Malware Detection Framework Using Hybrid Deep Learning Algorithms

Configuration Manual

Research Project
M.Sc. Cybersecurity

Ayaan Khan
Student ID: X21199728

School of Computing
National College of Ireland

Supervisor: Mr. Michael Pantridge

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ayaan Khan		
Student ID:	21199728		
Programme:	MSc. cybersecurity	**Year:**	2022-2023
Module:	Research project configuration manual		
Lecturer:	Mr. Michael Pantridge		
Submission Due Date:	18/09/2022		
Project Title:	Malware Detection Framework Using Hybrid Deep Learning Algorithms		
Word Count:	1148	**Page Count:**	7

Signature: Ayaan Khan

Date: 18/09/2022

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

Ayaan Khan
x21199728

1 Introduction

The present guidebook encompasses the essential guidelines and necessary equipment needed for the successful execution of the job at hand. The objective of this study is to develop a model that utilizes machine learning and ensemble learning methods for the purpose of detecting a specific type of intrusion, namely a remote access trojan (RAT), on Android devices. The inclusion of the configuration manual is of utmost importance as it encompasses all essential elements such as software, hardware, and implementation methodologies required for the development of this project.

2 System Requirements

To optimize the efficiency of the model and save processing time, it is necessary to utilize the following hardware and software components:

Hardware Requirement's

The operating system in question is Windows 11. The Random Access Memory (RAM) of the system is 16.0 gigabytes (GB).

The system is equipped with an 11th generation Intel Core i5-11320H processor, operating at a base frequency of 3.20GHz and a maximum turbo frequency of 2.50GHz. Additionally, it features a storage capacity of 512 GB in the form of a solid-state drive (SSD).

The system in question is characterized by a 64-bit operating system and a processor based on the x64 architecture.

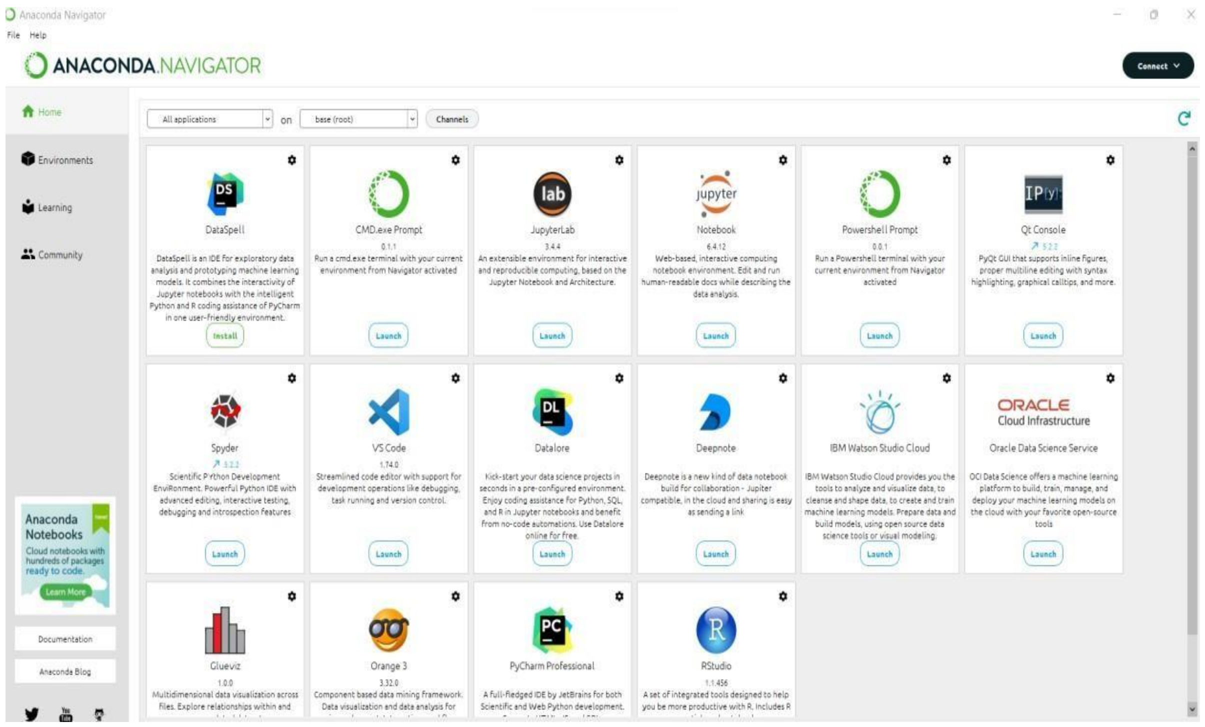
Software Requirements

The software tools utilized in this study are Anaconda Navigator, Jupyter Notebook, and Python version 3.11.1.

The programming language employed in this study was Python, and the research utilized Jupyter Notebook as the integrated development environment (IDE) of choice. Various Python libraries were employed for the purpose of analysis and visualization, which will be further upon in the subsequent discussion.

The use of a complete software suite known as Anaconda Navigator, which includes Jupyter Notebook and the requisite Python configuration for code execution, was employed. The Anaconda 64-bit version has been successfully installed on a Windows 11 operating system. Following a successful installation, it is necessary to initiate the launch of Jupyter Notebook from the navigator.

https://www.anaconda.com/products/distribution



DataSpell
DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.

CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated

JupyterLab
3.4.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
6.4.12
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated

Qt Console
↗ 5.2.2
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
↗ 5.2.2
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

VS Code
1.74.0
Streamlined code editor with support for development operations like debugging, task running and version control.

Datalore
Kick-start your data science projects in seconds in a pre-configured environment. Enjoy coding assistance for Python, SQL, and R in Jupyter notebooks and benefit from no-code automations. Use Datalore online for free.

Deepnote
Deepnote is a new kind of data notebook build for collaboration - Jupiter compatible, in the cloud and sharing is easy as sending a link

IBM Watson Studio Cloud
IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

Oracle Data Science Service
OCI Data Science offers a machine learning platform to build, train, manage, and deploy your machine learning models on the cloud with your favorite open-source tools

Glueviz
1.0.0
Multidimensional data visualization across files. Explore relationships within and

Orange 3
3.32.0
Component based data mining Framework. Data visualization and data analysis for

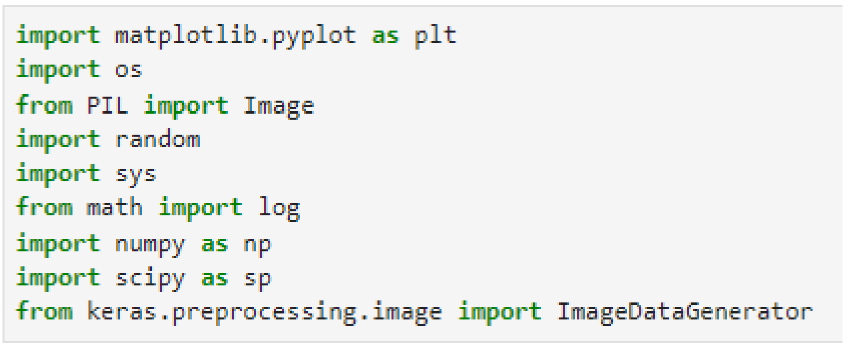
PyCharm Professional
A full-fledged IDE by JetBrains for both Scientific and Web Python development.

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R

3 Description of Dataset and Data preprocessing

This section encompasses the many procedures involved in data preparation for the machine learning model.

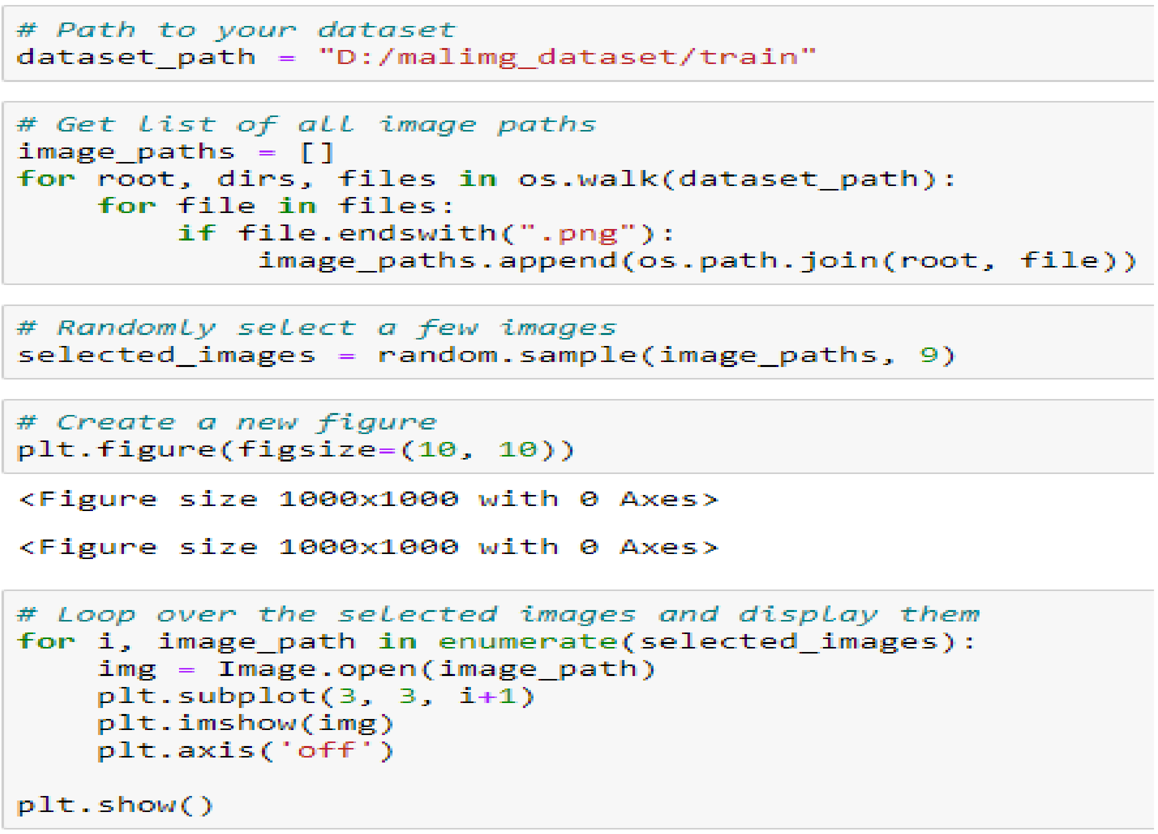
Importing Libraries



```python
import matplotlib.pyplot as plt
import os
from PIL import Image
import random
import sys
from math import log
import numpy as np
import scipy as sp
from keras.preprocessing.image import ImageDataGenerator
```

Importing the required libraries is the initial step in kicking off the data preparation and visualization process. The libraries covered in this set are os, Matplotlib, PIL, random, sys, and NumPy. Scipy provides a diverse range of algorithms and tools for the purpose of partitioning and transforming data.

Data Importing and processing.



```python
# Path to your dataset
dataset_path = "D:/malimg_dataset/train"
```

```python
# Get list of all image paths
image_paths = []
for root, dirs, files in os.walk(dataset_path):
    for file in files:
        if file.endswith(".png"):
            image_paths.append(os.path.join(root, file))
```

```python
# Randomly select a few images
selected_images = random.sample(image_paths, 9)
```

```python
# Create a new figure
plt.figure(figsize=(10, 10))
```

```
<Figure size 1000x1000 with 0 Axes>

<Figure size 1000x1000 with 0 Axes>
```

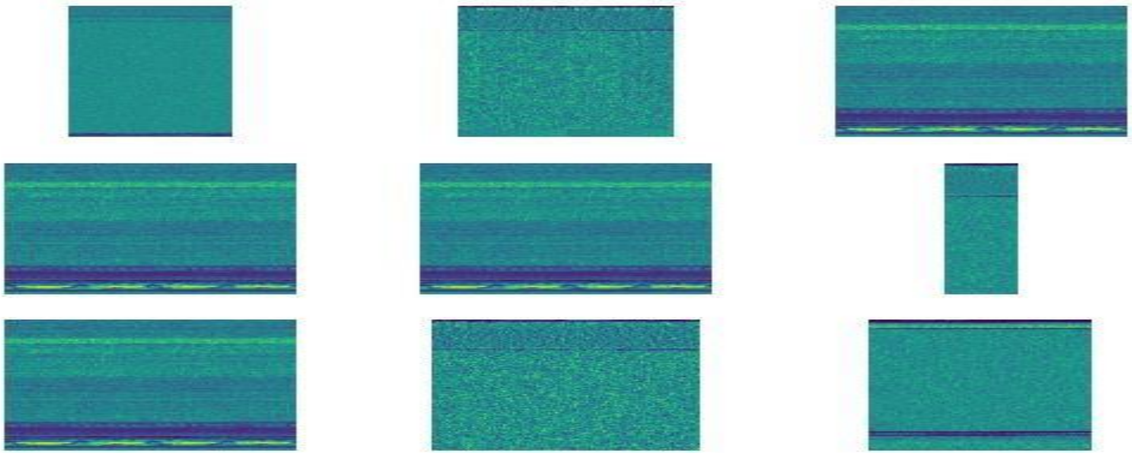
```python
# Loop over the selected images and display them
for i, image_path in enumerate(selected_images):
    img = Image.open(image_path)
    plt.subplot(3, 3, i+1)
    plt.imshow(img)
    plt.axis('off')

plt.show()
```

https://www.kaggle.com/code/mustafahamed/malimg-cnn/input

Before analyzing the data, it is necessary to lead the malimg_dataset/train into the operating system's data frame. The head () method may be utilized to display all the columns.

Data Visualization

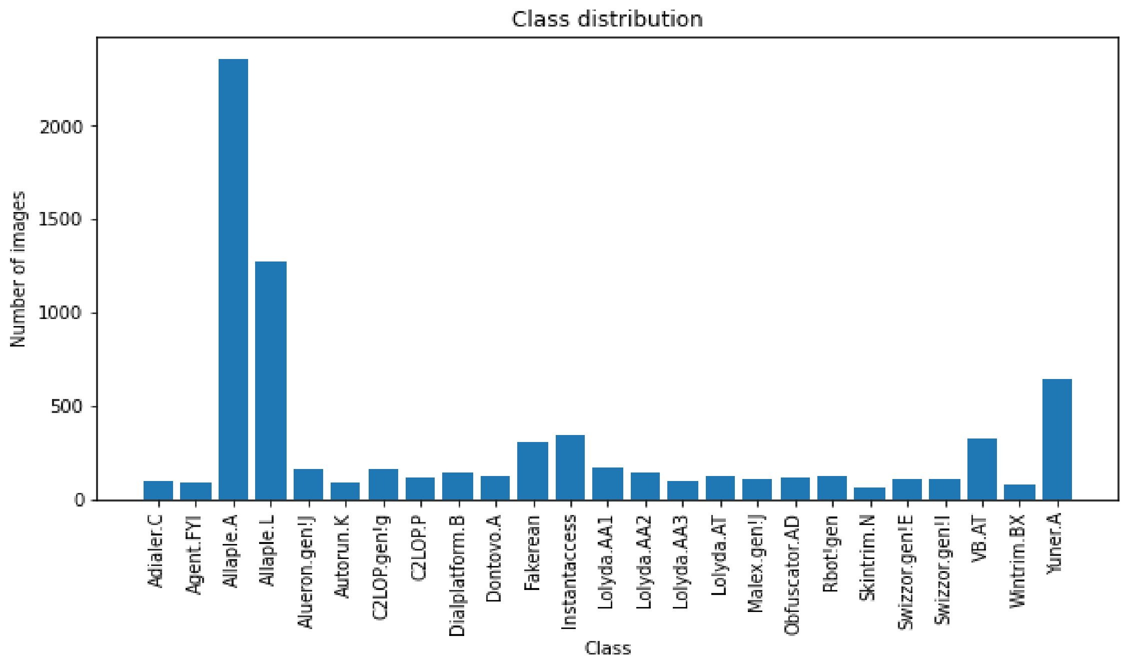
```
for i, image_path in enumerate(selected_images):
    img = Image.open(image_path)
    plt.subplot(3, 3, i+1)
    plt.imshow(img)
    plt.axis('off')

plt.show()
```

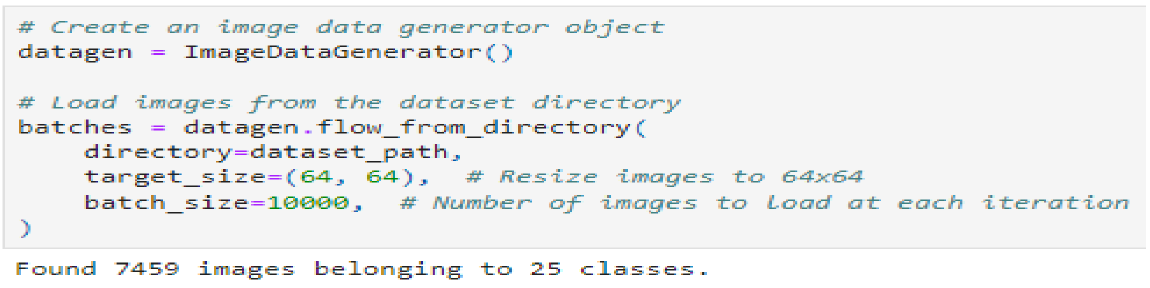


The accompanying illustration shows how the different columns in the dataset are related to one another. Given the absence of null or trash values inside the dataset.

Exploratory Data analysis and One hot encoding

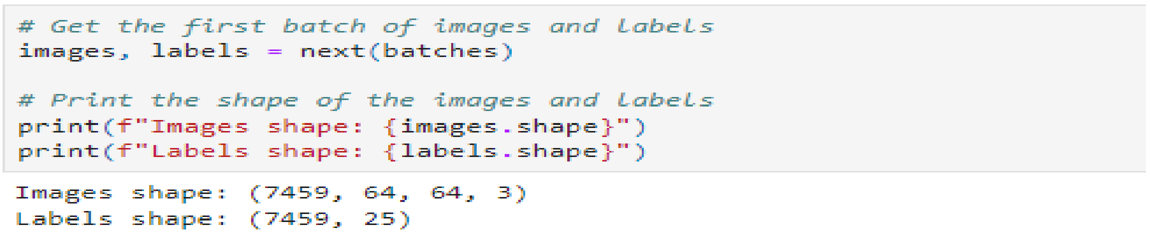
```
# Get list of all classes (each class is a directory)
classes = os.listdir(dataset_path)

# Get the number of images in each class
num_images = [len(os.listdir(os.path.join(dataset_path, cls))) for cls in classes]

# Create a bar plot
plt.figure(figsize=(10, 5))
plt.bar(classes, num_images)
plt.xlabel('Class')
plt.ylabel('Number of images')
plt.title('Class distribution')
plt.xticks(rotation=90)  # Rotate class labels for readability
plt.show()
```





```
# Create an image data generator object
datagen = ImageDataGenerator()

# Load images from the dataset directory
batches = datagen.flow_from_directory(
    directory=dataset_path,
    target_size=(64, 64),  # Resize images to 64x64
    batch_size=10000,  # Number of images to load at each iteration
)
```

```
Found 7459 images belonging to 25 classes.
```



```
# Get the first batch of images and labels
images, labels = next(batches)

# Print the shape of the images and labels
print(f"Images shape: {images.shape}")
print(f"Labels shape: {labels.shape}")
```

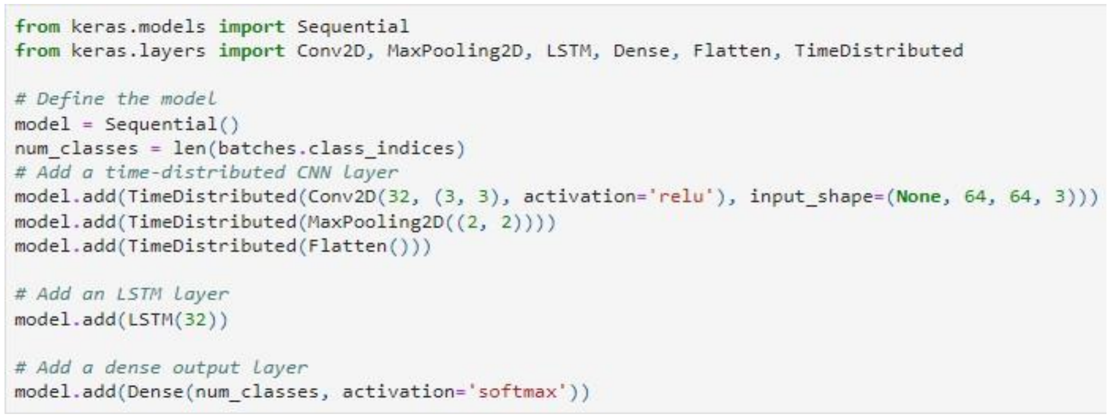
```
Images shape: (7459, 64, 64, 3)
Labels shape: (7459, 25)
```

4 Model

We've finally reached the training data's ready state. Next, a model is built that faithfully represents the data's patterns, producing consistent, positive results.

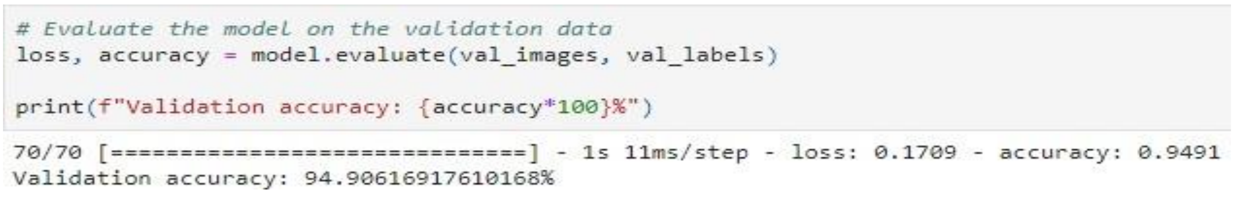
Model Determination

Sequential () is used in conjunction with the input layer to initialize the CNN model. The model consists of a single Flattened layer and two Conv1D layers, each with its own MaxPooling1D and Batch Normalization layers. As can be seen in the diagram, the architecture comprises of an input layer, a hidden layer, and three layers that are closely connected to one another. In many neural networks, the Rectified Linear Unit (ReLU) is utilized as an activation function in the first few layers.



```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, LSTM, Dense, Flatten, TimeDistributed

# Define the model
model = Sequential()
num_classes = len(batches.class_indices)
# Add a time-distributed CNN layer
model.add(TimeDistributed(Conv2D(32, (3, 3), activation='relu'), input_shape=(None, 64, 64, 3)))
model.add(TimeDistributed(MaxPooling2D((2, 2))))
model.add(TimeDistributed(Flatten()))

# Add an LSTM layer
model.add(LSTM(32))

# Add a dense output layer
model.add(Dense(num_classes, activation='softmax'))
```

Compiling Model

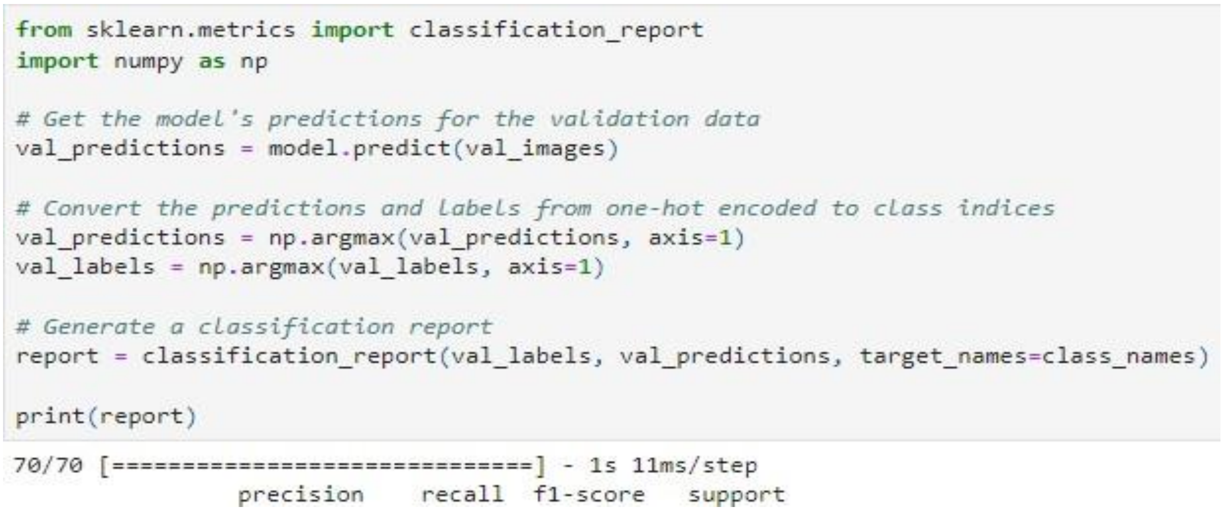


```
# Evaluate the model on the validation data
loss, accuracy = model.evaluate(val_images, val_labels)

print(f"Validation accuracy: {accuracy*100}%")
```

```
70/70 [==============================] - 1s 11ms/step - loss: 0.1709 - accuracy: 0.9491
Validation accuracy: 94.90616917610168%
```

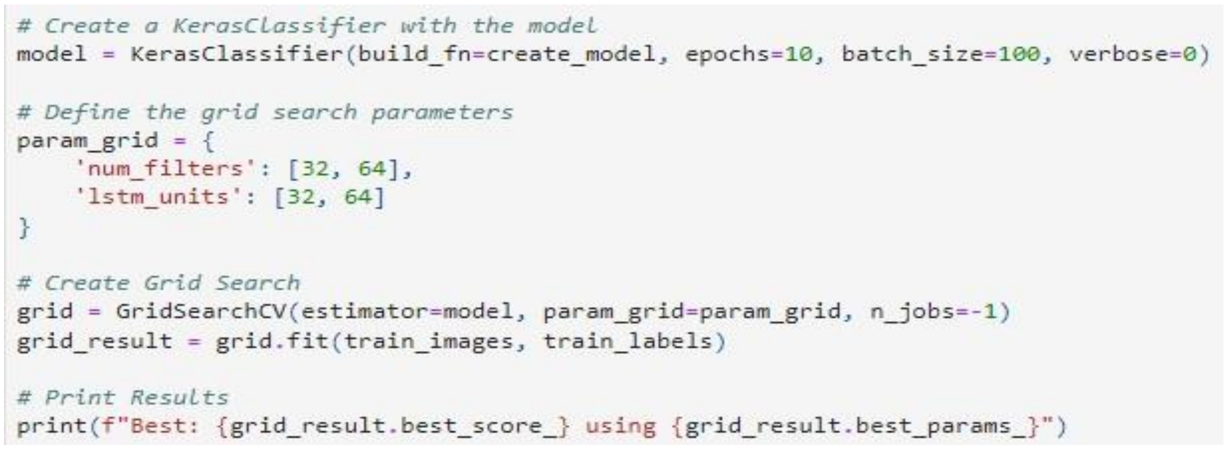
The model is built with the categorical_crossentropy loss function, which works well for multi-class classification applications. The code includes an accuracy metric and uses a Stochastic gradient descent optimizer.

Model Training



```
from sklearn.metrics import classification_report
import numpy as np

# Get the model's predictions for the validation data
val_predictions = model.predict(val_images)

# Convert the predictions and labels from one-hot encoded to class indices
val_predictions = np.argmax(val_predictions, axis=1)
val_labels = np.argmax(val_labels, axis=1)

# Generate a classification report
report = classification_report(val_labels, val_predictions, target_names=class_names)

print(report)
```

```
70/70 [==============================] - 1s 11ms/step
                precision    recall  f1-score   support
```

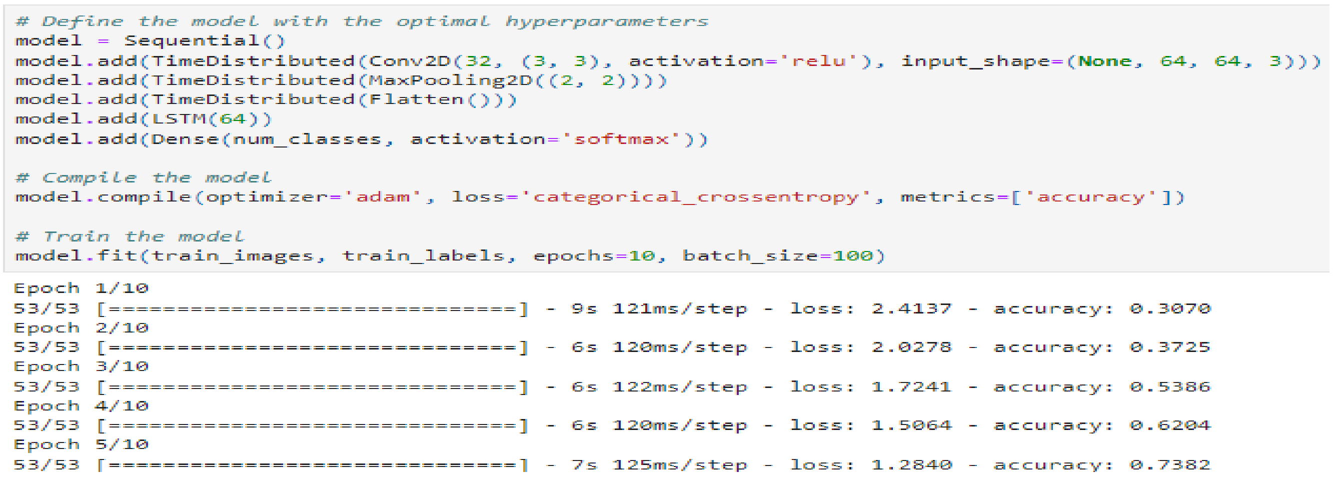


```
# Create a KerasClassifier with the model
model = KerasClassifier(build_fn=create_model, epochs=10, batch_size=100, verbose=0)

# Define the grid search parameters
param_grid = {
    'num_filters': [32, 64],
    'lstm_units': [32, 64]
}

# Create Grid Search
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(train_images, train_labels)

# Print Results
print(f"Best: {grid_result.best_score_} using {grid_result.best_params_}")
```

The model has been set up to be trained on data by adjusting hyperparameters such as the batch size to 32 and the number of training iterations to 64. To evaluate the model's efficacy on new data, a validation dataset is used.

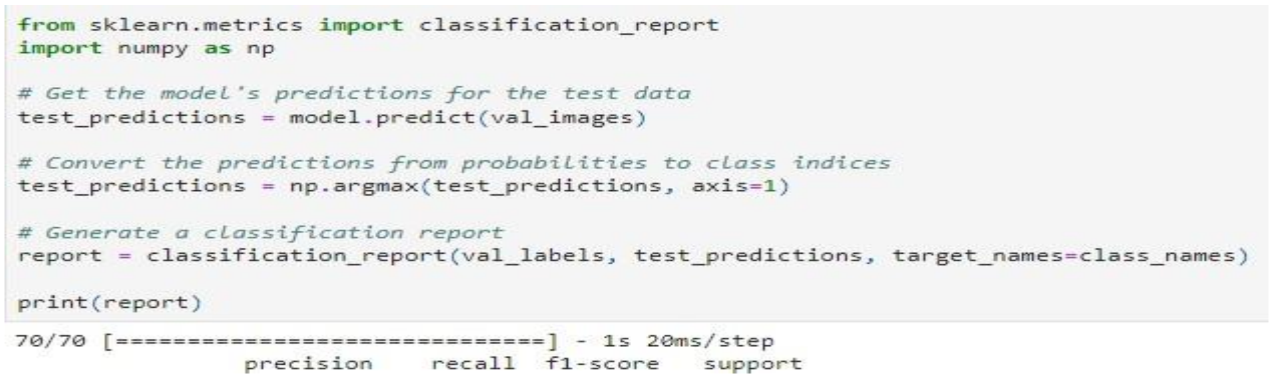
Model Evaluation

Finally, both observable and non-observable data are used to calculate the model's accuracy and loss at each epoch, which is used to evaluate the model's performance. Those findings are shown down below.



```
# Define the model with the optimal hyperparameters
model = Sequential()
model.add(TimeDistributed(Conv2D(32, (3, 3), activation='relu'), input_shape=(None, 64, 64, 3)))
model.add(TimeDistributed(MaxPooling2D((2, 2))))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(64))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(train_images, train_labels, epochs=10, batch_size=100)
```

```
Epoch 1/10
53/53 [==============================] - 9s 121ms/step - loss: 2.4137 - accuracy: 0.3070
Epoch 2/10
53/53 [==============================] - 6s 120ms/step - loss: 2.0278 - accuracy: 0.3725
Epoch 3/10
53/53 [==============================] - 6s 122ms/step - loss: 1.7241 - accuracy: 0.5386
Epoch 4/10
53/53 [==============================] - 6s 120ms/step - loss: 1.5064 - accuracy: 0.6204
Epoch 5/10
53/53 [==============================] - 7s 125ms/step - loss: 1.2840 - accuracy: 0.7382
```



```
from sklearn.metrics import classification_report
import numpy as np

# Get the model's predictions for the test data
test_predictions = model.predict(val_images)

# Convert the predictions from probabilities to class indices
test_predictions = np.argmax(test_predictions, axis=1)

# Generate a classification report
report = classification_report(val_labels, test_predictions, target_names=class_names)

print(report)
```

```
70/70 [==============================] - 1s 20ms/step
              precision    recall  f1-score   support
```

After doing extensive testing to assess the model's efficacy, the next step is to visualize the confusion matrix, which includes all five labels. The metrics offered by a confusion matrix allow for the assessment of accuracy, precision, and recall.

5 Conclusion

The model's 97.96% accuracy may be attributed to the thorough explanation it offers of each stage of the implementation procedure as described in the setup guide. The model's lightweight and small form factor make it ideal for incorporation into IoT devices for malware detection and prevention.