Malware Classification via API Call Frequency: a Deep Learning Approach

Praneeth Babu Marella[1,2]

[1]ISA 673: Operating Systems Security

[2]George Mason University, Fairfax, VA 22030, USA

Detecting and appropriately classifying malware can be challenging especially with the ever-changing threat landscape and evolution of malware techniques. Additionally, the increasing amount of data that anti-virus tools must ingest and process to take appropriate actions are making it hard to quickly and accurately predict the nature of the code executing on a host. This process can also be very computationally expensive to process the data and not robust enough. Traditional methods such as signature-based detections cannot be relied on as malware behavior constantly changes. Several methods have been proposed to improve this process and accurately predict malware classification. One of these methods is using the Windows operating system API call sequence a malware uses to execute its functionality as an indicator. By running malware in an isolated sandbox environment, such as Cuckoo's Sandbox, we can extract the API calls sequentially made by malware. In my research, I chose to use these Windows OS API call sequences extracted from a sandbox environment and apply Deep Learning sequential models using fully connected, Dense, layers to predict malware classifications. In the dataset I used for my model, the types of malware included were Adware, Backdoor, Downloader, Dropper, Spyware, Trojan, Virus, and Worm. Training and testing my model using the dataset of malware API call sequences resulted in an average total accuracy of 59% with individual class precision scores reaching up to about 91% when predicting malware classes using a multi-class classification method.

1. Introduction

New malware seems to be popping up every day as threat actors evolve and adapt to circumvent traditional or existing defense methods. This means defenders are constantly playing a game of whack-a-mole when attempting to successfully prevent malware execution attempts. But before we can even think about prevention measures, we need to focus on the detection component since prevention methods will be rendered useless without proper detection of potential malware. Failure to detect malware also means a potential security breach resulting from successful malware execution. In recent times, there has been a large upward trend of security breaches in companies from various sectors. Security breaches cause significant problems for organizations as they can lead to economic damages due to potential loss of data, the impact of the malware leading to downtime, and the time it will take to remediate the infection across the organizations. Some security breaches even led organizations to be completely shut down leading to larger impacts that affect the individual lives of workers within that organization on top of the economic losses. This paints us a picture of how large of a threat malware is to an organization and what its impacts are.

Malware is any malicious software that is designed to cause destruction or harm to an asset within an organization. Malware can be classified into different classes

depending on their functionality. These classes include but not limited to Dropper, Spyware, Trojan, Virus, and Worm. Each class of malware behaves differently due to their functional differences but are likely to be similar within each class. Traditionally, there are two types of classifications for malware detection techniques which are signature-based and anomaly-based. There are disadvantages to these traditional malware detection techniques. Signature-based detection is a type of detection that relies on a known attack database to detect attacks against an organization. Some of the disadvantages of this include easy evasion by modifying known malicious behavior of malware, novel attacks likely will not be detected, and can be computationally expensive as all the signatures need to be evaluated for detection. Anomaly-based detection is a type of detection that relies on identifying behavior that deviates from an implemented or learned baseline behavior. Some of the disadvantages of this technique include difficulty in defining a baseline of behavior, the potential for an increase in false-positive rate as responsibilities for users in an organization change, and potential for true negatives as behavior that is considered normal could be used by malware and go undetected for a long time.

Another approach for defense against malware is identifying malicious behavior through malware analysis. Malware analysis allows us to understand how a sample behaves and the potential threat it poses. There are two main types of malware analysis, static analysis, and dynamic analysis. Static analysis is done through the examination of the malware's source code directly and identifying the malicious behavior it is coded to execute. Dynamic analysis allows us to observe the malware's behavior through execution done in an isolated sandbox environment, typically set up to mimic a "real" system so malware can successfully execute. Some of the things sandbox analysis tools record when conducting dynamic analysis include process activity, network activity, registry changes, API calls, disk read/write, etc. After identifying the malicious behavior of malware, the next task is to appropriately classify it into a family of malware. The classes of malware are defined by the observed malicious activity patterns and the intended functionality. Accurate classification can be a challenging task especially with the ever-evolving malware and increased sophistication of malicious code. Let us take a report generated from VirusTotal on a random sample of malware, we will typically observe that various classifications are assigned to it depending on the scanner vendors. These classifications can vary vastly, especially if the sample is relatively unknown, due to the way the scanner chooses to label certain behaviors. But it could also be misclassified, like labeling a sample as a Trojan instead of a Worm, due to the complicated nature of the sample when performing analysis on it.

As malware gets more sophisticated, it gets harder to quickly detect them as malware and classifying them. This can cause lead to certain defense mechanisms not triggering leading to little or complete functional execution of the malware. In my research, I focus on using the API call sequences extracted from performing dynamic analysis on various malware sample spanning across eight different classes including Adware, Backdoor, Downloader, Dropper, Spyware, Trojan, Virus, and Worm to propose an efficient and accurate malware classifier. The sequence of operating systems API calls can be significant in understanding the behavior of malware because, typically, the system calls made by malware do not change unless the main functionality of the malware changes. This can help us understand the distinction between the classifications of malware and help us build models to accurately classify malware based on their functionality as described by their API calls. My research is focused on classifying malware based on their Windows operating system API calls using Deep Learning techniques. Several previous works use Machine Learning methods to analyze, specifically, API call sequences to classify malware. I sought to improve on the previous works by

interpreting the API call sequence data differently and building an artificial neural network that uses sequential models with fully connected, Dense, layers to predict malware classifications.

The rest of this paper is structured as follows. In section two, I discuss related works in brief detail. In section three, I introduce the dataset used and my model. In section four, I go over the results. In section five, I conclude my research and discuss potential future works.

2. Related Works

(Vinod & Golecha 2010) analyzed metamorphic malware using various generators and traced their API call sequences. A dataset was created and a signature is generated for an entire malware class using this data. Then the signatures were tested and resulted in an accuracy range of 75% and 80%. The challenge with using the method proposed by the authors of this work is that it based on signatures which we are trying to move away from.

(Ki *et al.* 2015) proposed a method that can detect the malware for all types of the ubiquitous devices. The authors identify a little over 2700 "important" API calls and categorize them into 26 different categorizes. The authors then extract the API call sequences of malware which they convert into a API category sequence. Then the category sequences are used to find least common sequences that are then checked against a pre-defined signatures of malicious behavior mapped to a sequence of API calls to execute that behavior. The authors achieve an accuracy score of 99% percent.

(Gupta *et al.* 2016) the authors used Windows API call sequences to capture the behaviour of malicious samples. The authors identified 534 important API calls which were mapped to 26 categories. Behaviour of any malicious application is captured through sequence of these 26 categories of APIs. The authors then applied a fuzzy hashing algorithm, ssdeep, to generate fuzzy hash based signatures. These signatures were used for training and testing on a set of 2000 samples. The authorized were able to achieve an average of 94% accuracy score in predicting the classification of a malware.

(Qiao *et al.* 2013) analyzed malware behavior based on API call sequences. The authors focused on the frequent messages in API call sequences, and hypothesized that frequent itemsets composed of API names can be valuable in the identification of the behavior of malware. The authors used Cuckoo's sandbox to extract API call sequence data by monitoring malware dynamic execution and create a new representation method of malware behaviors called BBIS to convert the API calls to byte-based sequential data. Then, the authors use clustering to evaluate their method and achieve an accuracy of about 88%.

(Pirscoveanu *et al.* 2015) developed a distributed malware testing environment by extending Cuckoo Sandbox and used to trace malware's behavioral data including network activity, process activity, file access, registry key access, etc. Then the extracted data was used to develop a classification method using supervised machine learning. Their proposed classification approach achieved a high classification rate with a weighted average (AUC) value of 0.98 using a Random Forests classifier. The approach has been extensively tested on a total of 42,000 malware samples, which is higher than the sample count in the dataset used in my research.

(Liu *et al.* 2017) proposed a machine learning based malware analysis system, which has a data processing, decision making, and malware detection modules. The data processing component dealt with gray-scale images, Opcode n-gram, and import functions to extracts data related to the malware. The decision making components uses the

extracted data to classify the malware. Finally, the detection component used the shared nearest neighbor (SNN) clustering algorithm to discover new malware families. Their approach was evaluated on more than 20000 malware instances. Their results showed that it can effectively classify the unknown malware with about 86% accuracy.

(Agrawal *et al.* 2018) propose a feature representation with a one-hot vector from API call name and top N frequent n-gram of the argument strings. The authors use a two-stage model using long short-term memory (LSTM) model for learning a set of features which are then input to a second classifier to predict unknown malware. These features included each system API call's two most input parameters. The authors show that they are able to achieve a true positive rate of about 78% when predicting whether a sample was malicious or not.

(Pascanu *et al.* 2015) propose a two-stage approach, a feature learning stage and a classification stage. The authors use recurrent neural networks (RNNs) to predict the next possible API call based on the previous API call sequence which is the feature learning stage. At the classification stage, they input a fixed-length representation for the event stream into a max-pooling layer. Then the extracted features were used for classification. The authors were able to achieve a 71% accuracy score on a dataset with 75000 samples.

(Kolosnjaji *et al.* 2016) in this paper the authors attempt to construct a neural network based on convolutional and recurrent network layers in order to obtain the best features for classification in an ANN. They believe that their feature extraction architecture that combines convolution of n-grams with full sequential modeling will provide better results. The authors were able to test the malware classification and achieved an average of 85.6% accuracy with their proposed neural network architecture.

(Catak *et al.* 2020) developed a new dataset containing API calls made on the windows operating system, which represents the behavior of malicious software. This dataset is open-sourced and is used in my research. They used LSTM (Long Short-Term Memory) layers classification method for sequential data. The results obtained by their classifier resulted in an accuracy up to 95% with binary classifications. They also run multi-class malware datasets to show the classification performance of the LSTM model which resulted in about 40% accuracy.

(Cheng *et al.* 2013) relied on a standard TF-IDF weighting scheme applied to a Windows API function call sequences and use the data to classify malware. The authors also measured the similarity between malware families and applied that to the data to improve classifications. The authors were able to achieve about 97% accuracy by applying these features to their classification models. This research uses a similar TF methodology to process their data as my research but the authors don't apply the same ANN model on the data that I use.

3. Background and Methodology

In this section I will go over what Windows API calls are, dataset used, data pre-processing methods used, and the ANN model created.

3.1. *Windows API Calls*

The Windows application programming interface (API) enables programs written by developers to interact with the Windows operating system. For example, if the developer wants to display a user interface element using the native Windows UI then the develop can use a Windows API call like "drawtextexa" to do that. Microsoft maintains an extensive library of API calls for developer to reference to when developing at MSDN.

MalwareFamily	*SampleCount*
Adware	379
Backdoor	1001
Downloader	1001
Dropper	891
Spyware	832
Trojan	1001
Virus	1001
Worms	1001

TABLE 1. Malware sample composition of the dataset

Typically, for a developer to perform an action or achieve expected behavior there is a series of API calls that need to be made. In the same way, malware will also need to perform a sequence of API calls to carry out their expected behavior. The behavioral overview can be inferred from analyzing all the API calls made by a code sample running on a system. Therefore, during dynamic analysis the API calls play a key factor and are widely used in determining the behavior of a malware sample. For my research, I used an open-source Windows malware API call sequence dataset to train my ANN in order to accurately classify malware.

3.2. *Malware API Call Sequence Dataset*

Gathering a large enough dataset of API call sequences from a variety of malware families can be time consuming and require sufficient resource.Therefore, I chose to take advantage of an open-source data set provided by (Catak 2019). This dataset contains 7107 API call sequences extracted from malicious sample ran in the Cuckoo's Sandbox environment. Cuckoo's sandbox is open source software for performing static and dynamic analysis of file samples. It also lets you download a detailed report that is easily accessible and manipulable to extract the relevant information. A sample output from Cuckoo's Sandbox behavior analysis that displays the sequential API call trace can be seen in Figure 1.

In their research, they translated the families produced by each of the software into 8 main malware families: Adware, Backdoor, Downloader, Dropper, Spyware, Trojan, Virus, and Worm. The authors define the malware families as follows:

- **Adware**: hides on your device and serves you advertisements
- **Backdoor**: a technique in which a system security mechanism is bypassed undetectable to access a computer or its data
- **Downloader**: share the primary functionality of downloading content
- **Dropper**: surreptitiously carries viruses, back doors and other malicious software so they can be executed on the compromised machine
- **Spyware**: enables a user to obtain covert information about another's computer activities by transmitting data covertly from their hard drive
- **Trojan**: misleads users of its true intent
- **Virus**: designed to spread from host to host and has the ability to replicate itself
- **Worm**: spreads copies of itself from computer to computer

(Catak 2019)

Table 1 shows the composition of malware sampling belonging to specific malware families in the dataset. As you can see observe the table, the number of samples of other malware families except Adware is balanced(Catak 2019). This could potentially lead to

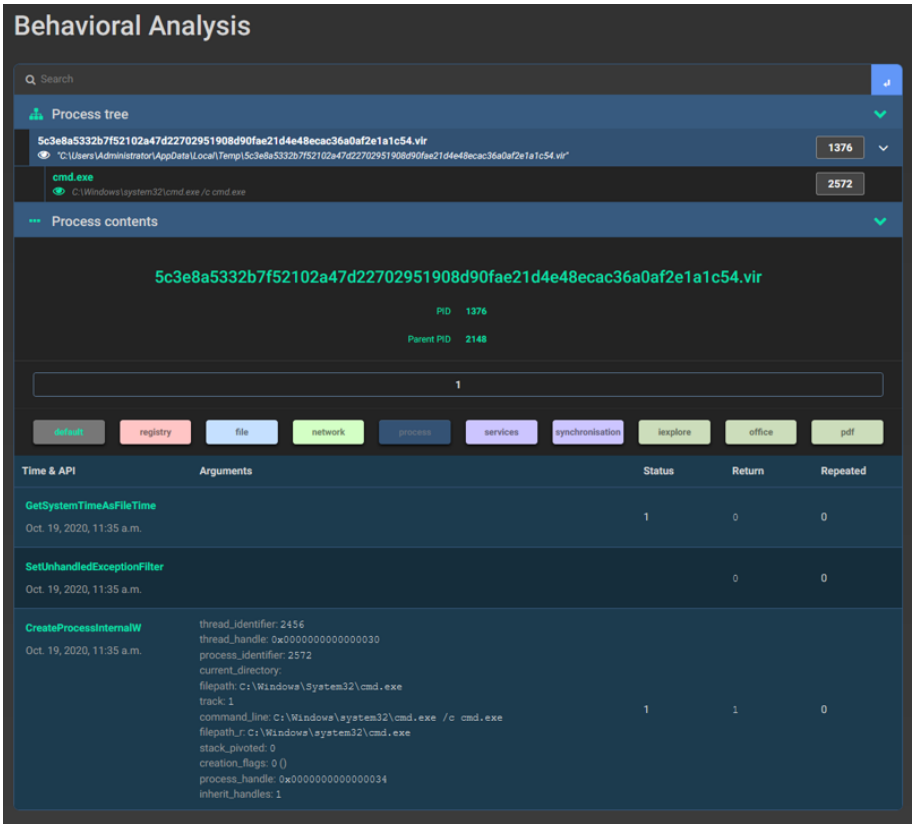

FIGURE 1. Sample of a API call trace from Cuckoo's Sandbox.

getkeystate	2694284
ntquerydirectoryfile	3053164
copyfilea	3257278
ntcreatefile	3769954
ntallocatevirtualmemory	3877058
findfirstfileexw	4094715
ntreadfile	5911672
process32nextw	6279599
ntclose	7022327
ntdelayexecution	13154471
getasynckeystate	36221591

FIGURE 2. The top 10 most API calls observed in the dataset.

skewed results when training and testing ANN models but I chose to use the dataset as-is since the rest of the families had a balanced composition.

Before we can move onto the data pre-processing step, we need to get a better understanding of the dataset we are working with. So, I ran some basic analysis to understand how many unique API calls are present in the dataset, which calls are the most common, and the least common. There are 278 unique API calls identified in the dataset and the most called API was identified as getasynckeystate, which determines whether a key is up or down (pressed) at the time the function is called. The least called

cryptdecodeobjectex	1
cryptgenkey	1
getusernameexa	1
ntdeletefile	1
rtlcompressbuffer	1
rtldecompressfragment	1
setinformationjobobject	1
internetopenurlw	2
netusergetlocalgroups	2
createdirectoryexw	3

FIGURE 3. The bottom 10 most API calls observed in the dataset.

on API call is actually a tie between a few of them. One of those API calls is ntdeletefile which specifies a file to be deleted immediately after the API call. Figure 2 and Figure 3 show the top and bottom 10 most API calls observed in the dataset.

3.3. *Approach*

• **Pre-process and Load Data**: The key to working with a neural network is processed data that we can feed into a ANN model. Typically this means working with numerical data which is normalized.

• **Define Model**: To feed our pre-processed data, we need to have a ANN model defined. A defined model includes choosing the layers, algorithms, activators, optimizer, and input/output sizes.

• **Train and Test**: Now that we have an ANN model defined, we need to choose a way to split our data and train our model. Here we also need to define the number of epochs. Then we use the leftover data to test and validate the accuracy of our model.

• **Interpret Results**: We need to interpret the results we get from our validation steps. Typically, this includes evaluating the average accuracy score and looking at the confusion matrix.

3.4. *Dataset Pre-processing*

Since we are trying to train our model to predict classification based on API call frequency, we need to find ways to process the dataset into numerical data that focuses on API call frequency across samples in the dataset. We concluded on two ways to process our dataset, one is to use the weighted API call frequency and another way is to categorize the API calls to their relative functionality and use the weighted API functionality frequency. For the weighted API call frequency method, we begin with some basic processing as detailed below:

(i) Assign a unique numerical value between the range of zero and the number of unique API calls observed (in this case 278).

(ii) Map the API calls in the sequence for each of the samples in dataset to the numerical value assigned to the unique API call.

(iii) Convert the mapped dataset to a frequency dataset. This means we count the number of times each of the unique API calls were invoked in each of the malware samples in the dataset.

Now that we have a frequency dataset, we want to figure out a way to normalize the data. One efficient way to do this apply a weight system typically used in information retrieval and text mining called Term Frequency (TF) and Term frequency-inverse document frequency (TF-IDF). Figure 4 shows hows we can calculate both TF and TF-IDF.

$$TF(t) = \frac{C(t)}{N} \quad (1)$$

Where:
T = API call
C = The frequency of API call
N = Total number of API call

Meanwhile, IDF is calculated using the equation below:

$$IDF(t) = \log\left(\frac{ND}{ND_t}\right) \quad (2)$$

Where:
T = API call
ND = Number of malware categories in the dataset
ND_t = Number of malware categories containing the API call

And finally, TF-IDF is calculated using the equation below:

$$TF - IDF(t) = TF(t) * IDF(t) \quad (3)$$

Where:
TF = Term frequency
IDF = Inverse document frequency

FIGURE 4. Formulas for calculating TF and TF-IDF (Murthy *et al.* 2019).

• **Term Frequency**: Measures how frequently a term appear within a given chunk of text. It is calculated by diving the number of times a term appears in the chunk of text by the total number of terms in the chunk of text. This method gives the same importance to every term.

• **Term Frequency-Inverse Document Frequency**: Measures how important a term is. First the term frequency is calculated then using the formulas in Figure 4, frequent terms are weighed down while the rarely appearing ones are scale up to calculate the TF-IDF. This method gives importance based on the rarity of a term.

Now we can use TF and TF-IDF formulas to create their respective datasets of the weighted API call frequency dataset. For the weighted API functionality frequency, we begin with some basic processing as detailed below:

(i) Map each of the unique API calls observed (in this case 278) to one of the 60 functional categories identified. A sample of the mapping can be seen in Figure 5.

(ii) Replace all the API calls for each sample with its mapped functionality.

(iii) Convert the functionality mapped dataset to a frequency dataset. This means we count the number of times each of the unique functionality was observed in each of the malware samples in the dataset.

(iv) Create the TF and TF-IDF datatsets of the API functionality frequency dataset.

Now that we are done processing the datasets in our proposed ways, we can move on to create a ANN model to input our processed data into to start predicting malware classifications.

isdebuggerpresent	AntiDebugging
outputdebugstringa	AntiDebugging
certopensystemstorea	CertificateAccess
iwbemservices_execmethod	CIM
iwbemservices_execquery	CIM
cocreateinstance	COMOLE
cocreateinstanceex	COMOLE
cogetclassobject	COMOLE
coinitializeex	COMOLE
coinitializesecurity	COMOLE

FIGURE 5. A sample of API calls mapped to their relative functionality.

```
antidebugging,certificateaccess,cim,comole,consolewrite,cryptocreate,cryptodecrypt,cryptoencrypt,cryptoexport,cryptor
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.07962407466283555,0.0,0.0,0.011217373819861967,0.0,
0.0,0.0,0.0,0.02454501742130025,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.614151434485485 4,0.103892170563658
0.15881579385436886,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.15576687989843796,0.0,0.0,0.1243510
0.0,0.0,0.0,0.002422592759674835 5,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.01706022925209734,0.0,0.0,0.0,0.0,0.04635397388369362
0.02714698635276013,0.0,0.0,0.012060035662780424,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.10650329567832456,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.4671853575295252,0.0,0.0,0.09324030656244686,0.0450
0.0,0.0,0.0,0.0015961824420596723,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.019969383595564025,0.0,0.0,0.0014
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.665793023229492,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0371111707394152 6,0.0,0.0,0.0,0.0,0.0,0.0,0.10132342000499665,0.0,0.0,0.0,0.0,0.0,0.1174373707867987 8,0
0.0,0.0,0.0,0.011934471256199494,0.0,0.0,0.0,0.0,0.0,0.0004967506790811939,0.0,0.0,0.0,0.0,0.0,0.0004862539853880523 7
0.0,0.0,0.0,0.00026939680873825327,0.0,0.0,0.0,0.0,0.0,0.0,0.002889563785313007,0.0,0.0010845977476943872,0.00115089 6
0.0,0.0,0.0,0.001016110118717964 7,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.04054832238794276,0.0,0.0,0.0,0.0,0.0097211431260801 9
0.0,0.0,0.0,0.00516220848298954 1,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.04748744045606991,0.0,0.0,0.0,0.00
0.0,0.0,0.0,0.0562096682537983 2,0.0,0.0,0.0,0.0,0.0,0.01579247076531476,0.0,0.0,0.0,0.0,0.0,0.0,0.1085859258573557,0.
```

FIGURE 6. A sample of the pre-processed malware API call sequence dataset.

3.5. *Defining our ANN Model*

Before we discuss how we modeled our ANN, we will go over some basics to get an understanding of how they work. Deep Learning is becoming a hot topic in the field of data science and artificial intelligence. Deep Learning is a field within Machine Learning and consists of a set of algorithms that are based on learning representations of the data. Deep Learning has also been becoming a popular topic for exploration in research related to discovery of new malware, malware detection and malware defense. Some of the popular libraries that allow you work with Deep Learning include Keras, TensorFlow, PyTorch, and more. In my project, I chose to use Kera which is a neural network API that runs on top of TensorFlow to build a multi-class prediction model.

Classification is a type of supervised machine learning algorithm used to predict a categorical label. A few useful examples of classification include predicting whether a customer will churn or not, classifying emails into spam or not, or whether a bank loan will default or not (Singh 2019).

The basic deep learning neural network consists of three main components:

- **Input Layer**: This is the layer where the inputs from the dataset are fed into the mode. The number of predictor variables is also specified here through the neurons.
- **Hidden Layers**: This is the layer where the outputs from the input layer are fed and exist in between the input and output layer. The deep neural network learns about the relationships involved in data in this component.
- **Output Layer**: This is the layer where the final output is extracted from what's happening in the previous two layers.

(Singh 2019)

We can visualize these layers and how a ANN model works by looking at the example in Figure 7. In this example, we are trying to figure if some random sample of wine is red or white based on a set of characteristics. The first layer, input layer, are fed

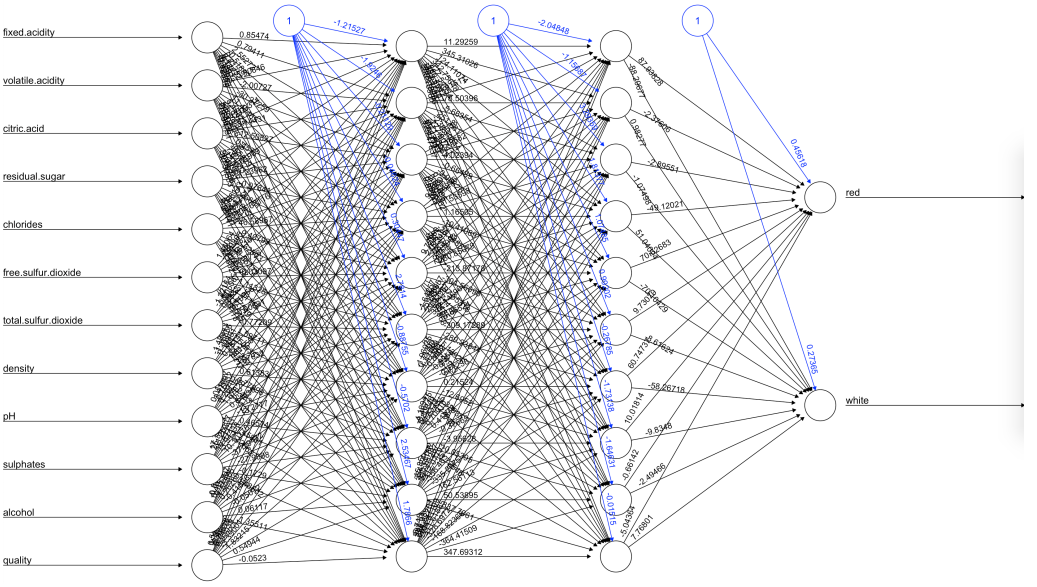

FIGURE 7. A simple ANN model visualized for classifying Wine classification.

```python
# 192 278 64 8
def baseline_model():
    model = Sequential()
    model.add(Dense(40, input_dim=60, activation="relu"))
    model.add(Dropout(0.2))
    model.add(Dense(20, activation="relu"))
    model.add(Dense(8, activation="softmax"))
    model.compile(
        loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"]
    )
    return model

# 120 10 1
estimator = baseline_model()
history = estimator.fit(
    X_train,
    Y_train,
    validation_data=(X_test, Y_test),
    epochs=120,
    batch_size=10,
    verbose=1,
)
```

FIGURE 8. One of our defined Keras ANN model and validator.

with the characteristics of the wine which includes alcohol %, pH, chlorides, and etc. Then the middle two layers, hidden layers, take in the input from the previous layers to understand the relationship between the data and activate a "neuron" when a threshold is met. Finally once the data reaches the output layer, the classification prediction is made.

For my multi-class classification model, I built a Keras sequential model with a non-linear activation method ReLU (Rectified Linear Unit). Then I use a dropout layer with the parameter 0.2, meaning 20% of the outputs from the input layer will randomly dropped before getting fed into the next layer. Then I use a fully connected layer, Dense, as one of the hidden where input data is fed and processed to learn about the relationships involved in data. Then the data is fed into an output layer with the

Softmax activation method with the loss function categorical crossentropy and the Adam optimizer. Categorical crossentropy is necessary because we want to specify that we have multiple classes. An example code of the implemented KNN model from my research can be seen in Figure 8.

4. Results and Future Works

Now that we have a defined model for our multi-class classification problem. We need to split the pre-processed dataset into train and test sets so that we can use it to train the model and then validate it by testing it. Understanding the definitions of the following terms can help interpret the validation results:

- **Accuracy Score**: Percent of predicted labels correctly matched to true labels.
- **Precision Score**: Proportion of predictions that were correct.
- **Recall Score**: Proportion of true positives identified.
- **Confusion Matrix**: 2D matrix comparing predicted category to true labels.
- **Epoch**: The numbers of a times a model run through an entire training set.

I tweaked the model parameters across several train and test runs before landing on the ones that gave me the best accuracy I could achieve with my model. Below are the results detailed for the best accuracy score I could achieve with the weighted API call frequency dataset and the weighted API functionality frequency dataset.

4.1. *Weighted API Call Frequency*

Looking at the results from the cross validation performed with predictions from my model on this dataset resulted in an average accuracy score of 59% across all classes of malware. This is not ideal but it is an improvement over previous conducted on this dataset. We can take a look at the confusion matrix Figure 9 to have a detailed look at how many accurate predictions we had for each malware class as well as the number of false positives.

We can look at the individual score for predicting different classes of malware Figure 10, we observe that my model did a great job predicting Adware, Backdoors, Downloaders, and Viruses (indicated by the higher precision score for each of these classes of malware).

4.2. *Weighted API Functionality Frequency*

Cross validation results of the predictions from this dataset actually show a reduced accuracy score of my model. So, we will not dive into the results until we figure out a better categorization method for API call functionality.

4.3. *Future Works*

There are several possibilities for future works to improve my pre-processed data and my KNN model so we can get a better accuracy score. This includes pre-processing API data using different methods like sequencing based on malicious behavior, tuning the ANN model to improve accuracy by using different parameters or algorithms, and gather a larger dataset to provide more data during training (esp. for types of malware where we had really low precision scores). All my code used for pre-processing the dataset and creating the KNN model is public on GitHub at https://github.com/pmarella2/Malware-Classification-using-ANN if they are interested in validating my results or working on exploring some of the future works ideas. I also uploaded all my pre-processed data so if any wants to play around with the KNN mode, they can do so with little effort.

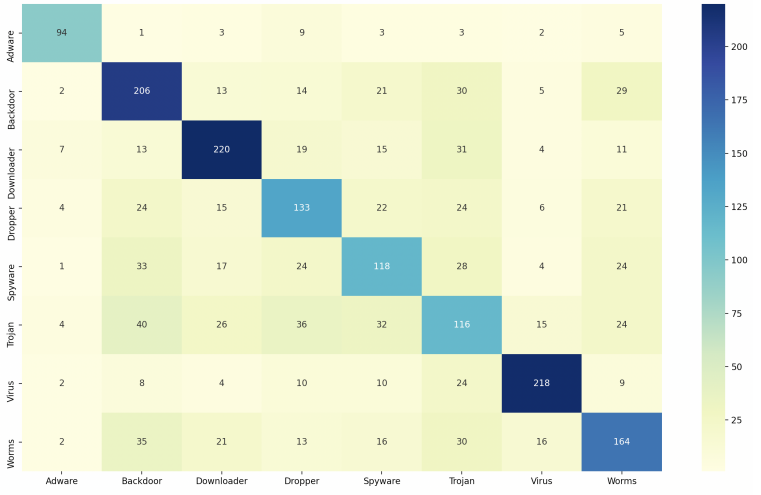

FIGURE 9. TF-IDF Weighted API Call Frequency Confusion Matrix.

```
Accuracy Score is: 59.774964838255976
Precision Score is: 60.539122651035
Recall Score is: 59.774964838255976
F1 Score is: 59.97799194984598
              precision    recall  f1-score   support

      Adware       0.91      0.73      0.81       120
    Backdoor       0.57      0.63      0.60       320
  Downloader       0.73      0.69      0.71       320
     Dropper       0.52      0.60      0.56       249
     Spyware       0.48      0.47      0.47       249
      Trojan       0.40      0.37      0.39       293
       Virus       0.82      0.73      0.78       285
       Worms       0.55      0.60      0.57       297

    accuracy                           0.60      2133
   macro avg       0.62      0.60      0.61      2133
weighted avg       0.61      0.60      0.60      2133
```

FIGURE 10. TF-IDF Weighted API Call Frequency Validation Results.

5. Conclusion

In this research I used the Windows operating system API call sequence a malware uses to execute its functionality as an indicator for classifying them into a family of malware. I used an open-source dataset with over 7000 malware samples that had their API call sequences extracted from Cuckoo's Sandbox dynamic analysis runs. I pre-processed the dataset with TF and TF-IDF weighted frequencies and applied a Deep Learning sequential model using fully connected, Dense, layers to predict malware classifications. In the dataset I used for my model, the types of malware included were Adware, Backdoor, Downloader, Dropper, Spyware, Trojan, Virus, and Worm. Training and testing my model using the dataset of malware API call sequences resulted in an average total accuracy of 59% with individual class precision scores reaching up to about 91% when predicting malware classes using a multi-class classification method. While the resulting accuracy may not be in the optimal range, it was an improvement over previous work conducted on the dataset using a multi-class classification model. I hope this will be another stepping

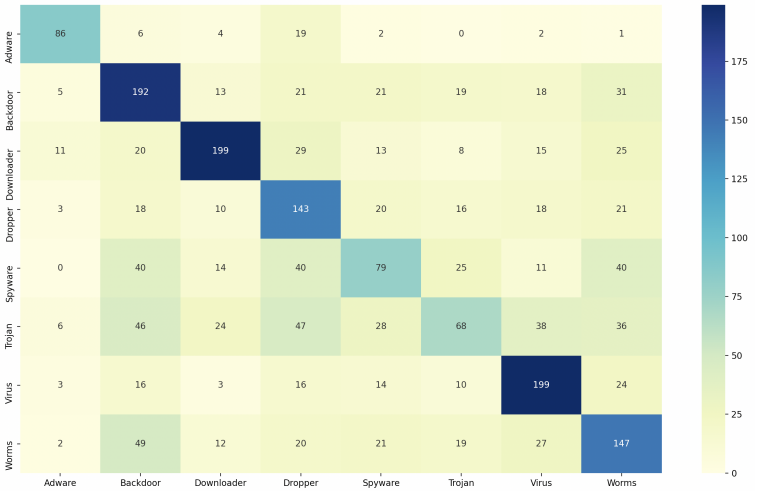

FIGURE 11. TF-IDF Weighted API Functionality Frequency Confusion Matrix.

```
Accuracy Score is: 52.18002812939522
Precision Score is: 52.02088405918756
Recall Score is: 52.18002812939522
F1 Score is: 51.39552092698321
              precision    recall  f1-score   support

      Adware       0.74      0.72      0.73       120
    Backdoor       0.50      0.60      0.54       320
  Downloader       0.71      0.62      0.66       320
     Dropper       0.43      0.57      0.49       249
     Spyware       0.40      0.32      0.35       249
      Trojan       0.41      0.23      0.30       293
       Virus       0.61      0.70      0.65       285
       Worms       0.45      0.49      0.47       297

    accuracy                           0.52      2133
   macro avg       0.53      0.53      0.52      2133
weighted avg       0.52      0.52      0.51      2133
```

FIGURE 12. TF-IDF Weighted API Functionality Frequency Validation Results.

stone in improving how we interpret the malware API call sequence data and build models that improve multi-class classification model accuracy.

REFERENCES

AGRAWAL, RAKSHIT, STOKES, JACK W, MARINESCU, MADY & SELVARAJ, KARTHIK 2018 Neural sequential malware detection with parameters. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2656–2660. IEEE.

CATAK, FERHAT OZGUR 2019 Malware api call dataset.

CATAK, FERHAT OZGUR, YAZI, AHMET FARUK, ELEZAJ, OGERTA & AHMED, JAVED 2020 Deep learning based sequential model for malware analysis using windows exe api calls. *PeerJ Computer Science* **6**, e285.

CHENG, JULIA YU-CHIN, TSAI, TZUNG-SHIAN & YANG, CHU-SING 2013 An information retrieval approach for malware classification based on windows api calls. In *2013 International conference on machine learning and cybernetics*, , vol. 4, pp. 1678–1683. IEEE.

GUPTA, SANCHIT, SHARMA, HARSHIT & KAUR, SARVJEET 2016 Malware characterization using

windows api call sequences. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pp. 271–280. Springer.

Ki, Youngjoon, Kim, Eunjin & Kim, Huy Kang 2015 A novel approach to detect malware based on api call sequence analysis. *International Journal of Distributed Sensor Networks* **11** (6), 659101.

Kolosnjaji, Bojan, Zarras, Apostolis, Webster, George & Eckert, Claudia 2016 Deep learning for classification of malware system call sequences. In *Australasian Joint Conference on Artificial Intelligence*, pp. 137–149. Springer.

Liu, Liu, Wang, Bao-sheng, Yu, Bo & Zhong, Qiu-xi 2017 Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering* **18** (9), 1336–1347.

Murthy, GSN, Chowdary, MVV, Sangameswar, MV & Vital, TPR 2019 Exploring the api calls for malware behavior detection using concordance and document frequency .

Pascanu, Razvan, Stokes, Jack W, Sanossian, Hermineh, Marinescu, Mady & Thomas, Anil 2015 Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1916–1920. IEEE.

Pirscoveanu, Radu S, Hansen, Steven S, Larsen, Thor MT, Stevanovic, Matija, Pedersen, Jens Myrup & Czech, Alexandre 2015 Analysis of malware behavior: Type classification using machine learning. In *2015 International conference on cyber situational awareness, data analytics and assessment (CyberSA)*, pp. 1–7. IEEE.

Qiao, Yong, Yang, Yuexiang, Ji, Lin & He, Jie 2013 Analyzing malware by abstracting the frequent itemsets in api call sequences. In *2013 12th IEEE international conference on trust, security and privacy in computing and communications*, pp. 265–270. IEEE.

Singh, Deepika 2019 Classification with keras.

Vinod, P Jain & Golecha, Gaur 2010 Laxmi.(2010). medusa: Metamorphic malware dynamic analysis using signature from api. In *Proceedings of the 3rd International Conference on Security of Information and Networks (SIN 2010)*, pp. 263–269.