National College of Ireland

Configuration Manual

MSc Research Project
Msc. in Cybersecurity

Annamalai Shanmugam
Student ID:X21222240

School of Computing
National College of Ireland

Supervisor:**Evgeniia Jayasekera**

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: **.....ANNAMALAI SHANMUGAM.....**

Student ID: **.....X21222240.....**

Programme **Msc in cybersecurity** ..... **Year:** **2023-24** .....

Module: **Msc Research Project/Internship** .....

Supervisor: **.....Evgeniia Jayasekera.....**

Submission Due Date: **21-12-2023** .....

Project Title: **.....CONFIGURATION MANUAL.....**

Word Count: **...900.....** **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: .....

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

Name

Student ID:

1 Introduction

This study, titled "A Comparative Analysis of Kernel-Based Support Vector Machines (SVM) and Convolutional Neural Networks (CNN) for Zero-Day Malware Detection," delves into the critical field of cybersecurity, specifically addressing the escalating threat of zero-day malware. It aims to evaluate and compare the effectiveness of SVM and CNN 1D algorithms in detecting such malware. This research stands out for its innovative approach and potential contribution to the development of more robust malware detection systems. The introduction sets the stage for a detailed exploration, outlining the methodology and the significance of the findings in the broader context of cybersecurity advancements.

2 System Specifications

Python, chosen for its extensive ecosystem of data science and machine learning libraries.

- Scikit-learn library for SVM implementation, offering comprehensive support for various machine learning algorithms.
- TensorFlow and Keras for the backend of the CNN 1D model, providing a high level of abstraction for building complex neural networks.
- Hardware Specifications: A high-performance computing environment equipped with multi-core CPUs is necessary to handle the intensive computation required for training and testing the models.
- GPUs with CUDA support are essential for the CNN 1D model to facilitate faster training through parallel processing.
- Data Handling: Adequate storage solutions are required to manage the large datasets involved in the training process.

2.1 Hardware Requirements

- Operating System: Windows 10, preferably the latest version to ensure compatibility with all required software and libraries.

- Processor: A powerful multi-core processor (Intel i7, i9, or equivalent AMD processors) to handle intensive computations.
- Graphics Card: A high-end GPU with CUDA support (such as NVIDIA GeForce RTX series) to accelerate the training process of CNN 1D models.

- RAM: At least 16GB, though 32GB or more is recommended for handling large datasets and

intensive computing tasks.

- Storage: Adequate SSD storage (at least 1TB) for fast data access and handling large datasets.

- Software Compatibility: Compatibility with Python, Scikit-learn, TensorFlow, and Keras, as well as other data science and machine learning tools and libraries.

2.2 Software Requirements

- Google Colab
- Python (Version 3.10)

3 Data Collection

The dataset used in the research on "A Comparative Analysis of Kernel-Based Support Vector Machines (SVM) and Convolutional Neural Networks (CNN) for Zero-Day Malware Detection" is characterized by the following attributes:

- Content: It includes malware binaries and legitimate files. The malware samples represent software designed to disrupt, damage, or gain unauthorized access to computer systems, while the legitimate files are harmless and useful software.

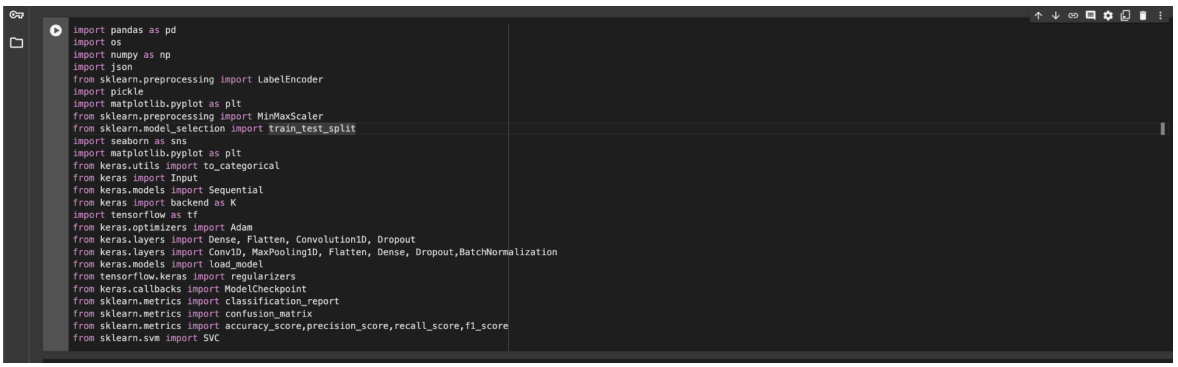
- Analysis Conducted: Detailed statistical analysis was performed on these files, notably the extraction of Portable Executable (PE) information and the calculation of entropy in different sections of the files. These are key indicators of file behavior and security traits.

- Dynamic Nature: The dataset is unique in its dynamic nature, with the potential addition of new data such as zero-day viruses as the research progresses. This evolving aspect is designed to test the adaptability and robustness of anti-malware algorithms, simulating the real-world challenges faced by anti-malware software giants.

- Sourcing Malware Samples: The malware samples were sourced from various online repositories, including security research databases and anonymized collections from cybersecurity firms.

- Practical Relevance: This dataset serves as a valuable tool for developing and testing cybersecurity solutions, and it provides a practical learning experience that mirrors the high-pressure environment of the cybersecurity industry.

These libraries that were used in our study include



```
import pandas as pd
import os
import numpy as np
import json
from sklearn.preprocessing import LabelEncoder
import pickle
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras import Input
from keras.models import Sequential
from keras import backend as K
import tensorflow as tf
from keras.optimizers import Adam
from keras.layers import Dense, Flatten, Convolution1D, Dropout
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout,BatchNormalization
from keras.models import load_model
from tensorflow.keras import regularizers
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
from sklearn.svm import SVC
```

Data Handling and Analysis:

- pandas (imported as pd): Essential for data manipulation and analysis, particularly useful for handling structured data like CSV files.
- numpy (imported as np): Provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- matplotlib.pyplot (imported as plt): Used for creating static, interactive, and animated visualizations in Python.
- seaborn (imported as sns): An advanced visualization library based on matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.
- os: Interacts with the operating system, used for file and directory operations.

Data Preprocessing:

- sklearn.preprocessing.LabelEncoder: Converts categorical labels into a numeric format, making them readable and processable by machine learning algorithms.
- sklearn.preprocessing.MinMaxScaler: Normalizes the dataset within a particular range, often required for optimized performance of machine learning algorithms.
- pickle: Implements binary protocols for serializing and de-serializing a Python object structure, useful for saving models or other large data structures.

Model Building and Training (Keras and TensorFlow):

- keras.utils.to_categorical: Converts a class vector (integers) to binary class matrix, necessary for classification tasks.
- keras.Input, keras.models.Sequential, keras.backend as K, keras.optimizers.Adam, keras.layers, and keras.callbacks.ModelCheckpoint: These components from Keras are used for building and training neural network models, including setting up the layers, activation functions, and optimization strategies.
- tensorflow (imported as tf): Provides the backend for Keras and additional functionalities for creating complex machine learning models.
- tensorflow.keras.regularizers: Applies regularization techniques to the model, which can prevent overfitting.

Model Evaluation and Metrics:

- sklearn.metrics: Includes functions like classification_report, confusion_matrix, accuracy_score, precision_score, recall_score, and f1_score, which are crucial for evaluating the performance of the machine learning models.
- keras.models.load_model: Used for loading a saved Keras model.

SVM Implementation:

- sklearn.svm.SVC: Part of scikit-learn's support vector machine (SVM) library, used for implementing the SVM algorithm for classification tasks.

Each of these libraries brings essential functionalities required for different stages of the research, from data preparation and processing to model development, training, and evaluation. Their combined use enables a comprehensive approach to analyzing and modeling the data for effective malware detection.

4 Data Pre-processing:

The data pre-processing steps carried out in the research on included several key procedures:

- Duplicate Removal: Identifying and removing duplicate files from the dataset to prevent bias.
- Data Balancing: Employing the Synthetic Minority Over-sampling Technique (SMOTE) to balance the dataset. This step is particularly important in malware detection, where dataset imbalance is a common issue.
- Irrelevant Data Filtering: Filtering out non-executable files and irrelevant data to focus exclusively on potential vectors for malware.
- Standardization: Standardizing the collected executables to a consistent format for feature extraction. This included normalizing file sizes where appropriate.

These pre-processing steps were essential to ensure the quality and reliability of the data, making it suitable for effective analysis and modeling using SVM and CNN 1D algorithms.

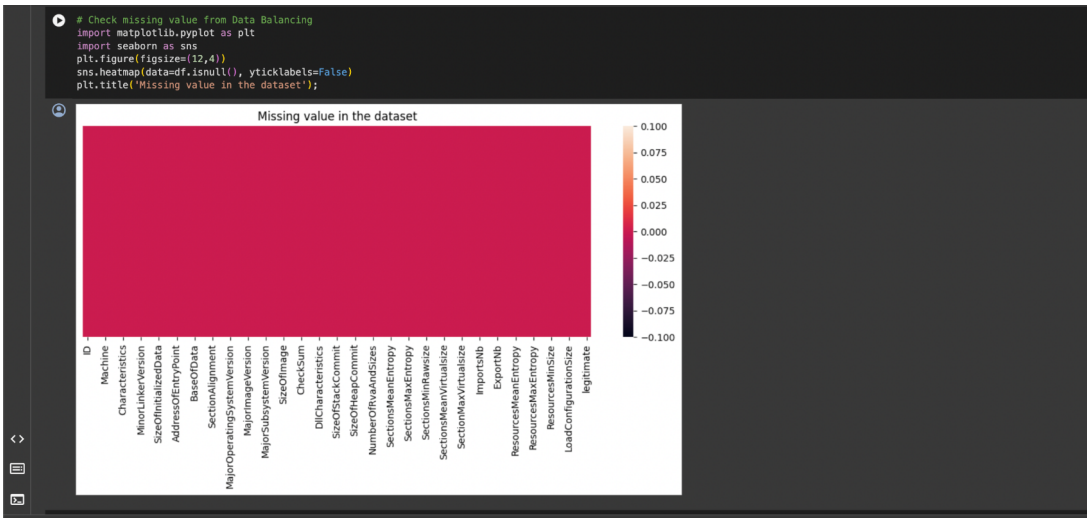


Fig 2: Fixing the missing value list

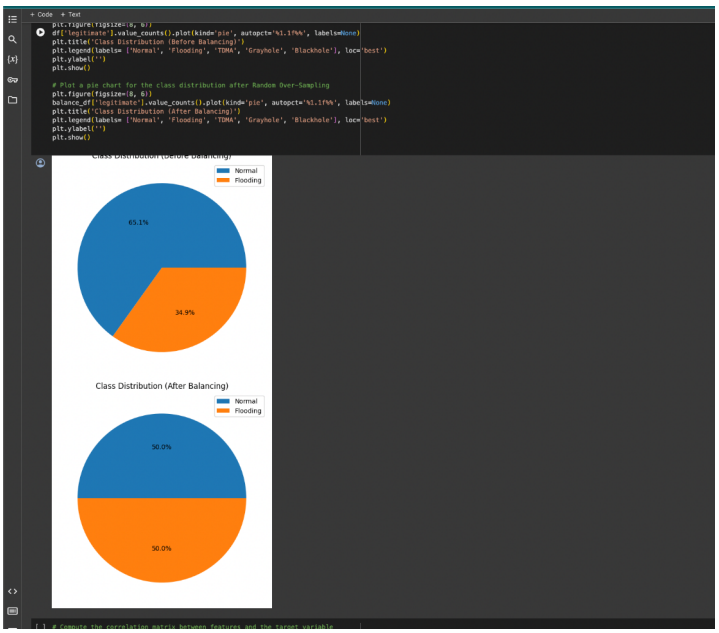


Fig 3: Balancing the data using SMOTE

- Identifying Imbalance: The research first identified the imbalance in the dataset, where one class (e.g., malware files) was underrepresented compared to another class (e.g., legitimate files).

- Generating Synthetic Samples: SMOTE works by creating synthetic samples from the minority class instead of creating exact copies. It does this by taking samples of the minority class and creating new, synthetic samples that are similar but slightly altered. This is typically achieved by finding the k-nearest neighbors of a minority class sample and interpolating between these neighbors to create a new sample.

- Balancing the Dataset: By adding these synthetic samples to the minority class, SMOTE helps in balancing the class distribution. This balanced dataset can then be used for training machine learning models, ensuring that the models do not become biased towards the majority class.

- Improving Model Performance: Using a balanced dataset helps in improving the performance of the machine learning models. It ensures that the models are equally sensitive to both classes and can generalize better when predicting on new, unseen data.

- In the context of the research on SVM and CNN for malware detection, using SMOTE for data balancing ensured that the models developed were robust and not biased towards predicting one class (like benign files) more accurately than the other (like malware files). This is crucial in cybersecurity applications where missing a malware instance (false negative) can be highly detrimental.

1. Data Correlation with target variable

Correlation Scores: The correlation scores are calculated between each feature and the target variable. The scores represent the strength and direction of the linear relationship between each feature and the target. A score close to 1 or -1 indicates a strong positive or negative correlation, respectively, while a score close to 0 indicates a weak or no linear correlation.

Sorted Features:

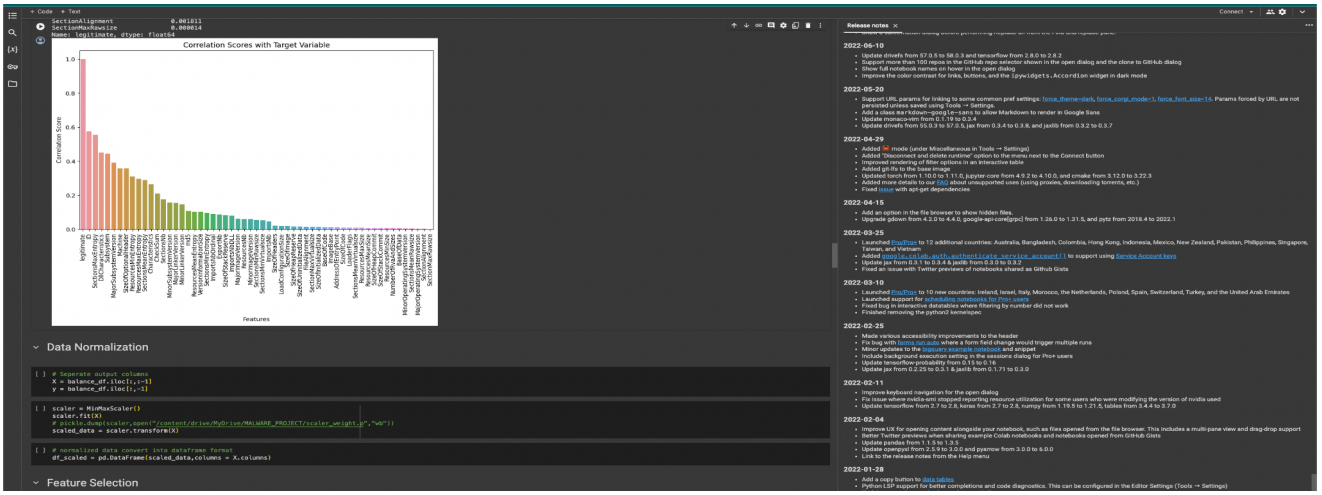


Fig 3: The features are sorted based on the absolute values of their correlation scores in descending order.

2. Feature Selection:

The feature selection process in the research involved several systematic steps to ensure the most informative features were used for the machine learning models:

Automated Feature Extraction: Custom scripts were developed to automate the extraction of features from a large volume of data. This process was critical for handling the complex and extensive datasets involved in the study.

Dimensionality Reduction: Techniques such as Principal Component Analysis (PCA) were employed. PCA is a method used to reduce the number of features in a dataset by transforming the original features into a new set of features (principal components) that retain the most significant variance in the data.

Information Gain: This step involved ranking the features according to their information gain with respect to the classification task. Features that provided the most insight into the data's class labels were prioritized. Information gain is a measure of how well a feature separates the classes in terms of the information it provides about the class distinction.

Mutual Information: Mutual information metrics were calculated to assess the dependency between features and the classification outcomes. This helped in identifying features that had a strong relationship with the target variable, thereby being more relevant for the models.

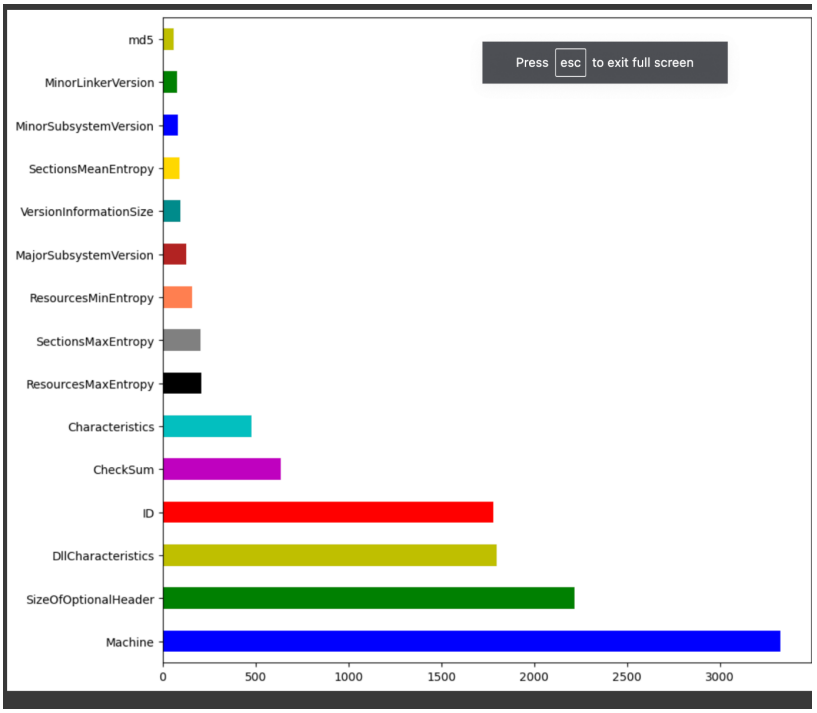


Fig 4: The selected features for training model

The Data is split into train and test with a 80:20 split.

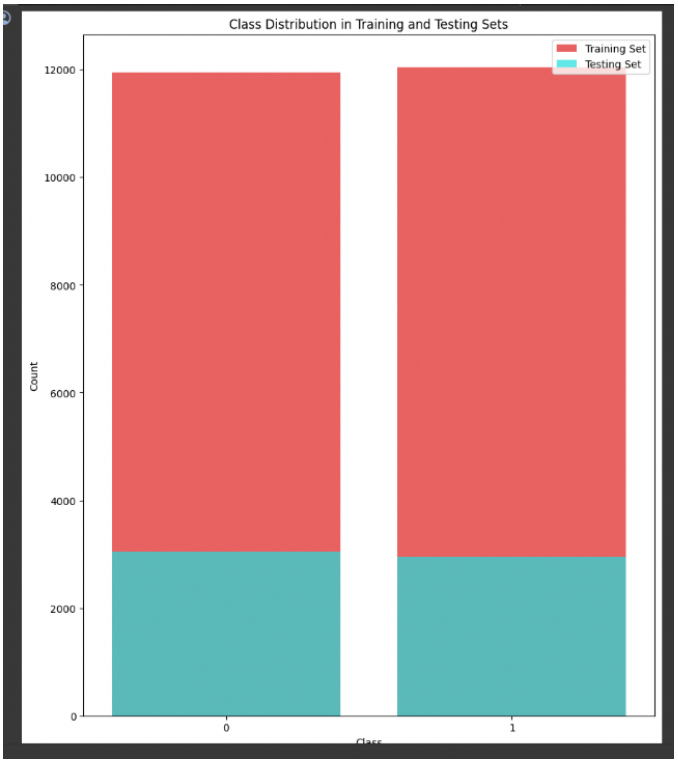


Fig 4: Visualization of training and test data split

5. Model Development:

4.1 CNN 1D Modelling:

Layer Configuration: The CNN 1D model was constructed with several convolutional layers. These layers are fundamental in CNN architecture, as they perform the convolution operation, extracting features from the input data.

Pooling Layers: Following the convolutional layers, pooling layers were included. Pooling layers are used to reduce the spatial dimensions (width, height) of the input volume for the next convolutional layer. They are essential for decreasing the computational load and for extracting dominant features, which provides robustness to the model.

Dropout for Regularization: Dropout layers were incorporated as a regularization technique. Dropout helps prevent overfitting in neural networks by randomly setting a fraction of input units to 0 at each update during training time.

Dense Layer for Classification: A dense layer was added for the purpose of classification. In neural networks, dense layers are fully connected layers where each input node connects to each output node.

Hyperparameter Optimization: Key hyperparameters such as the number of filters, kernel size, and learning rate

were optimized. This was achieved through a combination of manual tuning and automated methods like random search. Hyperparameter tuning is crucial for improving model performance and achieving more accurate results.

Evaluation Methodology: The performance of the CNN 1D model was evaluated using a hold-out validation set, which was crucial for assessing the generalization capability of the model to new, unseen data.

Performance Metrics: Various metrics were used to evaluate the model's performance, including accuracy, precision, recall, F1 score, and Area Under the Receiver Operating Characteristic Curve (AUC-ROC). These metrics provided a comprehensive view of the model's performance across both positive and negative classes.

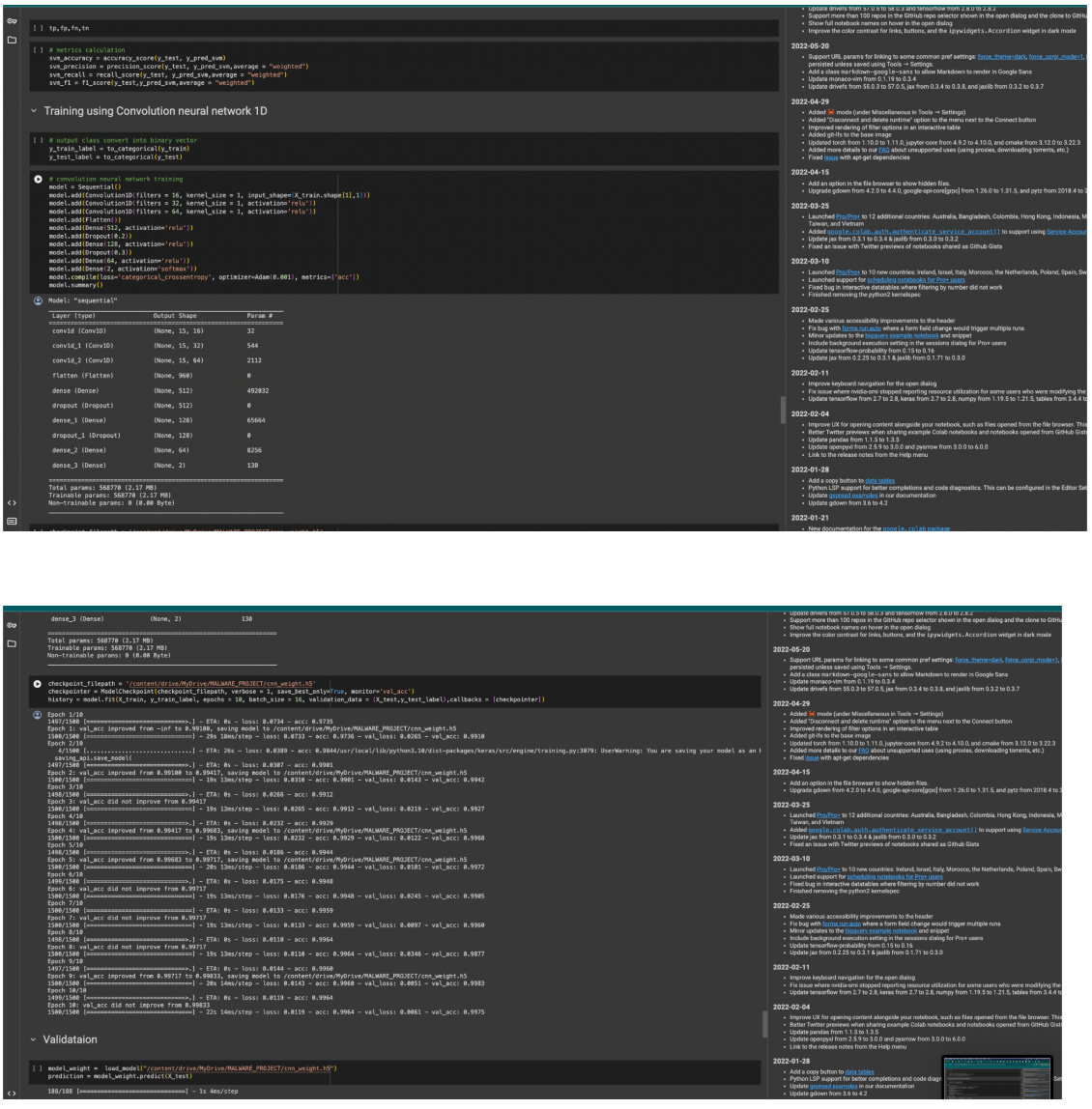


Fig 5: Implementation of CNN 1D algorithm

4.2 SVM Algorithm Modelling and report generation:

The SVM (Support Vector Machine) algorithm implemented in the research for zero-day malware detection involved a structured and methodical approach:

Input: The SVM algorithm processed preprocessed feature vectors derived from the dataset. These feature vectors represented the characteristics of the malware samples in a format suitable for machine learning analysis.

Process:

Mapping of Input Vectors: The feature vectors were mapped into a high-dimensional feature space. This is a typical characteristic of SVM, where it transforms the input data into a higher-dimensional space to make it easier to find a separating hyperplane.
Finding the Optimal Separating Hyperplane: The core of the SVM algorithm involves finding the hyperplane that best separates the classes (in this case, benign versus malicious software). The optimal hyperplane is the one that maximizes the margin between the classes.
Classification Decision: Based on the position of the data points relative to the hyperplane, the SVM makes a classification decision, determining whether a sample is benign or malicious.
Output: The output of the SVM algorithm was a binary classification indicating whether a sample in the dataset was benign or malicious.

This implementation highlights the SVM algorithm's strength in handling high-dimensional data and its effectiveness in binary classification tasks, which are critical in the context of malware detection.

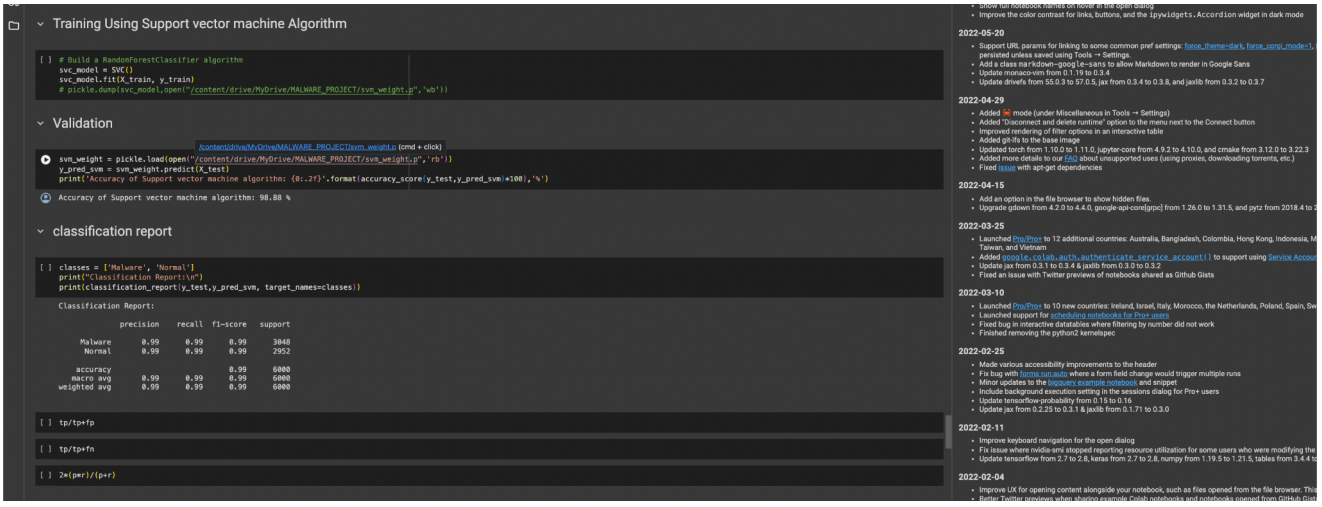
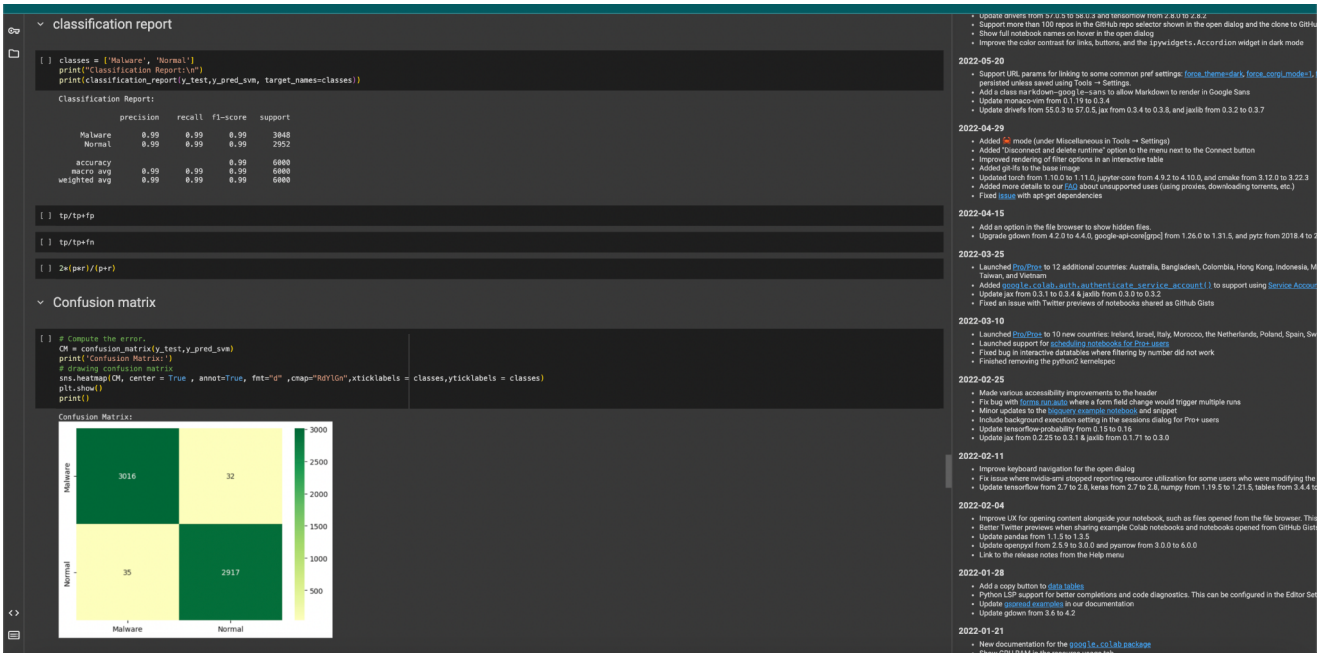


Fig 5: Implementation of SVM algorithm

6. Report Generation of Output:



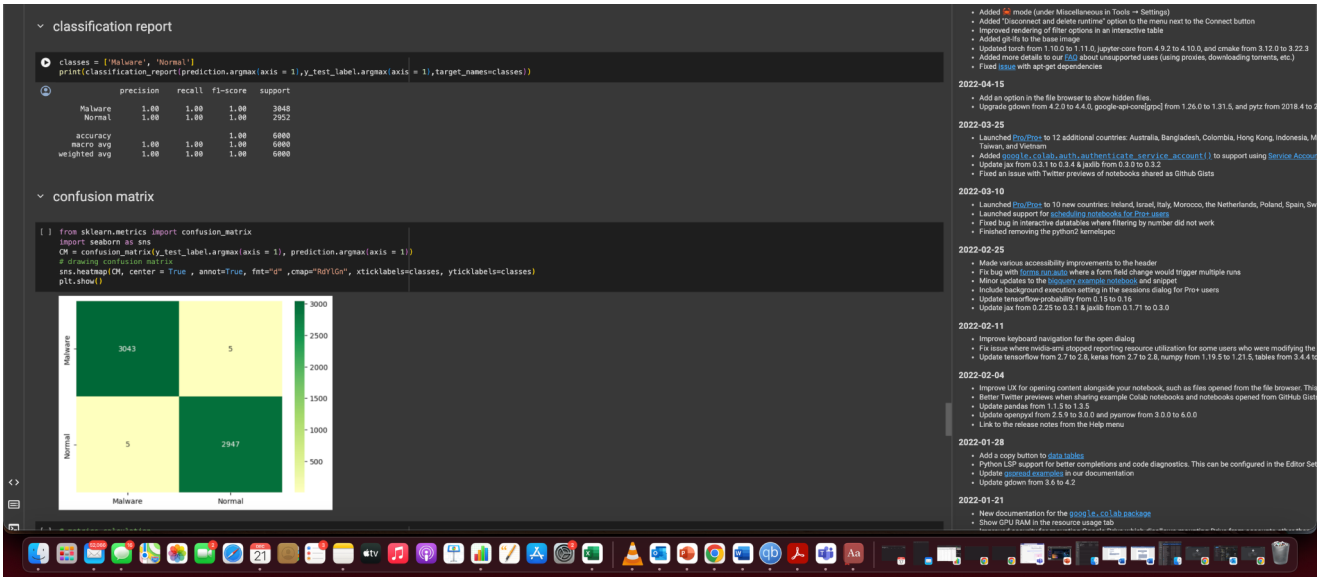


Fig 6: Results of SVM algorithm and CNN 1D algorithm

References:

Malware detection (2018) Kaggle. Available at: https://www.kaggle.com/competitions/malware-detection/data

“colab.google,” *colab.google*. https://colab.google/