Mini Project Report on

# DEEP LEARNING-BASED FOR MALWARE ATTACK DETECTION.

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Submitted by:

AASHNAN GOYAL 2021632

*Under the Mentorship of*

**Prof. (Dr.) MOHAMMAD WAZID**

**Associate Professor**

Department of Computer Science and Engineering
Graphic Era (Deemed to be University)
Dehradun, Uttarakhand

January 2024

Graphic Era
Deemed to be
University
DEHRADUN

# CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the project report entitled **"Deep learning – Based for Malware Attack Detection"** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering of the Graphic Era (Deemed to be University), Dehradun shall be carried out by the under the mentorship of **Prof.(Dr.) Mohammad Wazid, Associate Professor,** Department of Computer Science and Engineering, Graphic Era (Deemed to be University), Dehradun.

AGoyal

Aashnan Goyal 2021632

# Table of Contents

# Chapter 1

# Introduction

## 1.1 Introduction

Cyberattacks are currently the most pressing concern in the realm of modern technology. The word implies exploiting a system's flaws for malicious purposes, such as stealing from it, changing it, or destroying it. In layman's language, it's the offensive process, adopted by a hacker to harm the digital assets of an individual or an organization. The impacting assets could be computers, laptops, networks, servers, information systems, security infrastructures, and many more. The core aim of a cyber-attack is to gain admin-like access to the targeted asset or corrupt it so that the stored information can be accessed or related functions can malfunction [1]. There are many types of cyber attacks -

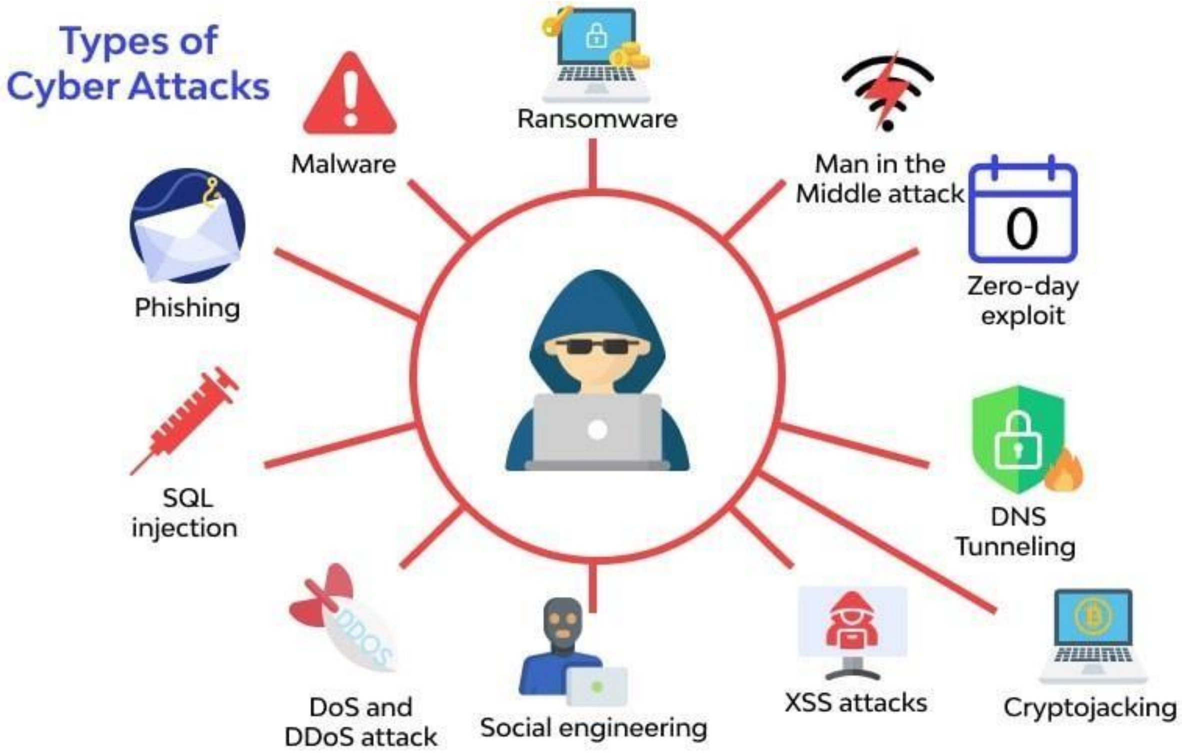


**Figure 1.1** Types of Cyber Attacks [1]

For this project, we are concerning ourselves with the detection of one particular type of cyberattack known as Malware.

Malware is any program or set of instructions that are designed to harm a computer, user, business, or computer system. The term “malware” encompasses a wide range of threats, including viruses, Trojan horses, ransomware, spyware, adware, rogue software, wipers, scareware, and so on. Malicious software, by definition, is any piece of code that is run without the user’s knowledge or consent [2].

Mobile malware has become a significant threat to the security of mobile devices in recent years. Android, the most widely used mobile operating system, is particularly vulnerable to malware attacks. The increasing number of malware attacks on Android devices poses a significant risk to users and their personal information. As a result, the task of detecting malware on Android devices is crucial to protect users from these malicious attacks. In this project, we propose a machine learning-based approach for the detection of malware on Android devices using the attributes extracted from Android applications as features.

The use of machine learning in malware detection is an active research area, as it allows for the automation of the detection process and the ability to adapt to new threats. In this project, we aim to extract meaningful features from Android applications that can be used to identify malware accurately. These features include information about the application's permissions, activities, services, and receivers. We use these attributes as input to our machine learning model, which is trained to distinguish between benign and malicious applications.

## 1.2. Dataset

The dataset used in this project was obtained from Kaggle, an online community platform and the dataset contains a set of attributes extracted from both benign and malicious Android applications. The dataset consists of feature vectors of 215 attributes extracted from 15,036 applications (5,560 malware apps from the Drebin project and 9,476 benign apps).

## Chapter 2

# Literature Survey

There has been a significant amount of research conducted in recent years on the use of machine learning for malware detection. Some of the latest research work in this area includes.

1. **"A Review of Android Malware Detection Approaches Based on Machine Learning" by Liu et al. (2020)** [3] **-** This paper proposed a deep learning-based approach for detecting Android malware. The authors used a convolutional neural network (CNN) to extract features from Android Application Package (APK) files and then used these features to train a classifier for detecting malware. The proposed approach achieved an accuracy of 97.5% on a dataset of over 10,000 APK files.
2. **"A Survey of Malware Detection Techniques based on Machine Learning" by El Merabet et al. (2019)** [4] **-** This paper provided a comprehensive survey of machine learning-based malware detection techniques. The authors discussed various techniques, including decision trees, random forests, support vector machines, neural networks, and ensemble methods. They also discussed the use of feature extraction and selection techniques, as well as the evaluation of performance using various metrics.
3. **"Malware detection using machine learning and deep learning techniques" by Hemant Rathore, et al. (2019)** [5] **-** This paper reviewed various machine learning and deep learning techniques for malware detection. The authors discussed the use of decision trees, random forests, support vector machines, and neural networks for this purpose. Their results showed that the Random Forest outperforms the Deep Neural Network with opcode frequency as a feature.
4. **"Malware detection using machine learning: A review" by Khan et al. (2016) -** This paper reviewed various machine learning-based malware detection techniques. The authors discussed the use of decision trees, random forests, support vector machines, and neural networks for this purpose. They also compared the performance of these techniques on several datasets and found that decision tree-based approaches performed best.

Some key observations that can be made from these studies include:

- Deep learning-based approaches are showing promising results in malware detection. Studies such as **"A Review of Android Malware Detection Approaches Based on Machine Learning" by Liu et al. (2020)**[2] have demonstrated that convolutional neural networks (CNNs) and long short-term memory (LSTM) networks are effective in detecting malware, achieving high accuracy rates.
- Combining multiple machine-learning models can improve the performance of malware detection. Studies such as **"A Survey of Malware Detection Techniques based on Machine Learning" by El Merabet et al. (2019)** [3] have suggested that ensemble methods, which combine the predictions of multiple machine learning models, can improve the performance of malware detection.
- Feature extraction and selection are important for machine learning-based malware detection. Studies such as **"A Survey of Malware Detection Techniques based on Machine Learning" by El Merabet et al. (2019)** have highlighted the importance of extracting relevant features from malware samples and selecting the most informative features for training a classifier.
- Evaluation of performance using various metrics is important for machine learning-based malware detection. Studies such as **"A Survey of Malware Detection Techniques based on Machine Learning" by El Merabet et al. (2019)** [3] have emphasized the importance of evaluating the performance of machine learning-based malware detection using various metrics, such as accuracy, precision, recall, and F1-score.

Overall, the current situation of malware detection using machine learning is promising, with deep learning-based approaches showing the most potential. However, there is still room for improvement, especially in combining multiple machine-learning models and feature extraction and selection. Additionally, there is a need for more robust evaluation methods.

## Chapter 3

# Methodology

### 3.1. Libraries Used [6]

1. **pandas -** pandas is a software library written for the Python programming language for data manipulation and analysis. It has functions for analyzing, cleaning, exploring, and manipulating data. It can clean messy datasets to make them readable and relevant.
2. **matplotlib** - it is a plotting library for Python and it helps to create a wide variety of static, animated, and interactive visualizations in Python, including line plots, scatter plots, bar plots, histograms, 3D plots, and more.
3. **tensorflow** - it provides a set of tools for building and deploying machine learning models, as well as a flexible ecosystem of tools and libraries that can be used to extend its functionality.
4. **keras** - Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.
5. **numpy** - it is a Python library that is used for scientific computing and data manipulation. It provides an array object, which is similar to a list but allows for more efficient mathematical operations.
6. **Scikit-learn (Sklearn) -** it is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reductionvia a consistency interface in Python. This library, which is largely written in Python,is built upon NumPy, SciPy and Matplotlib.
7. **random** - it is a built-in library that provides various functions to generate random numbers and data.

The proposed methodology of this project is provided below. To provide a more complete understanding of the proposed machine learning method for malware detection, Figure 3.1 illustrates the workflow process from start to finish.

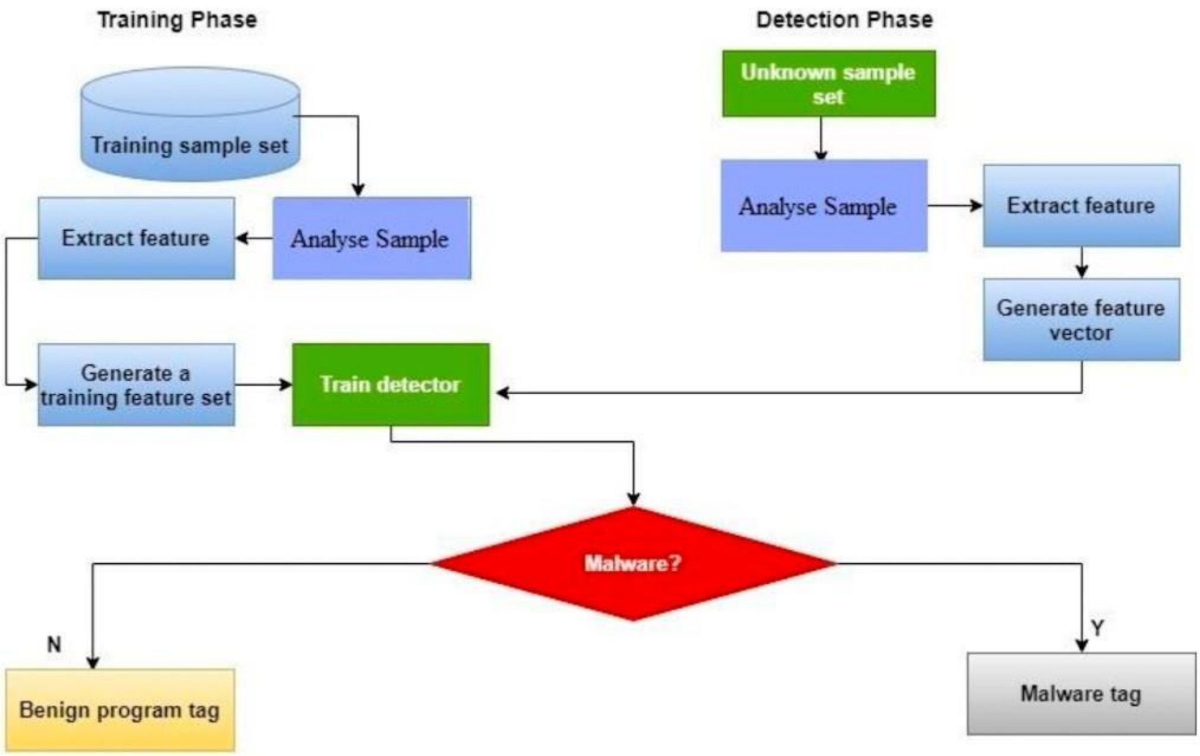


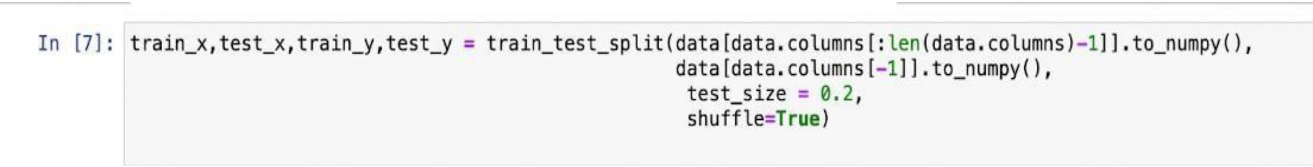
**Figure 3.1** Proposed ML Malware Detection Model [7]

## 3.2. Data Preprocessing

Data pre-processing is necessary for data cleaning and for preparing data for the machine learning model, which also improves the accuracy and effectiveness of the model. The-processing of data has been done in three phases, i.e., reading, checking, and cleaning the data. The dataset contains special characters like "?' and 'S'. The pre-processing data is cleaned by setting them to NaN values and using the dropna() function to remove all the rows containing NaN values from the data frame.

## 3.3. Data Classification

This code is using the train_test_split function from the sklearn.model_selection module to split the data into training and test sets. These sets will be used for training and testing a machine-learning model. It is important to split the data into training and test sets so that the model can be evaluated on unseen data and prevent overfitting.



```
In [7]: train_x,test_x,train_y,test_y = train_test_split(data[data.columns[:len(data.columns)-1]].to_numpy(),
                                                         data[data.columns[-1]].to_numpy(),
                                                          test_size = 0.2,
                                                          shuffle=True)
```

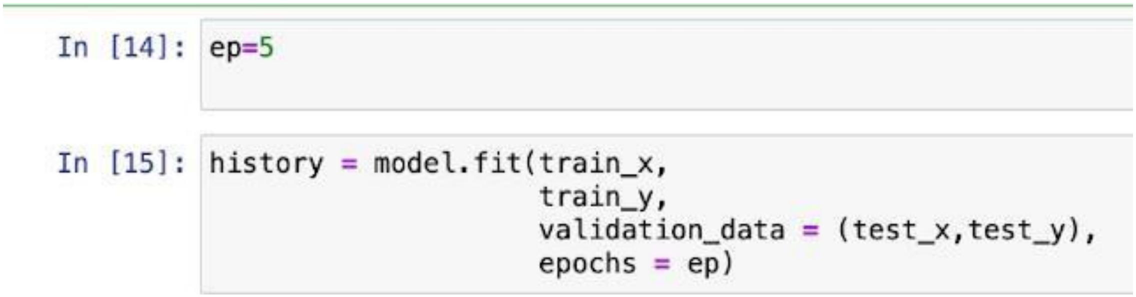
**Figure 3.2**

In the code, the splitting of the dataset is done in such a way that the training set contains all the features except the last column, which is the target variable, and the test set contains only the last column, which is the target variable. This will make it easy to feed the data to the model for training and testing.

<< test_size = 0.2 >> sets the proportion of the data that should be used for testing. In this case, 20% of the data will be used for testing and 80% will be used for training. The data is then shuffled to make sure the samples from each class are distributed randomly between the train and test sets.

## 3.4. Training the Model

A crucial component of machine learning algorithms that understand such data and memorize the information for future prediction is training data. A system that learns from humans and makes predictions for humans must be trained, and training data is  the foundation of every AI and ML project. [8]



```
In [14]: ep=5

In [15]: history = model.fit(train_x,
                             train_y,
                             validation_data = (test_x,test_y),
                             epochs = ep)
```

**Figure 3.3**

This code uses the fit() method of the Sequential model to train the model on the training data. train_x is the input features of the training set and train_y is the target variable of the training set. The validation set is used to monitor the performance of the model during training and to prevent overfitting. Each iteration over the training data is called an epoch.

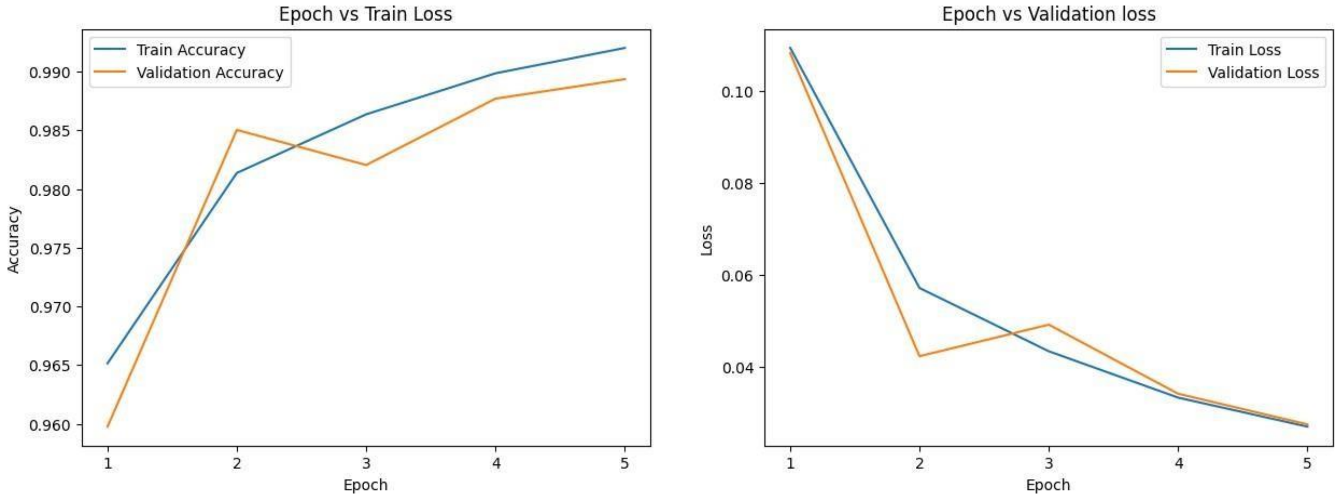


**Figure 3.4** Plots of the Training Accuracy and Validation Accuracy as a function of epochs.

The first subplot is a plot of the training accuracy and validation accuracy over a range of epochs. The second subplot is a plot of the training loss and validation loss over the same range of epochs. These subplots allow us to compare the performance of the model on the training and validation set during the training process and will allow us to identify if the model is overfitting or underfitting the data.

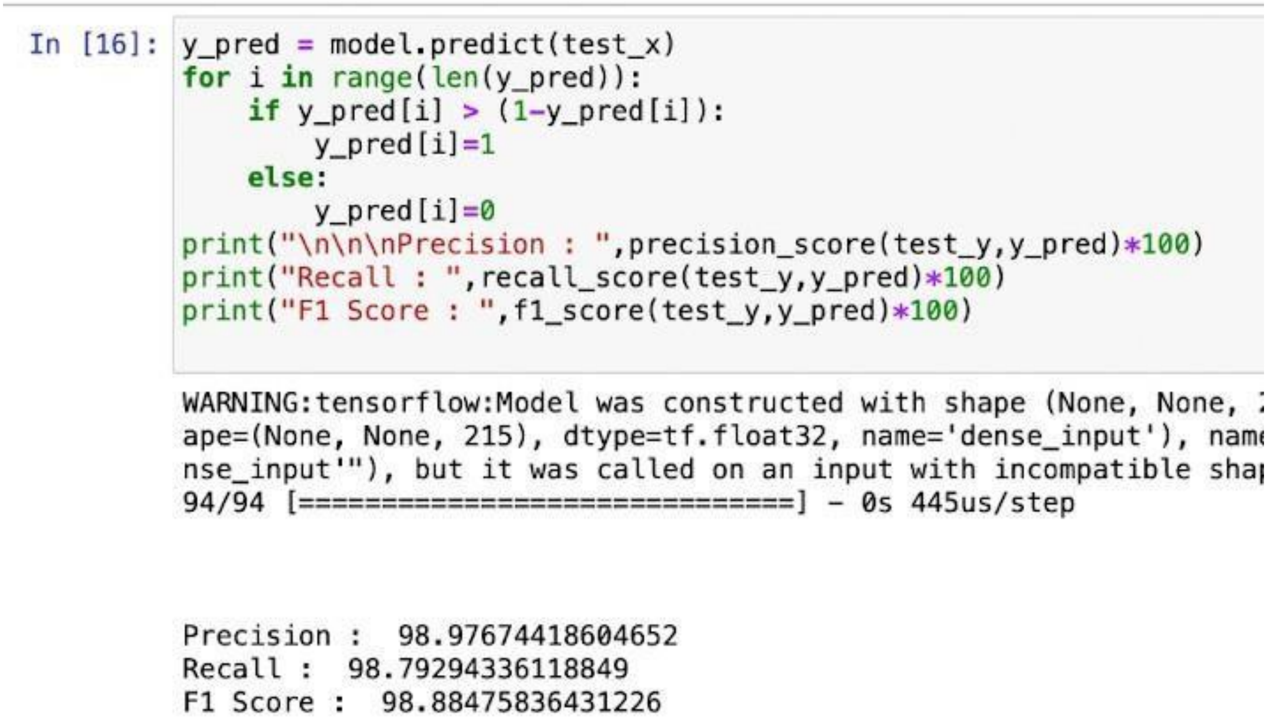
## 3.5. Testing the Model

The trained model is used to make predictions on the test set, and the model's performance is evaluated using three evaluation metrics: precision, recall, and F1 score. Precision is a metric that measures the proportion of true positive predictions among all positive predictions made by the model. It is a measure of the model's ability to avoid false positives. Recall is the proportion of true positives among the actual positives. It measures the ability of the model to find all the positive examples. F1 score is the harmonic mean of precision and recall, it balances the precision and recall and gives a single score that represents the overall performance of the classifier. It is useful when we want to seek a balance between precision and recall and is commonly used in imbalanced datasets.

# Chapter 4

## Result and Discussion

### 4.1. Result



```
In [16]: y_pred = model.predict(test_x)
         for i in range(len(y_pred)):
             if y_pred[i] > (1-y_pred[i]):
                 y_pred[i]=1
             else:
                 y_pred[i]=0
         print("\n\n\nPrecision : ",precision_score(test_y,y_pred)*100)
         print("Recall : ",recall_score(test_y,y_pred)*100)
         print("F1 Score : ",f1_score(test_y,y_pred)*100)
```

```
WARNING:tensorflow:Model was constructed with shape (None, None, 
ape=(None, None, 215), dtype=tf.float32, name='dense_input'), nam
nse_input'"), but it was called on an input with incompatible sha
94/94 [==============================] - 0s 445us/step



Precision :  98.97674418604652
Recall :  98.79294336118849
F1 Score :  98.88475836431226
```

**Figure 4.1** Code and Result of Accuracy

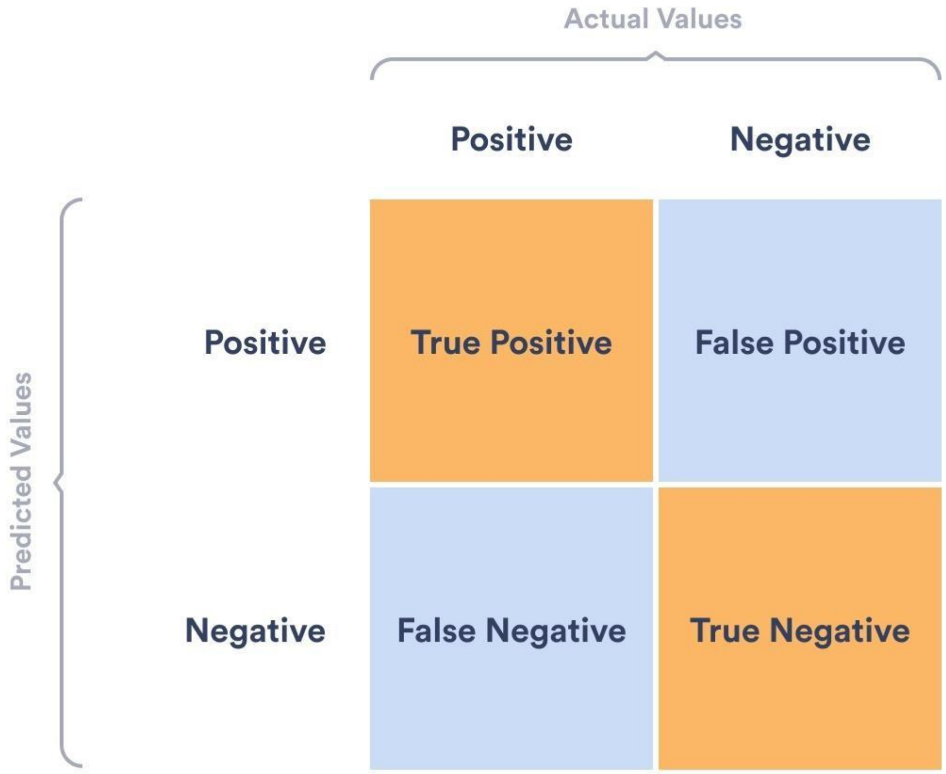
In our malware detection model, high precision indicates that the model is very good at identifying the positive class (i.e. identifying malware samples as malware) while minimizing the number of false positives (i.e. identifying benign samples as malware). High recall indicates that the model is very good at identifying all the positive classes (i.e. identifying malware samples as malware) while minimizing the number of false negatives (i.e. identifying malware samples as benign). A high F1 score indicates that the model has both high precision and high recall.

These evaluation metrics provide a comprehensive understanding of the model's performance on the test set and can be used to compare different models and fine-tune the parameters of the model.

## 4.2. Confusion Matrix

A confusion matrix is created to visualize the performance of the model on the test set. A confusion matrix is a table that is used to define the performance of a classification algorithm. It is a way to visualize the performance of the model by comparing the predicted labels to the true labels. It is used to understand the overall performance of the model and identify where the model is making errors.

The confusion matrix can be represented as follows:



| | | Actual Values | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted Values | Positive | True Positive | False Positive |
| | Negative | False Negative | True Negative |

**Figure 4.2** A Confusion Matrix

We have obtained the following confusion matrix as the result -

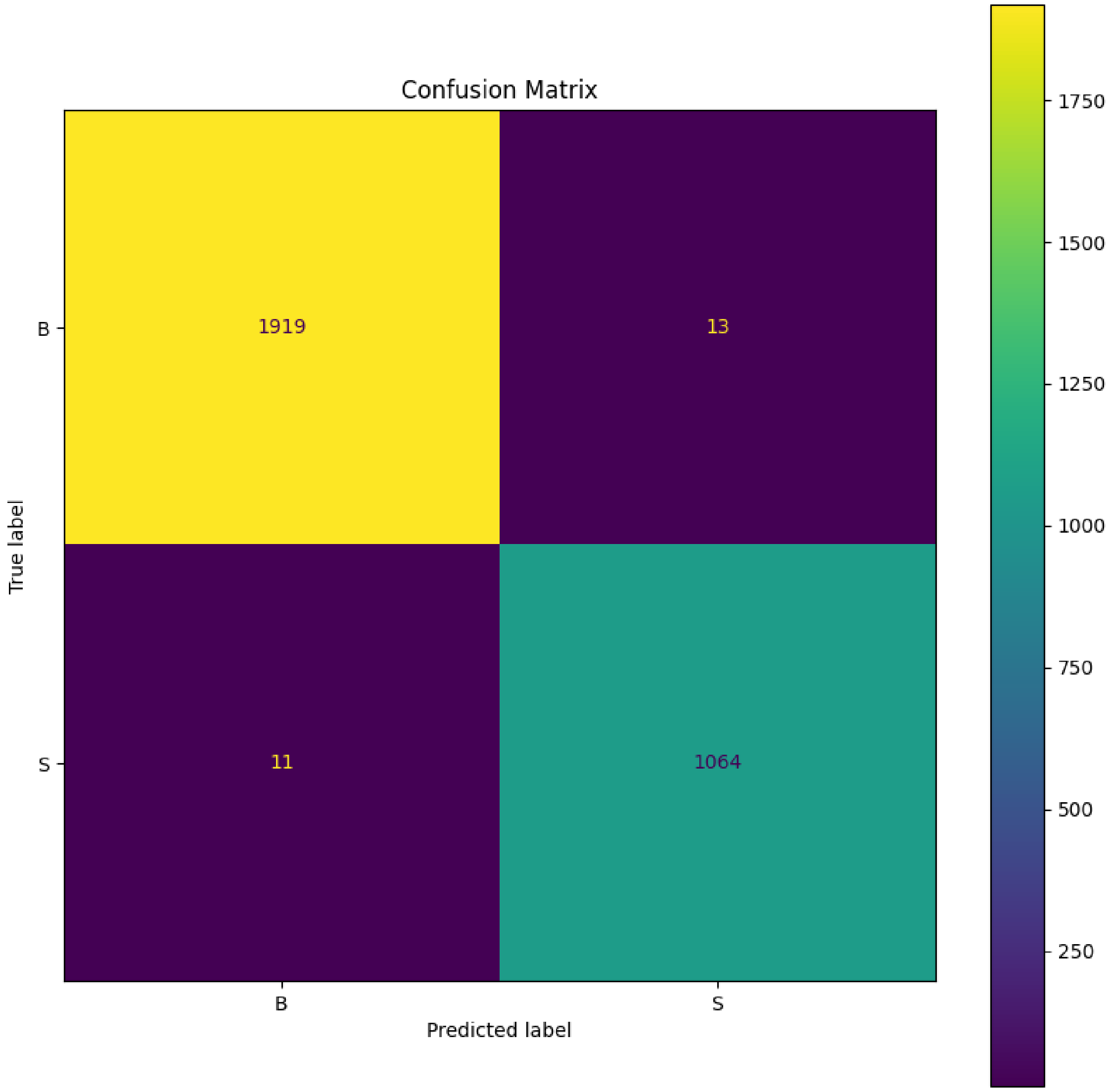


**Figure 4.3** Resultant Confusion Matrix

In this case, "B" represents the benign class and "S" represents the malicious class.

The above indicates the following:

- The model predicted 1919 true positives (i.e. actual malicious samples that were correctly predicted as malicious)
- The model predicted 13 false positives (i.e. actual benign samples that were incorrectly predicted as malicious)
- The model predicted 11 false negatives (i.e. actual malicious samples that were incorrectly predicted as benign)
- The model predicted 1064 true negatives (i.e. actual benign samples that were correctly predicted as benign)

## 4.3. Discussion

From the above confusion matrix, we can see that the model has high precision and recall, which is a good indication for a malware detection model. Hence, we can say that the model is performing well in terms of malware detection, as it can identify most of the malware samples correctly, while also having a low number of false positives.

# Chapter 5

# Conclusion and Future Work

## 5.1. Conclusion

In this project, we used a dataset consisting of 5560 malware samples and 9476 benign samples to train and evaluate a neural network-based malware detection model. The dataset was pre-processed by removing missing values and performing label encoding. We then split the dataset into training and testing sets. We evaluated the model's performance using some evaluation metrics including accuracy, precision, recall, and F1-score. We also plotted the model's performance over the training and validation sets using a confusion matrix.

The results of the model evaluation showed that the model performed very well in detecting malware, with a high precision, recall and F1-score. The confusion matrix provided a clear visualization of the model's performance and confirmed that the model was able to correctly identify most of the malware samples while minimizing the number of false positives.

In conclusion, the neural network-based malware detection model developed in this project demonstrated a high level of accuracy and robustness in identifying malware samples, making it a suitable approach for use in real-world malware detection systems. However, it's important to note that the dataset used in this project was from a specific period and this model would need to be retrained with the updated and diverse dataset to be able to detect new and unknown malware.

## 5.2. Future Work

The field of malware detection is an active area of research that is likely to continue to evolve in the future and many promising solutions can be used to improve the performance of malware detection models.

One potential area for future research is to explore the use of other machine learning algorithms such as Random Forest or Gradient Boosting. These algorithms have been known to produce better results in other classification problems and could potentially improve the performance of the model in identifying malware samples.

Another approach that can be used in the future is the use of unsupervised learning methods such as anomaly detection algorithms, which can  identify abnormal behavior or patterns in the data.  These algorithms can be used to detect unknown malware samples that have not been seen before and could be used in conjunction with supervised learning methods to improve the overall performance of the model.

In addition to these solutions, the use of big data analytics, cloud computing, and edge computing can be used to handle the large amount of data generated by malware detection systems and make the model more scalable. The combination of these solutions can also be used to develop a real-time malware detection system that can identify and block malicious activities before they can cause any damage.

## References

[1] Wallarm.com [Online] https://www.wallarm.com/what/what-is-a-cyber-attack

[2] Akhtar, M.S.; Feng, T. Malware Analysis and Detection Using Machine Learning Algorithms. Symmetry2022,14,2304. https:// doi.org/10.3390/sym14112304

[3] Liu, Kaijun & Xu, Shengwei & Xu, Guoai & Zhang, Miao & Sun, Dawei & Liu, Haifeng. (2020). A Review of Android Malware Detection Approaches Based on Machine Learning. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.3006143.

[4] Hoda El Merabet, Abderrahmane Hajraoui.(2019) A Survey of Malware Detection Techniques based on Machine Learning. (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 10, No. 1, 2019

[5] Hemant Rathore, Swati Agarwal, Sanjay K. Sahay and Mohit Sewak.(2019) Malware Detection using Machine Learning and Deep Learning. arXiv:1904.02441v1 [cs.CR] 4 Apr 2019

[6] For learning the Python Libraries used in this project - GeeksForGeeks

For learning Tensorflow - https://www.tensorflow.org/

For learning Scikit-learn - https://scikit-learn.org/stable/

[7] Akhtar, M.S.; Feng, T. Malware Analysis and Detection Using Machine Learning Algorithms. Symmetry2022,14,2304. https:// doi.org/10.3390/sym14112304

[8] Understanding the Importance of Training Data in Machine Learning by Roger Brown, TheAITechnology- https://medium.com/the-ai-technology/understanding-the-importance-of-training-data-in-machine-learning-da4235332904