Configuration Manual

MSc Research Project
MSc In Cybersecurity

Harisankar Kalathil Salim
Student ID: x23151552

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

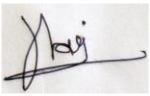
MSc Project Submission Sheet

School of Computing

Student Name:	Harisankar Kalathil Salim
Student ID:	X23151552
Programme:	MSc In Cybersecurity **Year:** 2023-2024
Module:	MSc in Research Project
Lecturer:	Vikas Sahni
Submission Due Date:	12-8-2024
Project Title:	Deep Learning-based Android Malware Detection with CNN-GRU Model

Word Count: **892** **Page Count:** **12**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 

Date: 12-08-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Harisankar Kalathil Salim
Student ID:x23151552

1 Introduction

The configuration manual details the tools and technologies used throughout the research implementation. Section 2 thoroughly describes the experimental setup. In Section 3, the various technologies and software tools employed are discussed. Section 4 provides a comprehensive, step-by-step guide for executing the machine learning workflow, beginning with importing the necessary libraries. It includes loading and pre-processing the dataset, followed by selecting key features from the pre-processed data. The section also covers balancing the class counts in the dataset. It covers splitting the dataset into training and testing sets, developing the CNN-GRU model architecture, and training the model with the prepared data. It addresses the performance evaluation of the trained model. Section 5 concludes with references for the software guide.

2 Experimental Setup

The experiment was conducted on a personal computer configured specifically for this purpose.

- **Hardware Specifications**: The system runs on a fifth-generation Intel i7 processor, has 24GB of RAM, and includes a 252GB SSD.
- **Operating System**: Windows 11.
- **Experimental Setup**: The setup includes Windows 11, Anaconda3 2023.07-2, Python 3.9, and Visual Studio Code.
- **Google Colab pro**: 300 Unit paid units were bought for model training purposes

3 Technologies and Software Used for Implementation

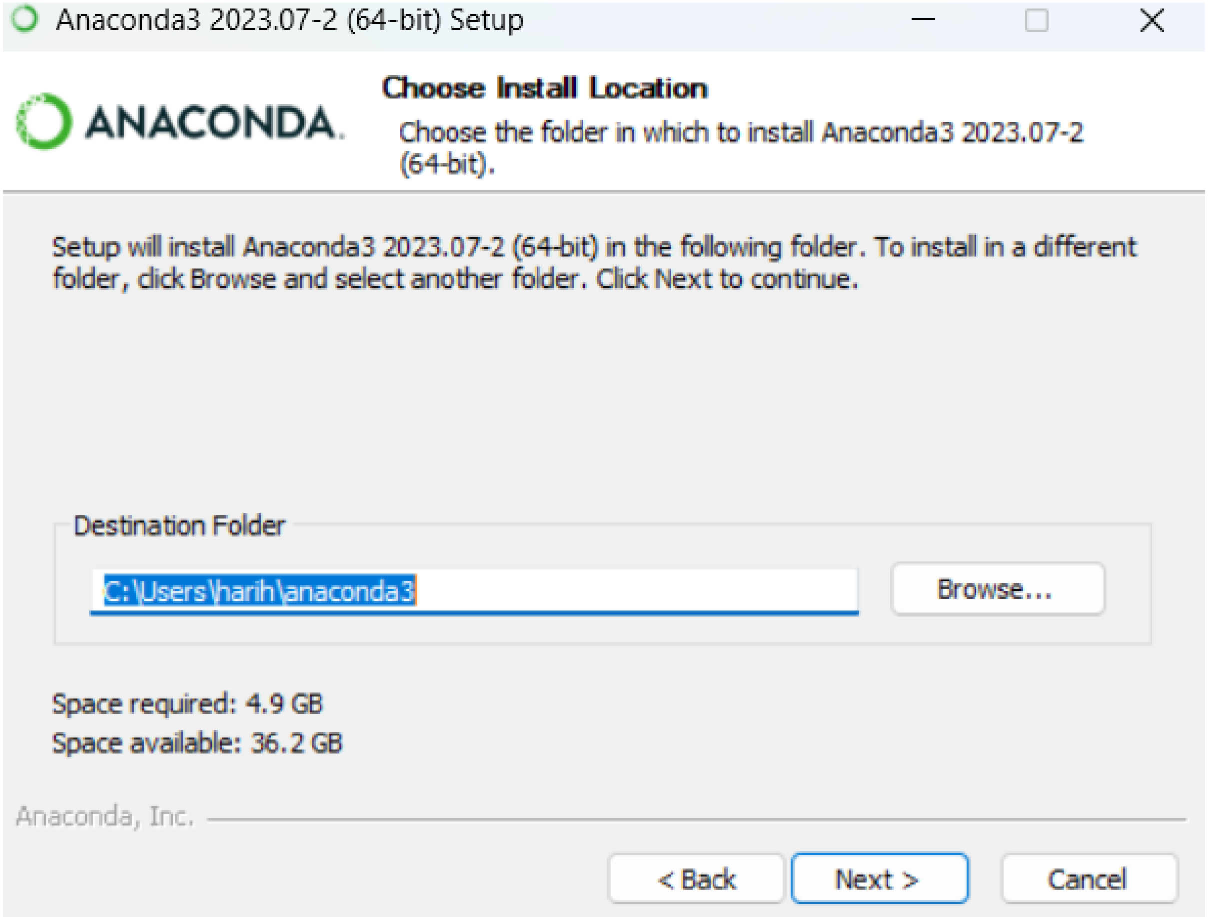
- **Software Used:** VS Code, Anaconda3 2023.07-2[1], Python 3.9

- **Anaconda3** is an open-source distribution of Python and R, specifically designed for scientific computing (docs.anaconda.com, n.d.). It simplifies package management and deployment with its Conda package manager and supports creating isolated environments for different projects, ensuring compatibility and ease of use. **Python 3.9** is a high-level, interpreted programming language celebrated for its readability and versatility (Mészárosová, 2015). This version includes new syntax features, performance enhancements, and updates to the standard library, making it ideal for a wide range of applications, from web development to data science. VS Code is a free and open-source code editor created by Microsoft. It is well-known for its powerful

[1] https://docs.anaconda.com/.

features, including debugging, syntax highlighting, intelligent code completion, and an extensive collection of extensions. These features make it highly customizable and adaptable, allowing it to support numerous programming languages and projects.

4 installation of anaconda



4 Implementation

Step 1: Import the required libraries for the implementation.

```python
import pandas as pd
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pickle
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GRU, Dense, Flatten
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GRU, Dense, Flatten
from tensorflow.keras.layers import BatchNormalization, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint
```

Figure (1): Import libraries for the implementation

Step 2: Load the dataset and perform pre-processing steps.

```python
dataset = pd.read_csv("drebin.csv")

print("Dataset shape:", dataset.shape)
print(dataset.head())

print(dataset.tail())

columns_with_no_data = dataset.columns[dataset.isnull().all()]

print("Columns with no data:", columns_with_no_data)

print(dataset.info())

print(dataset.isnull().sum())

dataset.drop(dataset.tail(1).index, inplace=True)

print(dataset.tail())

for column in dataset.columns:
    null_count = dataset[column].isnull().sum()
    if null_count > 0:
        print(f"Column '{column}' has {null_count} null values.")

target_count = dataset['class'].value_counts()

print(target_count)

columns_with_question_mark = []
for column in dataset.columns:
    if dataset[column].astype(str).str.contains('\?').any():
        columns_with_question_mark.append(column)

print("Columns containing '?':")

print(columns_with_question_mark)

dataset.drop(columns=['TelephonyManager.getSimCountryIso'], inplace=True)

print(dataset.head())

label_encoder = LabelEncoder()

y_encoded = label_encoder.fit_transform(dataset['class'])

dataset['class'] = y_encoded
print(dataset)
```

Figure (2): Loading Data and Pre-processing

Step 3: Select the important features from the pre-processed dataset using the Chi-2 algorithm

```python
X = dataset.drop('class', axis=1)
y = dataset['class']

best_features = SelectKBest(score_func=chi2, k=150)
fit = best_features.fit(X, y)

top_feature_indices = fit.get_support(indices=True)

scores = fit.scores_[top_feature_indices]
p_values = fit.pvalues_[top_feature_indices]
selected_features = X.iloc[:, top_feature_indices]

feature_scores = pd.DataFrame({'Feature': X.columns[top_feature_indices], 'Score': scores, 'P-Value': p_values})

print(feature_scores)

print(selected_features)
```

Figure (3): Feature selection using the Chi-2 Algorithm

Step 4: Balance the count of each class in the dataset using the SMOTE algorithm

```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(selected_features, y)

print("Class count before balancing:")
print(target_count)

print("Class count after balancing:")
print(pd.Series(y_resampled).value_counts())

balanced_dataset = pd.DataFrame(X_resampled, columns=selected_features.columns)
balanced_dataset['class'] = y_resampled

balanced_dataset.to_csv("drebin_balanced.csv", index=False)

print("Balanced dataset saved to 'drebin_balanced.csv'")
```

Figure (4): Data Balancing using the SMOTE Algorithm

Step 5: Divide the dataset into training and testing subsets.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure (5): Train and Test Split

Step 6: Develop the architecture for the CNN-GRU model.

```
# Build the model
model = Sequential()

# CNN layers
model.add(Conv1D(filters=256, kernel_size=3, activation='relu', input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2])))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(BatchNormalization())
model.add(Dropout(0.3))

#  GRU layers
model.add(GRU(128, return_sequences=True, dropout=0.3, recurrent_dropout=0.3))
model.add(GRU(64, dropout=0.3, recurrent_dropout=0.3))

# Dense layers
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.summary()
```

Figure (6): CNN-Gru Model Architecture

Step 7: Train and save the constructed model using the dataset.

```
checkpoint_path = "best_model.h5"
checkpoint_callback = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

history = model.fit(X_train_reshaped, y_train, epochs=200, batch_size=32, validation_data=(X_test_reshaped, y_test), callbacks=[checkpoint_callback])
```

Figure (7): Train and Save Trained Model

Step 8: Evaluate the performance of the model.

```python
best_model = tf.keras.models.load_model(checkpoint_path)
loss, accuracy = best_model.evaluate(X_test_reshaped, y_test)
print(f'Accuracy: {accuracy}, Loss: {loss}')

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

plt.tight_layout()
plt.show()

# Confusion matrix
y_pred = model.predict_classes(X_test)
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)
```

Figure (8): Performance Evaluation

Step 9: Import necessary libraries and define models path for the gui.

```python
import tkinter as tk
from tkinter import ttk
import pandas as pd
import numpy as np
import pickle
from tensorflow.keras.models import load_model

# Define file paths
MODEL_PATH = "model.h5"
LABEL_ENCODER_PATH = "label_encoder.pkl"
FEATURE_SELECTOR_PATH = "chi2_selector.pkl"
column_names=['transact', 'onServiceConnected', 'bindService', 'attachInterface', 'ServiceConnection',
            'android.os.Binder', 'SEND_SMS', 'Ljava.lang.Class.getCanonicalName', 'Ljava.lang.Class.getMethods',
            'Ljava.lang.Class.cast', 'Ljava.net.URLDecoder', 'android.content.pm.Signature', 'android.telephony.SmsManager',
            'READ_PHONE_STATE', 'getBinder', 'ClassLoader', 'Landroid.content.Context.registerReceiver',
            'Ljava.lang.Class.getField', 'Landroid.content.Context.unregisterReceiver', 'GET_ACCOUNTS', 'RECEIVE_SMS',
            'Ljava.lang.Class.getDeclaredField', 'READ_SMS', 'getCallingUid', 'Ljavax.crypto.spec.SecretKeySpec',
            'android.intent.action.BOOT_COMPLETED', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'android.content.pm.PackageInfo',
            'KeySpec', 'TelephonyManager.getLine1Number', 'DexClassLoader', 'HttpGet.init', 'SecretKey',
            'Ljava.lang.Class.getMethod', 'System.loadLibrary', 'android.intent.action.SEND', 'Ljavax.crypto.Cipher',
            'WRITE_SMS', 'READ_SYNC_SETTINGS', 'AUTHENTICATE_ACCOUNTS', 'android.telephony.gsm.SmsManager',
            'WRITE_HISTORY_BOOKMARKS', 'TelephonyManager.getSubscriberId', 'mount', 'INSTALL_PACKAGES',
            'Runtime.getRuntime', 'CAMERA', 'Ljava.lang.Object.getClass', 'WRITE_SYNC_SETTINGS',
            'READ_HISTORY_BOOKMARKS', 'Ljava.lang.Class.forName', 'INTERNET', 'android.intent.action.PACKAGE_REPLACED',
            'Binder', 'android.intent.action.SEND_MULTIPLE', 'RECORD_AUDIO', 'IBinder', 'android.os.IBinder',
            'createSubprocess', 'NFC', 'ACCESS_LOCATION_EXTRA_COMMANDS', 'URLClassLoader', 'WRITE_APN_SETTINGS',
            'abortBroadcast', 'BIND_REMOTEVIEWS', 'android.intent.action.TIME_SET', 'READ_PROFILE',
            'TelephonyManager.getDeviceId', 'MODIFY_AUDIO_SETTINGS', 'getCallingPid', 'READ_SYNC_STATS',
            'BROADCAST_STICKY', 'android.intent.action.PACKAGE_REMOVED', 'android.intent.action.TIMEZONE_CHANGED',
```

Figure (9): Importing libraries and setting up path

Step 10: Create the GUI and add functionality.

```python
def on_predict():
    # Clear previous results
    text_box.delete(1.0, tk.END)



    if model is not None and label_encoder is not None and feature_selector is not None:

        input_features_csv = entry1.get()

        # Parse the input features
        input_features = list(map(int, input_features_csv.split(",")))
        # Make predictions
        predict_malware(input_features, model, label_encoder, feature_selector, column_names, text_box)
    else:
        text_box.insert(tk.END, "Failed to load necessary components.")

gui=tk.Tk()
gui.geometry("900x600")
gui.resizable(0,0)
gui.configure(background="orange")
gui.title("Android Malware Detector")

# Create a Notebook widget
notebook = ttk.Notebook(gui)
notebook.pack(fill='both', expand=True)
```

Figure (10): Defining gui

Step 11: Load the model and preprocessing tools.

```python
# Load the saved model
def load_saved_model(model_path):
    try:
        model = load_model(model_path)
        return model
    except Exception as e:
        print("Error loading model:", str(e))
        return None

# Load the label encoder
def load_label_encoder(label_encoder_path):
    try:
        with open(label_encoder_path, "rb") as file:
            label_encoder = pickle.load(file)
        return label_encoder
    except Exception as e:
        print("Error loading label encoder:", str(e))
        return None

# Load the chi2 feature selector
def load_feature_selector(feature_selector_path):
    try:
        with open(feature_selector_path, "rb") as file:
            feature_selector = pickle.load(file)
        return feature_selector
    except Exception as e:
        print("Error loading feature selector:", str(e))
        return None
```

Figure (11): Functions for loading the models and prepocessing tools

Step 12: Define functions for prediction.

```python
# Define a function to make predictions
def predict_malware(input_features, model, label_encoder, feature_selector, column_names, text_box):
    try:
        # Preprocess input features
        input_features_processed = preprocess_features(input_features, column_names, feature_selector)

        if input_features_processed is not None:
            # Get the selected feature indices
            selected_feature_indices = feature_selector.get_support(indices=True)

            # Get the selected feature names
            selected_feature_names = [column_names[i] for i in selected_feature_indices]

            # Display selected features in the text box
            text_box.insert(tk.END, "\n\nSelected Features:\n")
            for feature in selected_feature_names:
                text_box.insert(tk.END, feature + "\n")

            # Reshape the input features for model prediction
            input_features_reshaped = input_features_processed.reshape((1, input_features_processed.shape[1], 1)
            text_box.insert(tk.END, "\n\nreshaped input features:\n")
            text_box.insert(tk.END, input_features_reshaped)
            text_box.insert(tk.END, "\n\nShape of reshaped input features :\n")
            text_box.insert(tk.END, str(input_features_reshaped.shape))

            # Make predictions using the loaded model
            predictions = model.predict(input_features_reshaped)

            text_box.insert(tk.END, "\n\nprobability value of prediction :\t")
            text_box.insert(tk.END, predictions)
```

Figure (12): Function for model prediction

Step 13: Compiling the GUI.

Figure (13): Home page

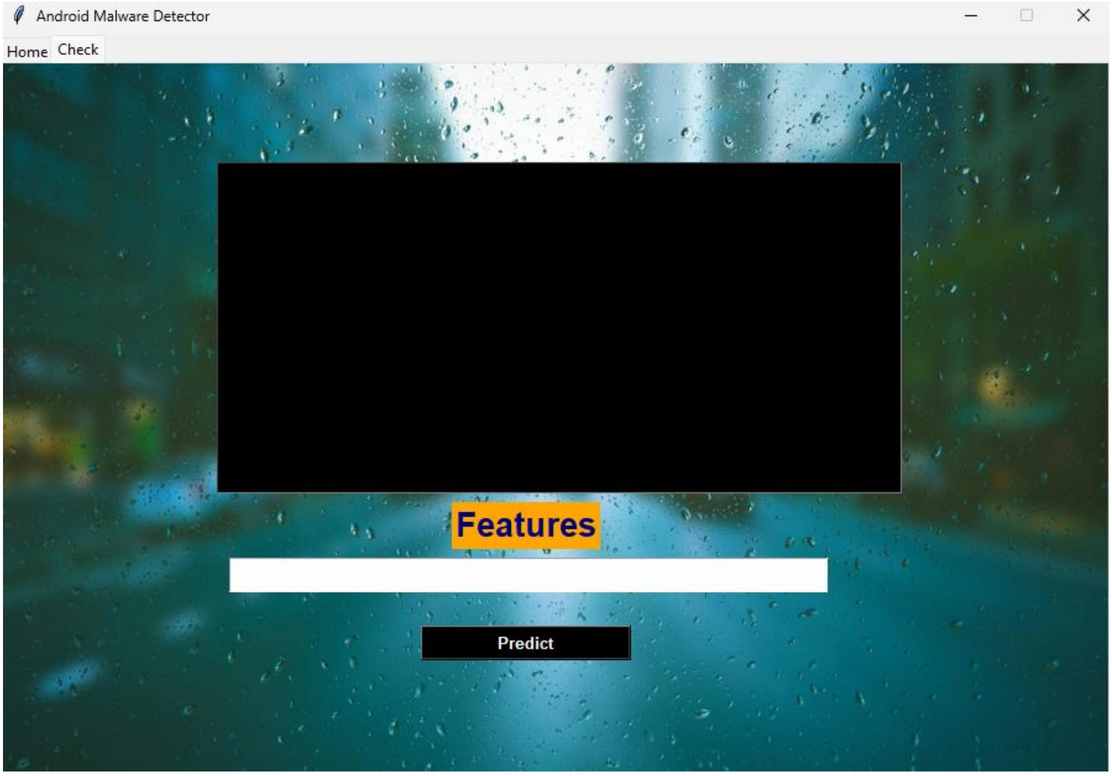


Figure (14): Prediction page

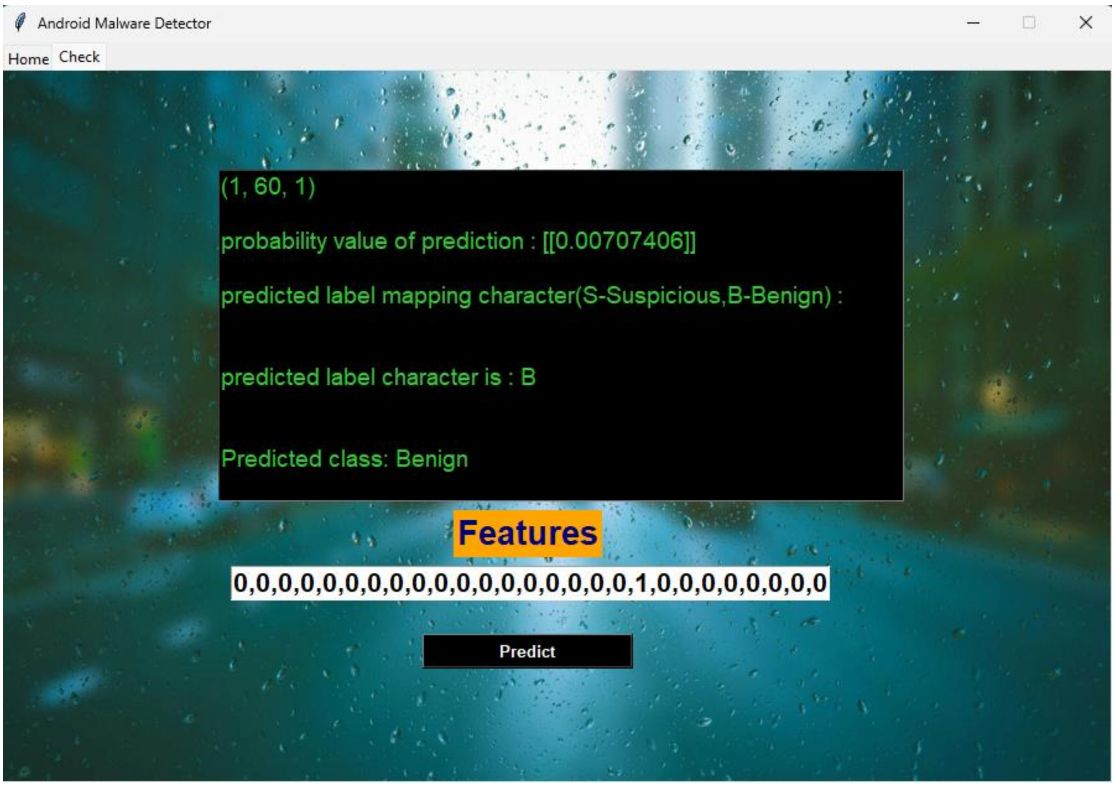


Figure (14): Prediction results(Suspicious test data)

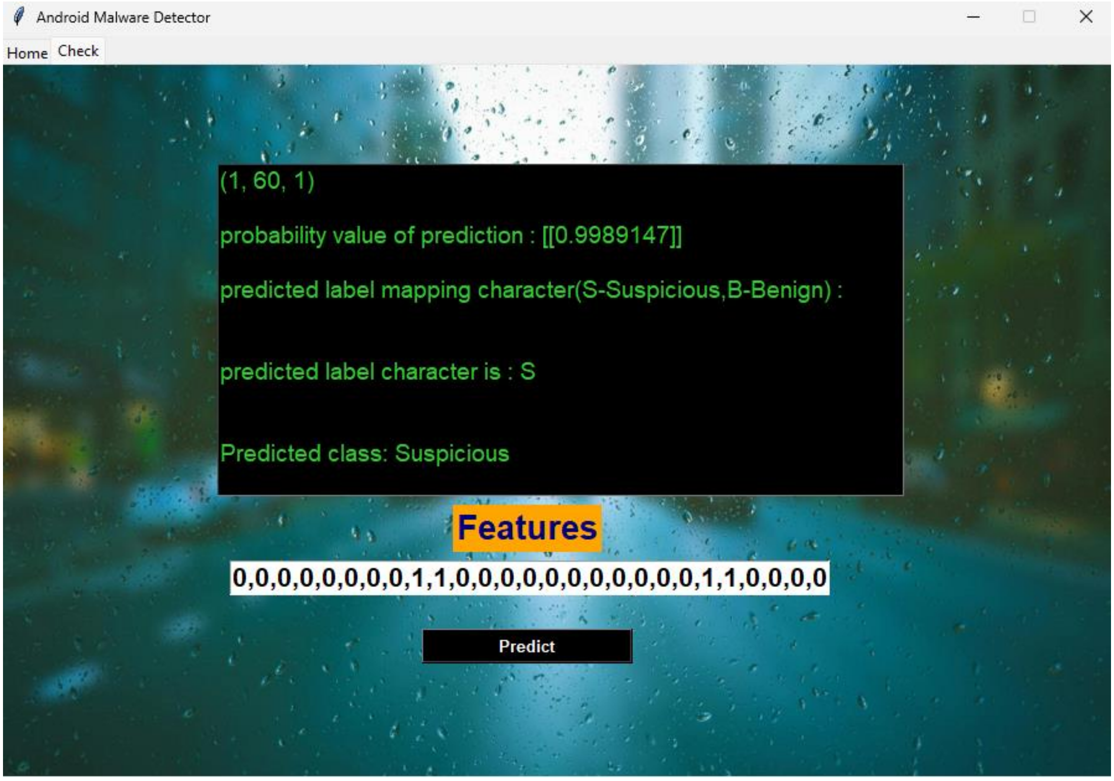


Figure (15): Prediction results(Normal test data)

1 References

Mészárosová, E. (2015). Is Python an Appropriate Programming Language for Teaching Programming in Secondary Schools? International Journal of Information and Communication Technologies in Education, 4(2), pp.5–14. doi:https://doi.org/10.1515/ijicte-2015-0005.