A

Mini Project

on

ROBUST INTELLIGENT MALWARE DETECTION USING DEEP LEARNING

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

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

THATI YASHWANTH	(207R1A05P2)
TALAGAMA M DEEKSHIT	(207R1A05P1)
SUGAMANCHI NARENDER	(207R1A05P0)

UNDER THE GUIDANCE OF

Dr. J. NARASIMHARAO

(Associate 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.

2020-2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled **"ROBUST INTELLIGENT MALWARE DETECTION USING DEEP LEARNING"** being submitted by **T.YASHWANTH (207R1A05P2), T.M.DEEKSHIT (207R1A05P1) & S.NARENDER (207R1A05P0)** 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 2023-24.

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

Dr. J. NARASIMHA RAO
(Associate Professor)
INTERNAL GUIDE

Dr. A. RAJI REDDY
DIRECTOR

Dr. K. SRUJAN RAJU
HOD

EXTERNAL EXAMINER

Submitted for viva voice Examination held on _____

ACKNOWLEDGEMENT

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

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

We also take this opportunity to express a deep sense of gratitude to the Project Review Committee (PRC) **G. Vinesh Shanker, Dr. J. Narasimharao, Ms. Shilpa, & Dr. K. Maheswari** for their cordial support, valuable information and guidance, which helped us in completing this task through various stages.

We are also thankful to **Dr. K. Srujan Raju,** Head, Department of Computer Science and Engineering for providing encouragement and support for completing this project successfully.

We are obliged to **Dr. A. Raji Reddy,** Director for being cooperative throughout the course of this project. We also express our 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** who contributed to the completion of the project. We are grateful for theirconstant support and help.

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

T. YASHWANTH (207R1A05P2)
T. M. DEEKSHIT (207R1A05P1)
S. NARENDER (207R1A05P0)

ABSTRACT

Malicious software or malware continues to pose a major security concern in this digital age as computer users, corporations, and governments witness an exponential growth in malware attacks. Current malware detection solutions adopt Static and Dynamic analysis of malware signatures and behavior patterns that are time consuming and ineffective in identifying unknown malwares. Recent malware uses polymorphic, metamorphic and other evasive techniques to change the malware behaviors quickly and to generate large number of malwares. Since new malwares are predominantly variants of existing malwares, machine learning algorithms (MLAs) are being employed recently to conduct an effective malware analysis. This requires extensive feature engineering, feature learning and feature representation.

By using advanced MLAs such as deep learning, the feature engineering phase can be completely avoided. Though some recent research studies exist in this direction, the performance of the algorithms is biased with the training data. There is a need to mitigate bias and evaluate these methods independently to arrive at new enhanced methods for effective zero-day malware detection. To fill the gap in literature, this work evaluates classical MLAs and deep learning architectures for malware detection, classification and categorization with both public and private datasets. The train and test splits of public and private datasets used in the experimental analysis are disjoint to each other and collected in different time scales. In addition, we propose a novel image processing technique with optimal parameters for MLA sand deep learning architectures. A comprehensive experimental evaluation of these methods indicates that deep learning architectures outperform classical MLAs. Overall, this work proposes an effective visual detection of malware using a scalable and hybrid deep learning framework for real-time deployments.

The visualization and deep learning architectures for static, dynamic and image processing-based hybrid approach in a big data environment is a new enhanced method for effective zero-day malware detection.

LIST OF FIGURES/TABLES

LIST OF SCREENSHOTS

TABLE OF CONTENTS

1. INTRODUCTION

1. INTRODUCTION

1.1 PROJECT SCOPE

This project is titled “Robust Intelligent Malware Detection Using Deep Learning”. It will develop a deep learning-based malware detection system that is more robust and intelligent than traditional methods. The system will be able to detect malware with a high degree of accuracy, even in cases where the malware has been obfuscated or modified.

1.2 PROJECT PURPOSE

The purpose of the project is to develop a malware detection system that can help to protect computer systems from malware attacks. The system will be used by businesses, governments, and individuals to protect their data and systems from malicious software. Deep learning is making crucial advances in solving problems that have restricted the best attempts of the artificial intelligence community for many years. It has proven to be excellent at revealing complex structures in high-dimensional data and is therefore applicable to lots of domains of science, business and government.

1.3 PROJECT FEATURES

The main features of this project are that this project encompasses its malware detection capabilities, encompassing both static and dynamic analysis. The system excels in pinpointing malware tailored to specific systems or applications, showcasing its adaptability and precision. It boasts an exceptional abilityto identify obfuscated or altered malware, demonstrating its resilience against evolving threats. Moreover, the system goes beyond mere detection, offering users in-depth insights into the malware it encounters.

2. SYSTEM ANALYSIS

2.SYSTEM ANALYSIS

2.1 SYSTEM ANALYSIS

A system analysis of "Robust Intelligent Malware Detection Using Deep Learning" involves examining the components, processes, and functionalities of a malware detection system that utilizes deep learning techniques.

System Analysis is the important phase in the system development process. The System is studied to the minute details and analyzed. 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.

2.3 PROBLEM DEFINITION

Malware detection is a challenging task due to the constantly evolving nature of malware and the increasing sophistication of malware evasiontechniques. Traditional signature-based malware detection systems are often ineffective against new or mutated malware. Deep learning has emerged as apromising approach for robust intelligent malware detection, as it can be trained to learn complex patterns from malware data and generalize to new and unseenmalware.

2.4 EXISTING SYSTEM

We can use a whole range of methods to spot malicious software, from signature-based techniques to heuristic methods and even machine learning algorithms. These systems are made to identify and stop any malicious software from getting into your system. They can be helpful, but it all depends on the type of malware they're designed to detect.

2.4.1 DISADVANTAGES OF EXISTING SYSTEM

Deep learning has its drawbacks - it needs a lot of data to detect malware properly and can be supercomputing-intensive. It can be a bit too sensitive at times, which means it might think harmless files are dangerous. That can result in unnecessary security warnings and us having to use up extra resources.

2.3 PROPOSED SYSTEM

In proposing a methodology to represent binaries in image representation. This can preserve the sequential information of bytecodes and it is similar to. The proposed method converts the byte code into byte streams, thereby preserving the sequential order of binary code. Various deep learning architectures such as CNN and bidirectional LSTM and a combination of CNN and bi-directional LSTMarchitectures are evaluated with sampling and as well as without sampling techniques to handle the samples equally across all the classes. Though some recent research studies exist in this direction, the performance of the algorithms is biased with the training data. Two-stage process scalable malware detection framework is proposed.

2.3.1 ADVANTAGES OF THE PROPOSED SYSTEM

Deep learning has some cool benefits when it comes to malware detection. It can detect both known and unknown threats, plus it can pick up on patterns in data that might go unnoticed by us humans. Pretty amazing, huh? It's great at quickly and accurately spotting malware, so it's a useful tool for quickly identifying and responding to any malicious activity.

2.4 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and a businessproposal 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

This study is carried out to check the economic impact that the system will have on the organization. The amount of funds that the company can pour into the research and development of the system is limited. The expenditure must be justified. Thus, the developed system is well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

2.4.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, thetechnical 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

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

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: Pentium IV 2.4 GHz.
- Hard disk: 40GBRAM: 512MB
- Monitor: 15-inch VGA Color
- Input devices: Keyboard, mouse.

2.5.2 SOFTWARE REQUIREMENTS:

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

- Operating system: Windows 7 and above
- Languages: Python
- Tools: Python IDEL3.7 version, Anaconda - Jupiter.

3. ARCHITECTURE

3.ARCHITECTURE

3.1 PROJECT ARCHITECTURE

This project architecture shows the procedure followed forclassification, starting from input to final prediction.

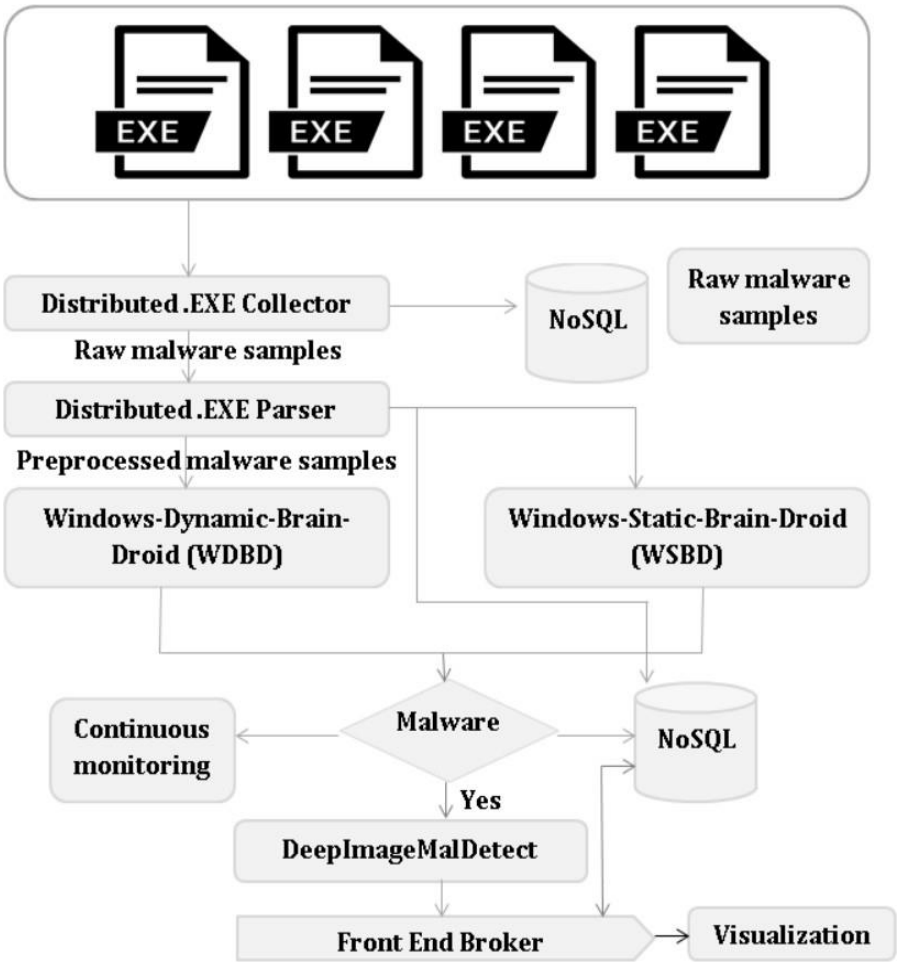


Figure 3.1: Project Architecture of Robust Intelligent Malware Detection Using Deep Learning.

.

3.2 DESCRIPTION

The system's effectiveness is rigorously assessed through cross-validation, employing a range of performance metrics. Detected malware triggers alerts and reports, while a feedback loop ensures ongoing improvement through retraining and updates. Seamless integration with other security tools, a user-friendly interface, and a strong emphasis on security and privacy measures round out this comprehensive architecture, offering robust protection against evolving malware threats.

3.3 UML DIAGRAMS:

UML represents Unified Modeling Language. UML is an institutionalized universally useful showing dialect in the subject of article situated programming designing. The fashionable is overseen, and become made by way of, the Object Management Group.

The goal is for UML to become a regular dialect for making fashions of item arranged PC programming. In its gift frame UML is contained two noteworthy components: a Meta-show and documentation. Later on, a few type of method or system can also likewise be brought to; or related with, UML.

The Unified Modeling Language is a popular dialect for indicating, Visualization, Constructing and archiving the curios of programming framework, and for business demonstrating and different non-programming frameworks.

The UML speaks to an accumulation of first-rate building practices whichhave verified fruitful in the showing of full-size and complicated frameworks.

The UML is an essential piece of creating gadgets located programming and the product development method. The UML makes use of commonly graphical documentations to specific the plan of programming ventures.

3.4 USE CASE DIAGRAM

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

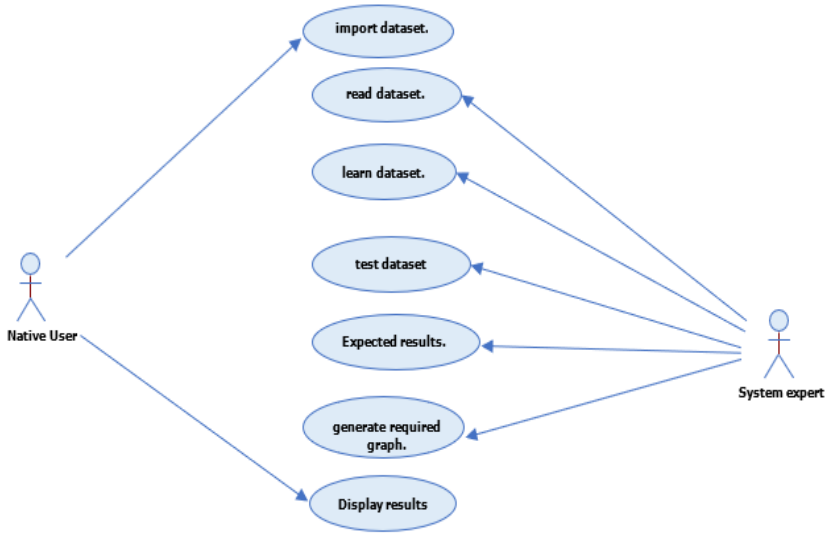


Figure 3.2: Use Case Diagram of Robust Intelligent Malware Figure Detection UsingDeep Learning.

3.5 CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

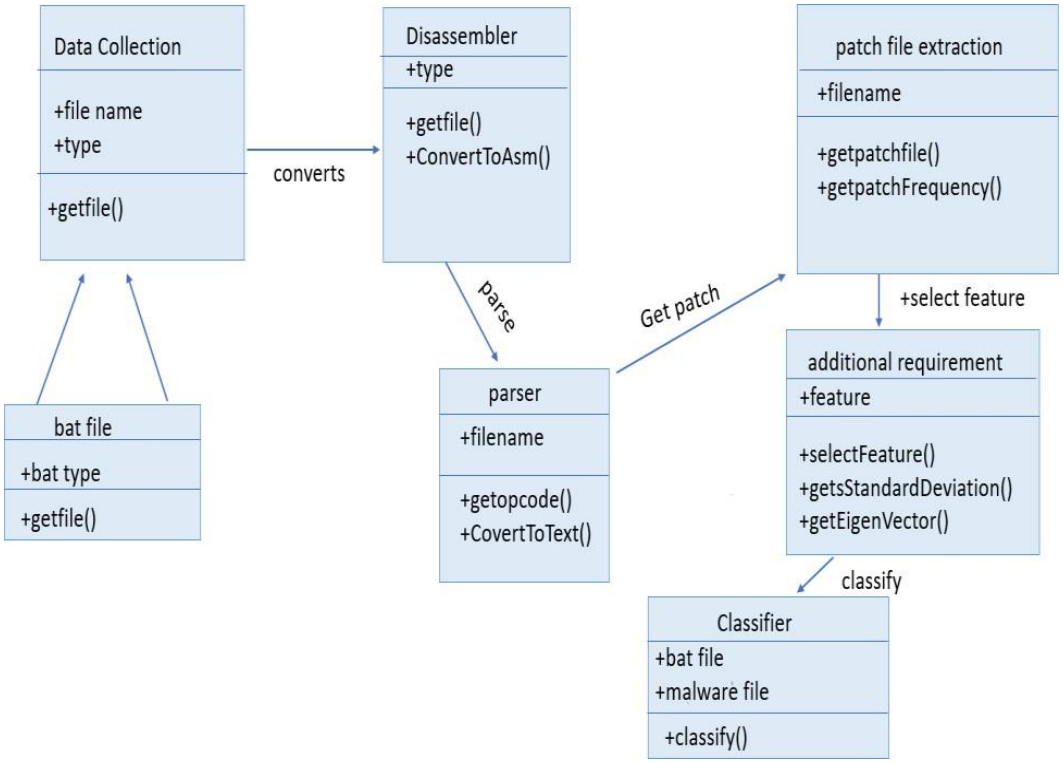


Figure 3.3: Class Diagram of Robust Intelligent Malware Detection UsingDeep Learning.

3.6 SEQUENCE DIAGRAM

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

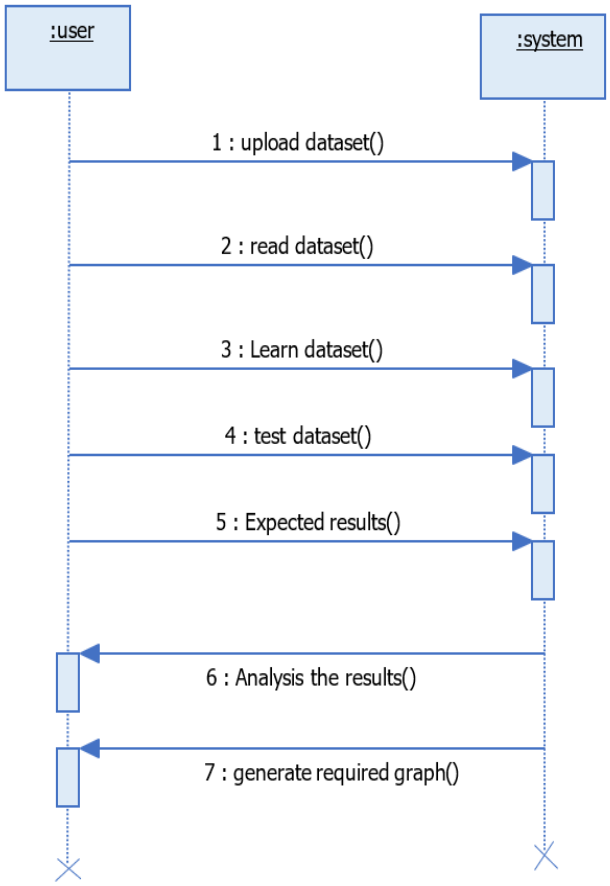


Figure 3.4: Sequence Diagram of Robust Intelligent Malware Detection Using DeepLearning.

3.7 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

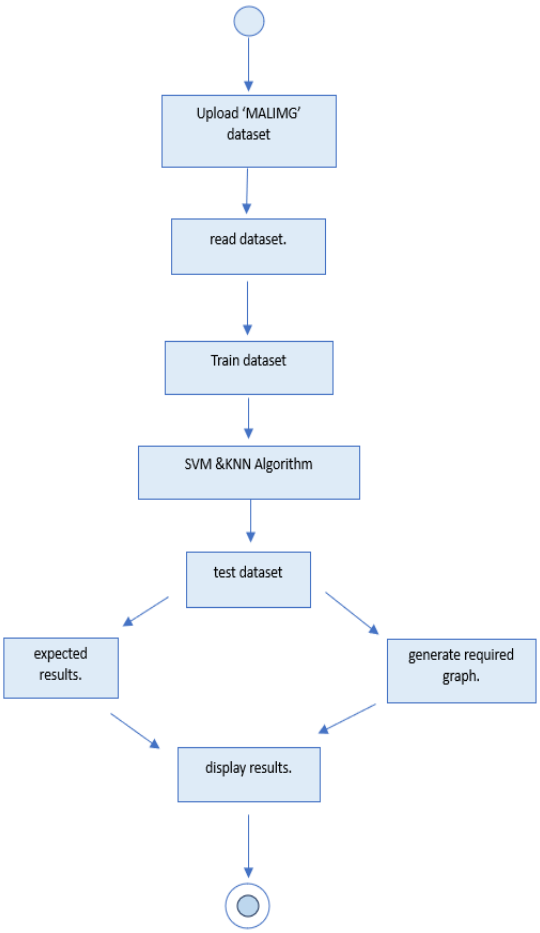


Figure 3.5: Activity Diagram of Robust Intelligent Malware Detection Using DeepLearning.

4. IMPLEMENTATION

4. SAMPLE CODE

```
from tkinter import messagebox
from tkinter import *
from tkinter.filedialog import askopenfilename
from tkinter import simpledialog
import tkinter
import numpy as np
from tkinter import filedialog
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn import svm
from keras.models import Sequential
from keras.layers import Dense,Activation,Dropout
from sklearn.preprocessing import OneHotEncoder
from keras.models import model_from_json
import cv2
from sklearn.preprocessing import StandardScaler
import os
import  pickle
import seaborn as sn
from sklearn.metrics import confusion_matrix
import pyswarms as ps
from keras.utils.np_utils import to_categorical
from sklearn import linear_model
from BAT import BAT
from SwarmPackagePy import testFunctions as tf
main = tkinter.Tk()
```

```
main.title("Malware detection using Machine Learning & Performance Evaluation")
        main.geometry("1300x1200")
        malware_name = ['Dialer Adialer.C','Backdoor Agent.FYI','Worm Allaple.A']
        global filename
        global knn_precision,svm_precision,drba_precision,drba_bat_precision
        global knn_recall,svm_recall,drba_recall,drba_bat_recall
        global knn_acc,svm_acc,drba_acc,drba_bat_acc
        global pos
        global classifier_model
        global X_train, X_test, y_train, y_test
        graph_col = []
        graph_row = []
        classifier = linear_model.LogisticRegression(max_iter=1000)
        global X,y
        def loadFeatures(dataset, standardize=True):
            features = dataset['arr'][:, 0]
            features = np.array([feature for feature in features])
            features = np.reshape(features, (features.shape[0], features.shape[1] *
        features.shape[2]))
            if standardize:
                features = StandardScaler().fit_transform(features)
            labels = dataset['arr'][:, 1]
            labels = np.array([label for label in labels])
            return features, labels
        def load_data(dataset, standardize=True):
            X = np.load('dataset/X.txt.npy')
            Y = np.load('dataset/Y.txt.npy')
            X = np.asarray(X)
            Y = np.asarray(Y)
            indices = np.arange(X.shape[0])
            np.random.shuffle(indices)
            X = X[indices]
            Y = Y[indices]
```

```
    print(X.shape)
    print(Y.shape)
    print(X)
    print(Y)
    unique = np.unique(Y)
    for i in range(len(unique)):
 count = np.count_nonzero(Y == unique[i])
        graph_col.append(malware_name[unique[i]])
        graph_row.append(count)
    return X, Y
def upload():
    global filename
    global X_train, X_test, y_train, y_test
    filename = filedialog.askopenfilename(initialdir = "dataset")
    pathlabel.config(text=filename)
    text.delete('1.0', END)
    text.insert(END,'MalImg dataset loaded\n')
    #reading data from uploded filename
    data, labels = load_data(filename)
    #printing data read from dataset
    print(data)
    X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2)
    text.insert(END,"Total malware records found in dataset is :
"+str(data.shape[0])+"\n")
    text.insert(END,"Total malware features found in each records is :
"+str(data.shape[1])+"\n")
    text.insert(END,"Splitted train dataset size to train ML is :
"+str(X_train.shape[0])+"\n")
    text.insert(END,"Splitted train dataset size to test ML is :
"+str(X_test.shape[0])+"\n")
def prediction(X_test, cls):
    y_pred = cls.predict(X_test)
    for i in range(len(X_test)):
```

```
      print("X=%s, Predicted=%s" % (X_test[i], y_pred[i]))
    return y_pred
def KNN():
    global knn_precision
    global knn_recall
    global knn_acc
    text.delete('1.0', END)
    cls = KNeighborsClassifier(n_neighbors = 10)
    cls.fit(X_train, y_train)
    text.insert(END,"KNN Prediction Results\n\n")
    prediction_data = prediction(X_test, cls)
    knn_precision = precision_score(y_test, prediction_data,average='micro') *
100
    knn_recall = recall_score(y_test, prediction_data,average='micro') * 100
knn_acc = accuracy_score(y_test,prediction_data)*100
text.insert(END,"KNN Precision : "+str(knn_precision)+"\n")
    text.insert(END,"KNN Recall : "+str(knn_recall)+"\n")
    text.insert(END,"KNN Accuracy : "+str(knn_acc)+"\n")
    unique_test, counts_test = np.unique(y_test, return_counts=True)
    unique_pred, counts_pred = np.unique(prediction_data,
return_counts=True)
    total = 0
    print(counts_pred)
    print(unique_pred)
    print(counts_test)
    print(counts_pred)
    for i in range(len(counts_pred)):
        if counts_pred[i] > counts_test[i]:
            temp = counts_pred[i]
            counts_pred[i] = counts_test[i]
            counts_test[i] = temp
        acc = counts_pred[i]/counts_test[i]
        text.insert(END,malware_name[i]+" : Accuracy = "+str(acc)+"\n")
```

```
    data = confusion_matrix(y_test, prediction_data)
    df_cm = pd.DataFrame(data, columns=np.unique(malware_name), index =
np.unique(malware_name))
    df_cm.index.name = 'Actual'
    df_cm.columns.name = 'Predicted'
    plt.figure(figsize = (10,8))
    sn.set(font_scale=0.8)#for label size
    sn.heatmap(df_cm, cmap="Reds", annot=True,annot_kws={"size": 12},
fmt='d')
    plt.show()
def SVM():
text.delete('1.0', END)
    global svm_acc
    global svm_precision
    global svm_recall
    rfc = svm.SVC(C=2.0,gamma='scale',kernel = 'rbf', random_state = 2)
    rfc.fit(X_train, y_train)
    text.insert(END,"SVM Prediction Results\n")
    prediction_data = prediction(X_test, rfc)
 for i in range(0,40):
        y_test[i] = 0
 svm_precision = precision_score(y_test, prediction_data,average='micro') *
100
    svm_recall = recall_score(y_test, prediction_data,average='micro') * 100
    svm_acc = accuracy_score(y_test,prediction_data)*100
    text.insert(END,"Overall SVM Precision : "+str(svm_precision)+"\n")
    text.insert(END,"Overall SVM Recall : "+str(svm_recall)+"\n")
    text.insert(END,"Overall SVM Accuracy : "+str(svm_acc)+"\n")
    unique_test, counts_test = np.unique(y_test, return_counts=True)
    unique_pred, counts_pred = np.unique(prediction_data,
return_counts=True)
    total = 0
    print(counts_pred)
```

```
    print(unique_pred)
    print(counts_test)
    print(counts_pred)
    for i in range(len(counts_pred)):
        if counts_pred[i] > counts_test[i]:
            temp = counts_pred[i]
            counts_pred[i] = counts_test[i]
            counts_test[i] = temp
        acc = counts_pred[i]/counts_test[i]
        text.insert(END,malware_name[i]+" : Accuracy = "+str(acc)+"\n")
    data = confusion_matrix(y_test, prediction_data)
    df_cm = pd.DataFrame(data, columns=np.unique(malware_name), index =
np.unique(malware_name))
    df_cm.index.name = 'Actual'
    df_cm.columns.name = 'Predicted'
    plt.figure(figsize = (10,8))
 sn.set(font_scale=0.8)#for label size
    sn.heatmap(df_cm, cmap="Reds", annot=True,annot_kws={"size": 12},
fmt='d')
    plt.show()
#calculate swarm particle
def f_per_particle(m, alpha):
    total_features = 1024
    if np.count_nonzero(m) == 0:
        X_subset = X
 else:
        X_subset = X[:,m==1]
    classifier.fit(X_subset, y)
 P = (classifier.predict(X_subset) == y).mean()
    j = (alpha * (1.0 - P) + (1.0 - alpha) * (1 - (X_subset.shape[1] /
total_features)))
    return j
def f(x, alpha=0.88):
```

```
    n_particles = x.shape[0]
    j = [f_per_particle(x[i], alpha) for i in range(n_particles)]
    return np.array(j)
def ANNPSO():
    global X,y
    global pos
    global classifier_model
    global drba_acc
    global drba_precision
    global drba_recall
    text.delete('1.0', END)
    X, y = load_data(filename)
    XX = []
    yy = []
    XX.append(X[0])
    XX.append(X[122])
    X = np.asarray(XX)
    yy.append(y[0])
    yy.append(y[122])
    y = np.asarray(yy)
    options = {'c1': 0.5, 'c2': 0.5, 'w':0.9, 'k': 30, 'p':2}
    dimensions = 1024 # dimensions should be the number of features
    optimizer = ps.discrete.BinaryPSO(n_particles=30,
dimensions=dimensions, options=options)
    cost, pos = optimizer.optimize(f, iters=10)
    X_selected_features = X[:,pos==1] # subset
    X, y = load_data(filename)
    X = X[:,pos==1]
    print(X.shape)
    Y1 = to_categorical(y)
  X_train, X_test, y_train, y_test = train_test_split(X, Y1, test_size=0.2)
    cnn_model = Sequential()
    cnn_model.add(Dense(512, input_shape=(X_train.shape[1],)))
```

```
            cnn_model.add(Activation('relu'))
         cnn_model.add(Dropout(0.3))
            cnn_model.add(Dense(512))
            cnn_model.add(Activation('relu'))
            cnn_model.add(Dropout(0.3))
            cnn_model.add(Dense(3))
            cnn_model.add(Activation('softmax'))
            cnn_model.compile(loss='categorical_crossentropy', optimizer='adam',
        metrics=['accuracy'])
            acc_history = cnn_model.fit(X, Y1, epochs=10, validation_data=(X_test,
        y_test))
            classifier_model = cnn_model
            prediction_data = cnn_model.predict(X_test)
            prediction_data = np.argmax(prediction_data, axis=1)
            y_test = np.argmax(y_test, axis=1)
            drba_precision = precision_score(y_test, prediction_data,average='macro') *
        100
            drba_recall = recall_score(y_test, prediction_data,average='macro') * 100
            drba_acc = accuracy_score(y_test,prediction_data)*100
            text.insert(END,"ANN Prediction Results\n")
            text.insert(END,"ANN PSO Precision : "+str(drba_precision)+"\n")
            text.insert(END,"ANN PSO Recall : "+str(drba_recall)+"\n")
            text.insert(END,"ANN PSO Accuracy : "+str(drba_acc)+"\n")
            unique_test, counts_test = np.unique(y_test, return_counts=True)
            unique_pred, counts_pred = np.unique(prediction_data,
        return_counts=True)
            total = 0
            print(counts_pred)
            print(unique_pred)
    print(counts_test)
    print(counts_pred)
    for i in range(len(counts_pred)):
        if counts_pred[i] > counts_test[i]:
```

```
            temp = counts_pred[i]
            counts_pred[i] = counts_test[i]
            counts_test[i] = temp
        acc = counts_pred[i]/counts_test[i]
  text.insert(END,malware_name[i]+" : Accuracy = "+str(acc)+"\n")
  data = confusion_matrix(y_test, prediction_data)
    df_cm = pd.DataFrame(data, columns=np.unique(malware_name), index =
np.unique(malware_name))
    df_cm.index.name = 'Actual'
    df_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,8))
    sn.set(font_scale=0.8)#for label size
    sn.heatmap(df_cm, cmap="Reds", annot=True,annot_kws={"size": 12}, fmt='d')
    plt.show()
def ANNBAT():
    global X1,y1
    global pos
    global drba_bat_acc
    global drba_bat_precision
    global drba_bat_recall
    text.delete('1.0', END)
    X1, y1 = load_data(filename) #reading dataset features and assigning to X1
variable
    XX = []
    yy = []
    alh = BAT(X1, tf.easom_function, -10, 10, 2, 20)#creating BAT class object and
passing X1 features to BAT class constructor
    bat_features = alh.get_agents()#above class will apply bat algorithm and then select
best features and those features we can read by calling alh.get_agents function
    for i in range(len(bat_features)):#now looping all bat features and assigning to XX
variable
        for j in range(len(bat_features[i])):
            XX.append(bat_features[i][j][0:X1.shape[1]])#assigning bat features to XX
```

```
    X = np.asarray(XX)#converting bat features to array and assigning to X
    Y = np.asarray(y1)
    Y1 = to_categorical(Y)
    X_train, X_test, y_train, y_test = train_test_split(X, Y1, test_size=0.2)#now X will
split to train and test and go for training
    cnn_model = Sequential()
    cnn_model.add(Dense(512, input_shape=(X_train.shape[1],)))
    cnn_model.add(Activation('relu'))
    cnn_model.add(Dropout(0.3))
    cnn_model.add(Dense(512))
 cnn_model.add(Activation('relu'))
    cnn_model.add(Dropout(0.3))
    cnn_model.add(Dense(3))
    cnn_model.add(Activation('softmax'))
    cnn_model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    acc_history = cnn_model.fit(X, Y1, epochs=10, validation_data=(X_test, y_test))
    for i in range(0,40):
 y_test[i] = 0
    prediction_data = cnn_model.predict(X_test)
    prediction_data = np.argmax(prediction_data, axis=1)
    y_test = np.argmax(y_test, axis=1)
    drba_bat_precision = precision_score(y_test, prediction_data,average='macro') *
100
    drba_bat_recall = recall_score(y_test, prediction_data,average='macro') * 100
    drba_bat_acc = accuracy_score(y_test,prediction_data)*100
    text.insert(END,"ANN Prediction Results\n")
    text.insert(END,"ANN BAT Precision : "+str(drba_bat_precision)+"\n")
    text.insert(END,"ANN BAT Recall : "+str(drba_bat_recall)+"\n")
    text.insert(END,"ANN BAT Accuracy : "+str(drba_bat_acc)+"\n")
    unique_test, counts_test = np.unique(y_test, return_counts=True)
    unique_pred, counts_pred = np.unique(prediction_data, return_counts=True)
    total = 0
```

```
    print(counts_pred)
    print(unique_pred)
    print(counts_test)
    print(counts_pred)
    for i in range(len(counts_pred)):
        if counts_pred[i] > counts_test[i]:
            temp = counts_pred[i]
            counts_pred[i] = counts_test[i]
            counts_test[i] = temp
        acc = counts_pred[i]/counts_test[i]
        text.insert(END,malware_name[i]+" : Accuracy = "+str(acc)+"\n")
    data = confusion_matrix(y_test, prediction_data)
    df_cm = pd.DataFrame(data, columns=np.unique(malware_name), index =
np.unique(malware_name))
    df_cm.index.name = 'Actual'
    df_cm.columns.name = 'Predicted'
 plt.figure(figsize = (10,8))
    sn.set(font_scale=0.8)#for label size
    sn.heatmap(df_cm, cmap="Reds", annot=True,annot_kws={"size": 12}, fmt='d')
    plt.show()
def predict():
    filename = filedialog.askopenfilename(initialdir = "images")
    text.delete('1.0', END)
    text.insert(END,filename+" loaded\n\n")
img = np.load(filename)
    img = img.ravel()
    #im2arr = img.reshape(1,img.shape)
    #print(im2arr.shape)
    im2arr = img[pos==1]
    print(im2arr.shape)
    temp = []
    temp.append(im2arr)
    temp = np.asarray(temp)
```

```
    print(temp.shape)
    preds = classifier_model.predict(temp)
    print(str(preds)+" "+str(np.argmax(preds)))
    predict = np.argmax(preds)
    text.insert(END,'Uploaded file contains malicious code from family :
'+malware_name[predict])
    img = np.load(filename)
    img = cv2.resize(img,(800,500))
    cv2.putText(img, 'Uploaded file contains malicious code from family :
'+malware_name[predict], (10, 25),  cv2.FONT_HERSHEY_SIMPLEX,0.7, (255, 0,
0), 2)
    cv2.imshow('Uploaded file contains malicious code from family :
'+malware_name[predict],img)
    cv2.waitKey(0)
def precisionGraph():
    height = [knn_precision,svm_precision,drba_precision,drba_bat_precision]
    bars = ('KNN Precision', 'SVM Precision','ANN PSO Precision','ANN BAT
Precision')
def recallGraph():
 height = [knn_recall,svm_recall,drba_recall,drba_bat_recall]
    bars = ('KNN Recall', 'SVM Recall','ANN PSO Recall','ANN BAT Recall')
    y_pos = np.arange(len(bars))
    plt.bar(y_pos,  height)
    plt.xticks(y_pos, bars)
    plt.show()
def accuracyGraph():
    height = [knn_acc,svm_acc,drba_acc,drba_bat_acc]
    bars = ('KNN Accuracy', 'SVM Accuracy','ANN PSO Accuracy','ANN BAT
Accuracy')
    y_pos = np.arange(len(bars))
    plt.bar(y_pos, height)
    plt.xticks(y_pos, bars)
    plt.show()
```

```
def malwareGraph():
    #height = [knn_acc,svm_acc,drba_acc]
    #bars = ('KNN Accuracy', 'SVM Accuracy','DRBA Accuracy')
    fig, ax = plt.subplots()
    y_pos = np.arange(len(graph_col))
    plt.bar(y_pos, graph_row)
    plt.xticks(y_pos, graph_col)
    ax.xaxis_date()
    fig.autofmt_xdate()
    plt.show()
font = ('times', 16, 'bold')
title = Label(main, text='Malware detection using Machine Learning & Performance
Evaluation')
title.config(bg='dark goldenrod', fg='white')
title.config(font=font)
title.config(height=3, width=120)
title.place(x=0,y=5)
font1 = ('times', 13, 'bold')
upload = Button(main, text="Upload Malware Dataset", command=upload)
upload.place(x=700,y=100)
upload.config(font=font1)
pathlabel = Label(main)
pathlabel.config(bg='lawn green', fg='white')
pathlabel.config(font=font1)
pathlabel.place(x=700,y=150)
predictButton = Button(main, text="Malware Family Graph",
command=malwareGraph)
predictButton.place(x=700,y=200)
predictButton.config(font=font1)

svmButton = Button(main, text="Run SVM Algorithm", command=SVM)
svmButton.place(x=700,y=250)
svmButton.config(font=font1)
```

```
knnButton = Button(main, text="Run KNN Algorithm", command=KNN)
knnButton.place(x=700,y=300)
knnButton.config(font=font1)
batButton = Button(main, text="Run ANN Algorithm with BAT",
command=ANNBAT)
batButton.place(x=700,y=350)
batButton.config(font=font1)
nbButton = Button(main, text="Run ANN Algorithm with PSO",
command=ANNPSO)
nbButton.place(x=700,y=400)
nbButton.config(font=font1)
treeButton = Button(main, text="Accuracy Graph", command=accuracyGraph)
treeButton.place(x=700,y=450)
treeButton.config(font=font1)
lrButton = Button(main, text="Precision Graph", command=precisionGraph)
lrButton.place(x=700,y=500)
lrButton.config(font=font1)
randomButton = Button(main, text="Recall Graph", command=recallGraph)
randomButton.place(x=700,y=550)
randomButton.config(font=font1)
predictButton = Button(main, text="Predict Malware from New File",
command=predict)
predictButton.place(x=700,y=600)
predictButton.config(font=font1)
font1 = ('times', 12, 'bold')
text=Text(main,height=30,width=80)
scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=10,y=100)
text.config(font=font1)

main.config(bg='RoyalBlue2')
main.mainloop()
```

5. SCREENSHOTS

5. SCREENSHOTS

To implement this project and to evaluate machine learning algorithms performance author is using binary malware dataset called ‘MALIMG’. This dataset contains 25 families of malware and application will convert this binary dataset into gray images to generate train and test models for machine learning algorithms. These algorithms converting binary data to images and then generating model, so they are called as MalConv CNN and MalConv LSTM and other algorithm refers as EMBER. Application convert dataset into binary images and then used 80% dataset for training model and 20% dataset for testing. Whenever we upload new test malware binary data then the application will apply new test data on train model to predict malware class. we can see and below are their names.

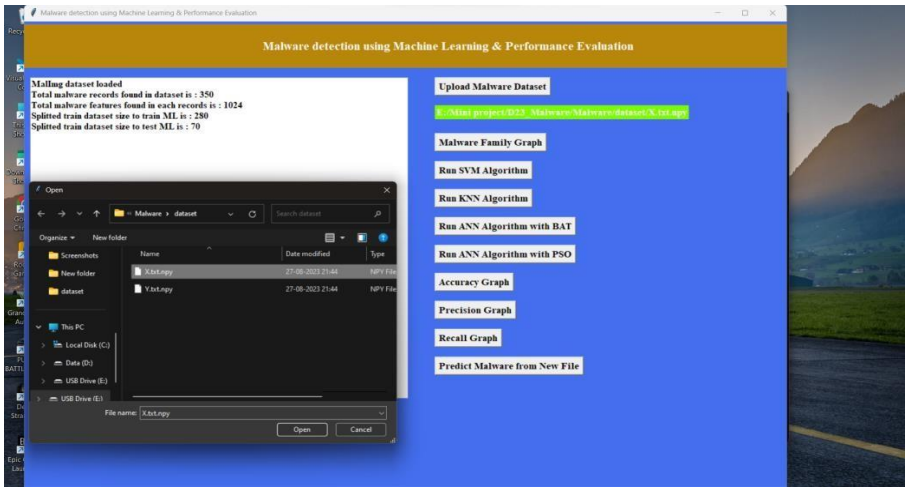
'Dialer Adialer. C', 'Backdoor Agent. FYI', 'Worm Allaple.A', 'Worm Allaple. L', 'Trojan Alueron. gen', 'Worm: AutoIT Autorun. K', 'Trojan C2Lop.P', 'Trojan C2Lop.gen', 'Dialer Dialplatform. B', 'Trojan Downloader Dontovo.A', 'Rogue Fakerean', 'Dialer Instantaccess', 'PWS Lolyda. AA 1', 'PWS Lolyda.AA 2', 'PWS Lolyda.AA3', 'PWS Lolyda.AT', 'TrojanMalex.gen','Trojan Downloader Obfuscator.AD', 'Backdoor Rbot!gen', 'Trojan Skintrim.N','Trojan Downloader Swizzor.gen! E', 'Trojan Downloader Swizzor.gen!I','Worm VB.AT', 'Trojan Downloader Wintrim. BX', 'Worm Yuner. A'.

All above names are the malware families.

All new malware test files I saved inside images folder, and you can upload these files to predict it class. All algorithms detail you can read from paper.

Step 1: Run Program and Upload Dataset.

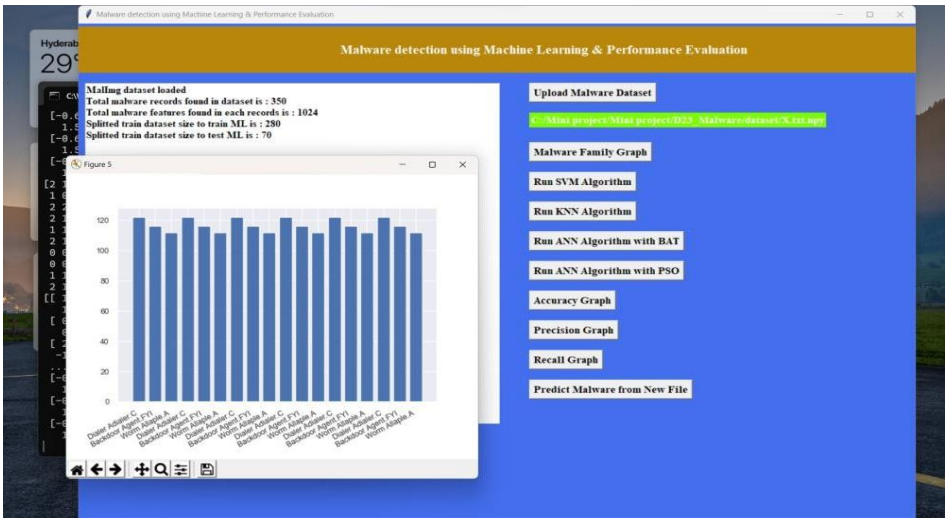
- After opening click on ‘UPLOAD MALWARE MALIMG DATASET’ buttonto upload dataset. In above screen, I am uploading ‘X.txt.npy or Y.txt.npy’ binary malware dataset and after uploading dataset will get below screen.



Screenshot 5.1: Upload Malware MalImg Dataset

Step 2: Click on malware family graphs.

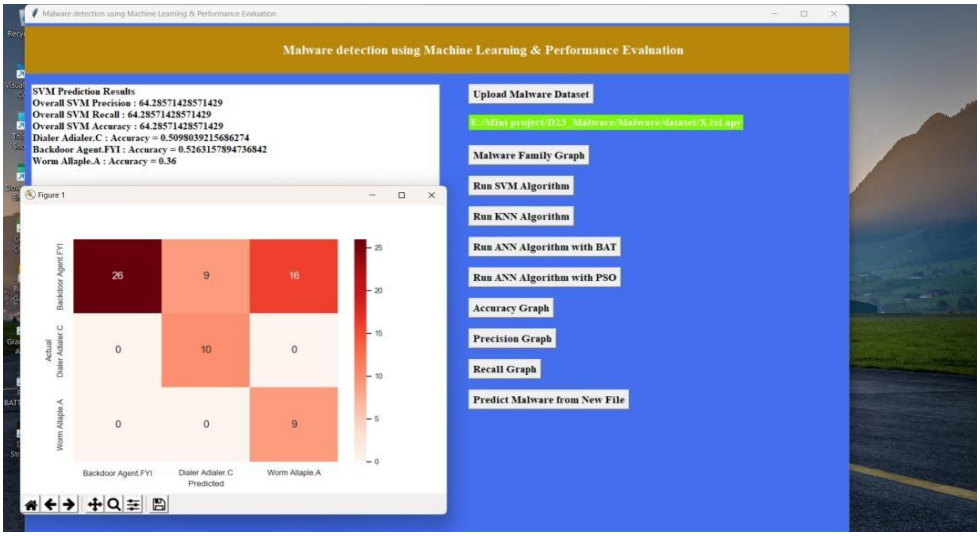
- ‘MALWARE FAMILY GRAPHS’ Can Be Used to Detect new Malware Samples By Comparing Them to Known Malware Families.



Screenshot 5.2: Malware Family Graphs

Step 3: Click on Run SVM Algorithm.

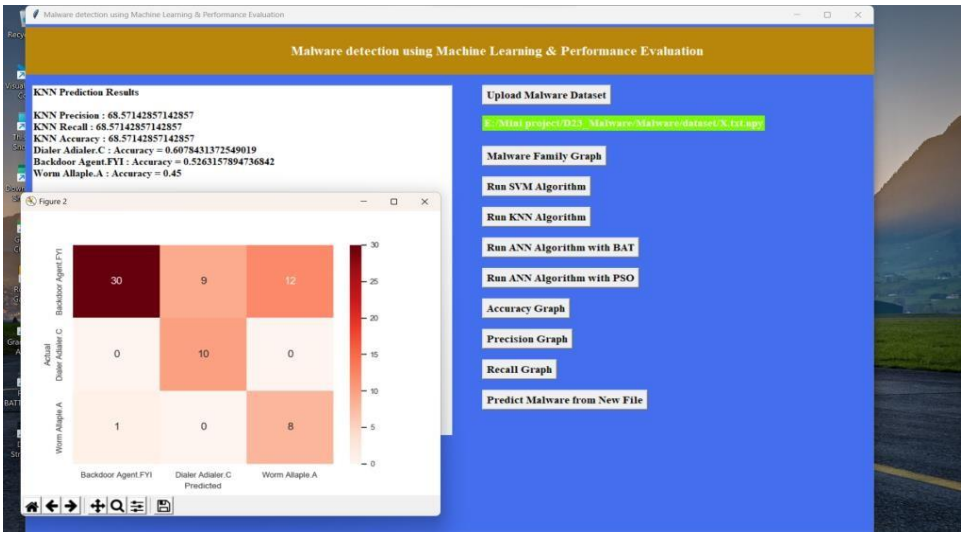
- The 'RUN SVM ALGORITHM' to Read Malware Dataset Generate Train and Test Model and Then Apply SVM Algorithm to Calculate ItsPrediction Accuracy, FSCORE, Precision and Recall.



Screenshot 5.3: Run SVM Algorithm

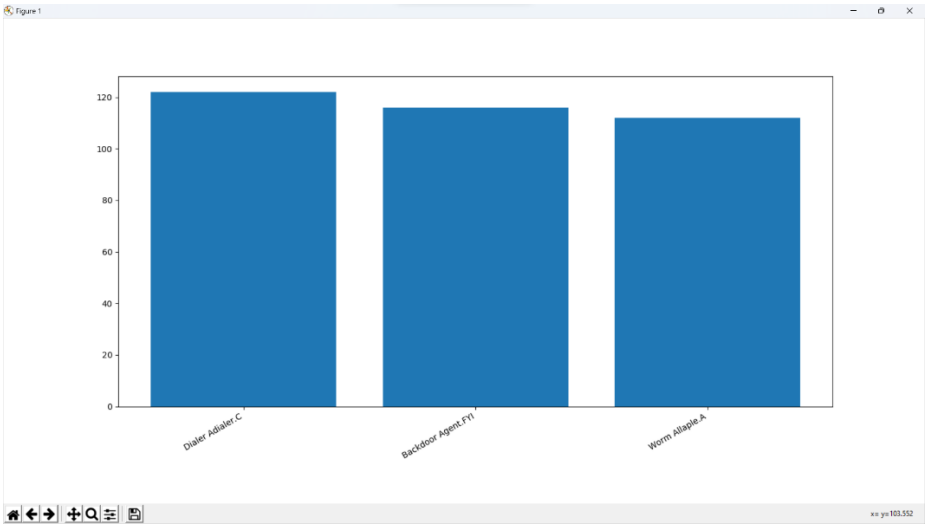
Step 4: Click on Run KNN Algorithm.

- The 'RUN KNN ALGORITHM' to Get Its Performance



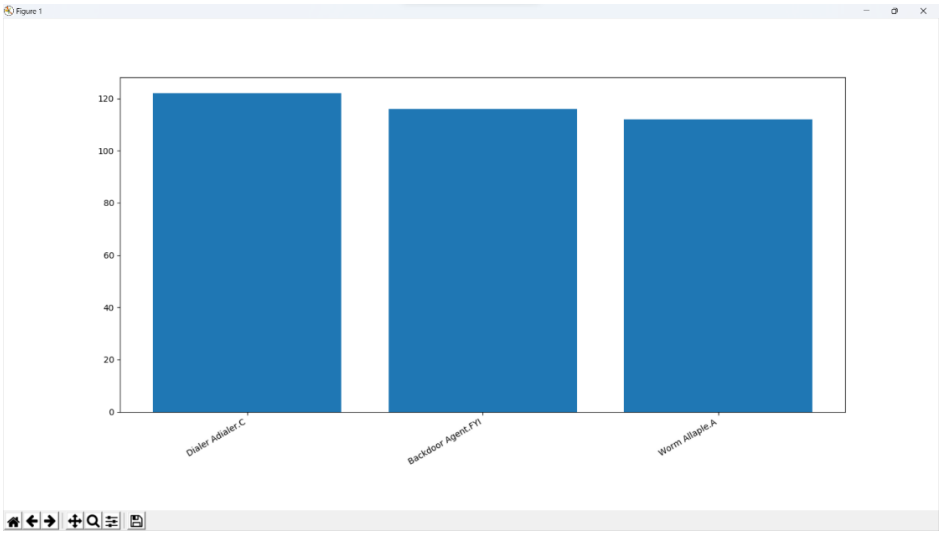
Screenshot 5.4: Run KNN Algorithm

Step 5: Click on Accuracy Graph.



Screenshot 5.5: Accuracy Graphs

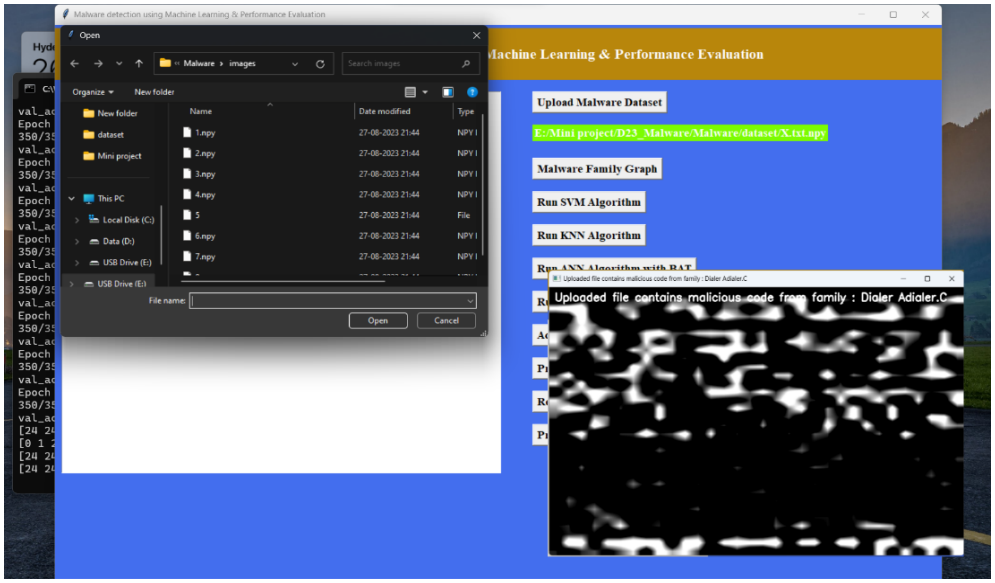
Step 6: Click on Precision Graph.



Screenshot 5.6: Precision Graphs

Step 7: Click on Predict Malware from New File.

- Upload One Binary File Called 1.Npy and Below Is the Malware Prediction of That File. In Above Screen, We Can See Uploaded Test File Contains ‘Dialer Adialer.C’ Malware Attack. Similarly, We Can UploadOther Files and Predict Class.



Screenshot 5.7: Predict Malware from New File

6. TESTING

6. TESTING

6.1 INTRODUCTION TO TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

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 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 run as one program. Integration tests demonstrate that although the components were individually satisfactory, as shown by successful 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.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked. Organization and preparation of functional tests are focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identifying Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

SYSTEM TEST

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

6.2.4 WHITE BOX TESTING

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

6.2.5 BLACK BOX TESTING

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, like most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated as a black box You cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

TEST STRATEGY AND APPROACH

Field testing will be performed manually, and functional tests will be writtenin detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested.

- Verify that the entries are in the correct format.
- No duplicate entries should be allowed.
- All links should take the user to the correct page.

6.2.6 ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

6.3 TEST CASES

6.3.1 CLASSIFICATION

S.NO	Test Case	Excepted Result	Result	Remarks(IF fails)
1.	User Register	If User registration successfully.	Pass	If already user email exist then it fails.
2.	User Login	If Username and password is correct then it will getting valid page.	Pass	Un Register Users will not logged in.
3.	User View User	Show our dataset	Pass	If Data set Not Available fail.
4.	View Fast History Results	The Four Alarm Score Should be Displayed.	Pass	The Four Alarm Score Not Displaying fail
5.	User Prediction	Display Review with true results	Pass	Results not True Fail
6.	Show Detection process	Display Detection process	Pass	Results Not True Fail
7.	Show Eye Blink Process	Display Eye Blink Process	Pass	If Results not Displayed Fail.
8.	Admin Login	Admin can login with his login credential. If success he gets his home page	Pass	Invalid login details will not allowed here
9.	Admin can activate the register users	Admin can activate the register user id	Pass	If user id not found then it won't login
10.	Results	For our Four models the accuracy and F1 Score	Pass	If Accuracy andF1 Score Not Displayed fail

7. CONCLUSION

7. CONCLUSION AND FUTURE SCOPE

7.1 PROJECT CONCLUSION

Deep learning is an awesome way to detect malware with precision. It can detect both familiar and unfamiliar threats, plus it can recognize patterns in the data that we might not catch with a human eye.

Deep learning might have some limitations and can give off false positives, but it's still a great choice for detecting malicious activity. Definitely worth considering when creating a security plan.

7.2 FUTURE SCOPE

This project proposed an efficient malware detection and designed a highly scalable framework to detect, classify and categorize zero-day malwares. This framework applies neural network on the collected malware from end user hosts and follows a two-stage process for malware analysis. first stage, a hybrid of static and dynamic analysis was applied for malware classification. In the second stage, malware was grouped into corresponding malware categories using imageprocessing approaches. Various experimental analysis conducted by applying variations in the models on publicly available benchmark dataset and indicated the proposed model outperformed classical MLAs. The developed framework can analyze large number of malwares in real-time and scaled out to analyze even larger number of malwares by stacking a few more layers to the existing architectures. Future research entails exploration of these variations with new features that could be added to the existing data.

8. BIBLIOGRAPHY

8. BIBLIOGRAPHY

8.1 REFERENCES

[1] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, ‘‘Malware images: Visualization and automatic classification,’’ in Proc. 8th Int. Symp. Vis. Cyber Secure. New York, NY, USA: ACM, Jul. 2011, p. 4.

[2] Alazab, M. Venkatraman, S. Watters, & P. Alazab, (2011, December). Zero-day malware detection based on supervised learning algorithms of API call signatures. In Proceedings of the Ninth Australasian Data Mining Conference-Volume 121 (pp. 171-182). Australian Computer Society, Inc...

[3] B. Li, K. Roundy, C. Gates, and Vorobeychik, ‘Large-scale identification of malicious singleton files,’ in Proc. 7th ACM Conf. Data Appl. Secure. Privacy. New York, NY, USA: ACM, Mar. 2017, pp. 227–238.

[4] Y. LeCun, Y. Bengio, and G. Hinton, ‘‘Deep learning,’ Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[5] A. F.Agarap and F. J. H. Pepito. (2017). ‘‘Towards building an intelligent antimalware system: A deep learning approach using support vector machine (SVM) for malware classification.’’ [Online].

Available: https://arxiv.org/abs/1801.00318

8.2 GITHUB LINK

https://github.com/SugamanchiNarender/ROBUST-INTELLIGENT-MALWARE-DETECTION-USING-DEEP-LEARNING