow is used for Machine Learning and Artificial Intelligence. Sequential model is imported, CNN layers like Input, Dropout, Dense, Flatten are imported, additional to these Conv2D, Pooling layers are included.
Lines 31- 51	The dataset directories are allotted. Then image size is set to 150 x 150 and the batch size is given as 32. The shuffle command is included to shuffle the images in the directories.
Lines 53 - 59	Types of families are plotted in these lines.
Lines 61 -82	The files in the dataset are splitted, shuffled and the partitions are made such that 70% is the training dataset, 30% is the testing dataset.
Lines 84 - 101	Machine is defined here, the machine is a Sequential Model, it is arranged according to the architecture with various layers like Conv2D, MaxPooling, Flatten, Dense, Dropout.
Lines 103 -121	Accuracy and Loss are plotted. Then the model is evaluated and is saved.
Lines 123 - 131	This is the last part of the program, where a image from outside the system is taken, tested and later determined whether it is classified accurately or not. If it is accurate, it is passed and if not, it is not.

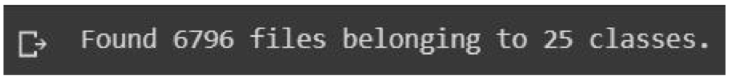
5. RESULTS

5. RESULTS

TRAINING DATASET



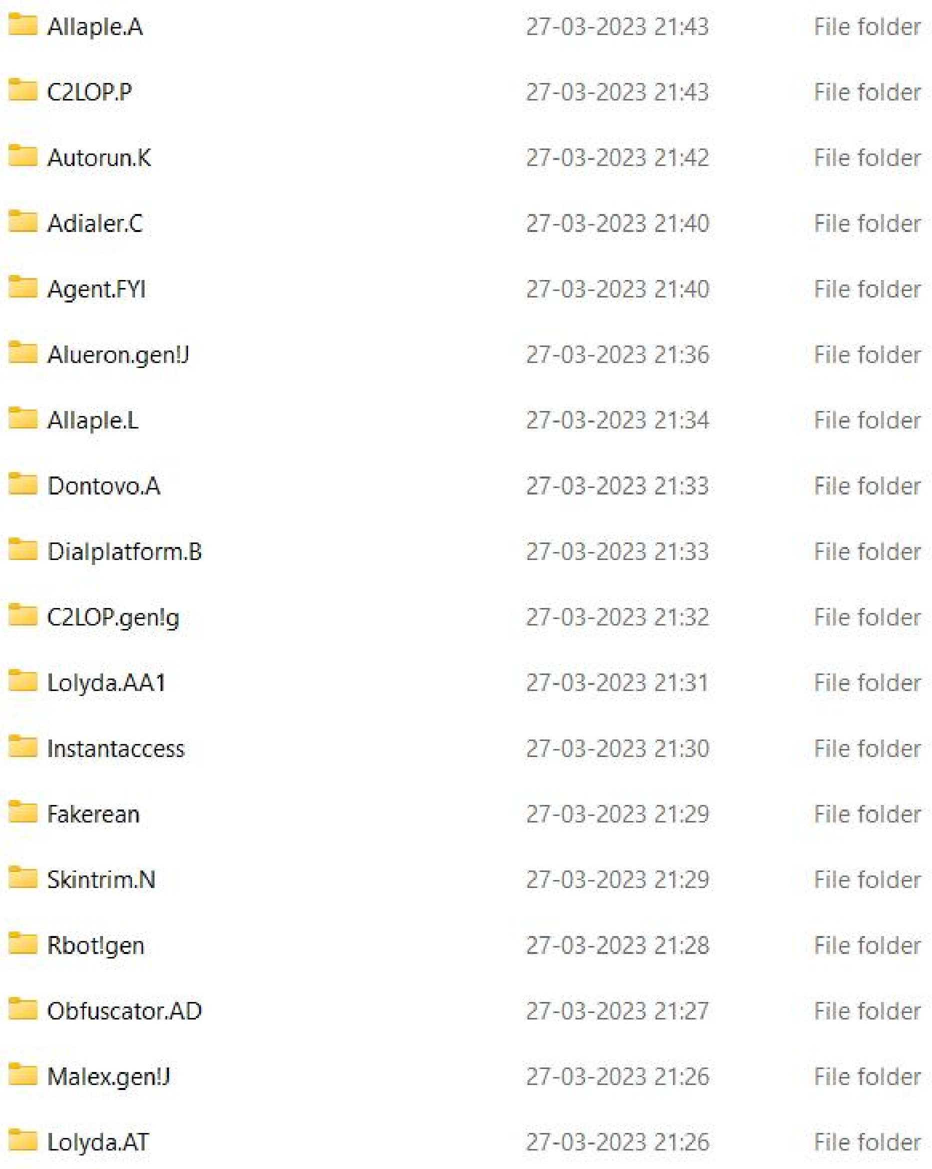
Allaple.A	27-03-2023 21:43	File folder
C2LOP.P	27-03-2023 21:43	File folder
Autorun.K	27-03-2023 21:42	File folder
Adialer.C	27-03-2023 21:40	File folder
Agent.FYI	27-03-2023 21:40	File folder
Alueron.gen!J	27-03-2023 21:36	File folder
Allaple.L	27-03-2023 21:34	File folder
Dontovo.A	27-03-2023 21:33	File folder
Dialplatform.B	27-03-2023 21:33	File folder
C2LOP.gen!g	27-03-2023 21:32	File folder
Lolyda.AA1	27-03-2023 21:31	File folder
Instantaccess	27-03-2023 21:30	File folder
Fakerean	27-03-2023 21:29	File folder
Skintrim.N	27-03-2023 21:29	File folder
Rbot!gen	27-03-2023 21:28	File folder
Obfuscator.AD	27-03-2023 21:27	File folder
Malex.gen!J	27-03-2023 21:26	File folder
Lolyda.AT	27-03-2023 21:26	File folder



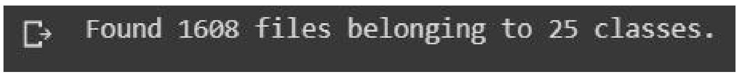
```
Found 6796 files belonging to 25 classes.
```

Screenshot 5.1 Training Data set

VALIDATION DATASET



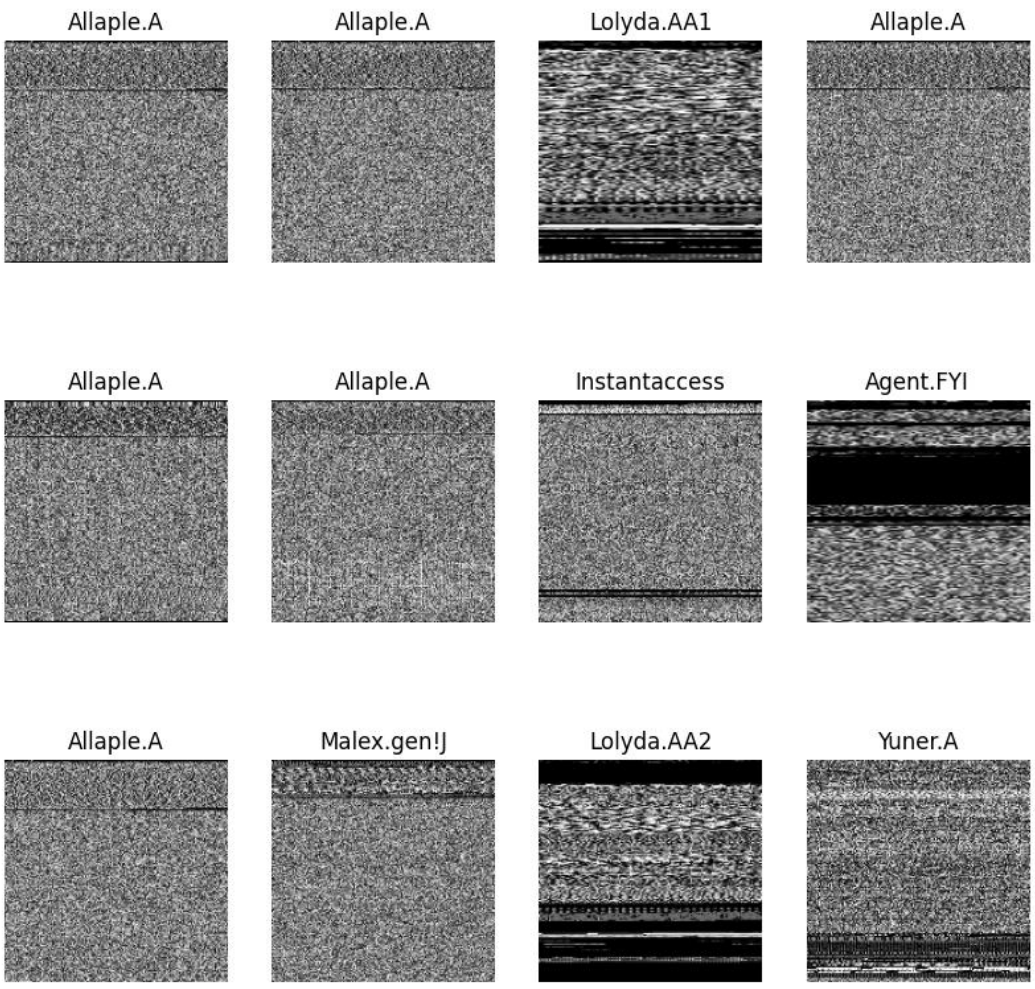
Allaple.A	27-03-2023 21:43	File folder
C2LOP.P	27-03-2023 21:43	File folder
Autorun.K	27-03-2023 21:42	File folder
Adialer.C	27-03-2023 21:40	File folder
Agent.FYI	27-03-2023 21:40	File folder
Alueron.gen!J	27-03-2023 21:36	File folder
Allaple.L	27-03-2023 21:34	File folder
Dontovo.A	27-03-2023 21:33	File folder
Dialplatform.B	27-03-2023 21:33	File folder
C2LOP.gen!g	27-03-2023 21:32	File folder
Lolyda.AA1	27-03-2023 21:31	File folder
Instantaccess	27-03-2023 21:30	File folder
Fakerean	27-03-2023 21:29	File folder
Skintrim.N	27-03-2023 21:29	File folder
Rbot!gen	27-03-2023 21:28	File folder
Obfuscator.AD	27-03-2023 21:27	File folder
Malex.gen!J	27-03-2023 21:26	File folder
Lolyda.AT	27-03-2023 21:26	File folder



```
Found 1608 files belonging to 25 classes.
```

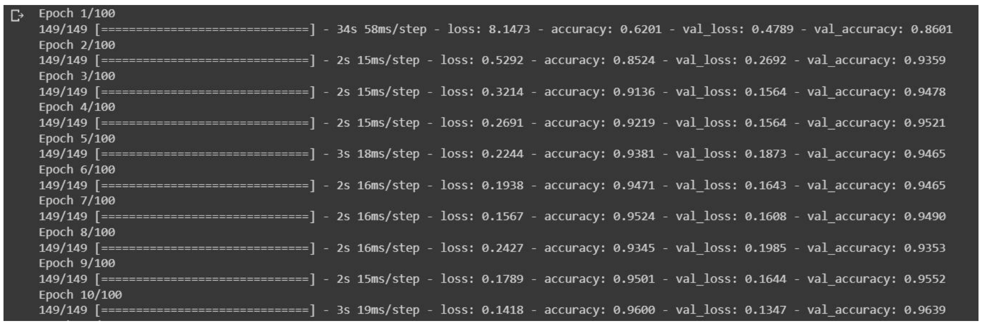
Screenshot 5.2 Validation Data Set

TYPES OF MALWARES AS PNG IMAGES



Screenshot 5.3 : Types of Malwares as Png Images.

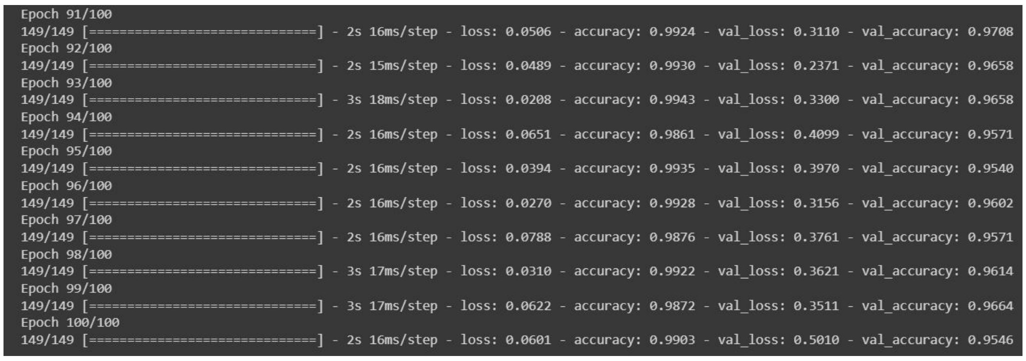
INITIAL EPOCHS FOR MACHINE WHEN NOT RESCALED



```
Epoch 1/100
149/149 [==============================] - 34s 58ms/step - loss: 8.1473 - accuracy: 0.6201 - val_loss: 0.4789 - val_accuracy: 0.8601
Epoch 2/100
149/149 [==============================] - 2s 15ms/step - loss: 0.5292 - accuracy: 0.8524 - val_loss: 0.2692 - val_accuracy: 0.9359
Epoch 3/100
149/149 [==============================] - 2s 15ms/step - loss: 0.3214 - accuracy: 0.9136 - val_loss: 0.1564 - val_accuracy: 0.9478
Epoch 4/100
149/149 [==============================] - 2s 15ms/step - loss: 0.2691 - accuracy: 0.9219 - val_loss: 0.1564 - val_accuracy: 0.9521
Epoch 5/100
149/149 [==============================] - 3s 18ms/step - loss: 0.2244 - accuracy: 0.9381 - val_loss: 0.1873 - val_accuracy: 0.9465
Epoch 6/100
149/149 [==============================] - 2s 16ms/step - loss: 0.1938 - accuracy: 0.9471 - val_loss: 0.1643 - val_accuracy: 0.9465
Epoch 7/100
149/149 [==============================] - 2s 16ms/step - loss: 0.1567 - accuracy: 0.9524 - val_loss: 0.1608 - val_accuracy: 0.9490
Epoch 8/100
149/149 [==============================] - 2s 16ms/step - loss: 0.2427 - accuracy: 0.9345 - val_loss: 0.1985 - val_accuracy: 0.9353
Epoch 9/100
149/149 [==============================] - 2s 15ms/step - loss: 0.1789 - accuracy: 0.9501 - val_loss: 0.1644 - val_accuracy: 0.9552
Epoch 10/100
149/149 [==============================] - 3s 19ms/step - loss: 0.1418 - accuracy: 0.9600 - val_loss: 0.1347 - val_accuracy: 0.9639
```

Screenshot 5.4 Initial Epochs for Machine when not re-scaled

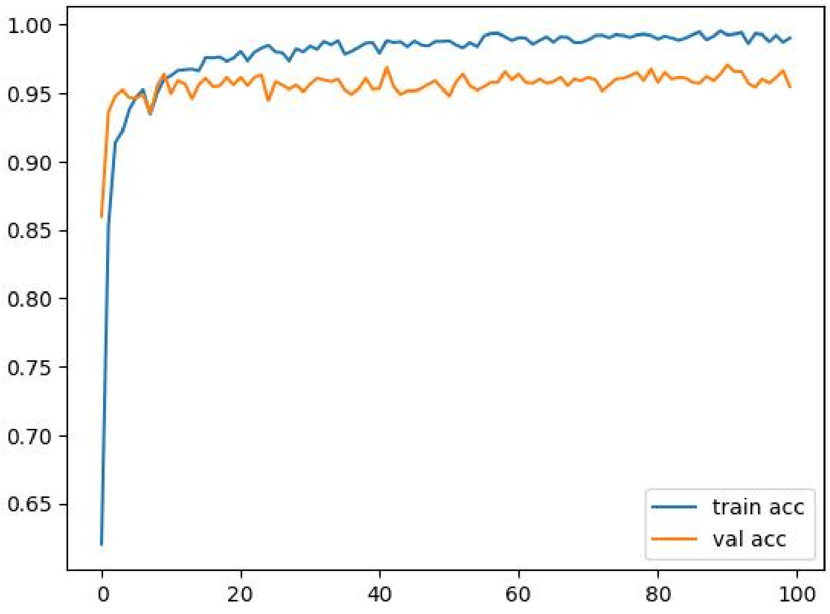
FINAL EPOCHS FOR MACHINE

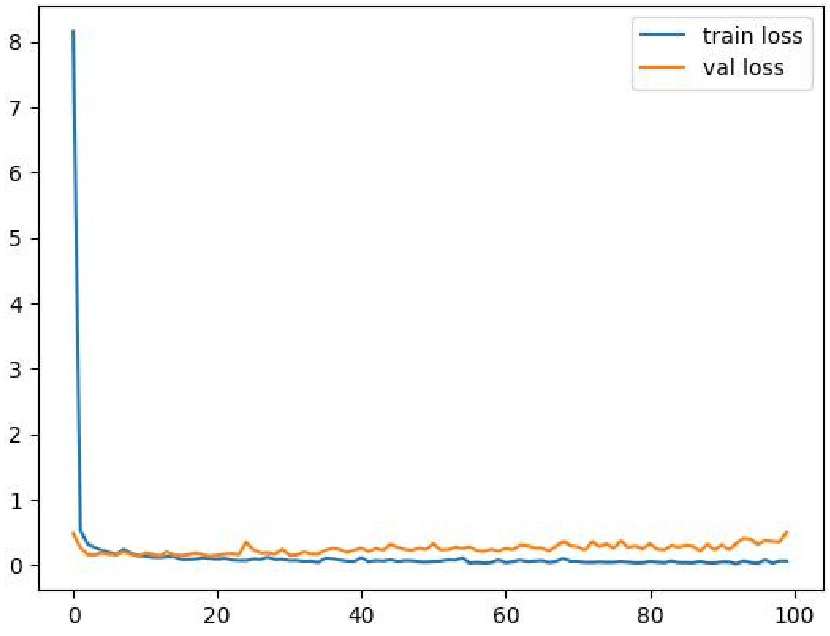


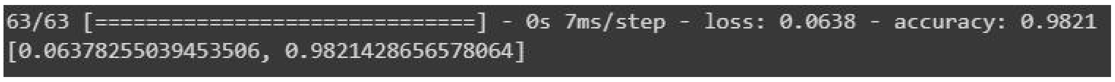
```
Epoch 91/100
149/149 [==============================] - 2s 16ms/step - loss: 0.0506 - accuracy: 0.9924 - val_loss: 0.3110 - val_accuracy: 0.9708
Epoch 92/100
149/149 [==============================] - 2s 15ms/step - loss: 0.0489 - accuracy: 0.9930 - val_loss: 0.2371 - val_accuracy: 0.9658
Epoch 93/100
149/149 [==============================] - 3s 18ms/step - loss: 0.0208 - accuracy: 0.9943 - val_loss: 0.3300 - val_accuracy: 0.9658
Epoch 94/100
149/149 [==============================] - 2s 16ms/step - loss: 0.0651 - accuracy: 0.9861 - val_loss: 0.4099 - val_accuracy: 0.9571
Epoch 95/100
149/149 [==============================] - 2s 16ms/step - loss: 0.0394 - accuracy: 0.9935 - val_loss: 0.3970 - val_accuracy: 0.9540
Epoch 96/100
149/149 [==============================] - 2s 16ms/step - loss: 0.0270 - accuracy: 0.9928 - val_loss: 0.3156 - val_accuracy: 0.9602
Epoch 97/100
149/149 [==============================] - 2s 16ms/step - loss: 0.0788 - accuracy: 0.9876 - val_loss: 0.3761 - val_accuracy: 0.9571
Epoch 98/100
149/149 [==============================] - 3s 17ms/step - loss: 0.0310 - accuracy: 0.9922 - val_loss: 0.3621 - val_accuracy: 0.9614
Epoch 99/100
149/149 [==============================] - 3s 17ms/step - loss: 0.0622 - accuracy: 0.9872 - val_loss: 0.3511 - val_accuracy: 0.9664
Epoch 100/100
149/149 [==============================] - 2s 16ms/step - loss: 0.0601 - accuracy: 0.9903 - val_loss: 0.5010 - val_accuracy: 0.9546
```

Screenshot 5.5 Final Epochs for Machine when not re-scaled

ACCURACY AND LOSS FOR THE GIVEN DATASET WHEN NOT RE-SCALED



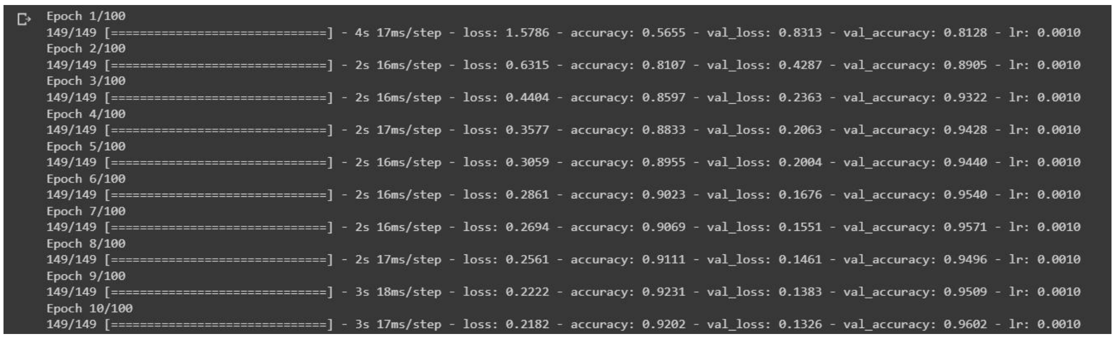




```
63/63 [==============================] - 0s 7ms/step - loss: 0.0638 - accuracy: 0.9821
[0.06378255039453506, 0.9821428656578064]
```

Screenshot 5.6 Accuracy and Loss for the Given Dataset when not re-scaled

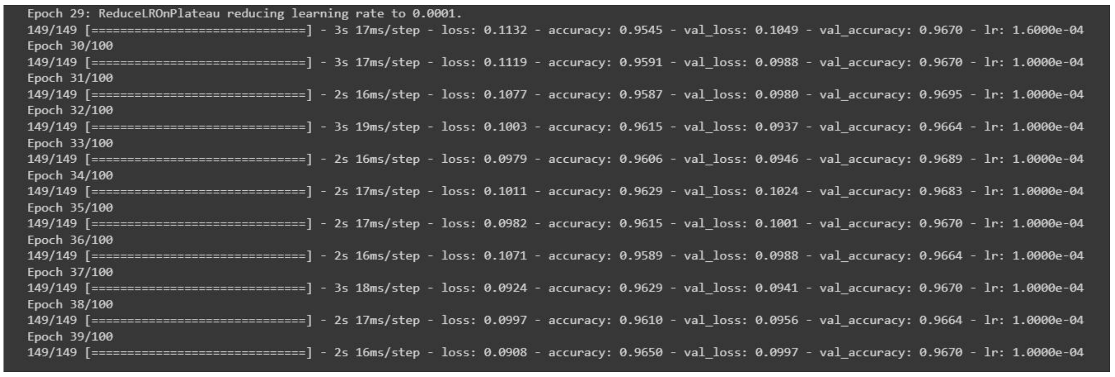
INITIAL EPOCHS FOR MACHINE WHEN RESCALED



```
Epoch 1/100
149/149 [==============================] - 4s 17ms/step - loss: 1.5786 - accuracy: 0.5655 - val_loss: 0.8313 - val_accuracy: 0.8128 - lr: 0.0010
Epoch 2/100
149/149 [==============================] - 2s 16ms/step - loss: 0.6315 - accuracy: 0.8107 - val_loss: 0.4287 - val_accuracy: 0.8905 - lr: 0.0010
Epoch 3/100
149/149 [==============================] - 2s 16ms/step - loss: 0.4404 - accuracy: 0.8597 - val_loss: 0.2363 - val_accuracy: 0.9322 - lr: 0.0010
Epoch 4/100
149/149 [==============================] - 2s 17ms/step - loss: 0.3577 - accuracy: 0.8833 - val_loss: 0.2063 - val_accuracy: 0.9428 - lr: 0.0010
Epoch 5/100
149/149 [==============================] - 2s 16ms/step - loss: 0.3059 - accuracy: 0.8955 - val_loss: 0.2004 - val_accuracy: 0.9440 - lr: 0.0010
Epoch 6/100
149/149 [==============================] - 2s 16ms/step - loss: 0.2861 - accuracy: 0.9023 - val_loss: 0.1676 - val_accuracy: 0.9540 - lr: 0.0010
Epoch 7/100
149/149 [==============================] - 2s 16ms/step - loss: 0.2694 - accuracy: 0.9069 - val_loss: 0.1551 - val_accuracy: 0.9571 - lr: 0.0010
Epoch 8/100
149/149 [==============================] - 2s 17ms/step - loss: 0.2561 - accuracy: 0.9111 - val_loss: 0.1461 - val_accuracy: 0.9496 - lr: 0.0010
Epoch 9/100
149/149 [==============================] - 3s 18ms/step - loss: 0.2222 - accuracy: 0.9231 - val_loss: 0.1383 - val_accuracy: 0.9509 - lr: 0.0010
Epoch 10/100
149/149 [==============================] - 3s 17ms/step - loss: 0.2182 - accuracy: 0.9202 - val_loss: 0.1326 - val_accuracy: 0.9602 - lr: 0.0010
```

Screenshot 5.7 Initial Epochs for Machine when re-scaled

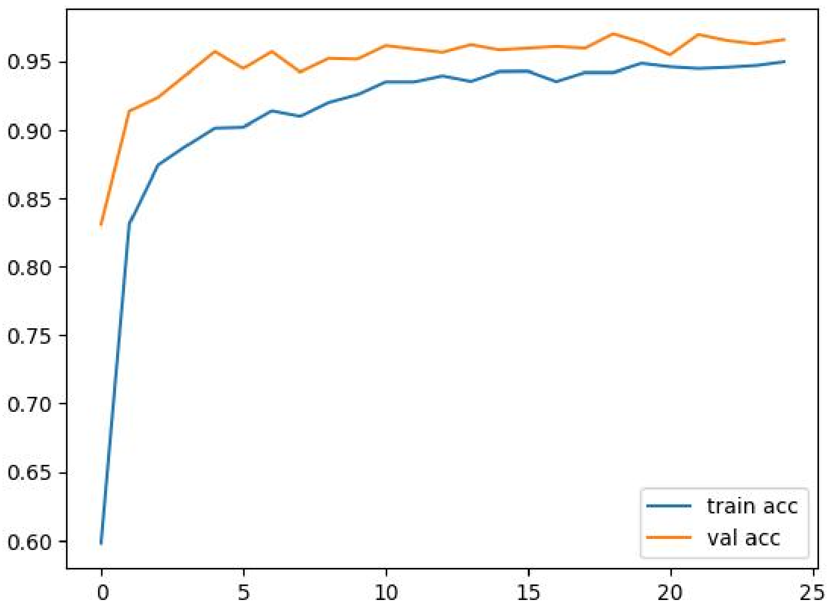
FINAL EPOCHS FOR MACHINE WHEN RESCALED

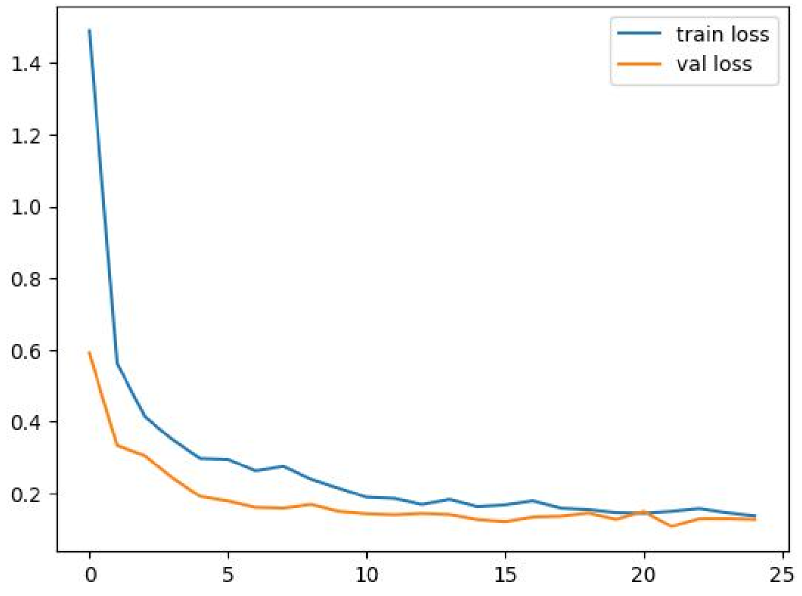


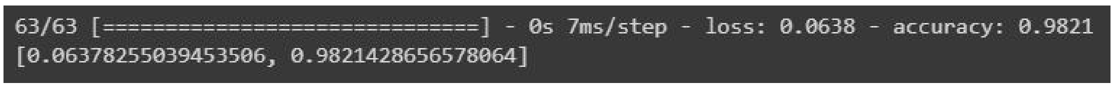
```
Epoch 29: ReduceLROnPlateau reducing learning rate to 0.0001.
149/149 [==============================] - 3s 17ms/step - loss: 0.1132 - accuracy: 0.9545 - val_loss: 0.1049 - val_accuracy: 0.9670 - lr: 1.6000e-04
Epoch 30/100
149/149 [==============================] - 3s 17ms/step - loss: 0.1119 - accuracy: 0.9591 - val_loss: 0.0988 - val_accuracy: 0.9670 - lr: 1.0000e-04
Epoch 31/100
149/149 [==============================] - 2s 16ms/step - loss: 0.1077 - accuracy: 0.9587 - val_loss: 0.0980 - val_accuracy: 0.9695 - lr: 1.0000e-04
Epoch 32/100
149/149 [==============================] - 3s 19ms/step - loss: 0.1003 - accuracy: 0.9615 - val_loss: 0.0937 - val_accuracy: 0.9664 - lr: 1.0000e-04
Epoch 33/100
149/149 [==============================] - 2s 16ms/step - loss: 0.0979 - accuracy: 0.9606 - val_loss: 0.0946 - val_accuracy: 0.9689 - lr: 1.0000e-04
Epoch 34/100
149/149 [==============================] - 2s 17ms/step - loss: 0.1011 - accuracy: 0.9629 - val_loss: 0.1024 - val_accuracy: 0.9683 - lr: 1.0000e-04
Epoch 35/100
149/149 [==============================] - 2s 17ms/step - loss: 0.0982 - accuracy: 0.9615 - val_loss: 0.1001 - val_accuracy: 0.9670 - lr: 1.0000e-04
Epoch 36/100
149/149 [==============================] - 2s 16ms/step - loss: 0.1071 - accuracy: 0.9589 - val_loss: 0.0988 - val_accuracy: 0.9664 - lr: 1.0000e-04
Epoch 37/100
149/149 [==============================] - 3s 18ms/step - loss: 0.0924 - accuracy: 0.9629 - val_loss: 0.0941 - val_accuracy: 0.9670 - lr: 1.0000e-04
Epoch 38/100
149/149 [==============================] - 2s 17ms/step - loss: 0.0997 - accuracy: 0.9610 - val_loss: 0.0956 - val_accuracy: 0.9664 - lr: 1.0000e-04
Epoch 39/100
149/149 [==============================] - 2s 16ms/step - loss: 0.0908 - accuracy: 0.9650 - val_loss: 0.0997 - val_accuracy: 0.9670 - lr: 1.0000e-04
```

Screenshot 5.8 Final Epochs for Machine when re-scaled

ACCURACY AND LOSS FOR THE GIVEN DATASET WHEN RE-SCALED



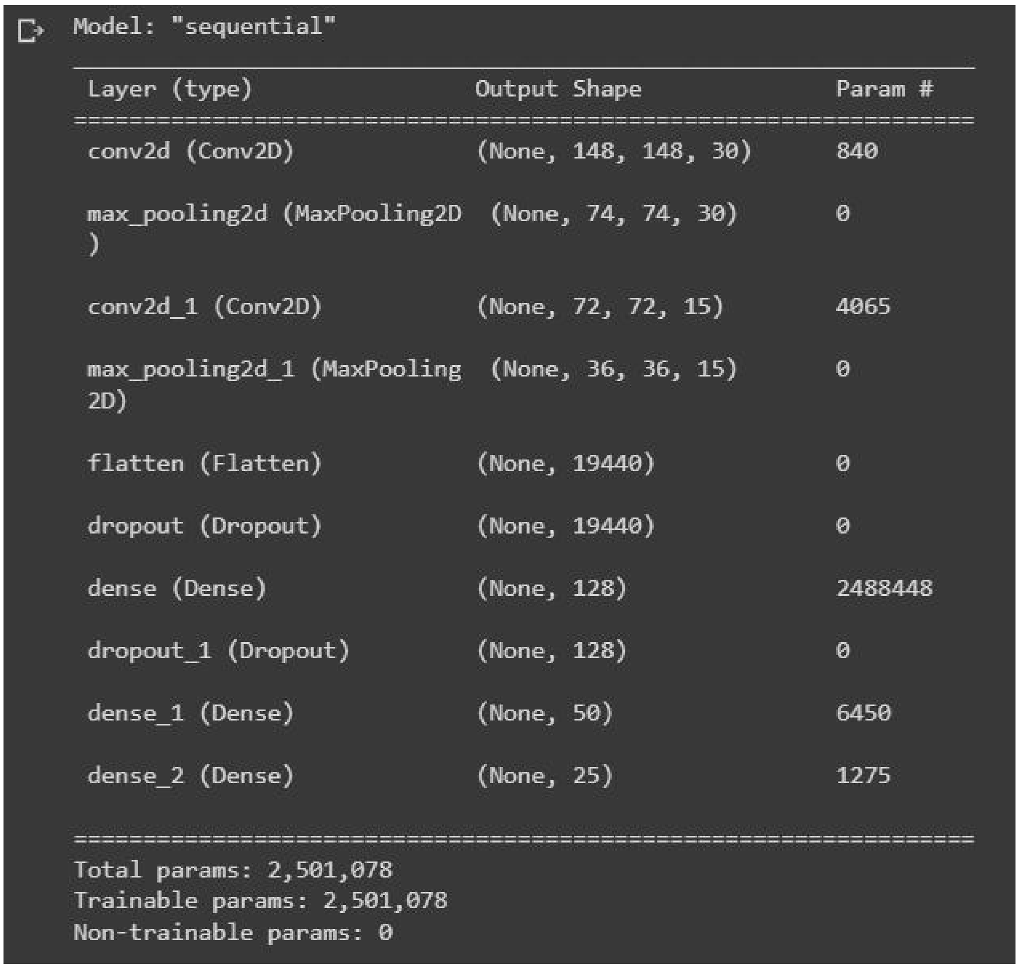




```
63/63 [==============================] - 0s 7ms/step - loss: 0.0638 - accuracy: 0.9821
[0.06378255039453506, 0.9821428656578064]
```

Screenshot 5.9 Accuracy and Loss for the Given Dataset when re-scaled

SUMMARY OF THE MACHINE



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 30)      840

 max_pooling2d (MaxPooling2D  (None, 74, 74, 30)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 15)        4065

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 15)       0
 2D)

 flatten (Flatten)           (None, 19440)             0

 dropout (Dropout)           (None, 19440)             0

 dense (Dense)               (None, 128)               2488448

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 50)                6450

 dense_2 (Dense)             (None, 25)                1275

=================================================================
Total params: 2,501,078
Trainable params: 2,501,078
Non-trainable params: 0
```

Screenshot 5.10 Summary of the Machine

6. TESTING

6. TESTING

6.1 INTRODUCTION TO TESTING

The purpose of testing in software development is to identify and rectify errors or faults in a work product, such as a software application or system. Testing is a crucial step in the software development life cycle (SDLC) as it helps to ensure that the software meets its intended functionality, requirements, and user expectations, and does not fail in unacceptable ways during its operation.Testing involves systematically checking the different components, sub-assemblies, assemblies, and/or the finished product to identify any potential faults or weaknesses. The goal is to exercise the software in various ways to uncover any defects or issues that may arise during normal or abnormal usage scenarios. By identifying and addressing these defects early in the development process, testing helps to minimize the risk of software failures, improve the quality and reliability of the software, and enhance overall user satisfaction.

6.2 TYPES OF TESTING:

6.2.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .It is done after the completion of an individual unit before integration. This is a structural testing that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

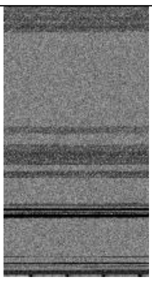
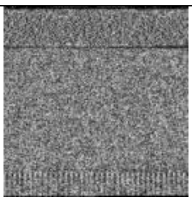
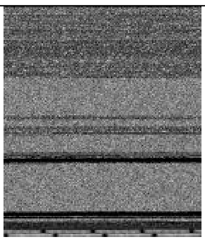
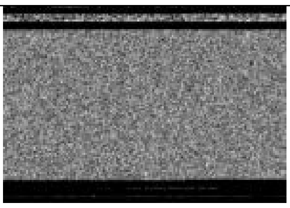
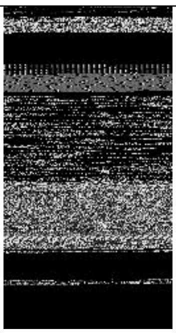
6.2.2 INTEGRATION TESTING

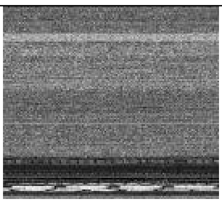
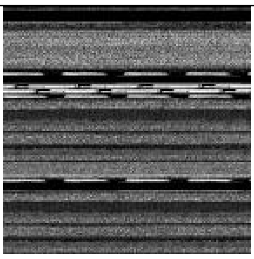
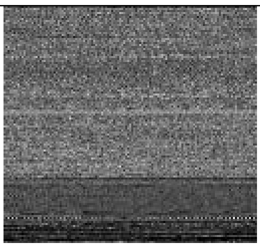
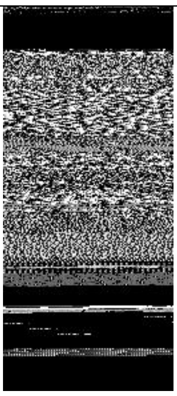
Integration tests are designed to test integrated software components to determine if they actually run as one program. Integration tests demonstrate that although the components were individually satisfactory, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

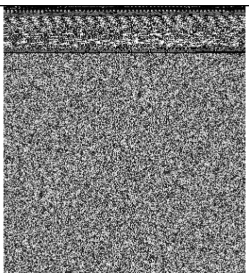
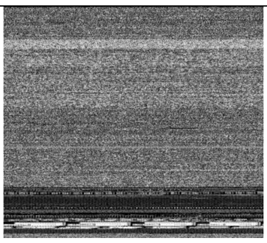
6.2.3 FUNCTIONAL TESTING

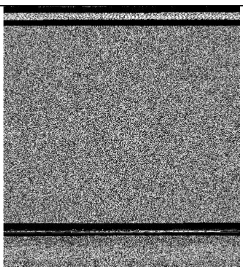
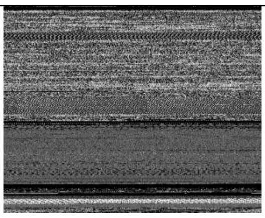
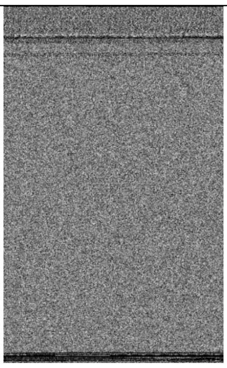
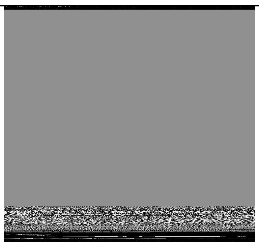
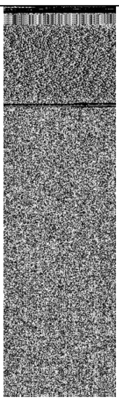
Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

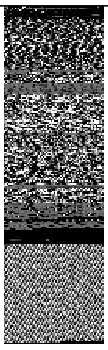
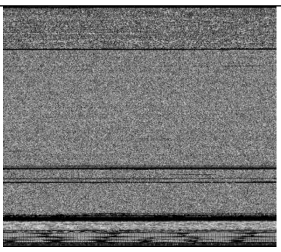
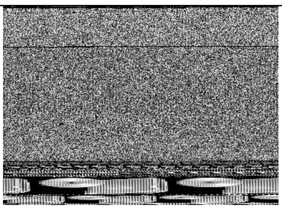
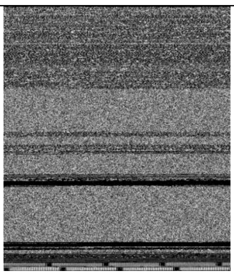
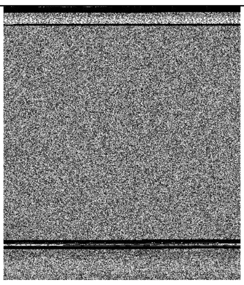
6.3 TEST CASES

S.No.	Image Uploaded	With shaping OUTPUT	Without Shaping OUTPUT	Result
1	 C2LOP.gen!g	C2LOP.gen!g	C2LOP.gen!g	Totally Passed.
2	 Allaple.A	Swizzor.gen!I	Allaple.A	Passed, if not reshaped.
3	 Swizzor.gen!I	Swizzor.gen!I	Swizzor.gen!I	Totally Passed
4	 Alueron.gen!J	Alueron.gen!J	Alueron.gen!J	Totally Passed
5	 Dontovo.A	Dontovo.A	Dontovo.A	Totally Passed

6	 Yuner.A	VB.AT	Yuner.A	Passed, if it is not reshaped.
7	Agent.FYI	Agent.FYI	Agent.FYI	Totally Passed
8	 VB.AT	VB.AT	VB.AT	Totally Passed
9	 Rbot!gen	Rbot!gen	Rbot!gen	Totally Passed
10	 Loylda.AA2	Loylda.AA2	Loylda.AA2	Totally Passed

11	 Malex.gen!J	VB.AT	Malex.gen!J	Passed, if it is not reshaped.
12	 Autorun.K	VB.AT	Autorun.K	Passed, if it is not reshaped.
13	Lolyda.AT	Lolyda.AT	Lolyda.AT	Totally Passed
14	Lolyda.AA1	VB.AT	Lolyda.AA1	Passed, if it is not reshaped.

15	 Obfuscator.AD	Obfuscator.AD	Obfuscator.AD	Totally Passed
16	 Adialer.C	Adialer.C	Adialer.C	Totally Passed
17	 Wintrim.BX	C2LOP.gen!g	Wintrim.BX	Passed, if it is not reshaped.
18	 Lolyda.AA1	VB.AT	Allaple.A	Totally Failed
19	 Allaple.L	Allaple.L	Allaple.L	Totally Passed

20	 Dialplatform.B	Dialplatform.B	Dialplatform.B	Totally Passed
21	 C2LOP.P	Swizzor.gen!E	Allaple.A	Totally Failed
22	 Fakerean	Fakerean	Fakerean	Totally Passed
23	 Swizzor.gen!E	Swizzor.gen!I	C2LOP.P	Totally Failed
24	 Instantaccess	Instantaccess	Instantaccess	Totally Passed

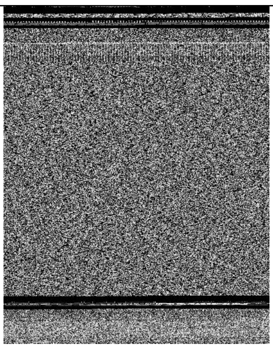
25	 Skintrim.N	VB.AT	Skintrim.N	Passed, if it is not reshaped.

Table 6.1 Test Cases

TEST CASES EXPLANATION

The testing here is done after a complete training of the machine for 100 epochs with the provided dataset. After the training the testing is done for about 10 cases and here is summary of them.

Test Case 01 :A byteplot(Grey Scale) image of **C2LOP.gen!g** is uploaded for the machine for the classification and the output received is **C2LOP.gen!g in both the cases**. Hence the test case is totally passed.

Test Case 02 :A byteplot(Grey Scale) image of **Allaple.A** is uploaded for the machine for the classification and the output received is **Allaple.A if it not reshaped and** the output received is **Swizzor.gen!I if reshaped.** Hence the test case is partially passed.

Test Case 03 :A byteplot(Grey Scale) image of **Swizzor.gen!I** is uploaded for the machine for the classification and the output received is **Swizzor.gen!I in both the cases**.. Hence the test case is totally passed.

Test Case 04 :A byteplot(Grey Scale) image of **Alueron.gen!J** is uploaded for the machine for the classification and the output received is **Alueron.gen!J in both the cases**. . Hence the test case is totally passed.

Test Case 05 :A byteplot(Grey Scale) image of **Dontovo.A** is uploaded for the machine for the classification and the output received is **Dontovo.A in both the cases**.. Hence the test case is totally passed.

Test Case 06 :A byteplot(Grey Scale) image of **Yuner.A** is uploaded for the machine for the classification and the output received is **Yuner.A if not reshaped and** the output received is **VB.AT**. Hence the test case is passed.

Test Case 07 :A byteplot(Grey Scale) image of **Agent.FYI** is uploaded for the machine for the classification and the output received is **Agent.FYI in both the cases**. Hence the test case is totally passed.

Test Case 08 :A byteplot(Grey Scale) image of **VB.AT** is uploaded for the machine for the classification and the output received is **VB.AT in both the cases**.Hence the test case is totally passed.

Test Case 09 :A byteplot(Grey Scale) image of **Rbot!gen** is uploaded for the machine for the classification and the output received is **Rbot!gen in both the cases**. Hence the test case is totally passed.

Test Case 10 :A byteplot(Grey Scale) image of **Loylda.AA2** is uploaded for the machine for the classification and the output received is **Loylda.AA2 in both the cases**. Hence the test case is totally passed.

Test Case 11 :A byteplot(Grey Scale) image of **Malex.gen!J** is uploaded for the machine for the classification and the output received is **Malex.gen!J if it not reshaped** and the output received is **VB.AT if reshaped.** Hence the test case is partially passed.

Test Case 12 :A byteplot(Grey Scale) image of Autorun.K is uploaded for the machine for the classification and the output received is **Autorun.K if it not reshaped** and the output received is **VB.AT if reshaped.** Hence the test case is partially passed.

Test Case 13 :A byteplot(Grey Scale) image of **Lolyda.AT** is uploaded for the machine for the classification and the output received is **Lolyda.AT in both the cases**. Hence the test case is totally passed.

Test Case 14 :A byteplot(Grey Scale) image of **Lolyda.AA1** is uploaded for the machine for the classification and the output received is **Lolyda.AA1 if it not reshaped** and the output received is **VB.AT if reshaped.** Hence the test case is partially passed.

Test Case 15 :A byteplot(Grey Scale) image of **Obfuscator.AD** is uploaded for the machine for the classification and the output received is **Obfuscator.AD in both the cases**. Hence the test case is totally passed.

Test Case 16 :A byteplot(Grey Scale) image of **Adialer.C** is uploaded for the machine for the classification and the output received is **Adialer.C in both the cases**. Hence the test case is totally passed.

Test Case 17 :A byteplot(Grey Scale) image of **Wintrim.BX** is uploaded for the machine for the classification and the output received is **Wintrim.BX if it not reshaped** and the output received is **C2LOP.gen!g if reshaped.** Hence the test case is partially passed.

Test Case 18 :A byteplot(Grey Scale) image of **Lolyda.AA1** is uploaded for the machine for the classification and the output received is **VB.AT if image is reshaped and Allaple.A if not reshaped**. Hence the test case is totally failed.

Test Case 19 :A byteplot(Grey Scale) image of **Allaple.L** is uploaded for the machine for the classification and the output received is **Allaple.L in both the cases**. Hence the test case is totally passed.

Test Case 20 :A byteplot(Grey Scale) image of **Dialplatform.B** is uploaded for the machine for the classification and the output received is **Dialplatform.B in both the cases**. Hence the test case is totally passed.

Test Case 21 :A byteplot(Grey Scale) image of **C2LOP.P** is uploaded for the machine for the classification and the output received is **Swizzor.gen!E if image is reshaped and Allaple.A if not reshaped.** Hence the test case is totally failed.

Test Case 22 :A byteplot(Grey Scale) image of **Fakerean** is uploaded for the machine for the classification and the output received is **Fakerean in both the cases**. Hence the test case is totally passed.

Test Case 23 :A byteplot(Grey Scale) image of **Swizzor.gen!E** is uploaded for the machine for the classification and the output received is **Swizzor.gen!I if image is reshaped and C2LOP.P if not reshaped**. Hence the test case is totally failed.

Test Case 24 :A byteplot(Grey Scale) image of **Instantaccess** is uploaded for the machine for the classification and the output received is **Instantaccess in both the cases**. Hence the test case is totally passed.

Test Case 25 : A byteplot(Grey Scale) image of **Skintrim.N** is uploaded for the machine for the classification and the output received is **Skintrim.N if not reshaped and** the output received is **VB.AT if reshaped**. Hence the test case is passed.

7. CONCLUSION & FUTURE ENHANCEMENT

7.CONCLUSION & FUTURE SCOPE

The project focused on programming a machine to process malware files, specifically .exe files, by converting them into grayscale images with a .png extension. Convolutional Neural Networks (CNNs) were used to train the machine, as they are designed for processing pixel data and image recognition. Two approaches were compared: one without rescaling the dataset, and one with rescaling.

The accuracy and loss metrics were used to evaluate the performance of the machine. The machine programmed without rescaling achieved a high accuracy of >99% and a low loss of <3%. On the other hand, the machine programmed with rescaling achieved a slightly lower accuracy of >98% and a slightly higher loss of <7%. The results were obtained using the provided dataset and with the test cases as previously mentioned.

However, it should be noted that deep learning architectures, including CNNs, are susceptible to adversarial environments. Generative Adversarial Network methods can be used to generate samples during testing or deployment phases, which can potentially fool deep learning architectures. The robustness of the deep learning architecture in the proposed work was not discussed, and this could be an important direction for future research, especially in security-critical environments where malware bias can have serious consequences for organizations.

This project also revealed that testing the malware families in two different ways, i.e., with and without rescaling, resulted in different outcomes in most cases. Only a few families completely passed the test, while the rest either partially passed or failed altogether. This highlights the complexity and variability of malware behavior and the need for robust and adaptable approaches in malware detection and classification.

In conclusion, the study demonstrated the effectiveness of using CNNs for processing malware files in the form of grayscale images. However, further research is needed to explore the robustness of deep learning architectures in adversarial environments, and to develop more resilient approaches for malware detection and classification. A classification error in this context can have severe consequences for organizations, emphasizing the importance of addressing this issue in security-critical applications.

FUTURE ENHANCEMENTS

The current project utilized a custom-made dataset consisting of grayscale PNG images of 25 classes of malware. However, there is potential for improvement by expanding the dataset with additional malware image files from various malware families. This can enhance the classification accuracy and allow for better generalization of the trained model to different types of malware.

The loss observed in the project was approximately 4% when the dataset was not rescaled, and around 7% when it was rescaled. While these results indicate good performance, further reduction in the loss can lead to even better processing, classification, and output accuracy. Fine-tuning the model and optimizing hyperparameters can be explored to achieve lower loss values, thus improving the performance of the machine in malware detection and classification tasks.

In addition to using Convolutional Neural Networks (CNNs), other approaches can also be considered to gain a deeper understanding of how the algorithm works and the potential impact it can bring to the machine. Exploring different deep learning architectures, such as recurrent neural networks (RNNs) or transformer-based models, may provide valuable insights into their effectiveness for malware detection. Utilizing ensemble methods, where multiple models are combined, can also potentially improve the overall performance and robustness of the system.

It is important to continue research and development in this field to stay ahead of constantly evolving malware threats. By adopting a multi-pronged approach and leveraging various techniques and algorithms, the machine can be further enhanced to achieve higher accuracy and robustness in detecting and classifying malware. This can contribute to the advancement of security-critical applications and help organizations in mitigating the risks associated with malware attacks.

8. BIBLIOGRAPHY

8.BIBLIOGRAPHY

R.Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran and S. Venkatraman, "Robust Intelligent Malware Detection Using Deep Learning," in IEEE Access, vol. 7, pp. 46717-46738, 2019, doi: 10.1109/ACCESS.2019.2906934.

Mahesh Arse, Kanhaiya Sharma, Shantanu Bindewari, Ashwin Tomar, Harshal Patil, Neha Jha, "Mitigating Malware Attacks using Machine Learning: A Review", 2023 International Conference on Artificial Intelligence and Smart Communication (AISC), pp.1032-1038, 2023.

P. Suresh Kumar, Umi Salma. B, Isa Mishra, Shitharth S, Diwakar Ramanuj Tripathi, Siva Rama Krishna T., "Malware Detection Classification using Recurrent Neural Network", 2022 2nd International Conference on Technological Advancements in Computational Sciences (ICTACS), pp.876-880, 2022.

Shivarti Naik, Amita Dessai, "Malware Classification Approaches Using Machine Learning Techniques: A Review", 2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT), pp.111-117, 2021

https://en.wikipedia.org/wiki/Malware

https://towardsdatascience.com/malware-classification-using-convolutional-neural-networks-step-by-step-tutorial-a3e8d97122f

GITHUB LINK : https://github.com/vivekbharu25/MalwareDetection.git

9. MANUSCRIPT

***Abstract:** The increasing frequency of protection breaches caused by malicious software, commonly known as malware, is a significant security concern in the digital age. With a growing number of computer users, organizations, and governments falling victim to malware attacks, detecting and mitigating malware has become a critical area of research. Traditional malware detection methods, which rely on static and dynamic analysis of malware signatures and behavior patterns, are time-consuming and have proven to be ineffective in identifying unknown malwares in real-time.*

Modern malwares are designed to evade detection by using techniques such as polymorphism and metamorphism, which rapidly change the behavior of the malware and generate numerous variants of the same malware. Machine learning algorithms (MLAs) have been increasingly employed to conduct effective malware analysis, particularly using feature engineering, feature learning, and feature representation. However, these methods also require significant time and resources.

To address this challenge, advanced MLAs, such as deep learning, can be utilized to eliminate the need for extensive feature engineering. Deep learning algorithms are capable of automatically extracting relevant features from the data, allowing for more efficient and effective malware analysis. By leveraging the power of deep learning, the time-consuming feature engineering phase in traditional MLAs can be circumvented, leading to faster and more accurate detection of unknown malwares in real-time.

In conclusion, the increasing sophistication and evasive techniques employed by modern malwares pose a significant security challenge. Traditional malware detection methods are ineffective and time-consuming. However, advanced MLAs, such as deep learning, offer promising solutions by automating feature extraction and enabling real-time detection of unknown malwares, thereby improving the effectiveness and efficiency of malware analysis in the virtual age.

INTRODUCTION

The project titled "Malware Detection Using Deep Learning with Image Processing" aims to leverage the power of deep learning to improve the efficiency of detecting malware in devices. Traditional methods of malware detection can be costly, inefficient, and time-consuming. However, with advancements in technologies like deep learning, there is an opportunity to upgrade these methods for better performance. By utilizing deep learning techniques, the project aims to enhance the accuracy and efficiency of malware detection. This project seeks to explore the potential of image processing in combination with deep learning for detecting malware. By leveraging these cutting-edge technologies, this project aims to provide a more effective solution for malware detection, addressing the limitations of traditional methods and improving the overall detection capabilities.

The main purpose of this project is to develop a malware detection system using deep learning and image processing techniques. The process involves converting a file into a grayscale image. The machine is then trained with a dataset to determine whether the converted image represents a malware or not. This approach leverages the power of deep learning algorithms to automatically extract relevant features from the grayscale images, allowing for efficient detection of malware. The system aims to improve the accuracy and efficiency of malware detection compared to traditional methods. The grayscale image conversion and deep learning-based detection process enable real-time analysis of files for malware presence. The project combines the fields of deep learning and image processing to create an effective and efficient malware detection system. This approach eliminates the need for extensive feature engineering, making the detection process faster and more accurate. The project has the potential to contribute to the advancement of malware detection techniques using cutting-edge technologies.

This project employs Convolutional Neural Networks (CNN), a type of deep learning approach, for the categorization of malware families. CNNs are widely used for image/object recognition and classification tasks due to their ability to automatically learn relevant features from input data. The project utilizes visualization techniques in combination with deep learning to categorize malware families based on their visual characteristics.

The CNN architecture consists of convolutional layers, which extract features from the input data by applying filters to capture local patterns. The output of convolutional layers is then passed through pooling layers, which down-sample the feature maps, reducing spatial dimensions while retaining important information. This helps in reducing computational complexity and improving efficiency.

Finally, the output of pooling layers is passed through one or more fully connected layers, which are responsible for making predictions or classifying the input data into different malware families. The fully connected layers learn complex patterns and relationships between features extracted from the input data.

The combination of CNN architecture, visualization techniques, and deep learning enables effective and efficient categorization of malware families based on their visual characteristics, contributing to the advancement of malware detection and analysis techniques..

METHODOLOGY

The project titled "Malware Detection Using Deep Learning with Image Classification" is centered around using deep learning techniques, specifically convolutional neural networks (CNNs), to detect malware. The goal of the project is to categorize malware into different families using visualization techniques with deep learning.

The project starts by converting a file, which is suspected to be malware, into a grayscale image. This image serves as the input to the CNN model. Grayscale images have only one channel (gray intensity), which simplifies the input data for the neural network.

The CNN model is trained on a dataset of labeled malware samples. The training process involves feeding the grayscale images into the CNN and adjusting the model's weights and biases to learn the patterns and features that distinguish malware from benign files. The CNN learns to extract relevant features from the images, such as pixel intensities, textures, and shapes, to make accurate predictions.

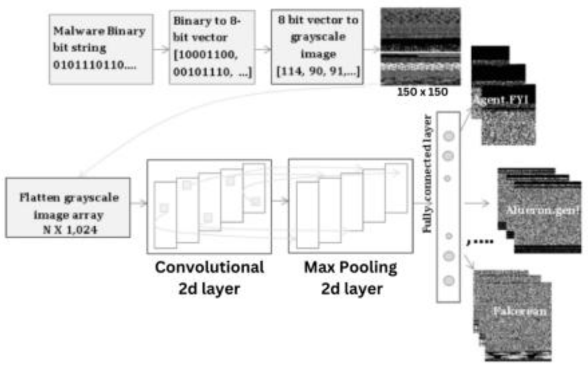

Malware Binary bit string 0101110110....
Binary to 8-bit vector [10001100, 00101110, ...]
8 bit vector to grayscale image [114, 90, 91,...]
150 x 150
Agent.FYI
Flatten grayscale image array N X 1,024
Convolutional 2d layer
Max Pooling 2d layer
Fully connected layer
Alueron.gen!
Fakerean

Figure1: System Architecture

Once the CNN model is trained, it can be used for malware detection. The grayscale image of a suspected file is fed into the trained CNN, and the model predicts whether the file is malware or not based on the patterns it has learned during training.

The project also explores two different approaches for detecting malware: "when rescaled" and "when not rescaled." The "when rescaled" approach likely refers to rescaling the grayscale image to a different size before feeding it into the CNN, while the "when not rescaled" approach may involve using the original grayscale image without any resizing. This suggests that the project may investigate how the size of the input image affects the accuracy of the malware detection model.

In summary, the project utilizes deep learning techniques, particularly CNNs, to detect malware by converting files into grayscale images and using visualization techniques for malware family categorization. It explores different approaches for image preprocessing and detection, and aims to provide an effective solution for malware detection using deep learning.

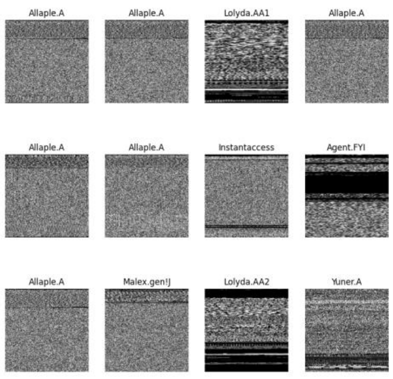


Figure 2: Types of Malwares as Greyscale png

EXPERIMENTAL RESULTS

The results of the experiment are shown below as the form of images.

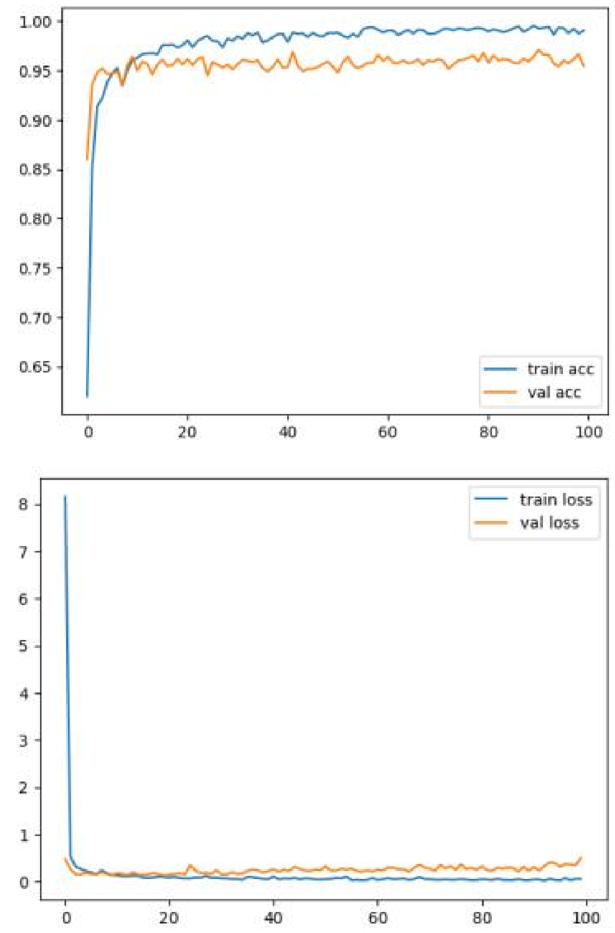
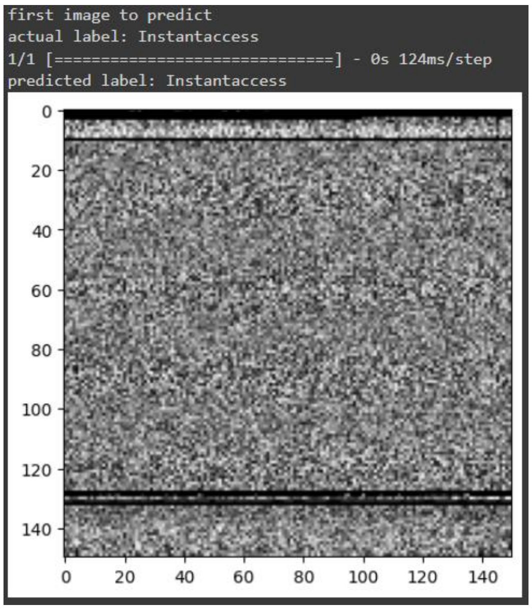


Figure 3: Accuracy and Loss of the machine when not rescaled.

The Accuracy received is 99.2% and loss we got is 03.82% .



```
first image to predict
actual label: Instantaccess
1/1 [==============================] - 0s 124ms/step
predicted label: Instantaccess
```

Figure 4: First Image to predict : Instantaccess

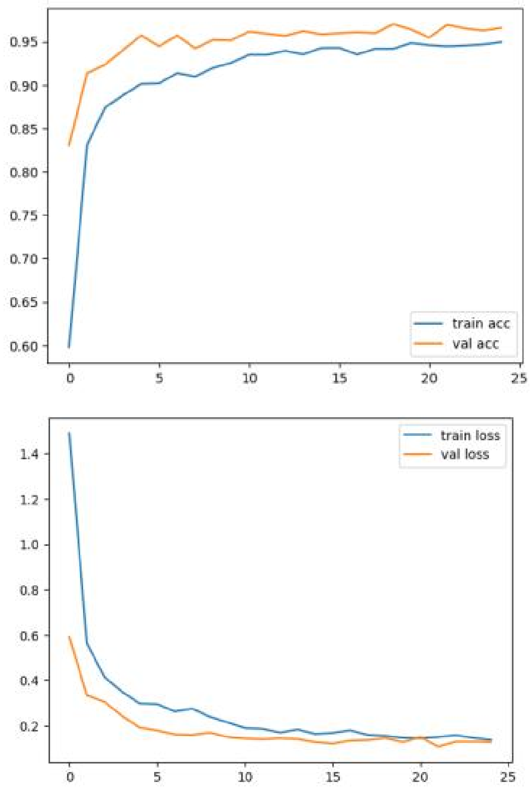


Figure 5: Accuracy and Loss of the machine when rescaled.

The Accuracy received is 98.2% and loss we got is 06.32% .

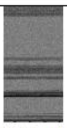
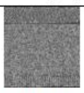
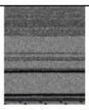
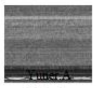
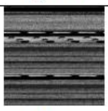
S.No.	Image Uploaded	With shaping OUTPUT	Without Shaping OUTPUT	Result
1	 C2LOP.gen!g	C2LOP.gen!g	C2LOP.gen!g	Totally Passed.
2	 Allaple.A	Swizzor.gen!I	Allaple.A	Passed, if not reshaped.
3	 Swizzor.gen!I	Swizzor.gen!I	Swizzor.gen!I	Totally Passed
4	Alueron.gen!J	Alueron.gen!J	Alueron.gen!J	Totally Passed
5	Dontovo.A	Dontovo.A	Dontovo.A	Totally Passed
6	 Yuner.A	VB.AT	Yuner.A	Passed, if it is not reshaped.
7	Agent.FYI	Agent.FYI	Agent.FYI	Totally Passed
8	 VB.AT	VB.AT	VB.AT	Totally Passed
9	Rbot!gen	Rbot!gen	Rbot!gen	Totally Passed
10	Loylda.AA2	Loylda.AA2	Loylda.AA2	Totally Passed

Table 1 : Test cases of the Experiment

FUTURE ENHANCEMENTS

The current project utilized a custom-made dataset consisting of grayscale PNG images of 25 classes of malware. However, there is potential for improvement by expanding the dataset with additional malware image files from various malware families. This can enhance the classification accuracy and allow for better generalization of the trained model to different types of malware.

The loss observed in the project was approximately 4% when the dataset was not rescaled, and around 7% when it was rescaled. While these results indicate good performance, further reduction in the loss can lead to even better processing, classification, and output accuracy. Fine-tuning the model and optimizing hyperparameters can be explored to achieve lower loss values, thus improving the performance of the machine in malware detection and classification tasks.

In addition to using Convolutional Neural Networks (CNNs), other approaches can also be considered to gain a deeper understanding of how the algorithm works and the potential impact it can bring to the machine. Exploring different deep learning architectures, such as recurrent neural networks (RNNs) or transformer-based models, may provide valuable insights into their effectiveness for malware detection. Utilizing ensemble methods, where multiple models are combined, can also potentially improve the overall performance and robustness of the system.

It is important to continue research and development in this field to stay ahead of constantly evolving malware threats. By adopting a multi-pronged approach and leveraging various techniques and algorithms, the machine can be further enhanced to achieve higher accuracy and robustness in detecting and classifying malware. This can contribute to the advancement of security-critical applications and help organizations in mitigating the risks associated with malware attacks.

CONCLUSION

The project focused on programming a machine to process malware files, specifically .exe files, by converting them into grayscale images with a .png extension. Convolutional Neural Networks (CNNs) were used to train the machine, as they are designed for processing pixel data and image recognition. Two approaches were compared: one without rescaling the dataset, and one with rescaling.

The accuracy and loss metrics were used to evaluate the performance of the machine. The machine programmed without rescaling achieved a high accuracy of >99% and a low loss of <3%. On the other hand, the machine programmed with rescaling achieved a slightly lower accuracy of >98% and a slightly higher loss of <7%. The results were obtained using the provided dataset and with the test cases as previously mentioned.

However, it should be noted that deep learning architectures, including CNNs, are susceptible to adversarial environments. Generative Adversarial Network methods can be used to generate samples during testing or deployment phases, which can potentially fool deep learning architectures. The robustness of the deep learning architecture in the proposed work was not discussed, and this could be an important direction for future research, especially in security-critical environments where malware bias can have serious consequences for organizations.

This project also revealed that testing the malware families in two different ways, i.e., with and without rescaling, resulted in different outcomes in most cases. Only a few families completely passed the test, while the rest either partially passed or failed altogether. This highlights the complexity and variability of malware behavior and the need for robust and adaptable approaches in malware detection and classification.

In conclusion, the study demonstrated the effectiveness of using CNNs for processing malware files in the form of grayscale images. However, further research is needed to explore the robustness of deep learning architectures in adversarial environments, and to develop more resilient approaches for malware detection and classification. A classification error in this context can have severe consequences for organizations, emphasizing the importance of addressing this issue in security-critical applications.

ACKNOWLEDGEMENT

We thank CMR Technical Campus for supporting this paper titled “Malware Detection Using Deep Learning with Image Processing” , which provided good facilities and support to accomplish our work. Sincerely thank our Chairman, Director, Deans, Head of the Department, Department of Computer Science and

Engineering, Guide and Teaching and Non-Teaching faculty members for giving valuable suggestions and guidance in every aspect of our work.

REFERENCE

1. R.Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran and S. Venkatraman, "Robust Intelligent Malware Detection Using Deep Learning," in IEEE Access, vol. 7, pp. 46717-46738, 2019, doi: 10.1109/ACCESS.2019.2906934.
2. Mahesh Arse, Kanhaiya Sharma, Shantanu Bindewari, Ashwin Tomar, Harshal Patil, Neha Jha, "Mitigating Malware Attacks using Machine Learning: A Review", 2023 International Conference on Artificial Intelligence and Smart Communication (AISC), pp.1032-1038, 2023.
3. P.Suresh Kumar, Umi Salma. B, Isa Mishra, Shitharth S, Diwakar Ramanuj Tripathi, Siva Rama Krishna T., "Malware Detection Classification using Recurrent Neural Network", 2022 2nd International Conference on Technological Advancements in Computational Sciences (ICTACS), pp.876-880, 2022.
4. Shivarti Naik, Amita Dessai, "Malware Classification Approaches Using Machine Learning Techniques: A Review", 2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT), pp.111-117, 2021