Anti-malware: Permission-based anti-malware detection

School of Computing | Computer Science with AI

University of Brighton | 06/05/2022

Jordan Nelson
School of Architecture, Technology and Engineering, University of Brighton, Lewes Road, Brighton, BN2 4GJ, United Kingdom
J.Nelson7@uni.brighton.ac.uk

Nikolaos Polatidis
School of Architecture, Technology and Engineering, University of Brighton, Lewes Road, Brighton, BN2 4GJ, United Kingdom
N.Polatidis@Brighton.ac.uk

Roger Evans
School of Architecture, Technology and Engineering, University of Brighton, Lewes Road, Brighton, BN2 4GJ, United Kingdom
r.p.evans@brighton.ac.uk

Abstract

The android platform has become populous with malware given the rise of the internet and mobile devices over the years, thus also giving the rising necessity and concern to protect ones-self. The rise of anti-malware technology has correlated with this threat, but so has faux anti-malware applications attempting trojan like attacks. A user may download said fake app and rely on the user agreeing to awarding certain permissions, that will allow the app to collect data and infect devices while not actually performing its disguised apparent role of removing malware. Expanding on existing work, we will be exploring the idea of identifying malware that disguises itself as anti-malware software. Using machine learning, the goal is to detect this malware using a permissions-based dataset. By using appropriate libraries and fine-tuning algorithms to train and test a neural network model and exploiting different classifiers to yield the highest performance against a given metric.

Contents

1. Introduction

Expanding on the work, HamDroid: Permission-based harmful Android anti-malware detection using neural networks (Seraj, Khodambashi, Pavlidis and Polatidis, 2022) we will be exploring and researching ways to expand and improve the approach. Based on the Android platform, currently the largest mobile platform in the world. Making up for 70% (Statcounter Global Stats, 2022) of the entire global market. This platform has also become the prime target for malware attacks and software. Because of this, a plethora of virus-scanning/detecting apps have flooded app-stores as the threat of malicious attacks increases over time. According to a 2019 report, (Android Test 2019 - 250 Apps, 2022), two-thirds of all anti-virus apps on the android platform are faux and do not function as advertised.

As discussed in the report, only 80 out of 250 anti-virus apps were able to detect over 30% of the malicious apps tested with zero false alarms. Following from this, we may say that the majority of anti-virus-esque apps use the listing approach in detecting malware, whether it be whitelisted or blacklisted software. The code of the software being scanned is seldom reviewed. This means that a majority of so-called anti-virus applications compare the package name of installed apps to that of the whitelist and deem them malware should it not be a member of said list. Looking at this, it would appear that the developers of these anti-virus apps seem to be amateur developers or software manufacturers not seeped in the cyber security world. In addition, several apps inherited the same user interface, presented a plethora of advertisements, and requested permissions and consent that would facilitate the collection of user data. According to recent studies, anti-malware solutions have been primarily machine learning based (Razgallaha, Khourya, Hallé and Khanmohammadi, 2021), however current existing methods that are commercial and freely available do not. Ergo we may assume that the technology to detect this anti-malware either does not exist or is not utilised.

This projects aim is to identify anti-malware that disguises itself as anti-malware software, by using machine learning techniques to distinguish between malware and non-malware using software permissions as the prerequisite. Given the vital role of anti-malware technology in today's cyber society, there comes the necessity to seek and provide an approach to identify harmful android anti-malware. The harmful anti-malware we will be looking at may be defined as: an application or software that inherently appears and to some degree, may even act like existing anti-malware software, to disguise itself as the genuine artifact. But is in fact, malware designed to act maliciously towards another computer, computer system or user. To achieve the proposed idea, our hypothesis will be that the anti-malware in question, may be identified purely based on the permissions that the software requests access, *e.g., requesting access to camera functions.*

This article attempts the following:

- Make modifications to the dataset (Seraj, 2021), to improve performance and quality of the data

- Use the dataset, to identify and build different neural network architectures to identify the malware
- To evaluate the new methodology using different metrics to prove the proposed methodology is indeed an improvement

2. Background

An article (Gorelik, 2020) that discusses the NGAV system, made the claim that traditional anti-virus software, and specifically machine learning based anti-virus software can't detect fileless malware. It goes on to state that the algorithms it talks about reviews code and behaviour. Thus, this supported the argument already made that the anti-malware posing as real anti-malware, would be able to slip through the net when using the traditional techniques of detecting malware. This put an emphasis on why the development of permission-based detection may be a better approach.

In a paper (Tchakounte, 2014) that reviewed permission-based malware detection, explicitly stated that when looking at this technique, one should look at more than just the 130 official permissions of android, and instead look deeper at a much wider range of permissions that could indeed cause harm to a user or device but where otherwise overlooked as permissions and not included in the 130. This meant that for the purpose of this project, and specifically with the idea of reducing the size of the dataset to remove redundant permissions, it was incredibly precisely carried out and reviewed manually rather than automatically executed by software. To have a human logically reason whether a permission is truly redundant based on another. As well as this a paper about coevolution of Malware and anti-malware (Sen et al., 2018) stated that new malware is being developed daily. With each new malware presenting new security risks, and anti-malware technologies being vulnerable to yet unknown risks, it emphasises how inept anti-malware is at doing its intended job. As discussed in section 1, the technologies to detect anti-malware simply doesn't exist or isn't utilised in modern approaches.

A paper that discussed malware detection techniques (Idika et al., 2007) suggested that the rising cases of malware could be considered an epidemic. Malware is constantly evolving and trying to hide itself. With some even making class-file names appear legitimate (Zhou et al., 2012) to try and disguise themselves reinforces the idea that traditional methods of looking at code, signatures and other relevant methods can be duped and bypassed by traditional approaches. Thus, meaning that the proposed method of detecting via permissions is a credible and feasible approach when compared to other methods. Further to this, we may consider the justification of this approach by looking at the 3 types of detection (Aung et al., 2013) attack/invasion detection, misuse detection and anomaly detection. With misuse detection being signature based (Aslan et al., 2020) we may rule this approach out, as we have already discussed why this is not a good approach in section 1. Anomaly detection may be referred to as behaviour-based detection and this issue with this is it typically monitors behaviour over time, which necessitates the creation of vast volumes of data to successfully train good and bad behaviour. The approach proposed in this report does not. Instead, we can limit our data to a finite set of permissions, to detect the anti-malware without having to rely on its behaviour. The other draw back to that is also that malware can in some instances, behave like the software it is imitating, thus bypassing behaviour detection altogether.

The previously stated paper in section 1, *Hamdroid*, goes into detail about the different approaches and techniques currently prevalent. Going on to state that all current anti-malware technology deals with malware in the same fashion, including that of the anti-malware described in that, and this report. With both named static and dynamic approaches functioning in different ways, it is claimed that regardless of the methodology, this specific type of anti-malware can still sneak through detection. Because of this, it necessitates the need for a robust approach to detecting it. And concluding that their MLP approach provides a simplified and accurate approach when compared to the likes of others that are based in ensemble or more complex approaches, thus meaning the detection using permissions, is feasible in terms of speed and practicality.

This strongly inspired this project as it highlighted the prevalent issue with the current technologies in the market and real world. The paper was well backed and reasoned for evidence and need of such a technology but left open the possibility of exploring this approach further. It failed to explore specific deep learning techniques as opposed to traditional machine learning. Ergo this enticed the idea that deep learning for malware detection may have been oversighted in the Hamdroid report and was yet, unexplored in this context.

In the research of machine learning techniques such as the cases of KNN, decision trees and some applications were looked at (Ertel, Black and Mast, n.d.), as well as artificial neural networks (ANN's), ensemble learning and classification using linear models (Russell, Norvig and Chang, n.d.) as a reference point for their theoretical and practical functionality and application both in this project as well as the real world helped develop an understanding of the technologies used within this project and helped identify potential candidates for the planning of the project in terms of classification and the respective approach to classification problems. It was useful to identify which algorithms would be beneficial to explore and which ones would be redundant based on the type of problems that they are useful for and their real-world applications. After consideration, the CNN and Dense model were settled on as the approach with other common classifiers for comparison of performance. With *Hamdroid* and other approaches (Arif et al., 2021) not utilising deep learning, specifically CNN or the dense model. It justified the exploration of the proposed methodology to diverge from typical machine learning techniques and explore a true deep learning approach.

3. Methodology

3.1 Programming Language

The programming language of choice for this project, is Python. Python is an object-orientated language most often used for rapid application development. This, in part, is due to it taking a lot less time to develop a piece of software using Python. Python programs can typically expect between three to five times (Comparing Python to Other Languages, 2022) less lines of code when compared to say, the Java equivalent. This is due to Python being dynamically typed, meaning the declaration of types for variables is not necessary. Thus, this type-system is fluid and reduces the physical coding-time needed.

The other added benefits of Python for this project are that for some-time now it has been the language of choice for Machine Learning (ML). This is due to an array of libraries that support Machine Learning such as Scikit-Learn and TensorFlow or admission the Keras

API. This support for Machine Learning has meant that Python has a vast amount of API's (Application Programming Interface), for algorithms and evaluation components. Adding all of this up, not only is Python famously easy to use, but because of the vast support for Machine Learning it makes Python quicker to build and easier to find support for libraries than found in other languages for the equivalent purpose and implementation within this project.

3.2 Integrated Development Environment (IDE)

In the programming world, there are many environments to choose from. Each with its own interface and set of built-in libraries and support for languages. The Integrated Development Environment: IDE for short, we will be using for all Python programming in the constraints of this project will be PyCharm. PyCharm community edition is a free, open-source Python dedicated IDE that is specifically geared towards full support for Python as well as all the tools and libraries prior mentioned as well as those mentioned in sections 3.3 and 3.4. PyCharm is an extremely popular environment developed by JetBrains, with reportedly 99 Fortune 100 companies, 92 Fortune Global 100 companies, and 97 Forbes Top 100 Digital Companies (Our Customers: 92 of Fortune Global 100 Companies - JetBrains, 2022) using the platform today including the likes of Google, Twitter and tech company HP. PyCharm is a professional, industry used environment, with support for Machine Learning that will provide us with the tools and means to execute this project with ease. For more information you may find it here: https://www.jetbrains.com/pycharm/

3.3 Scikit-Learn

Scikit-Learn is a Machine Learning API with library support that enables supervised and unsupervised learning, exposing a very wide variety of machine learning based algorithms. One journal (Pedregosa et al., 2011) found that Scikit-Learn outperformed other Machine Learning libraries in 4 out of 6 tests in terms of speed of computation. Scikit-Learn provides us with many useful algorithms in which we may build and test our given dataset against. The algorithms we will concern ourselves with that we may find within Scikit-Learn are, K-Nearest Neighbour (KNN), Decision Tree (DT), Random Forest (RF) and Ensemble classifiers. These will be useful for the classification task we have set for this project.

3.4 Keras

Keras is an open-source deep learning Application Programming Interface (API) built on-top of TensorFlow 2. It is used for solving Machine Learning problems with a specific focus on deep learning. It was designed to enable fast experimentation meaning that it provides the perfect requirement in this research-based project of being able to achieve results as fast as possible. The original idea of this project came from the prior mentioned report in section 1, to use deep learning as well as the machine learning algorithms mentioned in section 3.3 to achieve a more desirable result. The architecture we will be using from the Keras API will be that of the Convolutional Neural Network (CNN) and Dense architecture to train and test using our dataset and compare this against different metrics.

3.5 Project Planning

Planning for this project was somewhat abstract. As this project is heavily research based, and with the nature of the project itself, it is difficult to plan for.

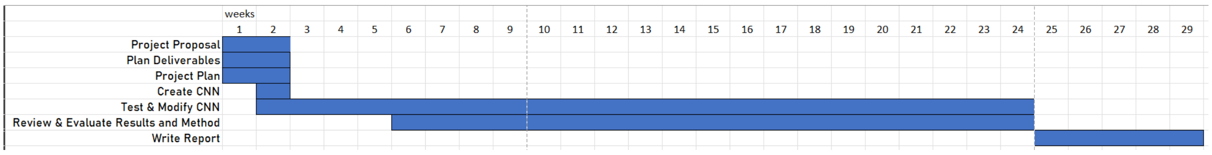

Figure 1: Gantt chart proposal

As we can see from figure 1, this is what the time management could look like for what is proposed for the project. But this is not an accurate representation of how the project ran or will run. With the project being based on machine learning while also being quite research based, there is no way of planning precisely how long it will take to achieve feasible results. Once run, a result of accuracy will be given, which can then be acted upon. In an ideal world, the result is accurate, and no action is needed on the first iteration. But this will most likely not be the case in which instance the whole process will be looped and re-iterated with different parameters each time to try and reach a desirable result.

Unfortunately, the plan may fail, and so a new model may need to be used to try and achieve a feasible result.

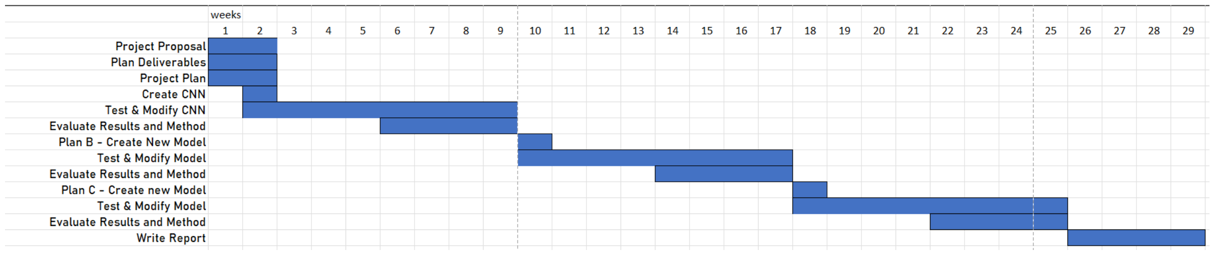

Figure 2: Gantt chart alternate

This may then cause the gantt chart to look something similar to figure 2. The issue, as previously mentioned is that there is no way of knowing in advance whether the project will play out as seen in figure 1, or figure 2 so both eventualities were planned for. This is because of the way a machine learning project functions, that the time it takes to run the program with one set of parameters is determined on a few factors such as the size of model, the dataset and the speed of the computer running the algorithm.

Going ahead with this, it was reasonable to plan for both eventualities as it is completely unpredictable on what would happen. Because the project is both researched based and in the realm of machine learning, it was sensible to be prepared for the proposed methodology to fall flat and have a plan B or even plan C in place, just in case.

Trello was used as a management tool, to keep track of tasks that needed to be completed and as you can see from figure 3, which was taken from before this report had concluded, tasks were split between seven categories and periodically updated depending on their status.

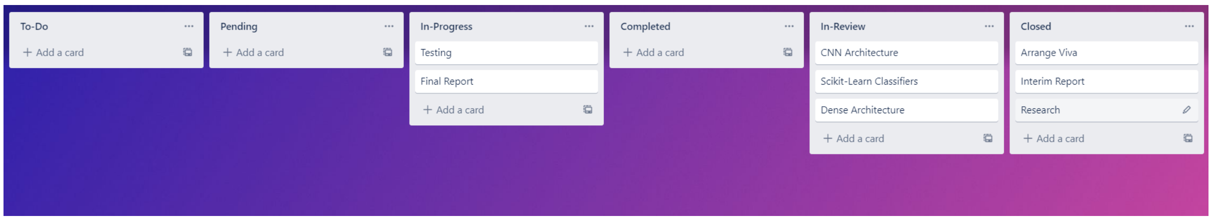

Figure 3: The Trello board

The coding of this project, due to previously mentioned benefits of using Python and its incredible support for machine learning, will make up for a small portion of the project time as it is easy to produce a machine learning model very quickly. The biggest concern will be running the software to achieve results and where necessary make modifications. This means time management will vary across the duration of the project as the testing and analysing phase will consistently vary.

3.6 Data Manipulation

The original *HamDroid* dataset, mentioned in section 1, that was used for this project had around 1,200 records of malware and non-malware. Each record having 328 individual permissions. The permissions represented software permissions that an application requests access to on installation. As can be seen from the prior mentioned report, the time recorded to execute their proposed algorithm was considerably long, at times exceeding an hour. In an experiment to try and reduce this time, it was conceived that some data processing may assist with this issue.

Reviewing the dataset, it was clear to see that of the 328 permissions, many were extremely alike. Often some permissions were even redundant from a logical point of view due to association. For example, if an application has permission to use the camera, all sub permissions of the camera would not need to be included in the dataset. Thus, this posed the hypothesis that reducing the size of the dataset, may decrease performance time at a little to no cost of accuracy.

4. Product Description

4.1 Convolutional Neural Network

Convolutional Neural Networks (CNNs) (Le Cun et al., 1990), are a type of biological inspired neural network. They are made up of neurons with learnable weights and biases. Each neuron will receive multiple inputs which will then take a given weighted sum across them, where it will then pass through an activation function and then provide an output.

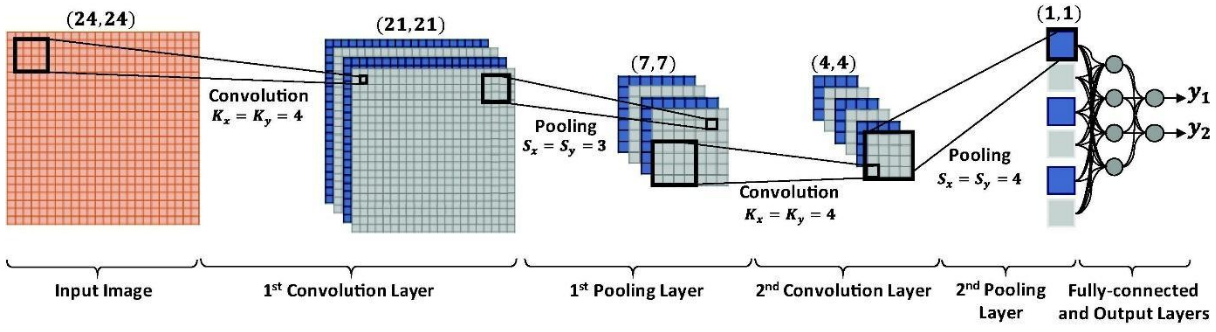

Figure 4: 1D Convolution with kernel of 3 and stride of 1 (Kiranyaz et al., 2022)

Convolutional neural networks typically are made up of three main layers,
the convolutional layer, the pooling layer, and a fully connected layer. The convolutional layer is the first layer of a convolutional neural network, and while this layer may be followed by other convolutional layers or indeed pooling layers. The fully connected layer is always the final layer, with each additional layer of the convolutional neural network increasing its complexity. For this project we used a one-dimensional convolutional neural network.

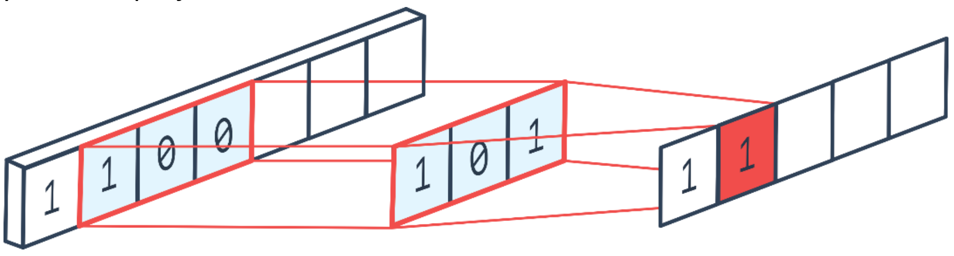

Figure 5: 1D Convolution with kernel of 3 and stride of 1 (Peltarion, 2022)

One way to think about how the 1D CNN works is to consider that we slide the kernel over the input image, then for each position of the kernel we multiply the values of the image and overlapping kernel positions together and produce the sum as the result. This sum is the value of the output image at the point of the image where the kernel is centred, as can be seen in figure 5.

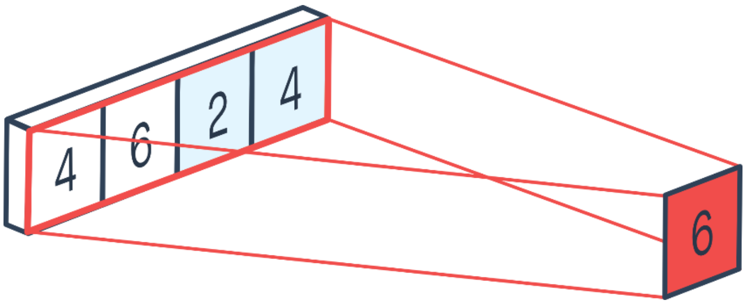

Figure 6: Max Pooling 1D (Peltarion, 2022)

After the convolution layer pooling was used to reduce the dimensionality. Pooling is typically used after a convolution layer to reduce the outputs dimension. This can also assist to reducing overfitting, the most prominent type of pooling is max pooling. Max pooling works not too dissimilar to the convolution, where in a given window, the max value is picked up and the window is slid over the input with a given stride after each iteration. A version of this can be seen in figure 6. For this project we used two 1D max pooling layers for our model.

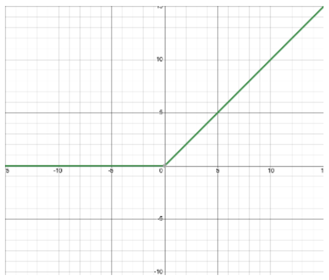

Figure 7: The ReLU Function (Singh, 2022)

For our convolutional layers we used the ReLU function, written: $Y = ReLU(X) = max(0,\ X)$ and plotted in figure 7. The activation function was applied to the weighted sum of all the inputs. ReLU is greatly used across neural networks for its speed and the fact that it can perform better than other activation functions, e.g., Sigmoid and Tanh, it is also specifically the commonly used activation function for convolutional neural networks.

After these layers we used a flatten layer. The flatten layer concatenates the output of the convolutional layers to create a “flat” structure so that it can then be fed into the dense layer at the end of the model.

In function, the dimension of the data in the dataset must match the dimension of the input layer. Each record of the dataset has a set of 328 permissions attributed to it, ergo the number of nodes in the neural network structure is equal to 328. In contrast, the output has only one singular node and one hidden layer. The reasoning for this, is because we are using binary classification. We are classifying whether; based on the permissions, if an app is malware, or not malware.

As for the hidden layer, the number of nodes can vary. This can alter performance of the neural network considerably and the only way to find the optimum number of nodes was by trial and error.

4.2 Keras Dense Model

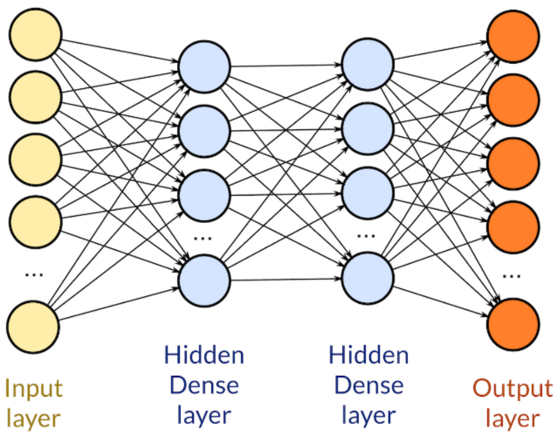

Figure 8: The Dense Layer (Chaturvedi, 2022)

After concluding the results from the convolutional neural network, we resorted to Plan B to try and improve on our work, as discussed in section 3.5. The dense layer (Keras, 2022) is just a standard densely connected neural network layer. This means that a single neuron within the network, derives its input from all the previous neurons in the previous layer of the network as can be seen in figure 8. The neurons within this layer, perform matrix-vector multiplication. This is a procedure where the row vector of the output from prior layers, is equal to the column vector of the dense layer.

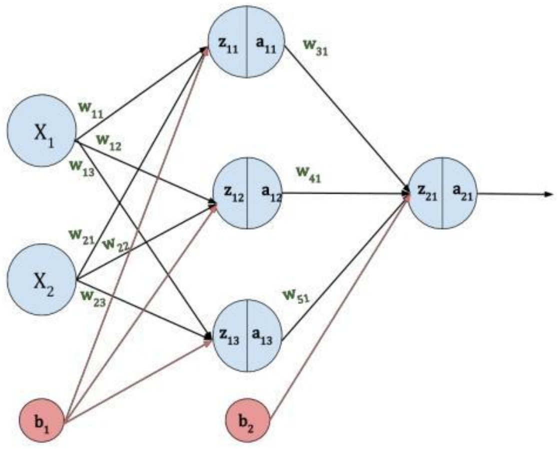

Figure 9: Neural Network Architecture (Kanani, 2019)

Each neuron is connected with only one parameter, a weight. For example, if the number 50 was passed through a connection with a weight of 0.2, it would turn into 10, these weights basically inform a neuron to respond more or less to a given input. For the duration of training, the weights are altered, and this is how the neural network learns. Each neuron is connected; however, they are not connected within a given layer, but connected from layer to layer instead. This way data can be considered to be moving in one direction, from the first layer to the next.

For our dense model we used two layers, followed by a flatten layer. As previously stated for the convolutional neural network, the flatten later concatenates the output to be fed into the final dense layer. As previously mentioned, the dimensionality must match that of the dataset but unlike before, we had by now made some changes to the dataset as mentioned in section 3.6. After making alterations to the dataset, this now sat at 205 permissions, ergo this was mirrored in the neural network. The output remained with one node as we are using binary classification. The optimum number of nodes for the hidden layer, again was by trial and error.

4.3 Scikit-Learn classifiers

After concluding the results from the dense neural network, we resorted to Plan C to try and improve on our work, as discussed in section 3.5. Plan C consisted of using the Scikit-Learn API, as discussed in section 3.3, to utilise various classifiers to see if any commonly used machine learning algorithm within that API would yield better results. The classifiers we will look at are: K-nearest neighbour (KNN), Decision trees, Random Forests, Multi-Layer Perceptron (MLP) and finally Ensemble.

4.3.1 K-Nearest Neighbour (KNN)

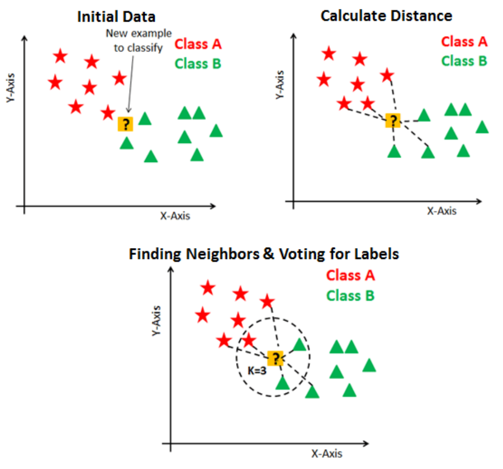

Figure 10: KNN Diagram (Band, 2020)

K-nearest neighbour (KNN) is a form of supervised learning algorithm. It is often used in both classification and regression methods. For this project we will only be looking at the former. K-nearest neighbour works by attempting to predict the correct class of the test data. It does this by calculating the distance between the test data and the training points, then looking at the given number (K) of points that appear closest to the test data. The algorithm then calculates the probability of the test data belonging to the classes of the given number (K) of points we looked at. This is then classified against the highest probability of the data looked at, as we can see from a simple diagram in figure 10.

4.3.2 Decision Tree

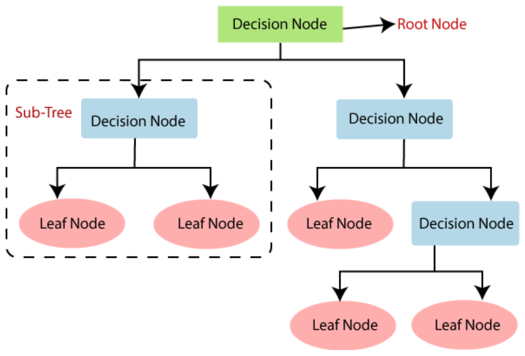

Figure 11: Abstract Decision Tree Diagram (Gupta, 2021)

Decision Trees are a type of non-parametric, supervised learning tools that like K-nearest neighbour can be used in classification and regression methodologies. It works by creating a training model. This model can then be used to predict the class of a given target by using simple decision rules that are inferred from the training data. The tree divides the dataset into subsets, meanwhile an associated tree is developed incrementally. A tree is typically made up of nodes that have different roles, such as testing for a specific attribute, and then connecting to the next node or a leaf node. Leaf nodes ultimately are the nodes that will provide the prediction thus completing the structure, as we can see in an abstract form in figure 11.

4.3.3 Random Forest

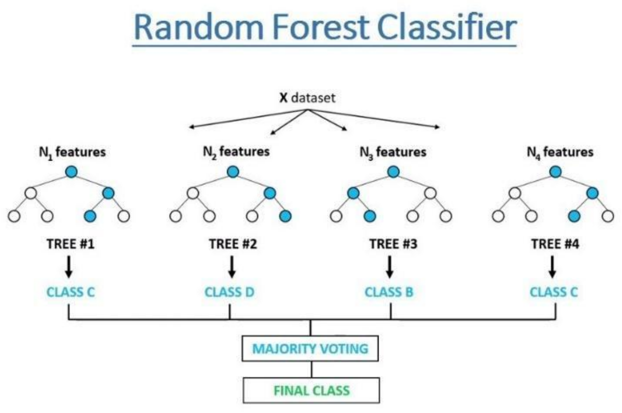

Figure 12: Abstract Random Forest Diagram (Chauhan, 2021)

Random forests are an ensemble type learning method that can be used for classification. Not too dissimilar to that of the decision tree. It works in the same way as discussed in section 4.3.2; however, it is an agglomeration of decision trees. For the classification task we have set, the tree derives its result by returning the classification that was output by the majority of the inherent trees as seen in figure 12, and not the average across them (regression). I.e., the most common or numerous results. One of the benefits of a random forest over decision trees is that it can be more accurate for larger datasets and have less tendency of overfitting. Thus, giving ample reason for the use of it in this project.

4.3.4 Multi-Layer Perceptron (MLP)

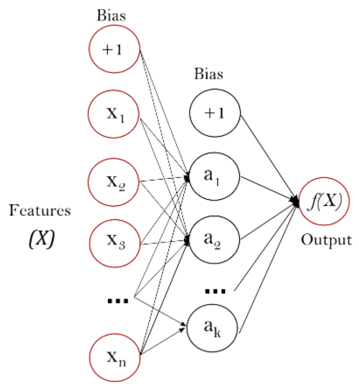

Figure 13: MLP diagram (Chauhan, 2021)

The multi-layer perceptron or MLP, works in very much the same sense as the dense model, as discussed in section 4.2. The data traverses in one direction between layers with each node having an activation function, weight and bias etc, as we can see from figure 13. MLP is very popular, widely utilised supervised learning tool that can be very powerful and can be considered a subset of dense neural networks. For this project, we are using the scikit-learn API to implement a multi-layer perceptron model.

Grid-Search and Hyperparameter Tuning

For this project it was necessary to use grid-search and parameter tuning techniques, to derive the best or more ideal parameters for the dense model as well as the MLP model. Grid search is essentially a brute force approach to deriving the ideal parameters. It uses a combination of all prior specified hyperparameters and given values to calculate the performance of every combination provided. It then returns the highest performing combination. Because of this brute force approach, it can become incredibly time consuming as well as computationally expensive depending on the size of the parameter space.

Fortunately, the scikit-learn API has very easy-to-use, implementable, and understandable documentation for grid-search. As such, it was successfully implemented and utilised to derive more ideal hyperparameters for the multi-layer perceptron model. The result of which found that, the best results were achieved with a hidden layer size of two hundred, the ReLU activation function, sigmoid for the solver, alpha of 0.0005, and an adaptive learning rate. Unfortunately, at the time of writing this report, hyperparameter tuning could not successfully be implemented for the Keras Dense model or CNN model. Instead, manual trial and error techniques were used.

4.3.5 Ensemble

Finally, we have ensemble. Ensemble is a technique rather than an actual neural network. Ensemble aims to do exactly as the word itself defines. Similar in abstract idea as random forests, we will ensemble all the prior mentioned scikit-learn algorithms and run them together and receive a result from each. We will then utilise both soft and hard voting to achieve a result. Hard voting referring to a majority vote, as previously mentioned for random forests. And soft voting refers to the average of the outputs.

5. Experimental Evaluation and Results

To evaluate our methods, we ran each classifier 5 times, over accuracy, precision, recall and f1 metrics to give us a good indicator of our algorithm's performance. For the sake of a fair experiment, where possible, some features were constant throughout testing. These include ten-fold cross validation and a training size of 0.2 (20%). The training size of 0.2 was found, through testing, to yield the highest results on average across all the classifiers. This means that no one algorithm was ran differently than another on the dataset, that would cause it to perform better or worse due to training and testing sizes.

5.2 Evaluation Metrics

To determine the overall performance of this project we will be testing what we make against several metrics. Accuracy, Precision, Recall and F1. To calculate the accuracy, we compare the classification result against the expected value for a given sample, then we count the number of correct classifications and divide by the number of given samples as we can see in equation 1. Precision, seen in equation 2, evaluates the proportion of predicted faux results. In this case it means the the proportion of results that are malware against not malware, that are in-fact actually malware. Recall, seen in equation 3, evaluates the proportion of correctly classified malware. F1, seen in equation 4, returns a value of one or zero and can be considered the mean of precision and recall.

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \tag{1}$$

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$Recall = \frac{TP}{TP+FN} \tag{3}$$

$$F1 = \frac{2\ x\ Precision\ x\ Recall}{Precision+Recall} \tag{4}$$

Key (Alphabetical): False Negative predictions (FN), False Positive predictions (FP), True Negative predictions (TN), True Positive predictions (TP)

5.3 Convolutional Neural Network

Number of Hidden Nodes	Number of Epochs	Accuracy (%)	Number of Hidden Nodes	Number of Epochs	Accuracy (%)
16	90	85.27	22	90	86.67
17	90	86.40	23	90	84.20
18	90	86.00	24	100	86.65
19	90	85.33	32	100	84.68
20	90	86.68	64	100	87.53
20	80	86.64	128	100	86.01
20	100	86.22	256	100	83.53
20	110	85.68	300	100	85.60
21	90	86.40	400	100	85.87

Table 1. Iterations ran with test size of 0.2

The CNN model was tested with 10-fold cross validation across 10 tests, where the results were then marked as an average across them. According to the test results as seen in table 1. The optimal number of hidden nodes for the first Conv1D layer was 64.

Test Number	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
1	0.86	0.75	0.71	0.69
2	0.86	0.74	0.73	0.73
3	0.88	0.73	0.68	0.75
4	0.87	0.75	0.75	0.71
5	0.87	0.74	0.72	0.72
AVG	**0.87**	**0.74**	**0.72**	**0.72**

Table 2. Convolutional Neural Network results

As we can see from table 2, we can state that the convolutional neural network created for this project performs at around 87% accuracy. We may also infer from these results presented, that it has a standard deviation of around one percent.

5.4 Dense Model

Number of Hidden Nodes	Number of Epochs	Accuracy (%)	Number of Hidden Nodes	Number of Epochs	Accuracy (%)
16	90	85.33	24	100	86.59
17	90	85.24	32	100	85.43
18	90	85.92	64	100	87.60
19	90	85.31	128	100	85.91
20	90	86.17	256	100	84.23
21	90	85.64	300	100	85.40
22	90	85.55	400	100	88.17
23	90	86.20			

Table 3. Iterations ran with test size of 0.2

The Dense model was tested with 10-fold cross validation across 10 tests, where the results were then marked as an average across them. According to the test results as seen in table 3. The optimal number of hidden nodes for the first dense layer was 400. However, due to the large number of nodes, the run-time was exceedingly long. Because of this, the final resultant number of nodes used, was 64. This is because it produced a similar result but on average took less than half the time to run.

Test Number	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
1	0.86	0.75	0.72	0.70
2	0.87	0.75	0.70	0.74
3	0.87	0.74	0.74	0.73
4	0.87	0.75	0.73	0.72
5	0.88	0.73	0.72	0.72
AVG	**0.87**	**0.74**	**0.72**	**0.72**

Table 4. Keras Dense Model results

As we can see from table 4, we may state that the Keras dense model created for this project performs at around 87% accuracy, similar to that of the CNN model. We may also infer from these results presented, that it has a standard deviation of less than or equal to one percent.

5.5 Scikit-Learn

K-Nearest Neighbour

Value of K	Accuracy (%)	Value of K	Accuracy (%)
1	76	9	77
2	78	10	78
3	77	15	74
4	77	20	77
5	76	25	76
6	77	30	76
7	75	40	76
8	80	50	76

Table 5. Iterations ran with test size of 0.2

To find the optimum value of K was a trial-and-error task. For our given dataset, the model was tested with 10-fold cross validation across 10 tests, where the result was then marked as an average across them. The optimum value of K was found to be 8, with an accuracy return of 80%.

Test Number	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
1	0.80	0.77	0.74	0.74
2	0.80	0.77	0.74	0.74
3	0.80	0.77	0.74	0.74
4	0.79	0.76	0.73	0.73
5	0.80	0.77	0.74	0.74
AVG	**0.80**	**0.77**	**0.74**	**0.74**

Table 6. K-nearest neighbour results

As we can see from table 6, we can state that the k-nearest neighbour algorithm created for this project performs at around 80% accuracy, which is nearly 10% less than the two previous methods. We may also infer from these results presented, that it has a standard deviation of less than one percent.

Decision Tree

Test Number	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
1	0.74	0.71	0.71	0.70
2	0.74	0.72	0.71	0.70
3	0.74	0.72	0.70	0.70
4	0.74	0.72	0.71	0.71
5	0.74	0.72	0.70	0.70
AVG	**0.74**	**0.72**	**0.71**	**0.70**

Table 7. Decision Tree results

As we can see from table 7, we can state that the decision tree algorithm created for this project performs at around 74% accuracy, which is worse than that of all previous classifiers. We may also infer from these results presented, that it has a standard deviation of less than one percent.

Random Forest

Test Number	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
1	0.79	0.77	0.74	0.76
2	0.80	0.76	0.75	0.76

3	0.79	0.77	0.75	0.75
4	0.79	0.77	0.75	0.75
5	0.79	0.78	0.75	0.75
AVG	**0.79**	**0.77**	**0.75**	**0.75**

Table 8. Random Forest results

As we can see from table 8, we can state that the random forest algorithm created for this project performs at around 79% accuracy, which puts it somewhere above the decision tree algorithm and below the k-nearest neighbour model in terms of performance. We may also infer from these results presented, that it has a standard deviation of less than one percent.

Multi-Layer Perceptron

Test Number	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
1	0.76	0.74	0.72	0.71
2	0.78	0.74	0.72	0.71
3	0.78	0.73	0.72	0.73
4	0.78	0.74	0.71	0.72
5	0.78	0.73	0.71	0.71
AVG	**0.78**	**0.74**	**0.72**	**0.72**

Table 9. Multi-layer Perceptron results

As we can see from table 9, we may state that the multi-layer perceptron model created for this project performs at around 78% accuracy, which puts it below k-nearest neighbour and the random forest models, but above the decision tree in terms of performance. We may also infer from these results presented, that it has a standard deviation of less than one percent.

Ensemble Hard Voting

Test Number	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
1	0.77	0.75	0.73	0.73
2	0.75	0.73	0.71	0.71
3	0.76	0.74	0.72	0.72
4	0.75	0.73	0.71	0.72
5	0.76	0.75	0.72	0.72
AVG	**0.76**	**0.74**	**0.72**	**0.72**

Table 10. Ensemble Hard Voting results

As we can see from table 10, we can state that the ensemble model created for this project performs at around 76% accuracy, which puts it as the second worst performing model in terms of accuracy. We may also infer from these results presented, that it has a standard deviation of equal to or less than one percent.

Ensemble Soft Voting

Test Number	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
1	0.77	0.75	0.73	0.73
2	0.76	0.74	0.72	0.72
3	0.76	0.74	0.72	0.73
4	0.77	0.75	0.73	0.72
5	0.76	0.74	0.72	0.72

AVG	**0.76**	**0.74**	**0.72**	**0.72**

Table 11. Ensemble Soft Voting results

As we can see from table 11, we can state that the ensemble model created for this project performs at around 76% accuracy, which puts it equal with hard voting in terms of performance. We may also infer from these results presented, that it has a standard deviation of equal to or less than one percent.

5.6 Overview

%	CNN Model	Dense Model	K-NN	Decision Tree	Random Forest	MLP	Ensemble (Combined)
Accuracy	0.87	0.87	0.80	0.74	0.79	0.78	0.76
Precision	0.74	0.74	0.77	0.72	0.77	0.74	0.74
Recall	0.72	0.72	0.74	0.71	0.75	0.72	0.72
F1-Score	0.72	0.72	0.74	0.70	0.75	0.72	0.72

Table 12. Comparison of the classifiers

Overall, we found that the convolutional neural network, and dense model performed at the same level (87%) under the accuracy metric as is evident in table 12, with a difference of less than 1%. We can also see that the decision tree consistently performed the worst in all metrics. One thing of note however is that the Random Forest performed the highest in all metrics other than accuracy. With its higher precision (77%) meaning that it returns a higher number of relevant results than irrelevant, and the higher recall (75%) meaning that it returns most of the relevant results, regardless of if it returns an irrelevant result. From this we can derive that, the convolutional neural network and the dense model are the highest performing methods in terms of accuracy, however the random forest model is the best algorithm in terms of its individual performance irrespective of its accuracy.

5.7 Performance Evaluation Comparison

5.7.1 Data Manipulation

As mentioned in section 3.6, we hypothesised that a reduction in the datasets size, may improve performance speed at little to no cost of accuracy. To do this, a simple comparative algorithm was set up. The entire header of the dataset was taken, and each permission was compared to all other permissions. At first, strings that were fifty percent similar were compared and recorded but from the results, this proved too inaccurate. After a series of trial and error, the optimum ratio settled around 75%. This meant that strings that were seventy-five present similar were recorded along with their position in the dataset. The columns of similar permissions were then merged into one. The dataset reduced from 328 columns, to 205, a 37.8% decrease in size. This new dataset was then tested on the convolutional neural network model discussed in section 4.1, with the time and accuracy recorded against the original, with the results shown in table 13.

Test Number	Time (minutes)	Accuracy (%)	Time (minutes)	Accuracy (%)
1	1:47	87.24	1:28	86.66
2	1:45	88.01	1:29	85.35
3	1:43	87.57	1:29	87.27
4	1:46	87.39	1:30	86.34
5	1:47	87.91	1:28	86.79
6	1:45	87.07	1:29	87.65
7	1:45	87.23	1:29	86.52
8	1:43	87.94	1:31	87.99
9	1:44	88.12	1:30	88.13
10	1:42	86.90	1:29	87.01
AVG	**1:44**	**87.53**	**1:29**	**86.97**

Table 13. Left: Original dataset, Right: Reduced dataset

As we can see from the results in table 13, the reduced dataset only performed 0.56% less accurate than the original dataset, but in contrast ran 16.85% faster. This means, if we were to hypothetically apply this to *Hamdroids* longest recorded results, it could shave almost 25 minutes from the runtime with only a decrease of accuracy by less than 1%.

5.7.2 CNN vs Dense Model

Test Number	Time (m)	Accuracy (%)	Time (m)	Accuracy (%)	Time (m)	Accuracy (%)	Time (m)	Accuracy (%)
1	1:47	87.24	1:28	86.66	5:01	88.42	3:20	86.89
2	1:45	88.01	1:29	85.35	5:01	86.33	3:18	87.54
3	1:43	87.57	1:29	87.27	5:02	87.58	3:18	87.29
4	1:46	87.39	1:30	86.34	4:59	87.74	3.20	87.69
5	1:47	87.91	1:28	86.79	5:00	86.91	3:21	88.10
6	1:45	87.07	1:29	87.65	5:01	87.95	3:17	87.33
7	1:45	87.23	1:29	86.52	5:02	88.14	3:20	86.80
8	1:43	87.94	1:31	87.99	5:01	88.39	3:18	87.21
9	1:44	88.12	1:30	88.13	5:00	87.97	3:18	88.11
10	1:42	86.90	1:29	87.01	5:00	87.25	3:19	87.48
AVG	**1:44**	**87.53**	**1:29**	**86.97**	**5:00**	**87.66**	**3:19**	**87.44**

Table 14. CNN vs Dense Model Left: CNN, Right: Dense Model
Light grey area: Reduced dataset results

Under further testing, we ran 10 tests to determine how the CNN and Dense model compare with the original and reduced dataset. As we can see from table 14, the results shed more light on how our reduced dataset universally reduces runtime. The simulation, as well as the one mentioned in 5.7.1, were executed on an Intel Core i7-9750H CPU @ 2.60GHz, 16.0 GB RAM using the Microsoft Windows 11 operating system. From the results we can conclude that the CNN model runs 16.9% faster, and the dense model runs 33.6% faster on average using the reduced dataset. Both of which only suffering a loss of accuracy of less than 1%.

When comparing the results, we see a 0.13% difference in accuracy between the CNN and dense model (87.53% vs 87.66%) while using the original dataset, and a 0.47% difference between them (86.97% vs 87.44%), while using the reduced dataset. The biggest divide between the two, is runtime. While using the original dataset, the dense model ran 188.5% slower than CNN and 123.6% slower using the reduced dataset. This means, that even if we compare the dense models best result of 87.6%, vs the CNN's lowest result, it will run 237% slower for an accuracy increase of only 0.69% therefore concluding that the CNN model is the more desirable choice.

6. Critical Review

Overall, the project has been somewhat a success. The aim of the project was set out to research and develop a method of successfully classifying malware, such that it could also outperform the prior named pre-existing method mentioned in section 1. This project has achieved as such in more than one way. As observed from this report, we can correctly state that the accuracy and speed of successfully detecting malware, have been a success as well as outperforming existing methods. Both the convolutional neural network model and dense model perform to a higher accuracy than the equivalent method discussed in *Hamdroid*. As well as this, the method(s) mentioned in this report also perform at a higher speed, ergo reducing run-time which is a very satisfying and functionally important outcome from a practicality point of view.

The outlook of this, in hindsight is optimistic. The highest accuracy achieved on average, was 87%. This leaves a 13% margin of error. Ideally the overall accuracy target was 95% and this project has fallen short of that by 8%. This short coming is entirely a time-based problem. With machine learning it is often noted that it is a time-consuming task, and in the professional world is often the debate whether or not the payoff is worthwhile. The reason for this, is because the way to improve the results presented in this project would be to explore wider parameters. This means experimenting and testing an even wider range of hidden layer sizes, test and train sizes and hyperparameter tuning than has already been observed. The cost of doing so is based entirely on time and computational speed. A higher level of computational power can lower run-times thus more testing can be complete in a shorter time frame. As well as this, more time in general would be beneficial as machine learning typically can take a long time to develop when compared to other software.

Because of this, the project has been a focal point of learning that can be reviewed and analysed for future projects in the realm of artificial intelligence. It has put a strong emphasis on time-management and forward projections as unlike other machine learning projects, this particular one has shown that the development of these technologies is not always so cut and dry. This project has emphasised the concept of planning for the inevitable, and in the case of this, has meant spending time and resources to develop an algorithm with the best given parameters as that given time allowed, just for it to return results that are not as desirable as predicted.

This in turn has spotlighted the importance of time management which had previously been a loose guide. This means, that if this project were to be repeated, more time could have been dedicated towards the development and testing phase rather than the research phase of the project. Thus, it may have provided a better result, having tested a wider range of possibilities, specifically in getting the hyperparameter tuning software for the convolutional neural network fully functioning.

Along with this, the project has shown throughout the importance of understanding and experience with the tools and technologies you use. In practice this means that the understanding of neural networks, how they function both in theory and their practical application is extremely beneficial in reducing development and testing time. As the project progressed, and a further understanding of the technologies, particularly the similarity of the dense model, MLP and to a smaller degree, the convolutional neural network (CNN). After changing from the CNN, the drawbacks, errors, and time wasted were not repeated in the development of the other two methods. This essentially meant, that redundant methods and code that provided undesirable results, or worse results could inherently be reduced as time progressed. An example of this may be a redundant method of trying to increase accuracy of the dense model which turned out to be erroneous, could then be omitted from the development of the MLP as these technologies are extraordinarily similar in function. Ergo, an error in the dense model, would likely produce the same outcome in MLP. This has caused a development in understanding of how a specific approach and change to these algorithms, will affect how it performs functionally as well as in terms of time-cost, where time is the difference in runtime, and cost being how big of a payoff it turns out to be, for this project it is in terms of the algorithm's accuracy.

All of this means that a future project in a related topic, would run in a much timelier, planned out fashion as at the beginning of this project, having not done anything quite like this before, it was difficult to plan and project how the project will run in terms of its development, and how over-time this would affect the project as a whole.

References

1. Seraj, S., Khodambashi, S., Pavlidis, M., & Polatidis, N. (2022). HamDroid: permission-based harmful android anti-malware detection using neural networks. *Neural Computing and Applications*, 1-10.
2. StatCounter Global Stats. 2022. *Mobile Operating System Market Share Worldwide | Statcounter Global Stats*. [online] Available at: <https://gs.statcounter.com/os-market-share/mobile/worldwide> [Accessed 16 March 2022].
3. AV-Comparatives. 2022. *Android Test 2019 - 250 Apps*. [online] Available at: <https://www.av-comparatives.org/tests/android-test-2019-250-apps/>.
4. Razgallaha, A., Khourya, R., Hallé, S. and Khanmohammadi, K., 2021. *A survey of malware detection in Android apps: Recommendations and perspectives for future research*. [online] Available at: <https://doi.org/10.1016/j.cosrev.2020.100358>.
5. Seraj, S., 2021. *A dataset for fake android anti-malware detection*. [online] Kaggle.com. Available at: <https://www.kaggle.com/datasets/saeedseraj/a-dataset-for-fake-android-antimalware-detection>.
6. Gorelik, M., 2020. *Machine Learning Can't Protect You From Fileless Attacks | NGAV*. [online] Blog.morphisec.com. Available at: <https://blog.morphisec.com/machine-learning-cant-protect-you-from-fileless-attacks>.
7. Tchakounte, F., 2014. *Permission-based Malware Detection Mechanisms on Android: Analysis and Perspectives*. [PDF] Scientific Online. Available at: <https://www.researchgate.net/profile/Franklin-Tchakounte/publication/271199472_Permission-based_Malware_Detection_Mechanisms_on_Android_Analysis_and_Perspectives/links/54c11b5c0cf2d03405c4de97/Permission-based-Malware-Detection-Mechanisms-on-Android-Analysis-and-Perspectives.pdf>.
8. S. Sen, E. Aydogan and A. I. Aysan, "Coevolution of Mobile Malware and Anti-Malware," in IEEE Transactions on Information Forensics and Security, vol. 13, no. 10, pp. 2563-2574, Oct. 2018, doi: 10.1109/TIFS.2018.2824250.
9. Idika, N. and Mathur, A.P., 2007. A survey of malware detection techniques. Purdue University, 48(2).
10. Zhou, Y. and Jiang, X., 2012, May. Dissecting android malware: Characterization and evolution. In 2012 IEEE symposium on security and privacy (pp. 95-109). IEEE.
11. Zarni Aung, W.Z., 2013. Permission-based android malware detection. International Journal of Scientific & Technology Research, 2(3), pp.228-234.

12. Aslan, Ö.A. and Samet, R., 2020. A comprehensive review on malware detection approaches. IEEE Access, 8, pp.6249-6271.
13. Ertel, W., Black, N. and Mast, F., n.d. *Introduction to artificial intelligence*. 2nd ed. Springer.
14. Russell, S., Norvig, P. and Chang, M., n.d. *Artificial intelligence A Modern Approach*. 3rd ed. Pearson.
15. Mohamad Arif J, Ab Razak MF, Awang S, Tuan Mat SR, Ismail NSN, Firdaus A (2021) A static analysis approach for Android permission-based malware detection systems. PLoS ONE 16(9): e0257968. https://doi.org/10.1371/journal.pone.0257968
16. Python.org. 2022. *Comparing Python to Other Languages*. [online] Available at: <https://www.python.org/doc/essays/comparisons/>.
17. JetBrains. 2022. *Our Customers: 92 of Fortune Global 100 Companies - JetBrains*. [online] Available at: <https://www.jetbrains.com/company/customers/> [Accessed 23 March 2022].
18. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M. and Duchesnay, M., 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research 12 (2011) 2825-2830,*.
19. Trello.com. 2022. *Trello*. [online] Available at: <https://trello.com/>
20. B Boser Le Cun, John S Denker, D Henderson, Richard E Howard, W Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In Advances in neural information processing systems. Citeseer, 1990.
21. Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M. and Inman, D., 2022. 1D convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, [online] Volume 151. Available at: <https://doi.org/10.1016/j.ymssp.2020.107398>
22. Peltarion, 2022. *A 1D convolution with a kernel sized 3 and stride 1.*. [image] Available at: <https://peltarion.com/static/1d_convolution_pa1.png> [Accessed 25 March 2022].
23. Peltarion, 2022. *Global max pooling 1D*. [image] Available at: <https://peltarion.com/static/1d_global_max_pooling.png> [Accessed 25 March 2022].
24. Singh, S., 2022. *ReLU as an Activation Function in Neural Networks*. [online] Deep Learning University. Available at: <https://deeplearninguniversity.com/relu-as-an-activation-function-in-neural-networks/>.
25. Chaturvedi, N., 2022. *Custom Layers in Keras*. [online] [image] Medium. Available at: <https://medium.datadriveninvestor.com/custom-layers-in-keras-de5f793217aa>.
26. Keras.io. 2022. *Keras documentation: Dense layer*. [online] Available at: <https://keras.io/api/layers/core_layers/dense/>.
27. Kanani, B., 2019. *Mathematics behind the Neural Network - Machine Learning Tutorials*. [online] Machine Learning Tutorials. Available at: <https://studymachinelearning.com/mathematics-behind-the-neural-network/>.
28. Band, A., 2020. *How to find the optimal value of K in KNN?*. [online] towards data science. Available at: <https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb>.
29. Gupta, S., 2021. *Decision Tree Implementation in Python with Example*. [online] Springboard.com. Available at: <https://www.springboard.com/blog/data-science/decision-tree-implementation-in-python/>.

30. Chauhan, A., 2021. *Random Forest Classifier and its Hyperparameters*. [online] Medium. Available at: <https://medium.com/analytics-vidhya/random-forest-classifier-and-its-hyperparameters-8467bec755f6>.

Appendices

Appendix A: Meetings

Date		Type of communication	Topic
2021	29th September	Email	Discuss project ideas AI project proposal
	5th October	Meeting	Discuss project ideas Anti-malware project
	11th October	Email	Project progress review Early scikit-learn planning
	28th October	Meeting	Project progress review Discussion of data processing and deep learning
	5th November	Email	Project progress review CNN creation
	9th November	Email	Project progress review, Synthetic code attempt failure
	30th November	Email	Viva setup
	8th December	Microsoft Teams	2nd reader meeting
2022	14th January	Email	Progress review
	17th February	Meeting	Progress review, data processing
	27th February	Email	Progress review, data processing results,
	28th February	Email	Progress review, transition to dense model
	3rd March	Meeting	Progress review, dense model results and algorithm performance evaluation
	13th March	Email	Progress review, advice to remove layers and change parameters to evaluate performance
	15th March	Email	Progress review, transition to scikit-learn model
	23rd March	Email	Progress review, report writing
	4th April	Email	Progress review, report first draft, restructure needed
	5th April	Email	2nd Draft complete, review of report, font change suggested

Appendix B: Source/Asset Files

https://github.com/jordannelson0/anti-malware