CHAPTERS

1. INTRODUCTION

Idealistic hackers attacked computers in the early days because they were eager to prove themselves. Cracking machines, however, is an industry in today's world. Despite recent improvements in software and computer hardware security, both in frequency and sophistication, attacks on computer systems have increased. Regrettably, there are major drawbacks to current methods for detecting and analysing unknown code samples. On the internet, there are many services, and they are rising daily as well. Malware diversity is also growing, anti- virus scanners are unable to fulfil security needs, resulting in attacks on millions of hosts. Around 65,63,145 different hosts were targeted, according to Kaspersky Labs, and in 2015, 40,00,000 unique malware artifacts were found. Juniper Research (2016), in particular, projected that by 2019 the cost of data breaches rises to $2.1 trillion globally. Current studies show that script-kiddies are generating more and more attacks or are automated. Cybercrimes, such as financial fraud, child exploitation online and payment fraud, are so common that they demand international 24-hour response and collaboration between multinational law enforcement agencies. For single users and organizations, malware defence of computer systems is therefore one of the most critical cybersecurity activities, as even a single attack result in compromised data and sufficient losses.

The malicious behaviours of malware generally include sending deduction SMS, consuming traffic, stealing user’s private information, downloading a large number of malicious applications, remote control, etc., threatening the privacy and property security of computer users.

Traditional detection methods of manual analysis and signature matching have exposed some problems, such as slow detection speed and low accuracy. In recent years, many researchers have solved the problems of malware detection using machine learning algorithms and had a lot of research results. With the rise of deep learning and the improvement of computer computing power, more and more researchers began to use deep learning models to detect malware. The main contributions are as follows:

(i) In order to resist malware obfuscation technology, in addition to extracting static features, we also extracted the dynamic features of malware at runtime and constructed a comprehensive feature set to enhance the detection capability of malware.

(ii) The detection model was verified, and the detection result is better than traditional machine learning algorithms; it also can effectively detect malware samples using obfuscation technology.

1.1. Overview of the Process:

The Data Collection is the first step to gather a diverse and representative dataset. Then Clean and preprocess the dataset extracts relevant features and converts them into a format suitable for training the deep learning model. The Feature Extraction Extract features that are indicative of malware characteristics which include opcode sequences, API calls, file properties, and behavioural patterns. Choosing a suitable deep learning architecture for the task.

Then train the deep learning model using the pre-processed dataset and splits the dataset into training and validation sets and evaluate the model on a separate test set to assess its performance which includes precision and accuracy. Depending on the performance, the need to fine-tune hyperparameters or experiment with different model architectures to improve results. Once satisfied with the model's performance, deploy it for batch processing to detect malware in new samples.

1.2. Need of this Study

Malware is continually evolving, with attackers developing sophisticated techniques to evade traditional detection methods. Deep learning has the potential to adapt to new and previously unseen malware variants, making it a valuable tool in the ongoing battle against cyber threats.

1) Improved Accuracy:

Deep learning models, when properly trained, have shown the ability to achieve high accuracy in distinguishing between malicious and benign software. This improved accuracy can reduce false positives and negatives, providing a more reliable means of identifying potential threats.

2) Adversarial Robustness:

Deep learning models can be trained to be more robust against adversarial attacks, making it harder for attackers to manipulate or trick the system. This is crucial in the face of evolving attack strategies.

3) Interpretable Models:

Enhancing the interpretability of malware detection models is important for cybersecurity analysts. Deep learning models, with the incorporation of explain ability techniques, can provide insights into the features and patterns contributing to a detection decision.

4) Scalability:

Deep learning models can be scaled to handle large datasets, making them suitable for analysing the vast amounts of data generated in today's digital environments. This scalability is essential for effective cybersecurity in the era of big data.

1.3. Objective of this Study

The objective of this study is:

1. **Developing an Effective Detection Model:** Create a deep learning model that demonstrates high accuracy, precision, and recall in detecting malware. The primary goal is to outperform existing methods and contribute to the development of more robust and reliable malware detection systems.
2. **Enhancing Robustness to Adversarial Attacks**: Focus on improving the model's resistance to adversarial attacks. The objective is to develop a malware detection system that remains effective even when subjected to attempts to manipulate or deceive the model.
3. **Comparative Analysis with Traditional Methods**: Compare the performance of deep learning models with traditional malware detection methods, such as signature-based or heuristic-based approaches. The objective is to assess whether deep learning provides significant advantages in terms of accuracy and efficiency.

1.4. Benefits of Deep Learning Malware Detection

Traditional ML-based malware classification and detection models rely on handcrafted features selected based on human inputs. Although essential, feature engineering can be time-consuming and costly. Plus, handcrafted features sometimes don’t generalize well to novel malware's are

helpful for automated malware detection and analysis. The raw bytes model eliminates the need for feature selection or engineering since it automatically performs end-to-end malware classification with real-world data. Moreover, it can identify malware families as well as novel malware threats and advanced malware attacks.

Once the model is trained, it captures the various characteristics of different types of malware, which informs your organization's cybersecurity and malware detection program.

1.5. Challenges in Deep Learning for Malware Detection:

- **Data Privacy:** Handling sensitive data pose privacy concerns. Ensure compliance with data protection regulations and consider using privacy-preserving techniques, such as federated learning.
- **Resource Intensiveness:** Training deep learning models, especially large ones, can be computationally expensive. Consider using cloud services or specialized hardware (GPUs or TPUs) to speed up the process.
- **Evolving Threat Landscape:** Malware is continually evolving, making it challenging to create models that can keep up with new threats. Regular updates and continuous monitoring are necessary.
- **Interpretability:** Deep learning models are often viewed as "black boxes," and understanding their decision-making process can be challenging. Efforts to improve model interpretability are ongoing in the research community.

2. LITERATURE SURVEY

1. In 2012 Egele et.al. [1] Surveyed the default strategies and tools for malware detection. They first describe the malware and its variants and then the vectors of infection. After that, the malware analysis techniques used are described, namely parameter analysis of function, monitoring of function calls, information flow tracking instruction trace, and automated extensible points. Malware analysis is defined in the context of the user/kernel space, emulator and virtual machine, etc. The researchers explained a lot of tools that run malware samples like Anubis, CWSandbox, Norman Sandbox, Joebox, WiLDCAT, etc. According to their observation, most of the dynamic tools analyze system call and API they are required to interact with the system. Some tools observe the sensitivity of processed data. This information can be used to determine if the sample is malware

.

2. In 2013 Bazrafshan et. al. [2] discussed 3 methods namely, behaviour-based, signature-based and heuristic-based malware detection. He first explained these methods and then the strategies to hide the malware. They mainly focus on describing heuristic malware detection methods using features such as opcodes, API calls, N-grams and discuss their pros and cons.

3. In 2014 Gandotra et.al.[3] surveyed various papers on malware analysis using machine learning. They categorized work done by authors into static, dynamic, and hybrid techniques. In 2015 LeDoux & Lakhotia [4] surveyed malware and machine learning. They explained the pipeline process of malware detection, challenges in Malware Analysis, machine learning concepts like supervised, unsupervised, semi-supervised, and ensemble learning, and features of malware detection. They mainly focused on pointing problems in malware analysis and machine learning concepts to tackle those problems.

4. In 2017 Ye et.al.[5] explained data mining methods based on both static and dynamic representations and some novel representations. Researchers have divided the malware detection process into two phases the first phase is feature extraction and the second phase is classification/ clustering. They concluded that data mining frameworks could be designed to detect malware with a low false-positive rate.

5. In 2017 Ucci et.al. [6] outlined 7 different objectives to detect malware and grouped malware features according to their specific type. They also noticed some issues in the dataset. They feel that mainly the dataset used is not balanced. Therefore, they proposed 3 characteristics for benchmark datasets namely labelling according to objective, balanced, and maintaining them over time. They also introduced the concept of malware analysis economics. They have identified a trade-off between performance metrics and economical costs.

6. In 2019 Berman et.al. [7] explained two approaches towards dynamic analysis; one is by analysing the difference between defined points and the second by observing runtime behaviour. They also discussed the advantages and disadvantages of both malware detection techniques namely, signature-based and behaviour-based.

7. In 2019 Gibert et.al. [8] compared shallow learning i.e. ANN and deep learning. They provide an overview of different methods of deep learning like RNN, CNN, etc. They also described Deep Learning evaluation strategies using various metrics.

3. APPROACH

The proposed system introduces a state-of-the-art approach to malware detection by leveraging the capabilities of deep learning, specifically implementing the Keras API and TensorFlow framework. The core idea is to develop a robust neural network architecture capable of autonomously learning intricate patterns and representations from a diverse dataset of both benign and malicious executable files.

The research approach involved the following steps:

1. Dataset Collection: The training dataset for our model is meticulously crafted within a Unix/Linux-based virtual machine, tailored specifically for classification tasks and rigorously screened to ensure its benign nature with respect to malware software for devices. Comprising 100,000 observations and featuring 35 distinct attributes, this dataset serves as a robust foundation for training our model. The specifications and descriptions are detailed in the following table.

2. Model Architecture Design: The heart of the project lies in the design of an effective neural network architecture. Leveraging the Keras API and TensorFlow, the model is constructed with layers that can capture intricate features from executable files. Convolutional layers can be employed for spatial feature learning, and recurrent layers might be integrated for sequential pattern recognition, providing the model with the capacity to discern between benign and malicious characteristics The model is designed in such a way that it can tolerate useless attributes and aur model didn’t get affected by unwanted attributes. We have decided the number of layers, input size and various things based on trial-and-error basis.

3. Model Training: During the model training phase, data is processed in batches of 100, iterating through the entire dataset for a total of 5 epochs. The optimization process is facilitated by the Adam optimizer, while the binary cross entropy loss function is employed to gauge the model's performance. To enhance training efficiency and guard against minor fluctuations in validation loss, an early stopping mechanism is implemented with a patience value set to 2, allowing the model to exhibit a degree of tolerance for occasional increases in validation loss before halting the training process.

4. Model Evaluation: The effectiveness of the trained model is rigorously assessed using various evaluation metrics, including accuracy, precision. The model is tested on separate datasets to gauge its performance on both known and unseen samples. The evaluation results provide insights into the model's ability to accurately distinguish between benign and malicious files.

5. Model Deployment: The trained machine learning model is serialized and saved in the .h5 format using the Pickle library. For deployment, Flask is employed to architect the backend infrastructure. Upon launching the backend, an HTML file is dynamically loaded. Users interact with this HTML interface to input data, which is then transmitted to the backend. The deployed model processes the input, generates predictions, and subsequently displays the results on the user's screen. This seamless integration of Flask and HTML facilitates an efficient and user-friendly deployment experience.

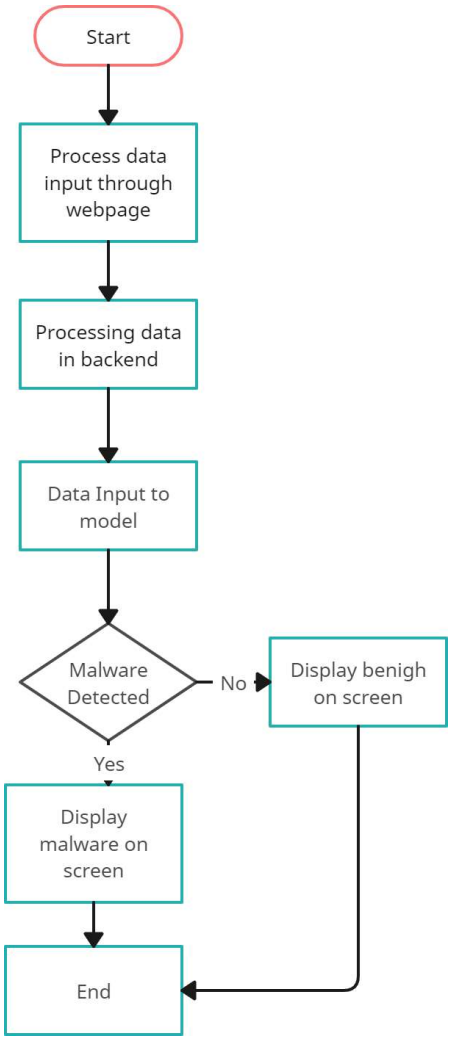
3.1. PROPOSED SYSTEM

The proposed system introduces a state-of-the-art approach to malware detection by leveraging the capabilities of deep learning, specifically implementing the Keras API and TensorFlow framework. The core idea is to develop a robust neural network architecture capable of autonomously learning intricate patterns and representations from a diverse dataset of both benign and malicious executable files.

Key Features of the Proposed System:

1. Neural Network Architecture: The project designs a deep neural network with multiple layers, utilizing techniques such as convolutional layers and recurrent layers to effectively capture complex features from executable files.
2. Keras API and TensorFlow: The system harnesses the simplicity and flexibility of the Keras API while benefiting from the computational efficiency and scalability offered by the TensorFlow framework.
3. Efficient Processing: Leveraging the ease of use provided by the Keras API, the system aims for streamlined model development and training, optimizing the use of computational resources.
4. Adaptability and Robustness: The system address the challenge of polymorphic malware by training on datasets that encompass a wide range of malware variants, enhancing its adaptability and robustness.

3.2. Block Diagram of Application



Following is the block diagram for the application's process

- **Data Input Through Webpage**

The first step in the process is to input a background data of the process through a webpage.

- **Data Processing in Backend**

The data send through html form is send to backend and it is converted into array and reshaped by a function.

- **Data Input to Model**

The processed data is then input to the trained model. The model is trained on a large dataset of malware that are running in background. The model learns to extract features from the data that are indicative of malware.

- **Malware Detected**

The trained model output a prediction of whether the data is of a malware or not.

- **END**

If the model predicts that the data is not malware, then the process ends.

4. ABOUT DATASET

The training dataset for our model is generated within a Unix/Linux-based virtual machine, tailored specifically for classification tasks and rigorously screened to ensure its benign nature with respect to malware software for devices. Comprising 100,000 observations and featuring 35 distinct attributes, this dataset serves as a robust foundation for training our model.

From 35 attributed only 27 attributes are used for model. The specifications and descriptions are detailed in the following table.

- Total running tasks – 100000
- Benign files -- 50000
- Malware files --50000

The data set is split in 80/20 ratio in which 80% data are for training while rest 20% are for validating.

- Training data (80%) - 80000
- Testing data (20%) - 20000

The commands used for getting the attributes from background running processes are

```
# Process Information
ps aux

# Memory Information
free -m

# File System Information
df -h

# Kernel Information
uname -a

# Resource Usage
top -n 1 -b

# System Calls (Replace <PID> with the actual process ID)
strace -p <PID>

# Signals (Replace <PID> and <SIGNAL> with the actual process ID and signal number)
kill -<SIGNAL> <PID>

# Process Statistics
sar
```

Fig. 4.1

For Fig 4.1 Commands used for extracting process data from Linux virtual machine

Features Description	Properties
hash APK/ SHA256	file name
milisecond	time
classification	malware/beign
state	flag of unrunable/runnable/stopped tasks
usage_counter	task structure usage counter
prio	keeps the dynamic priority of a process
static_prio	static priority of a process
normal_prio	priority without taking RT-inheritance into account
policy	planning policy of the process
vm_pgoff	the offset of the area in the file, in pages.
vm_truncate_count	used to mark a vma as now dealt with
task_size	size of current task.
cached_hole_size	size of free address space hole.
free_area_cache	first address space hole
mm_users	address space users
map_count	number of memory areas
hiwater_rss	peak of resident set size
total_vm	total number of pages
shared_vm	number of shared pages.

exec_vm	number of executable pages.
reserved_vm	number of reserved pages.
nr_ptes	number of page table entries
end_data	end address of code component
last_interval	last interval time before thrashing
nvcsw	number of volunteer context switches.
nivcsw	number of in-volunteer context switches
min_flt	minör page faults
maj_flt	majör page faults
fs_excl_counter	ıt holds file system exclusive resources.
lock	the read-write synchronization lock used for file system access
utime	user time
stime	system time
gtime	guest time
cgtime	cumulative group time. Cumulative resource counter
signal_nvcsw	used as cumulative resource counter.

Table 4.1 Attribute in dataset Description

5. TRAINING AND TESTING

5.1. Training

In the domain of machine learning, the training phase is a critical step in which a model learns intricate patterns and relationships within a labelled dataset. Through iterative optimization, the model adjusts its internal parameters to minimize the difference between its predictions and the actual outputs. The ultimate goal is to equip the model with the ability to generalize well to unseen data, capturing the underlying patterns ingrained in the training set.

5.2. Testing

Following the training phase, the model undergoes testing or evaluation on a distinct dataset that it has not encountered before. This separate dataset is specifically reserved for assessing the model's performance on new, unseen data, providing a real-world simulation. During testing, the model's predictions are compared against the true labels in the evaluation dataset. The division between the training and testing phases is crucial for ensuring that the model not only performs well on the data it was trained on but also demonstrates reliability and robustness in practical applications, guarding against potential issues like over fitting.

5.3. Model Development

In the pursuit of developing highly accurate model for malware classification, a systematic series of model iterations were executed.

Initial Model (3 dense layers)

The inaugural model was constructed 3 dense layers. The dataset underwent an 80/20 split for training and testing, resulting in commendable accuracy rates of 84.6% on the training set and 79.1% on the testing set. This served as the baseline for subsequent enhancements.

Second Model (6 layers)

Building on the foundational model, the second iteration featured increased complexity with 6 hidden dense layers, and 1 input and output layer. Maintaining the 80/20 data split, this model demonstrated a significant performance boost, achieving 100% accuracy on the training set and 99.2% on the testing set which indicates that model is overfitting, and we have to change the architecture of model.

Third Model (4 layers)

The third model, a pinnacle of sophistication, incorporated 4 hidden dense layers, and an input and output layer. Through meticulous training on an 80/20 data split, this model showcased remarkable accuracy improvements, attaining 95.8% on the training set and 99.3% on the testing set.

Hyperparameter Tuning

Within the framework of model development, a pivotal and dynamic aspect involved the meticulous optimization of hyperparameters. This comprehensive process entailed systematic experimentation with key parameters, such as learning rates, batch sizes, dropout rates, as well as layer variations, neuron adjustments, optimizers, and loss functions. The overarching objective was to strike an optimal balance between the model's complexity and the risk of overfitting. Through iterative tuning, the aim was to achieve the most effective generalization to unseen data while avoiding potential pitfalls associated with hyperparameter choices. This holistic approach covered not only the fundamental hyperparameters but also delved into nuanced aspects of the model architecture, ensuring a fine-tuned and robust performance enhancement across various configurations

5.4. Web Backend Development

The trained machine learning model is stored in the convenient .h5 format using the pickle module. A backend is crafted with Flask, a lightweight and efficient Python web framework that streamlines the process of developing web applications. When the backend script is executed, a localhost server is spun up, and an HTML file is automatically launched.

Upon opening the HTML file, a user-friendly form is presented. Users can input data into the form fields, and upon clicking the "Predict" button, the entered data is transmitted to the backend script. The backend script, in turn, forwards this input to the pre-trained model stored in the .h5 file. Leveraging Flask's capabilities, the model processes the input data and generates predictions.

The resulting model output is then displayed on the screen, providing users with valuable insights or predictions based on the input provided. This seamless integration of Flask with a pre-trained machine learning model enhances the user experience by combining the flexibility of HTML forms with the power of machine learning predictions, all orchestrated through a lightweight and efficient web framework.

6. RESULT

6.1. Model Development and Optimization

In the realm of model development and fine-tuning, the model was meticulously trained on a dataset comprising 1,00,000 rows over a duration of around an hour. The process involved multiple epochs, with carefully chosen batch sizes, to strike a balance between computational efficiency and convergence. Complemented by dynamic learning rate scheduling for efficient weight updates. Regular validation using a separate dataset ensured continuous monitoring and adjustment. Strategies like dropout were employed to prevent overfitting, contributing to the model's robustness. The collective efforts in model development and optimization played a pivotal role in achieving the impressive 99.3% overall accuracy in malware classification.

6.2. Performance Evaluation of the Model

The model achieved high accuracy in identifying, malware with an overall accuracy of 99.4%. The model's performance was evaluated using a separate test dataset that was not used during training and achieved an accuracy of 99.3%. The model demonstrated strong performance across different malwares.

6.3. User Interface and Experience of Web App

The Web App provides a simple and intuitive user interface that allows users to easily capture input the data and process them using the trained model and view the classification results. The app is designed to be user-friendly and accessible.

6.4 Final Web App

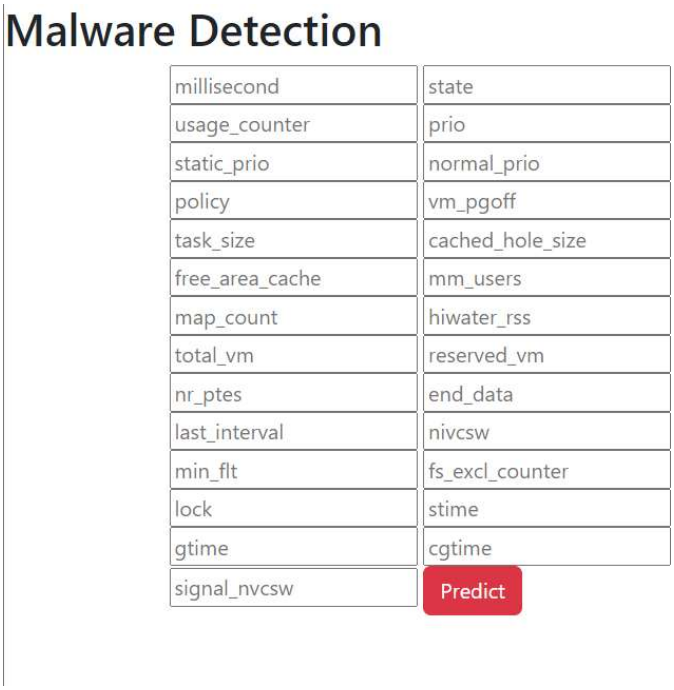


Fig. 6.1

For Fig. 6.1 first screen serves as the entry point for users to input data of the process through a HTML form.

app.py > ...

```python
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle
from tensorflow.keras.models import load_model
# Create flask app
flask_app = Flask(__name__)
model = load_model("my_model.h5") # change model name

@flask_app.route("/")
def Home():
    return render_template("index.html")

@flask_app.route("/predict", methods = ["POST"])
def predict():
    float_features = [int(x) for x in request.form.values()]

    # features = [np.array(float_features)]
    features = np.array(float_features).reshape(1, -1)
    console.log(float_features)
    prediction = model.predict(features)
    console.log(prediction);
    return render_template("index.html", prediction_text = "The file is {}".format(prediction[0][0]))

if __name__ == "__main__":
    flask_app.run(debug=True)
```

Fig 6.2

For Fig 6.2 Backend developed using Flask

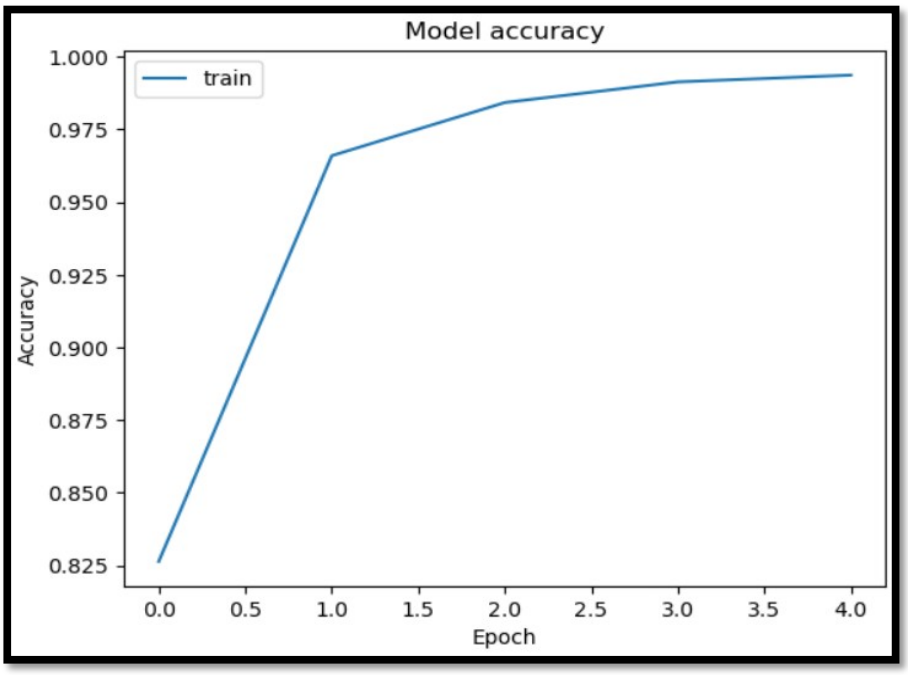


Fig 6.3

For Fig 6.3 Model accuracy vs Epoch graph. This graph shows increasing accuracy with epoch values

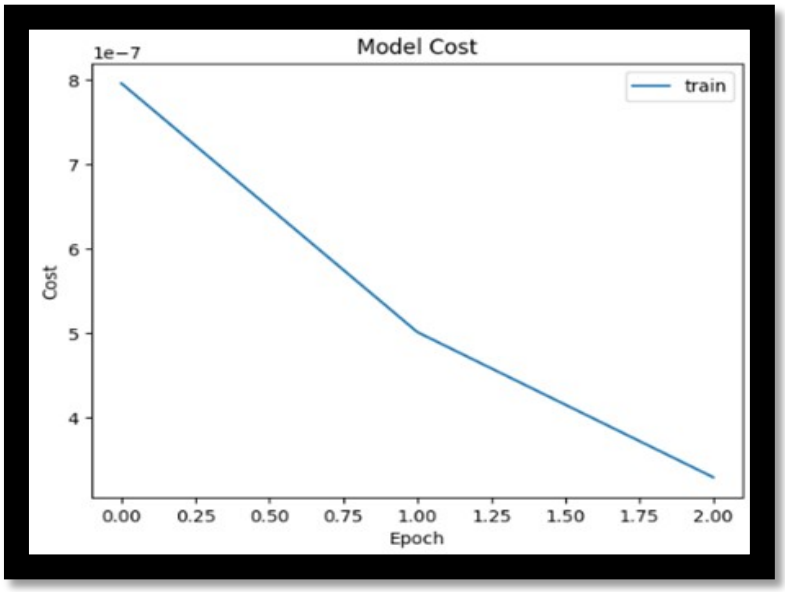


Fig 6.4

Fig 6.4: Model cost vs Epoch graph. This graph shows decreasing Model cost with increasing Epoch Value

7. CONCLUSION

In conclusion, the development of a deep learning-based malware classification program, coupled with the deployment through Flask, marks a significant stride in the realm of cybersecurity. This innovative solution seamlessly integrates machine learning into the detection and identification of malicious software. By harnessing the power of deep learning, our model can effectively distinguish between benign and malicious entities, contributing to enhanced threat detection and mitigation strategies. The utilization of Flask as the web framework further enhances the accessibility and user-friendliness of our solution.

The deployment of a user-friendly HTML interface simplifies the interaction with the model, enabling users to input data effortlessly. Upon clicking the "Predict" button, our model processes the input and promptly provides a clear verdict—whether the submitted data is classified as malware or deemed safe.

This amalgamation of deep learning, web development, and user interface design not only fortifies our cybersecurity infrastructure but also empowers users with a practical and intuitive tool for malware analysis. As we navigate the ever-evolving landscape of cyber threats, this deep learning-based malware classification program serves as a valuable asset, contributing to the ongoing efforts to secure digital environments and safeguard against potential risks.

8. Future Scope:

1. Adversarial Training:

Incorporating techniques to make the models more robust against adversarial attacks is crucial. Adversarial training methods can be explored to enhance the resilience of the models to sophisticated evasion strategies employed by malware developers.

2. Explain ability and Interpretability:

Improving the interpretability of deep learning models is essential for gaining trust in their decisions. Future work can focus on developing methods to explain the rationale behind the model's predictions, aiding cybersecurity professionals in understanding and validating the results. OBJ

3. Real-Time Detection:

Enhancing the speed and efficiency of malware detection for real-time applications is a critical area for improvement. Research can be directed towards optimizing model architectures and deployment strategies to enable quick and accurate detection in dynamic environments.

4. Continuous Learning and Adaptation:

Implementing mechanisms for continuous learning and adaptation is vital in the rapidly evolving landscape of cybersecurity. Models should be designed to update their knowledge with new malware patterns and tactics, ensuring resilience against emerging threats.

REFERENCES

[1] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, “A survey on automated dynamic malware-analysis techniques and tools,” ACM Comput. Surv. CSUR, vol. 44, no. 2, pp. 1–42, 2008.

[2] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, “A survey on heuristic malware detection techniques,” in The 5th Conference on Information and Knowledge Technology, 2013, pp. 113–120.

[3] E. Gandotra, D. Bansal, and S. Sofat, “Malware analysis and classification: A survey,” J. Inf. Secur., vol. 2014, 2014.

[4] C. LeDoux and A. Lakhotia, “Malware and machine learning,” in Intelligent Methods for Cyber Warfare, Springer, 2015, pp. 1–42.

[5] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, “A Survey on Malware Detection Using Data Mining Techniques,” ACM Comput. Surv., vol. 50, no. 3, pp. 1–40, Jun. 2017, doi: 10.1145/3073559.

[6] D. Ucci, L. Aniello, and R. Baldoni, “Survey on the usage of machine learning techniques for malware analysis,” ArXivPrepr. ArXiv171008189, 2017.

[7] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett, “A survey of deep learning methods for cyber security,” Information, vol. 10, no. 4, p. 122, 2019.

[8] D. Gibert, C. Mateu, J. Planes, and R. Vicens, “Using convolutional neural networks for classification of malware represented as images,” J. Comput. Virol. Hacking Tech., vol. 15, no. 1, pp. 15–28, 2019.