Spring 2023

Malware Classification using API Call Information and Word Embeddings

Sahil Aggarwal
San Jose State University

Malware Classification using API Call Information and Word Embeddings

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Sahil Aggarwal

May 2023

The Designated Project Committee Approves the Project Titled

Malware Classification using API Call Information and Word Embeddings

by

Sahil Aggarwal

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2023

Dr. Fabio Di Troia	Department of Computer Science
Dr. Wendy Lee	Department of Computer Science
Dr. Faranak Abri	Department of Computer Science

ABSTRACT

Malware Classification using API Call Information and Word Embeddings

by Sahil Aggarwal

Malware classification is the process of classifying malware into recognizable categories and is an integral part of implementing computer security. In recent times, machine learning has emerged as one of the most suitable techniques to perform this task. Models can be trained on various malware features such as opcodes, and API calls among many others to deduce information that would be helpful in the classification.

Word embeddings are a key part of natural language processing and can be seen as a representation of text wherein similar words will have closer representations. These embeddings can be used to discover a quantifiable measure of similarity between words. In this research, we conduct a series of experiments using hybrid machine learning techniques, where we generate word vectors and use them as features with various classifiers. We use Hidden Markov Models and Word2Vec to generate embeddings based on dynamic API call logs of the malware. Apart from these, we also use the popular BERT and ELMo models which are known for generating contextualized embeddings. The resulting vectors are used as input for our classifiers, specifically Support Vector Machines (SVM), Random forest (RF), k-Nearest Neighbors (kNN), and Convolutional Neural Networks (CNN). Using these, we conduct two distinct sets of experiments where we try to classify the family of malware as well as the category of malware. The results achieved here prove that embeddings of API calls can be a useful tool in malware classification, especially in the case of families.

Index Terms **– Word Embeddings, Dynamic Analysis, API Calls, Hmm2Vec, Word2Vec, ELMo, BERT, SVM, RF, kNN, CNN**

ACKNOWLEDGMENTS

I would like to thank my guide and advisor Prof. Fabio Di Troia for his constant guidance and support throughout this work. The valuable suggestions and feedback he procided went a long way in helping me achieve the milestones set for this project.

I am also grateful to my committee members Prof. Wendy Lee and Prof. Faranak Abri for their feedback and suggestions. Finally, I would like to thank my family and friends for their constant support throughout my time at SJSU.

TABLE OF CONTENTS

CHAPTER

1 Introduction 1

2 Related Work 4

3 Background 7

3.1 Malware Analysis 7

3.1.1 Malware Category and Family 7

3.1.2 Dynamic Analysis with Buster Sandbox Analyzer and Sandboxie 7

3.2 Word Embedding Techniques 8

3.2.1 HMM2Vec 9

3.2.2 Word2Vec 12

3.2.3 ELMo 13

3.2.4 BERT 15

3.3 Classifiers 16

3.3.1 Support Vector Machine 16

3.3.2 Random Forest 17

3.3.3 k-Nearest Neighbors 18

3.3.4 Convolutional Neural Networks 18

4 Dataset and Experiments 19

4.1 Dataset 19

4.2 Experiments 23

4.2.1 HMM2Vec Experiments 23

4.2.2 Word2Vec Experiments 24
4.2.3 ELMo 24
4.2.4 BERT 24
4.2.5 Classifiers 25
4.3 Results 27
4.3.1 Malware Category Classification 27
4.3.2 Malware Category Classification with Different Set of Calls 39
4.3.3 Malware Family Classification 44
5 Conclusion and Future Work 55
LIST OF REFERENCES 57
APPENDIX
Appendix 60
A.1 Additional Results 60

LIST OF FIGURES

1	General Layout of Experiments	3
2	Malware under analysis with BSA	9
3	Initial and final B transpose [1]	11
4	CBOW and Skip-Gram Architecture [2]	13
5	ELMo Architecture [3]	14
6	BERT Pre-training and Fine-tuning [4]	16
7	Sample Preprocessed API log	22
8	Confusion Matrix for HMM2Vec-RF Category Classification	28
9	Top 20 calls vs Top 40 calls for HMM2Vec Category Classificaiton	29
10	Confusion Matrix for HMM2Vec-RF Category Classification with Other Symbol (41 Calls)	30
11	Top 40 calls vs Top 40 calls and Other (41) for HMM2Vec Category Classificaiton	31
12	Confusion Matrix for Word2Vec-RF Category Classification	32
13	Top 20 calls vs Top 40 calls for Word2Vec Category Classification	33
14	Confusion Matrix for ELMo-RF Category Classification	34
15	Top 20 calls vs Top 40 calls for ELMo Category Classification	35
16	Confusion Matrix for BERT-RF Category Classification	36
17	Top 20 calls vs Top 40 calls for BERT Category classification	37
18	Comparison of hybrid machine learning approaches for categories	39
19	Confusion Matrix for HMM2Vec-RF Category Classification with calls in all categories	40

20 Confusion Matrix for Word2Vec-KNN Category Classification with calls in all categories 42

21 Comparison of HMM2Vec scores for categories with Top 40 calls and calls in all categories (31 calls) 43

22 Comparison of Word2Vec scores for categories with Top 40 calls and calls in all categories (31 calls) 44

23 Confusion Matrix for HMM2Vec Family Experiments 46

24 Training Model Accuracy vs Loss For HMM2Vec-CNN 47

25 Confusion Matrix for Word2Vec Family Experiments 48

26 Training Model Accuracy vs Loss For Word2Vec-CNN 49

27 Confusion Matrix for ELMo Family Experiments 50

28 Training Model Accuracy vs Loss For ELMo-CNN 51

29 Confusion Matrix for BERT Family Experiments 51

30 Training Model Accuracy vs Loss For BERT-CNN 52

31 Comparison of hybrid machine learning approaches for families . 54

A.32 Confusion Matrix for HMM2Vec-SVM Category Classification . . 60

A.33 Confusion Matrix for HMM2Vec-KNN Category Classification . . 61

A.34 Confusion Matrix for HMM2Vec-CNN Category Classification . . 62

A.35 Training Model Accuracy vs Loss For Hmm2Vec-CNN For Category 63

A.36 Confusion Matrix for Word2Vec-SVM Category Classification . . 63

A.37 Confusion Matrix for Word2Vec-KNN Category Classification . . 64

A.38 Confusion Matrix for Word2Vec-CNN Category Classification . . 65

A.39 Training Model Accuracy vs Loss For Word2Vec-CNN For Category 66

A.40 Confusion Matrix for ELMo-SVM Category Classification 66

A.41 Confusion Matrix for ELMo-KNN Category Classification 67

A.42 Confusion Matrix for ELMo-CNN Category Classification 68

A.43 Training Model Accuracy vs Loss For ELMo-CNN For Category . 69

A.44 Confusion Matrix for BERT-SVM Category Classification 69

A.45 Confusion Matrix for BERT-KNN Category Classification 70

A.46 Confusion Matrix for BERT-CNN Category Classification 71

A.47 Training Model Accuracy vs Loss For BERT-CNN For Category 72

A.48 Confusion Matrix for HMM2Vec-SVM Category Classification with Other Symbol (41 Calls) 72

A.49 Confusion Matrix for HMM2Vec-KNN Category Classification with Other Symbol (41 Calls) 73

A.50 Confusion Matrix for HMM2Vec-CNN Category Classification with Other Symbol (41 Calls) 74

A.51 Confusion Matrix for HMM2Vec-SVM Category Classification with calls in all categories 75

A.52 Confusion Matrix for HMM2Vec-KNN Category Classification with calls in all categories 76

A.53 Confusion Matrix for HMM2Vec-CNN Category Classification with calls in all categories 77

A.54 Confusion Matrix for Word2Vec-SVM Category Classification with calls in all categories 78

A.55 Confusion Matrix for Word2Vec-RF Category Classification with calls in all categories 79

A.56 Confusion Matrix for Word2Vec-CNN Category Classification with calls in all categories 80

LIST OF TABLES

1	Number of Samples for Malware Categories	20
2	Number of Samples for Malware Families	21
3	Percentage Distribution of Top Calls	22
4	Hyperparameters Tested for Classifiers	26
5	Classification Report for HMM2Vec Category Classification	29
6	Classification Report for Word2Vec Category Classification	33
7	Classification Report for ELMo-RF Category Classification	35
8	Classification Report for BERT-RF Category Classification	37
9	Hyperparameters Selected for Category Classification	38
10	Classification Report for HMM2Vec-RF Category Classification with calls in all categories	41
11	Classification Report for Word2Vec-KNN Category Classification with calls in all categories	43
12	Classification Report for best HMM2Vec Family Classification	45
13	Classification Report for best Word2Vec Family Classification	47
14	Classification Report for best ELMo Family Classification	47
15	Classification Report for best BERT Family Classification	52
16	Hyperparameters Selected for Family Classification	53

CHAPTER 1

Introduction

In the past few decades, technology has become integral to several parts of our daily lives and computers are omnipresent. With such advancement, it is no surprise that attackers have found different ways to disrupt and damage these computer systems for personal gain. Such attackers work by trying to inject some malicious code into normal-looking programs which can then in turn do several different kinds of damage. They may steal money, and personal information, bombard the computer with downloads so the system fails or even encrypt your data asking for a ransom in exchange for a safe resolution. This malicious code can be defined as malicious software or malware. Over the years, malware has advanced to the point that it can infect virtually any system such as laptops, mobiles, and even internet services. According to the SonicWall Cyber Threat Report in 2023, the number of malware hits has increased for the first time in over three years and is recorded at a very high value of 5.5 billion [5]. Owing to such circumstances, the task of malware detection and classification has become ever so critical.

Malware classification can be defined as figuring out which class or family a particular malware belongs to. The rationale behind this stands that malware belonging to the same families and categories exhibit similar behavior in their execution. This is a fundamental problem in malware analysis. Current tools for the same employ a signature-based detection algorithm wherein they have a database of signatures of malware that have been previously analyzed. However, this technique often fails when the malware in question is new and has not been seen that much before. Making the job even more difficult is the fact that malware developers continuously find new ways to evade detection. To solve this problem, a lot of the new research has focused on how machine learning can be employed to detect and classify malware.

Malware can be divided into categories such as Worm and Trojan based on how they interact with the system and these categories can further be divided into another level which is families. Malware with the same characteristics such as a common code base, or common authors is considered to be part of the same family. These common characteristics can be used and exploited to classify malware.

In this research, the use of API calls as features are considered for malware classification. We use word embedding techniques to create vector representations from them and use these as input to different multi-class classifiers. The word embedding techniques considered in this research include HMM2Vec, Word2Vec, Embeddings from Language Model (ELMo), and Bidirectional Encoder Representations from Transformers (BERT). These techniques play a crucial role in our classification as the results are highly dependent on the ability of the generated embeddings to understand the latent features in the call sequence. In all these cases, we experiment with the same multi-class classifiers that are k-Nearest Neighbours (kNN), Support Vector Machines (SVM), Random Forest classifiers (RF), and Convolutional Neural Networks (CNN). These hybrid techniques are implemented and tests are conducted on a wide variety of malware and all the results are compiled into this report. Fig 1 below describes the layout for how the experiments were conducted starting with the analysis of malware to extract logs and concluding in attempting classification. The process remains unchanged for most techniques except HMM2Vec, where we add an extra step to map the calls to numbers. The relevant details have been provided in Chapter 4.

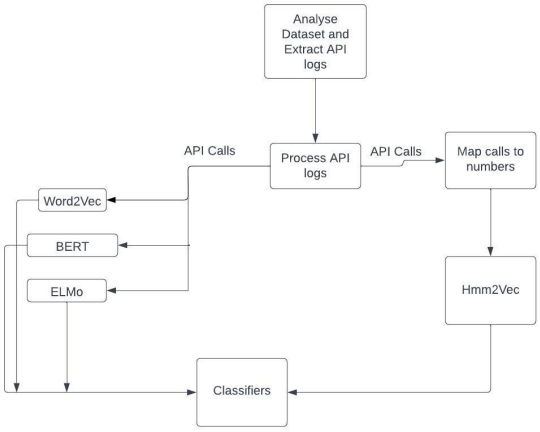


Figure 1: General Layout of Experiments

The remainder of this report is organized as follows. Chapter 2 discusses the related work and previous literature around malware classification followed by Chapter 3, which provides a deep dive into some of the background concepts that are needed to understand the work done. In Chapter 4, the data set is introduced and we go into the details of the proposed experiments as well as the results garnered. Finally, we end with Chapter 5 which concludes the work done and builds upon it with some ideas that could potentially improve the work done.

CHAPTER 2

Related Work

Malware classification is among some of the most researched problems because of it's global impact and as such several machine learning techniques have been identified to try to solve this problem. Within them, a common methodology is to study how the malware interacts with a system and extract features from them which could be helpful in understanding their behaviour. These features may include a number of things such as opcodes, register changes, file system changes, API calls made and so on. These features can be extracted using static analysis or dynamic analysis of the malware executable.

Several studies have used API call logs as features to detect or classify malware and they have achieved good results. A malware under execution may cause the system to make different API calls and these are good features for machine learning as usually they are used in a specific way. To extract these calls, we need to run the malware executable which often requires a virtual environment [6]. Authors in [7] proposed a method for selecting a subset of API calls and achieved 0.98 accuracy in malware classification using algorithms like SVM and Random Forest. In a similar scheme, the authors in [8] propose enhancing the API call information using Term Frequency and Inverse document frequency (TF-IDF) to select only the most important calls for malware detection achieving near perfect accuracy. These results provide enough evidence to argue the effectiveness of API calls as a feature.

While algorithms like SVM and Random Forest generally perform well at classification tasks, recent works have shown that techniques using deep neural networks can be successfully used because of their ability to uncover and learn complex relationships. In [9], the authors propose a CNN based architecture where they convert malware binaries to images recording an impressive 98% accuracy. In [10], the authors discuss

data mining techniques to select features and classify them using models like kNN and Naive Bayes. The selection of appropriate techniques to enhance the relevance of information in our features can be crucial in classifying malware.

Word embeddings are learned representations of text that are used in natural language processing (NLP). Since their advent, they have been employed in a number of varied tasks with very promising results. Among these, malware classification has also emerged as a field where these techniques prove useful. This statement is supported by the work done in [11], where API call sequences are converted to numeric vectors using various methods. Another popular algorithm used to generate these representations is Word2Vec, developed at Google [2]. In [12], the author uses Word2Vec to extract vectors from machine code instructions and demonstrates the algorithms ability in tasks of malware detection.

Hidden Markov Models (HMM) have found an effective use-case in malware analysis because of their ability determine patterns in sequences that are generally not observable otherwise. Several research works have tried modelling HMMs on various malware features and have shown promising results. In [13], the authors conduct a comparison between the efficacy of opcode sequences and API call sequences by training HMMs on them and conclude that the latter perform better. In a similar study, A.Damodaran et al. [14] use HMMs and reach a similar conclusion. They argue that while opcodes are generally good, they fail to predict some families due to obfuscation techniques which are likely to affect them more than API calls. In [15], the author shows HMM2Vec, a technique for generating vectors using HMMs opcodes as features. The work compares the performance of HMM2Vec with Word2Vec and deliver promising results with an accuracy of about 93 percent in both cases thereby validating the HMM2Vec approach.

One drawaback of the Word2Vec algorithm is the inability of producing vectors

that have contextual information. Further developments in the field of NLP led to the development of several other models for generating embeddings which could overcome this problem. One such model was introduced in [16] called Embeddings for Language Model (ELMo) which is based on the biLSTM model and it achieved state of the art results in several common NLP tasks. Another breakthrough came with the introduction of the transformer architecture and self-attention mechanism in [17]. With the increased interest in transformed based architectures, J.Delvin et al. [4] introduced Bidirectional Encoder Representations from Transformers (BERT) which again achieved state of the art results. In [18], the authors use malware opcode and compare the performance of these newer techniques with HMM2Vec and Word2Vec for the task of malware family classification. Both approaches performed relatively well and improved performance over the former two with results indicating that BERT slightly outperforms ELMo in the given task. More on how each of these techniques work is discussed in the next section. This research is highly motivated by the work done in [15][18]. The work done here follows a similar structure to these works but instead of opcode sequences, we rely on the API call sequences for classification.

CHAPTER 3

Background

This chapter explains the basics of the various tools and technologies that were used in this research. It starts with malware analysis and how we can extract API calls from a given executable. Following this, we briefly describe the workings of the word embedding models used namely HMM2Vec, Word2Vec, BERT, and ELMo. At last, we discuss the various multi-class classifiers used here. In particular, we talk about SVM, kNN, Random Forest, and CNN. Following this is Chapter 4 which talks about the dataset, the experiments performed, and the results that were achieved.

3.1 Malware Analysis

3.1.1 Malware Category and Family

Any program that is created with the intent to disrupt or harm a computer system can be defined as malware. As such, based on their behavior malware can take many different forms and present as one among various categories such as Ransomware and Trojan. This broad categorization dependent on a malware's characteristics, behavior, and aim, is referred to as the category of malware. On the other hand, a malware family is a more specific characterization within a category wherein we group them based on functional similarities such as the code base or origin i.e. a single category has multiple families. Due to this, the classification of malware by category is generally more complex than classification by family. In this paper, we focus on both of these categorizations and our experiments try to classify malware in both groups.

3.1.2 Dynamic Analysis with Buster Sandbox Analyzer and Sandboxie

The term dynamic analysis can be explained as the analysis of software that is executed in a controlled environment. In contrast, static analysis refers to the analysis of software that is not under execution. Although dynamic analysis usually incurs a larger overhead, previous work has shown that they are suitable for deriving an

accurate model of the malware [14] [7]. Common information that can be derived from dynamic analysis of a malware executable is API calls, system calls, and register changes and these can be useful as features in several malware tasks. In this research, we specifically focus on the API calls extracted. In the lifetime of its execution, malware may use various API calls for different purposes such as accessing a file system or connecting to a remote server. This behavior can be exploited to understand the malware better.

There are a lot of tools that can help us in extracting the required information. The tool that was used in this research is called Buster Sandbox Analyzer (BSA) [19]. BSA is a free, open-source dynamic analysis tool designed to detect if processes exhibit malicious activities. BSA works by executing the malware in a controlled virtual environment so the host system is not affected by any harmful activities. It does so by working in conjunction with another software called Sandboxie which provides a sandboxed environment to run programs in[20]. One of the best features of BSA is that it can capture an executable's workings and automatically generate reports based on it including details such as API calls network communications, file system changes, and various other information. Together, BSA and Sandboxie provide a powerful method of analyzing malware and as such are used in this research to analyze the malware samples. Fig 2. shows a screenshot of malware under analysis with BSA. The API call log that we use for our experiments can be seen as generated by the program.

3.2 Word Embedding Techniques

As mentioned previously, word embedding techniques are primarily used in natural language processing tasks to numerically quantify the distance among words in a vocabulary. In this research, we used these techniques to generate features for our classifiers. It is expected that by using these techniques, we will be able to identify

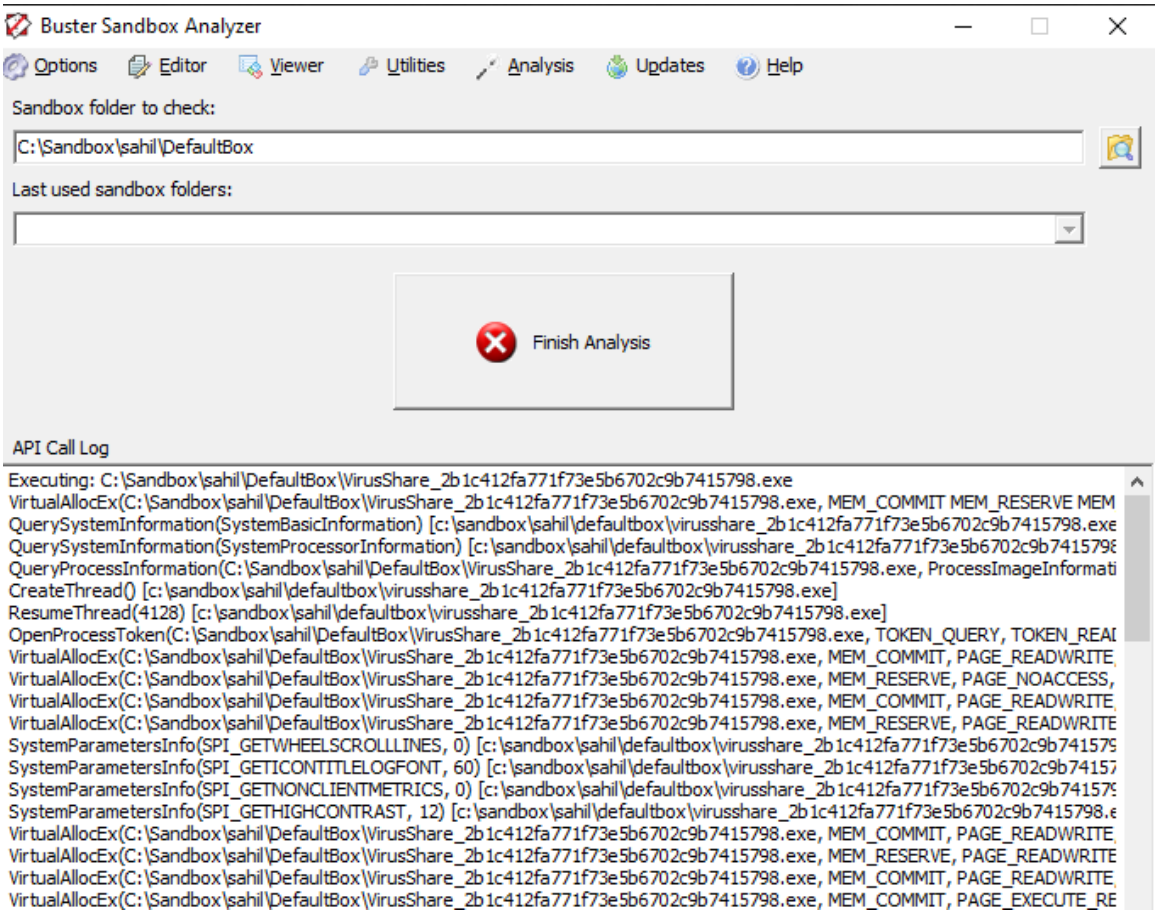


Figure 2: Malware under analysis with BSA

relationships that could be better understood by the classification algorithms. In this section, we discuss the embedding techniques used in this research starting with HMM2Vec, an embedding technique based on HMMs, followed by the well-known Word2Vec. Finally, we discuss some of the relatively newer techniques ELMo and BERT which can understand the context among the words in our vocabulary.

3.2.1 HMM2Vec

The hidden Markov Model is a statistical model in which a system is assumed to be a Markov process of order one with a number of hidden states that are not directly observable. The aim of an HMM is to figure out the most probable sequence

of states that could have generated the data. Usually, HMM can be represented by three matrices A, B, and Pi which denote the hidden state transition probability, the observation probability matrix and the initial state probabilities respectively [1]. The number of hidden states that the HMM may infer is donated by N and is usually selected by the user. Other notations that are used are T, to define the length of the observation sequence, and M, which represents the number of unique symbols in the vocabulary. The values of M and T are most commonly derived on the basis of the training data set.

To shed more light on the workings of HMM, we use the example as explained in [1]. The author considers the problem of training an HMM on English text where each letter is a part of M, our observation symbols. Therefore, the total number of observation symbols comes out to 27 including the 'space' as a letter. The HMM is trained on a large sample of English text and the number of hidden states is set to 2. Fig 3. shows the initial and final B-transpose of the HMM. Interestingly, we notice from the converged B matrix that the HMM has identified the two states to be vowels and consonants without any prior assumptions. This shows that the HMM was able to identify a statistically significant distinction in the English text.

HMM2Vec is a technique in which we calculate the cosine similarity between any two vectors. We consider the row of the converged B transpose matrix as the vector representation of that particular letter. This means that for each individual letter, we have a vector of length 2. The length of the vector depends on the number of hidden states.

The formula for cosine similarity between any two vectors X and Y is given by

$$cos(X, Y) = \frac{\sum_{i=0}^{n-1} X_i Y_i}{\sqrt{\sum_{i=0}^{n-1} X_i^2}\sqrt{\sum_{i=0}^{n-1} Y_i^2}} \tag{1}$$

The cosine similarity is the measure of the cosine angle and can be a value between

	Initial		Final	
a	0.03735	0.03909	0.13845	0.00075
b	0.03408	0.03537	0.00000	0.02311
c	0.03455	0.03537	0.00062	0.05614
d	0.03828	0.03909	0.00000	0.06937
e	0.03782	0.03583	0.21404	0.00000
f	0.03922	0.03630	0.00000	0.03559
g	0.03688	0.04048	0.00081	0.02724
h	0.03408	0.03537	0.00066	0.07278
i	0.03875	0.03816	0.12275	0.00000
j	0.04062	0.03909	0.00000	0.00365
k	0.03735	0.03490	0.00182	0.00703
l	0.03968	0.03723	0.00049	0.07231
m	0.03548	0.03537	0.00000	0.03889
n	0.03735	0.03909	0.00000	0.11461
o	0.04062	0.03397	0.13156	0.00000
p	0.03595	0.03397	0.00040	0.03674
q	0.03641	0.03816	0.00000	0.00153
r	0.03408	0.03676	0.00000	0.10225
s	0.04062	0.04048	0.00000	0.11042
t	0.03548	0.03443	0.01102	0.14392
u	0.03922	0.03537	0.04508	0.00000
v	0.04062	0.03955	0.00000	0.01621
w	0.03455	0.03816	0.00000	0.02303
x	0.03595	0.03723	0.00000	0.00447
y	0.03408	0.03769	0.00019	0.02587
z	0.03408	0.03955	0.00000	0.00110
space	0.03688	0.03397	0.33211	0.01298

Figure 3: Initial and final B transpose [1]

0 and 1. The closer the value is to 1, the more similar two vectors are and vice versa. Based on this understanding, consider the following example explained in [15].

V(a) = (0.13845, 0.00075) V(e) = (0.21404, 0.00000)

V(s) = (0.00000, 0.11042) V(t) = (0.01102, 0.14392)

Now, if we use the formula to calculate their similarity, we can see that the vowels are very close together as cos(V(a), V(e)) = 0.999 which is what we expect

since they belong to the same state. On the other hand, cos(V(a), V(t)) = 0.0817 representing that these letters are not very similar. Based on this, we can infer that these embeddings carry some information that may help us understand the vocabulary better. Therefore, we can train HMMs on distinct observation sequences and define numerical vectors using the B matrix. More detailed information on HMM and how they work can be accessed here [1].

3.2.2 Word2Vec

Word2Vec is a word embedding technique introduced by Tomas Mikolov at Google in 2013 [2]. The algorithm is based on a shallow neural network that can be used to embed features in a high-dimensional space. The trained extracted embeddings can be used in several tasks as the words that are similar in context are represented closely together.

Word2Vec consists of two algorithms that can be trained to generate the embeddings for the words which are

3.2.2.1 Common Bag of Words (CBOW)

In this algorithm, the embedding for a given word is generated by combining the distributed representations of the other words that appear in its context. The context is usually defined as a window where the word to be represented is the middle word and all others are the context.

3.2.2.2 Skip N-gram

In this algorithm, the model tries to model a given word by trying to predict the neighboring words that would appear in its context i.e. given an input word, the model tries to predict what the words are that either appears immediately before or after.

Looking at the architectures from Fig 4, we can see that both techniques are mirror

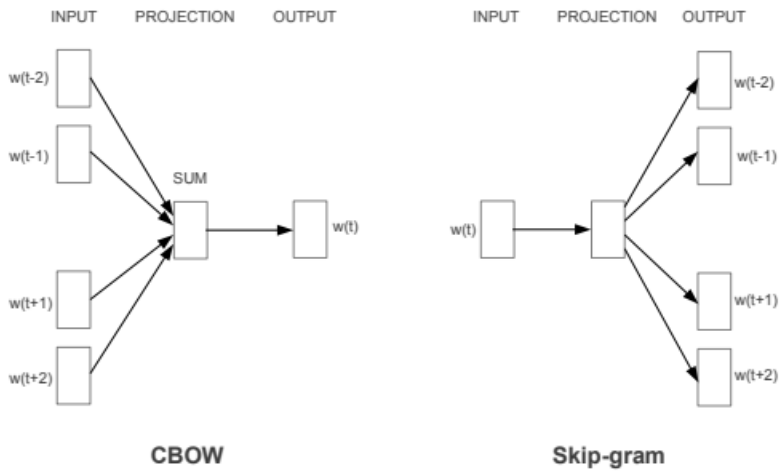


Figure 4: CBOW and Skip-Gram Architecture [2]

images of each other. While both of these models are quite effective for generating embeddings, the Skip-gram model is more powerful as it models every single word in the vocabulary. However, due to this it also incurs a high overhead compared to the CBOW. In our experiments, we use the default model available which is based on the CBOW architecture.

Since the Word2Vec similarity is also based on the cosine similarity discussed in the previous section, the experiments are somewhat analogous to the HMM2Vec experiments. However, Word2Vec offers some advantages over the HMM2Vec the main one being that HMM2Vec is based on a Markov model of order one which means it is limited in the context information it may capture compared to this. Another drawback with HMM is the training time, as it takes an order of N^2T work which means for larger values of N and T, training can be cumbersome.

3.2.3 ELMo

ELMo was introduced by Matthew E. Peters et al. in [16] in 2018. ELMo is a bi-directional language model capable of generating contextualized vectors for a particular

word by capturing both semantics as well as the syntax for said word [18]. This means that similar words represented in different contexts are represented differently which allows us to capture subtle differences in the language. The architecture of ELMo consists of two LSTM networks called the Forward layer and Backward layer as shown in Fig 5. These layers are trained on the tasks of next-word prediction and previous-word prediction respectively and work in conjunction thereby processing input in both directions and capturing meaningful relationships in both directions. The final word embedding is generated by concatenating these contextualized embeddings and scaling them with the help of a normalizing factor. These bidirectional layers are also supported by a task-specific layer on top which helps transform the embeddings from our layers into a suitable form for the task at hand, by usually projecting them into a lower dimension.

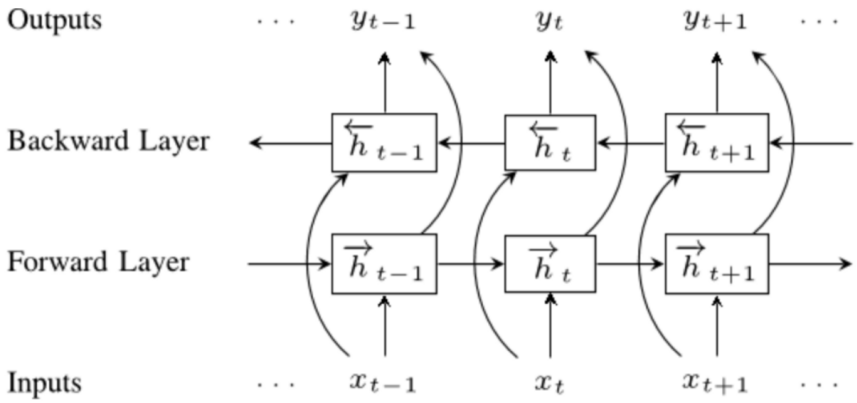


Figure 5: ELMo Architecture [3]

Apart from generating contextualized vectors, ELMo offers another advantage over the previously mentioned techniques wherein we do not necessarily have to handle out-of-vocabulary words as ELMo can create embeddings for these using character-level representations. This makes ELMo quite powerful in tasks that are highly domain-specific and may contain lots of out-of-vocabulary words. More information

on ELMo can be found here [16].

3.2.4 BERT

BERT, standing for Bidirectional Encoder Representations from Transformers, was introduced in [4] at Google. It is based on the transformer architecture that was introduced in 2017 by Vaswani et al [17]. The transformer architecture introduced the concept of self-attention, in which each input embedding is represented as three vectors namely the query, key, and value vectors. These vectors are then used to calculate attention scores among the different units and combined to produce a final sum. This was a key improvement over other models as it meant that we could now capture long-range dependencies without having fixed-size windows. Like ELMo, BERT uses self-attention and also produces contextualized word embeddings.

The architecture of BERT consists of just a stack of encoder models, twelve in number to be precise. The BERT model produces embeddings by following a two-step procedure of pre-training and fine-tuning. During pre-training, the model is trained on a very huge amount of text data and also goes under training on a couple of specific tasks which are next sentence prediction (NSP) and masked language modeling (MLM). In MLM, some of the words in the input are masked and the model is tasked with predicting the original masked word which helps the model learn robustly. In NSP, the model is tasked with deciding which among two of the input sentences precedes the other in a text. This helps the model learn relationships among sentences. Following all this, fine-tuning of these pre-trained parameters is done. This fine-tuning is based on the task to be performed and incorporates training on a labeled data set to optimize the parameters generated in pre-training.

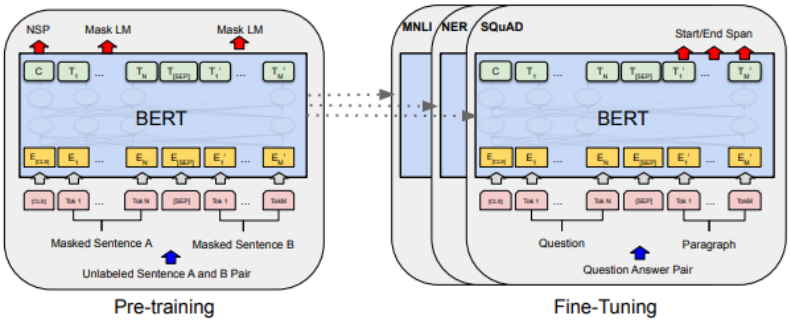


Figure 6: BERT Pre-training and Fine-tuning [4]

3.3 Classifiers

Given a labeled data set, classification can be defined as the process of predicting the label for a given input. This prediction is done on the basis of input features, which are analyzed to derive commonalities among the input and previously analyzed inputs. There are several models that have been developed and found to be of great use in classification tasks such as SVM. In our research, we use the vectors generated by the word embedding techniques as features to try and classify malware in their respective class or family. We use various classifiers and the following section describes the concepts on which these are based. All the classifiers used here have proven to be effective in the task of malware classification using different features as inferred by the works done in [18] [9] [13] [15]. This section marks the end of this chapter and is followed by Chapter 4, which contains some implementation-specific details on these classifiers related to our experiments.

3.3.1 Support Vector Machine

Support Vector Machines (SVM) is a set of supervised machine learning algorithms that are hugely popular in classification tasks. In order to maximize the distance between classes, SVM tries to construct a hyperplane that can act as a separator. In [18], the authors have used SVM for the classification of malware features generated

using word embedding techniques and achieved promising results. SVM can help us identify subtle changes in malware samples as the hyperplane is capable of working in a higher dimensional space with the help of a kernel trick in cases where the data is not linearly separable. There are a number of kernel functions that we can use to achieve this high-dimensional mapping. Another important part of the algorithm is the cost regularization parameter C which helps avoid overfitting. It does so by allowing some classes into the boundary of the hyperplane. Once the hyperplane is set, the algorithm predicts classes by mapping data points to the high-dimension space and judging the relative position with the hyperplane. More information on SVM and its mathematical proof has been provided here [21].

3.3.2 Random Forest

Random Forest (RF) is an ensemble learning technique that uses multiple decision trees to predict a class label introduced by Breiman in [22]. A decision tree is a supervised machine learning algorithm that works by recursively splitting up the input into smaller similar subsets. While they have their advantages, a single decision tree tends to overfit as the depth and complexity of the tree increase. Random Forest tries to solve this problem by combining the decisions from several decision trees. This is because, since the algorithm bags features along with the observations, the trees tend to protect each other from individual errors and avoid overfitting. The algorithm works by assigning samples at random to different decision trees and then averaging the results. The class label predicted is the one that received the most votes from the individual trees. A good explanation of how to use these algorithms can be found at [23].

3.3.3 k-Nearest Neighbors

k-Nearest Neighbors is a simple supervised machine learning algorithm that makes use of a sample's neighboring data points to predict where it belongs. First, the k nearest samples are taken and then the distance between them and the input sample to be predicted is calculated. The distance measure is a parameter that we can decide based on the use case such as Euclidean distance or Minkowski distance and this value is what the prediction is based on. The algorithm does not include a training phase which is why is referred to as a lazy classifier. When implementing kNN, The value of k is of paramount concern as too small values can lead to overfitting while very large values can lead to performance and accuracy degradation [18].

3.3.4 Convolutional Neural Networks

Neural networks are algorithms modeled to work like the human brain. A convolutional neural network or a CNN is a type of feedforward deep neural network introduced in [24] that has found success in several image classification tasks. A CNN consists of several layers including an input layer, several hidden layers, and an output layer. Among these, the convolutional layer is the core building block. It contains a set of filters that slide over an input to extract relevant features by exploiting any spatial patterns.

CHAPTER 4

Dataset and Experiments

4.1 Dataset

The dataset used in this project contains a number of windows executable files. The executables were analyzed under Buster Sandbox Analyzer to extract the list of API logs the malware called under execution. In total, we extracted 782 samples from the initial raw data set and created 2 distinct sets from this – one for categories with about 583 samples and one for families with about 492 samples. These datasets were used in the experiments that are described in the upcoming sections.

For categories, we separated the 583 samples into 11 malware categories. The description of these is given below:

Adware: This is a type of malware that floods the computer with unwanted ads in the form of pop-ups or banners.

Backdoor: This type of malware tries to create unauthorized entry points into a system. These entry points could be used at any time to gain access to said system.

Modifier: Modifiers attack the system by modifying the behavior of other programs ranging from changing a browser's homepage to even installing additional malware.

PWS: PWS or Password Stealer is a type of malware specifically designed to steal credentials and other sensitive information.

Rogue: Rogue software is a type of malware that disguises itself as a legitimate antivirus software and tries to get the user to pay for it.

Tool: These are programs that can do a number of harmful tasks such as creating viruses or preventing an antivirus from detecting malicious software.

Trojan: Trojans usually present as legitimate programs that are actually trying to do malicious activities such as gain control of a system or steal information.

Trojan Downloader: This type of malware downloads and installs a number of software on the system, including several malware.

Trojan Monitoring Service: This type of malware spies on the computer and tries to steal important information. It may do so by recording keystrokes and screenshots.

Virus: Viruses are type of malware that attach themselves to legitimate programs. These are able to spread rapidly and can cause significant damage.

Worm: This is a type of malware that create copies of itself and spreads it to other systems connected to a common network.

Table 1 shows the number of samples for each of these categories in the data set.

Table 1: Number of Samples for Malware Categories

Malware Category	Sample Count
Adware	50
Backdoor	53
Modifier	52
PWS	50
Rogue	53
Tool	60
Trojan	53
Trojan Downloader	52
Trojan Monitoring Service	53
Virus	55
Worm	52

For families, we were able to separate the samples into 7 malware families. A brief description of these is as follows:

Adload: Adload is a family of Adware and displays unwanted advertisements on a system under attack.

Bancos: : Bancos is a trojan that is concerned with banking details. It steals

sensitive information such as credit card numbers and bank credentials.

Onlinegames: This family of malware is designed to steal information such as passwords of online games by monitoring keystrokes and such.

VBInject: Is a type of malware family that injects malicious code in other programs and gathers data like system settings, network configuration and so on.

Vundo: Is again a malware family that is known to cause unwanted ads on a system thereby leading to performance degradation.

Winwebsec: Is a malware that disguises itself as antivirus or security software followed by asking the user to pay money to clean up the system.

Zwangi: Zwangi is a type of Browser Modifier malware that infects Windows systems and redirects the user to different webpages that may cause harm to the victim.

Table 2 shows the number of samples for each of these families in the data set.

Table 2: Number of Samples for Malware Families

Malware Family	Sample Count
Adload	70
Bancos	71
Onlinegames	70
VBInject	70
Vundo	71
Winwebsec	70
Zwangi	70

A single sample in either of the datasets consists of the API call log sequence for that malware. The logs are usually presented in the form of the API name with some other information such as file path as seen in Fig 2. We initially preprocess these files to remove all other information except the API call names. Therefore, we have for each sample a sequence of API calls that were invoked when the sample was under

execution. Fig 7 shows a sample preprocessed log with just the API names.

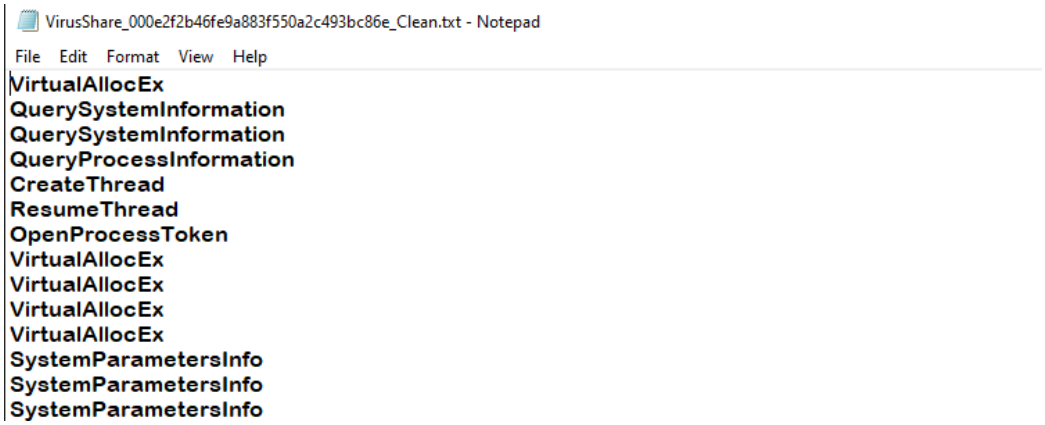


Figure 7: Sample Preprocessed API log

The number of distinct API calls that we saw in total was about 94. While using all calls can be good, it would incur significant processing times, and as such it is important to select the most relevant or the most important calls. To determine the importance of a single call, we simply use the frequency counts of all API calls. Work done by [18] follows a similar approach where they consider working with the Top 20 opcodes. For the experiments we conducted, we consider the top 20 and top 40 calls giving us two distinct sets of features. The total percentage of calls these cover over the entire set of calls is given in Table 15.

Table 3: Percentage Distribution of Top Calls

Calls	Count (Percentage)
Total (Category)	233539 (100%)
Top 40 (Category)	232080 (99.37%)
Top 20 (Category)	223698 (95.78%)
Total (Family)	160813 (100%)
Top 40 (Family)	159987 (99.48%)
Top 20 (Family)	154868 (96.3%)

In both the cases of families and categories, we see that the same API call has

the highest frequency which is the VirtualAllocEx.

4.2 Experiments

Similar techniques were used with regard to both categories and families i.e. even though the data may change, the process of how we conduct the experiments remain the same. This section provides some insight into this process.

4.2.1 HMM2Vec Experiments

The first technique used for feature extraction is HMM2Vec, which is based on Hidden Markov Models.The sequence of API calls that has been processed is used as the observation sequence to train the HMM. For the experiments, all HMMs were initialized with N = 2 as it is known to give the best classification accuracy [18]. The number of unique observation symbols i.e. M is either set to 20 or 40 depending on the number of calls retained. For training, we used a modified version of the HMM code available here [25]. For each of the malware samples, we obtain the best model after 20 random restarts. Another set of experiments is done with the total number of calls set to 41 instead of 40. In this case, all calls that lie outside the top 40 are replaced with a symbol denoting 'Other' instead of being deleted. This is done to try and see if retaining the whole sequence would be more helpful for HMM to understand. For training, each call in the sequence is mapped statically to a number for the training of the model which allows us to define positions for each call. The B matrix of the trained model is used to derive the vector representation. The final vector for a sample is created by appending the rows of the B matrix resulting in a one-dimensional vector of length N*M.

One of the challenges with HMM2Vec is maintaining consistency among the rows of the B matrix. To elaborate, consider the English language example we used earlier where any of the 2 rows of the matrix could represent the letter being a vowel or a

consonant i.e. the ordering of these rows is not fixed. To overcome this, we maintain that the row which shows convergence for the most popular call, VirtualAllocEx, will always be row 0 of the matrix we use. In cases where the convergence is in state 1, we swap the rows of the matrix thereby maintaining consistency among the features.

4.2.2 Word2Vec Experiments

Each call sequence is viewed as a stream of words during Word2Vec training. The implementation for this is done using the gensim module in Python [26]. The module allows us to specify various parameters for training such as vector size and the algorithm to use. As mentioned previously, we use the CBOW algorithm which is the default setting. In addition, we set the vector size to 2 to maintain consistency with the number of states from our HMM experiments. To obtain the final vector for a sample, we append the individual vectors for each word similar to HMM to obtain a 1D vector of length.

$$vector\_size * numberofwords \tag{2}$$

4.2.3 ELMo

The ELMo model we use is available to us in Python through the TensorFlow module. This model generates context-aware embeddings and is pre-trained on a large amount of English text. ELMo, just like Word2Vec, requires the input as a stream of words and outputs a vector of size 1024 for each of the tokens in its input sequence. Since each of our samples may contain a different number of words, the embeddings generated would be of different sizes. To maintain uniformity, we sum up the individual vectors for each word and divide them by the total number of samples. The final vector for a sample is a one-dimensional vector of length 1024.

4.2.4 BERT

BERT, like ELMo, uses the self-attention mechanism to generate context-aware embeddings. In this research, we use the BERT Base model available through the

transformers module. This Base model is trained on a large corpus and consists of 110 million parameters. It is a smaller version compared to the other BERT models but it has previously performed quite well in malware classification [18]. BERT only works with sequences of length up to 512 so any longer sequences are truncated to meet this constraint that is we use the first 512 tokens. BERT works by adding two special tokens, namely CLS and SEP, to the sequences. The CLS is a special token that is used to represent the entire sequence. For this BERT model, the output is a tensor of size

$$batchsize * numberofwords * 768 \tag{3}$$

For classification, we create a final vector of size 768 by grabbing the embedding for the CLS token.

4.2.5 Classifiers

All of the classifiers namely SVM, RF, kNN, and CNN have several different parameters that can be optimized to get the best possible result for each set of features. To determine these, we make use of the Grid Search module available in the sklearn library. It works by specifying several different values for each of the parameters we want to tune, and then the module performs an exhaustive search with all possible combinations of the parameters. The combination with the best accuracy is selected. It is also very helpful in the fact that it does a cross-validation over 5 folds for each of the combinations to find the best one.This means for the training set that is fed into the GridSearch, it will validate the different combinations over 5 different intermediate test sets created off the original training set. Since our experiments are conducted with generally less samples per label, the cross validation aspect of this module helps increase the reliability and generalizability of these results. Table 4 below shows the parameters that we tested for the different classifiers.

Table 4: Hyperparameters Tested for Classifiers

Classifier	Hyperparameter	Tested Values
SVM	Kernel	rbf, linear, poly
	C	0.1, 1, 10, 100, 1000
	Gamma (only rbf)	10, 1, 0.01, 0.0001
Random Forest	n_estimators	200 to 1000 step 100
	max_depth	10 to 40 step 10
	min_samples_split	2, 5, 10, 50
	min_samples_leaf	1, 2, 5, 10
kNN	n_neighbors	1 to 19 step 2
	metric	minkowski, manhattan, euclidean
	weights	distance, uniform
CNN	epochs	100, 200, 1000
	learning_rate	0.1, 0.01, 0.001
	activation	relu, tanh
	optimizer	sgd, adam

Note that for all the word embedding techniques, we test 60 combinations each for SVM and kNN, 576 combinations for Random Forest and at last about 48 combinations for CNN and we repeat these experiments for both the Top 20 and Top 40 calls. In total, we test

$$2 * (30 + 60 + 576 + 48) = 1428 \tag{4}$$

combinations each for category classification and family classification per word embedding technique to determine the best parameters. In addition, for SVM experiments, we scaled the data between 0 and 1 using the StandardScaler functionality of sci-kit learn. This is because SVM performs better on data that is scaled and we observed the same as the accuracy improved significantly. For our CNN experiments, we create networks with 4 hidden layers and since we are attempting multiclass classification, we use the spars_categorical_crossentropy as our loss function. All experiments were performed with the same split, with 80 percent of the data being used for training

and the remaining 20 percent for testing. In the case of CNN, we split the training data further into 70% for training and 10% for validation.

4.3 Results

4.3.1 Malware Category Classification

For this set of experiments, we consider eleven malware categories that were discussed in Section 4.1.

4.3.1.1 HMM2Vec

For the HMM2Vec experiments, the best accuracy we achieved came with the Random Forest classifier at 0.69. kNN performed very close reaching 0.68 followed by SVM at 0.62. The worst performance came with CNN with 0.61 accuracy. Fig A.32 to Fig A.34 show the confusion matrix for each of the classifiers.

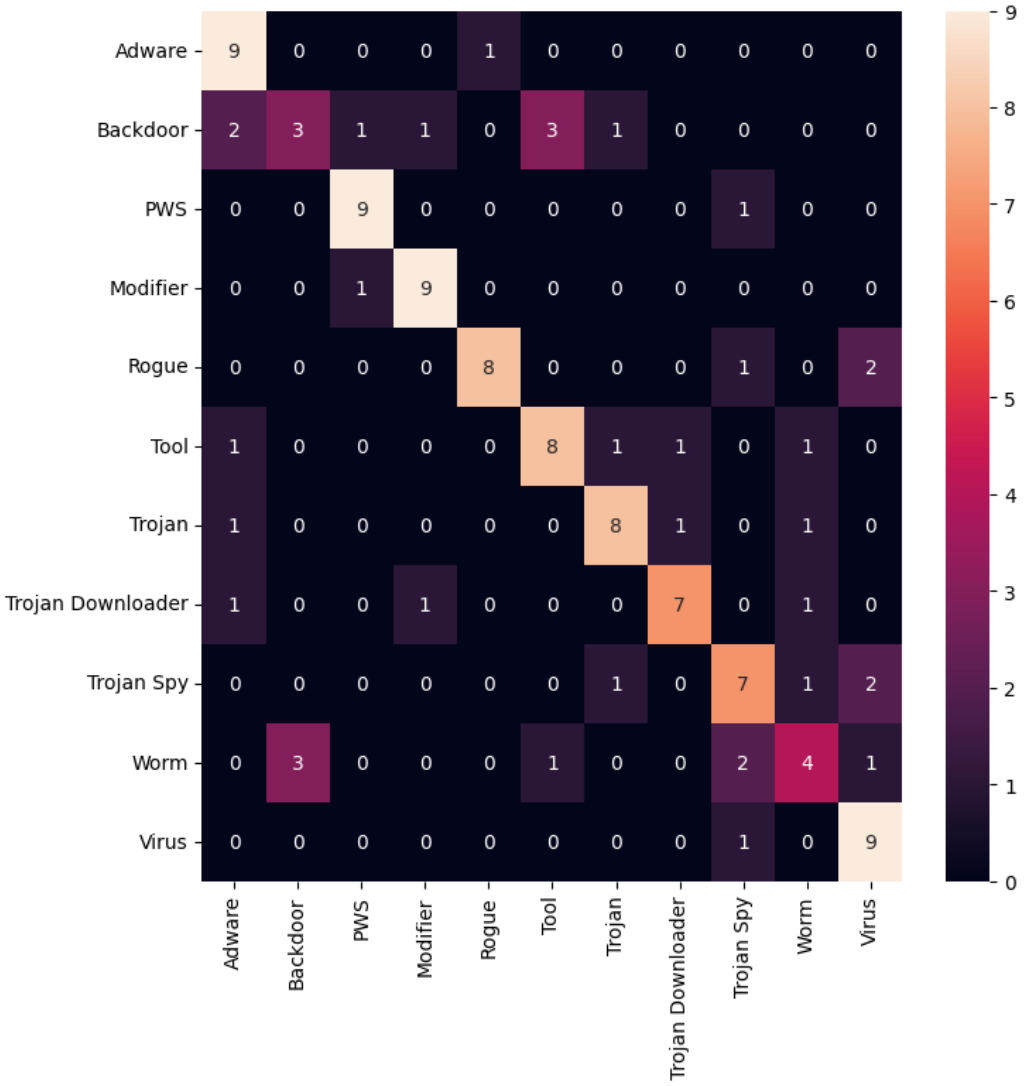


Figure 8: Confusion Matrix for HMM2Vec-RF Category Classification

From these results, we see a common pattern where the Worm category is continually misidentified with all classifiers. This may be because of high amount of variance within the samples in this class. In respect to the calls retained, for HMM2Vec we see that the Top 40 calls perform better than the Top 20 calls although not by too big of a margin. Fig 9 shows the accuracy achieved in both cases.

Table 5: Classification Report for HMM2Vec Category Classification

Category	Precision	Recall	F1-Score
Adware	0.64	0.90	0.75
Backdoor	0.50	0.27	0.35
PWS	0.82	0.90	0.86
Modifier	0.82	0.90	0.86
Rogue	0.89	0.73	0.80
Tool	0.67	0.67	0.67
Trojan	0.73	0.73	0.73
Trojan Downloader	0.78	0.70	0.74
Trojan Spy	0.58	0.64	0.61
Worm	0.50	0.36	0.42
Virus	0.64	0.90	0.75

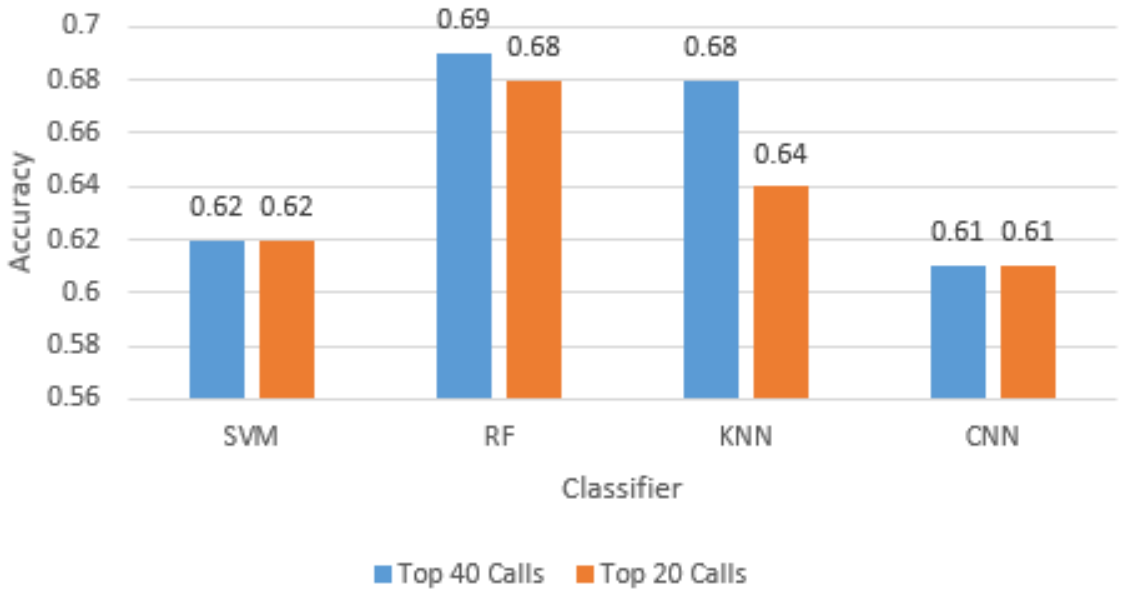


Figure 9: Top 20 calls vs Top 40 calls for HMM2Vec Category Classificaiton

For experiments done where we replaced all calls outside of the selected ones with a constant symbol, the performance decreased. This may be because the number of these calls that lie outside our selected set may add up to a higher number masking what the sequence actually represents in the output matrices. For these experiments,

the best accuracy we achieved came with the Random Forest classifier at 0.62. Fig 11 shows the accuracy of this approach compared to the one where we delete the calls.

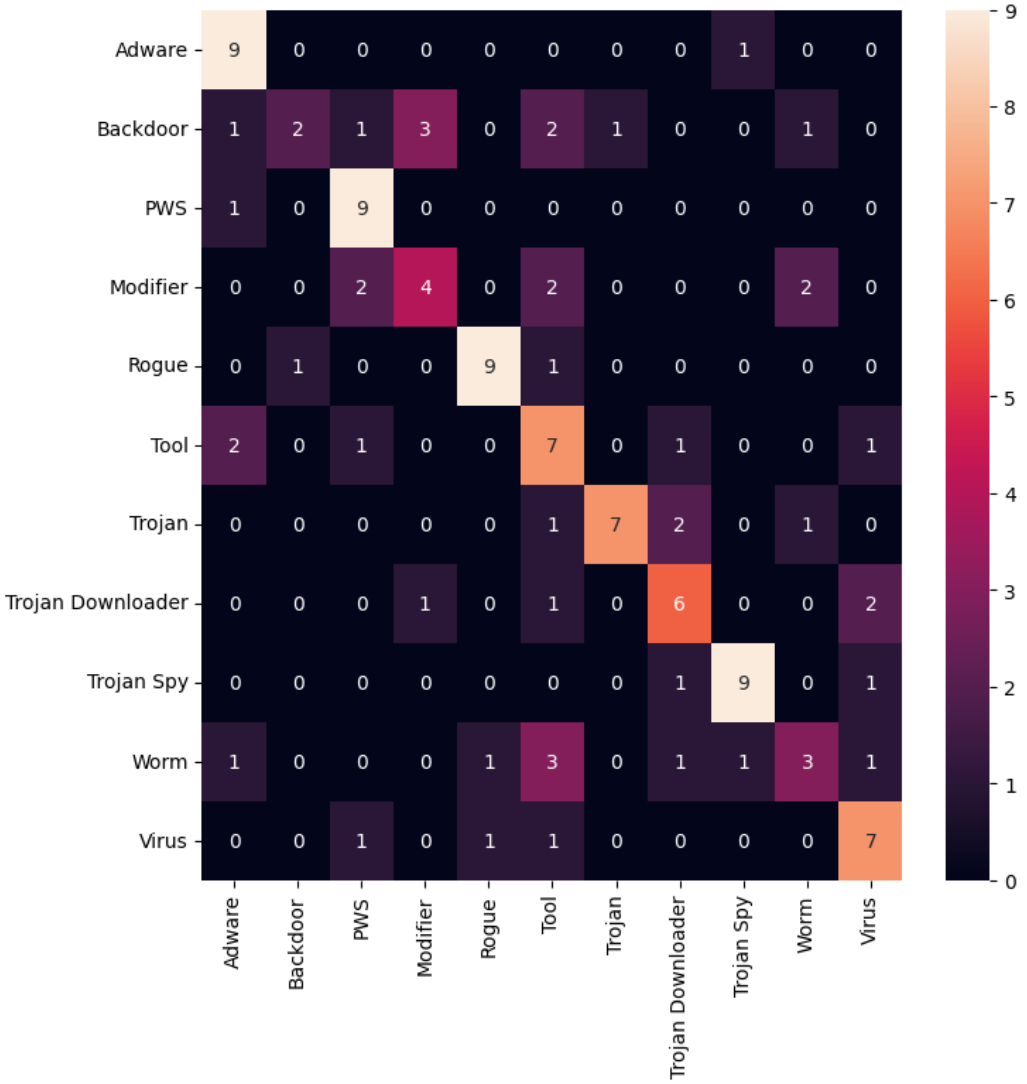


Figure 10: Confusion Matrix for HMM2Vec-RF Category Classification with Other Symbol (41 Calls)

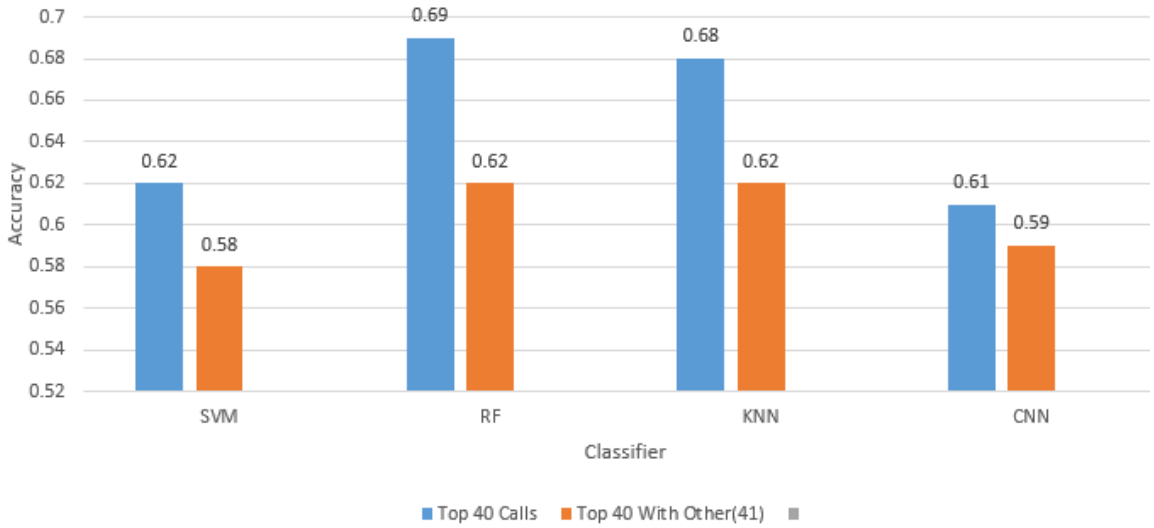


Figure 11: Top 40 calls vs Top 40 calls and Other (41) for HMM2Vec Category Classificaiton

4.3.1.2 Word2Vec

For the Word2Vec experiments, the best accuracy we achieved came with the Random Forest and kNN classifiers at 0.77. Overall, this was the highest accuracy achieved in the task of malware category classification. Word2Vec-SVM performed slightly worse at 0.76 whereas CNN came last with about 0.74.

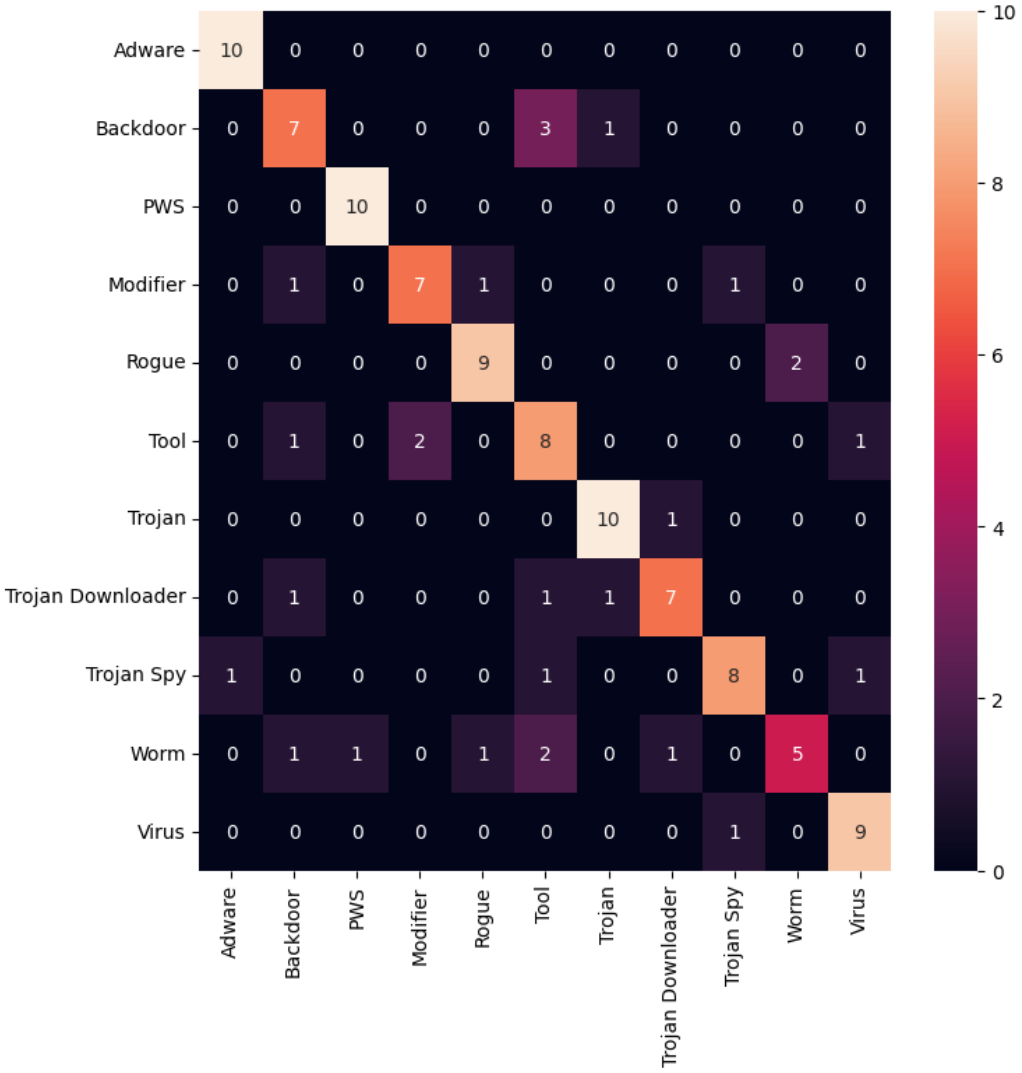


Figure 12: Confusion Matrix for Word2Vec-RF Category Classification

Again we see that the Worm class is the least accurate. However, compared to the HMM2Vec experiments we see that Word2Vec can predict the category Backdoor with much higher accuracy which is probably the cause of the boost in accuracy. With respect to the calls retained, for Word2Vec we see that the Top 40 calls perform significantly better than the Top 20 calls which was not the case for HMM2Vec. Fig 13 shows the accuracy achieved in both cases.

Table 6: Classification Report for Word2Vec Category Classification

Category	Precision	Recall	F1-Score
Adware	0.80	0.80	0.80
Backdoor	0.58	0.64	0.61
PWS	0.91	1.00	0.95
Modifier	0.80	0.80	0.80
Rogue	0.82	0.82	0.82
Tool	0.53	0.67	0.59
Trojan	0.83	0.91	0.87
Trojan Downloader	0.89	0.70	0.84
Trojan Spy	0.89	0.73	0.80
Worm	0.83	0.45	0.59
Virus	0.75	0.90	0.82

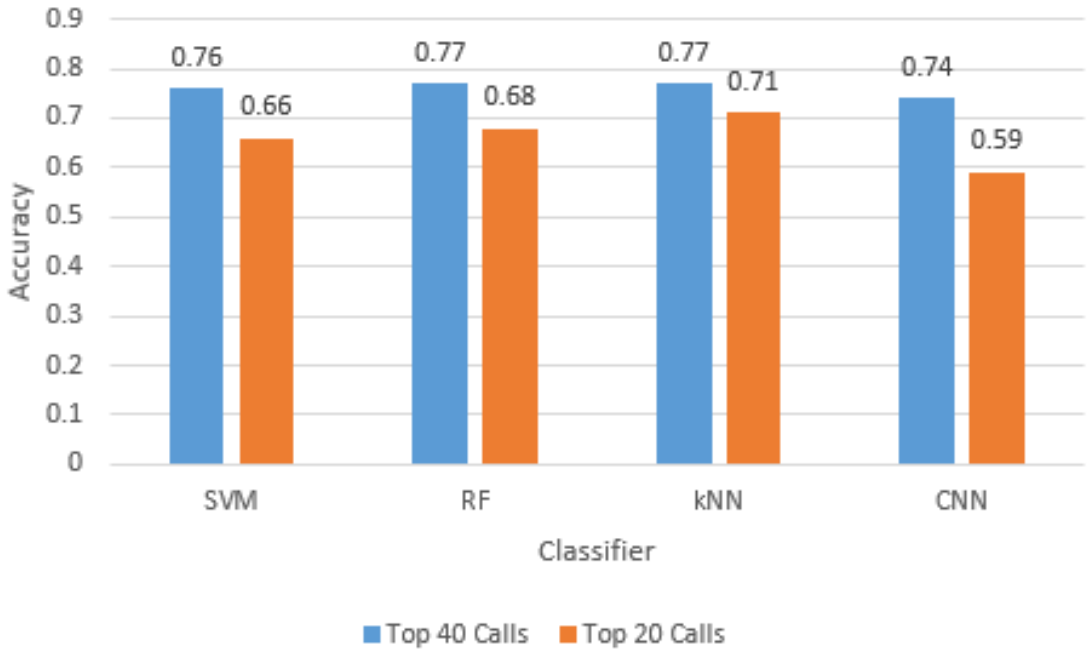


Figure 13: Top 20 calls vs Top 40 calls for Word2Vec Category Classification

4.3.1.3 ELMo

Within the ELMo experimetns, RF again performed the best and achieved an accuracy of 0.77. Surprisingly, kNN did not perform as well as it lags far behind with

0.71. Finally, similar to the experiments above SVM and CNN did not perform as well with only about 0.70 and 0.67 respectively. Fig A.40 to Fig A.42 show the confusion matrices for these experiments.

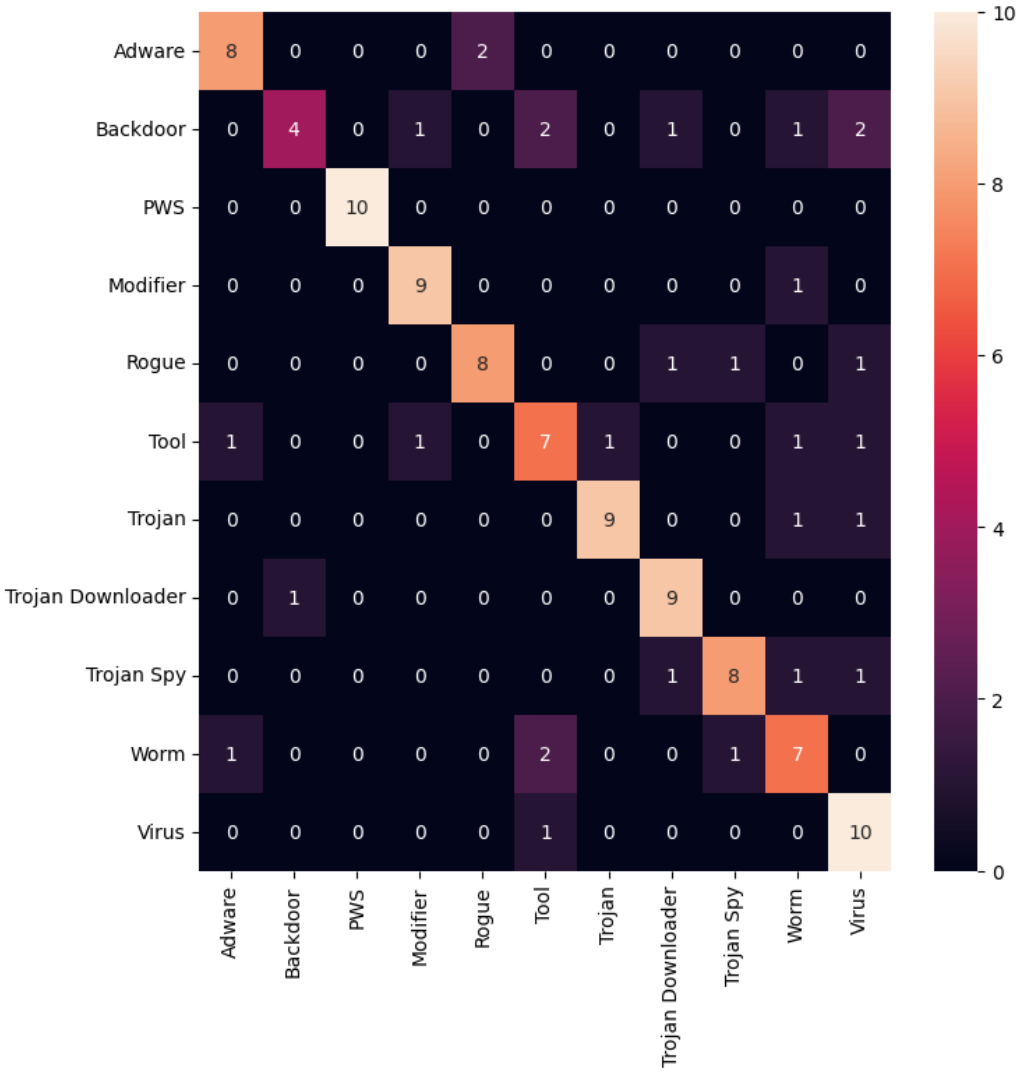


Figure 14: Confusion Matrix for ELMo-RF Category Classification

Similar to Word2Vec, ELMo performed significantly better with the Top 40 calls than it does with the Top 20 calls as seen in Fig 15.

Table 7: Classification Report for ELMo-RF Category Classification

Category	Precision	Recall	F1-Score
Adware	0.80	0.80	0.80
Backdoor	0.80	0.36	0.50
PWS	1.00	1.00	1.00
Modifier	0.82	0.90	0.86
Rogue	0.80	0.73	0.76
Tool	0.58	0.58	0.58
Trojan	0.75	0.82	0.78
Trojan Downloader	0.75	0.90	0.81
Trojan Spy	0.80	0.72	0.75
Worm	0.50	0.58	0.53
Virus	0.63	0.91	0.75

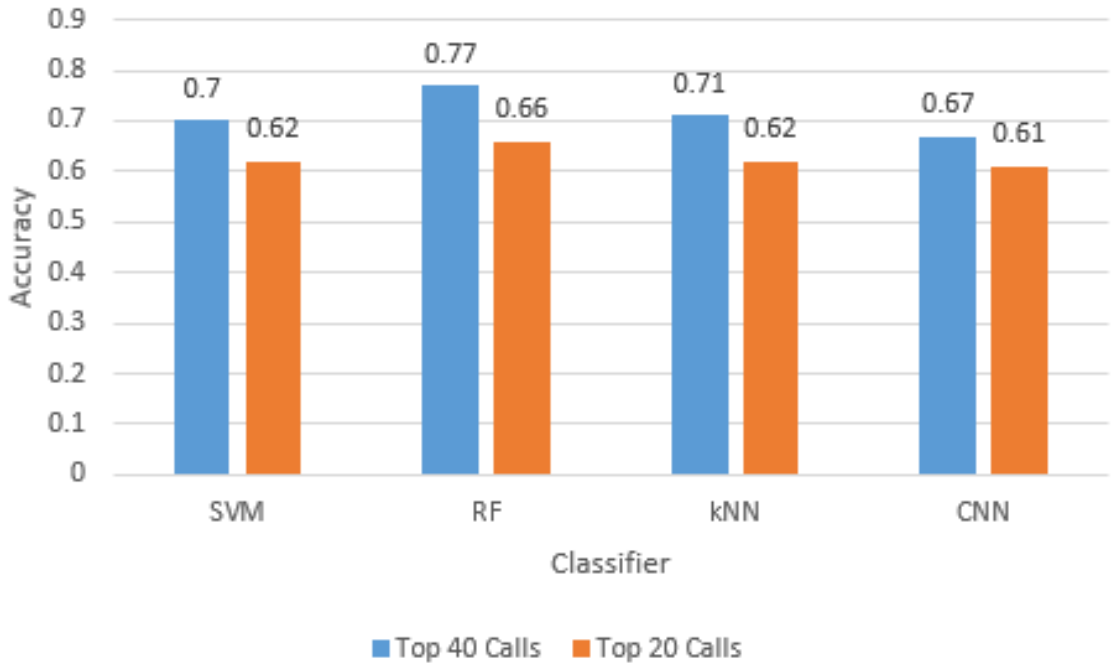


Figure 15: Top 20 calls vs Top 40 calls for ELMo Category Classification

4.3.1.4 BERT

The process for BERT is quite similar to ELMo and we expected close results. Analogous to previous results, kNN and RF got the highest accuracy both at 0.74.

While this is lower than ELMo, surprisingly SVM performed much better with BERT than the other techniques and was able to reach the maximum accuracy at 0.74. CNN was quite far behind all the other three techniques here at 0.67.

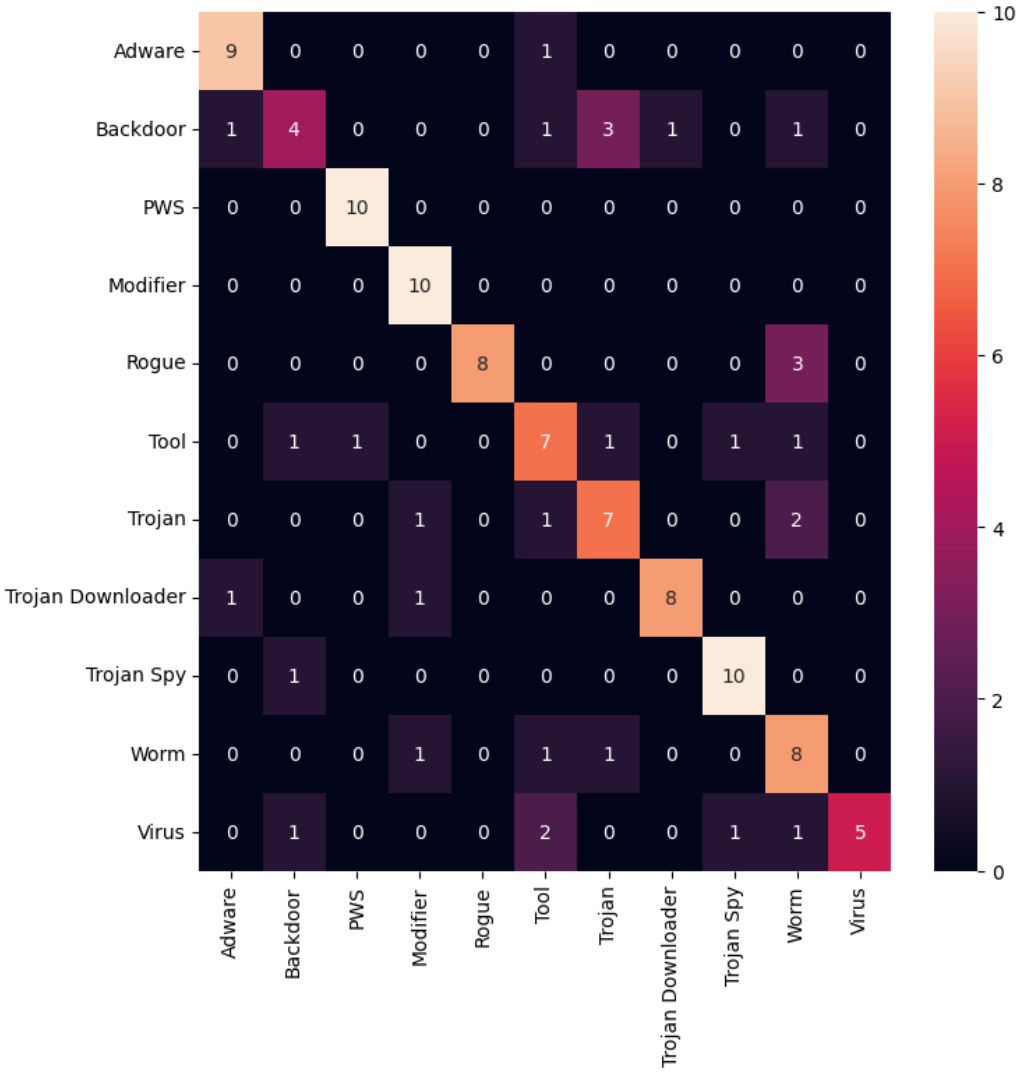


Figure 16: Confusion Matrix for BERT-RF Category Classification

Fig 17 shows the accuracy for the calls retained. While the Top 40 calls come out ahead again, the difference is not very much.

Table 8: Classification Report for BERT-RF Category Classification

Category	Precision	Recall	F1-Score
Adware	0.82	0.90	0.86
Backdoor	0.57	0.36	0.44
PWS	0.91	1.00	0.95
Modifier	0.77	1.00	0.87
Rogue	1.00	0.73	0.84
Tool	0.54	0.58	0.56
Trojan	0.58	0.64	0.61
Trojan Downloader	0.89	0.80	0.84
Trojan Spy	0.67	0.91	0.77
Worm	0.62	0.73	0.67
Virus	1.00	0.50	0.67

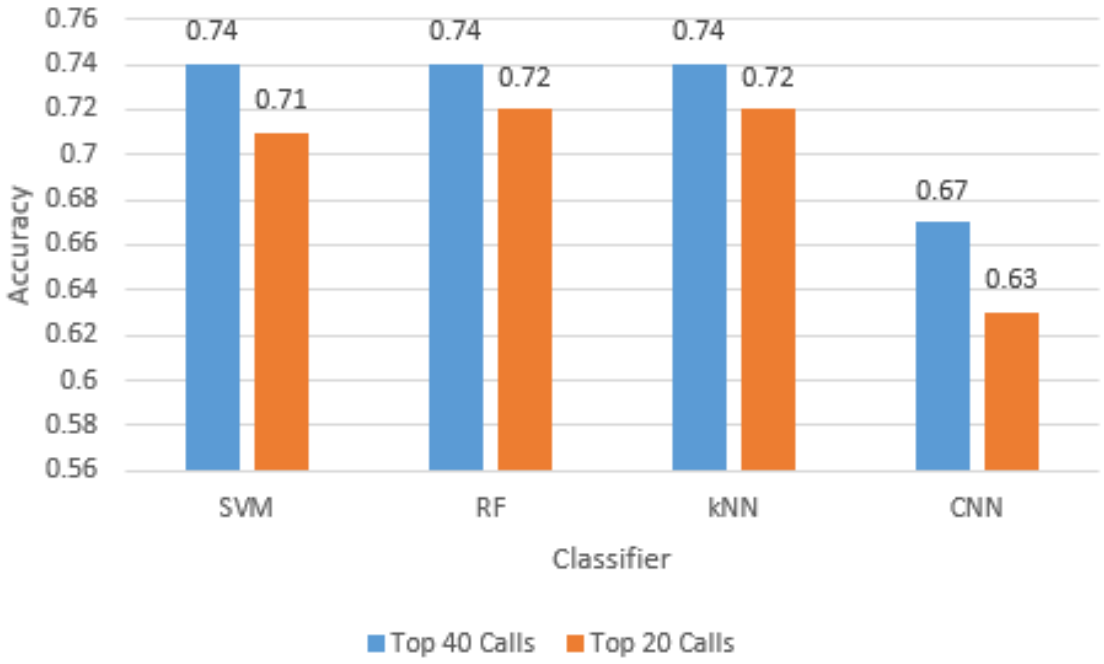


Figure 17: Top 20 calls vs Top 40 calls for BERT Category classification

4.3.1.5 Results on Malware Category Classification

Table 9 shows the optimal values we got for the hyperparameters of all classifiers.

Table 9: Hyperparameters Selected for Category Classification

Classifier	Hyperparameter	HMM2Vec	Word2Vec	ELMo	BERT
SVM	Kernel	rbf	rbf	rbf	rbf
	C	1000	1000	100	100
	gamma	0.1	0.1	0.0001	0.0001
RF	n_estimators	200	400	700	400
	max_depth	40	40	30	40
	min_samples_split	2	2	3	3
	min_samples_leaf	2	1	1	2
kNN	n_neighbors	3	3	3	3
	metric	manhattan	euclidean	euclidean	euclidean
	weights	distance	distance	distance	distance
CNN	epochs	200	200	1000	1000
	learning_rate	0.0001	0.0001	0.0001	0.0001
	activation	relu	relu	relu	relu
	optimizer	adam	adam	adam	adam

Overall, the classification of categories does not perform as well as compared families. This is expected as the malware category is much more difficult to predict since each malware in a category may belong to a different family and may function differently. In our dataset, among 11 categories we have about 90 distinct families. This high number of families may be a contributing factor to the low accuracy score we see here. Fig 18 shows the accuracies achieved for the different hybrid techniques. It was observed that a high number of misclassifications occur for the Worm and Backdoor categories. The maximum accuracy we achieved came with Word2Vec-RF at about 77%. ELMo performed similarly well with Random Forest the as classifier. The best results we achieved using HMM2Vec for category classification were 69%. Among the embedding techniques, Word2Vec and ELMo perform the best and produce similar results. In general, we see that there is agreement among the embedding techniques for the best classifier which is RF. In addition, we see that in most cases RF and kNN perform similarly well whereas SVM and CNN are somewhat lagging.

However, interestingly SVM performed on par with RF and kNN in the case of BERT. This may be because of SVM's ability to handle high-dimensional data.

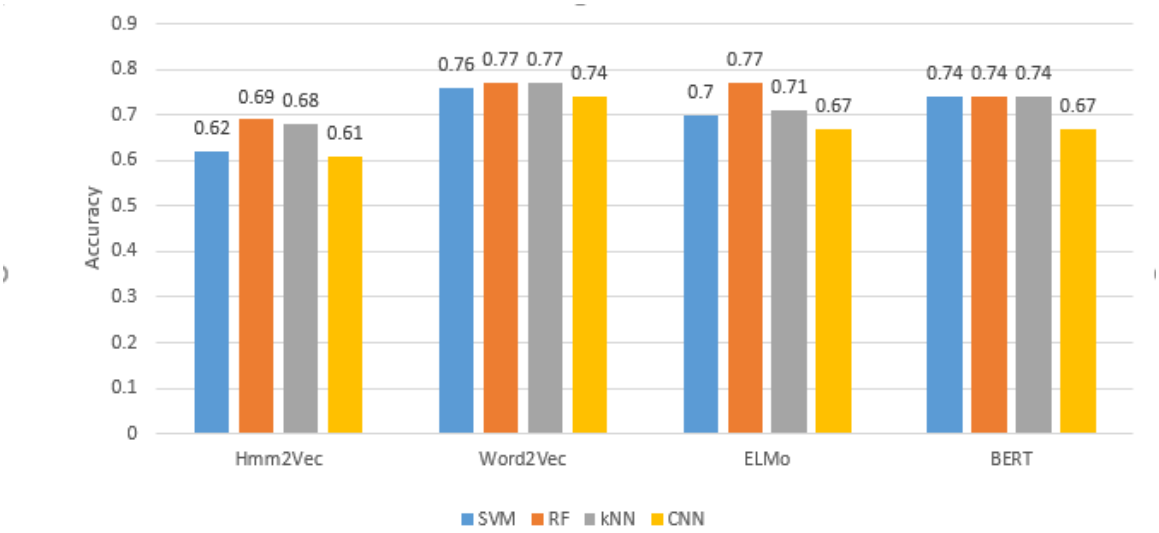


Figure 18: Comparison of hybrid machine learning approaches for categories

4.3.2 Malware Category Classification with Different Set of Calls

While selecting calls by frequency has given some decent results, there is a possibility that these calls are dominated by a few categories and we are missing out on some essential calls that may be representative of the other categories. To this extent, we perform an experiment on the Category classification of samples using all common calls across the categories. In this, rather than frequency we would use a counter to maintain how many categories a particular call appears in and in the end we retain the ones that appear in all. The total number of these calls was 31 and the experiment was only performed with the embeddings generated using HMM2Vec and Word2Vec. The best results achieved with this experiment with each classifier can be seen below.

4.3.2.1 HMM2Vec

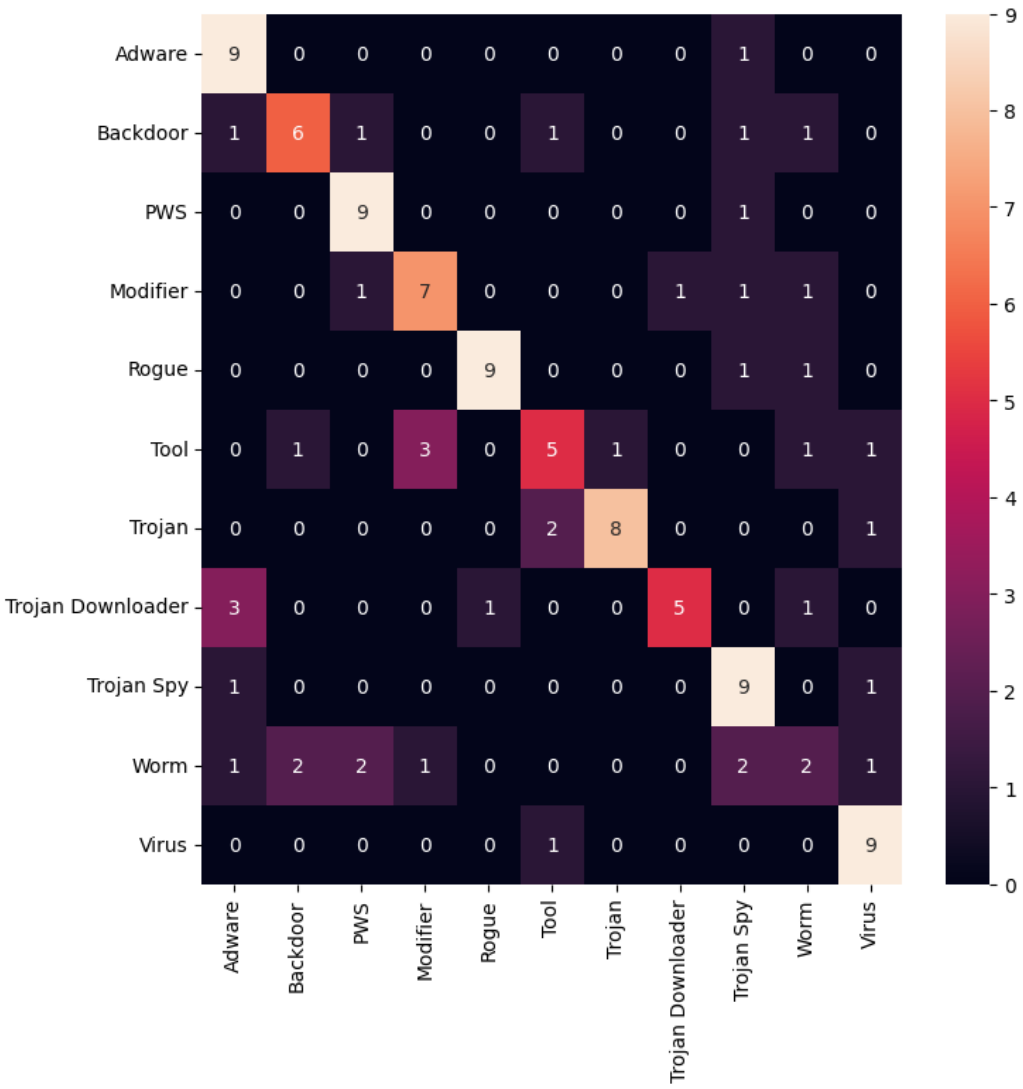


Figure 19: Confusion Matrix for HMM2Vec-RF Category Classification with calls in all categories

Table 10: Classification Report for HMM2Vec-RF Category Classification with calls in all categories

Category	Precision	Recall	F1-Score
Adware	0.60	0.90	0.72
Backdoor	0.67	0.55	0.60
PWS	0.69	0.90	0.78
Modifier	0.64	0.70	0.67
Rogue	0.90	0.82	0.86
Tool	0.56	0.42	0.48
Trojan	0.89	0.73	0.80
Trojan Downloader	0.83	0.50	0.62
Trojan Spy	0.56	0.82	0.67
Worm	0.33	0.18	0.24
Virus	0.69	0.90	0.78

4.3.2.2 Word2Vec

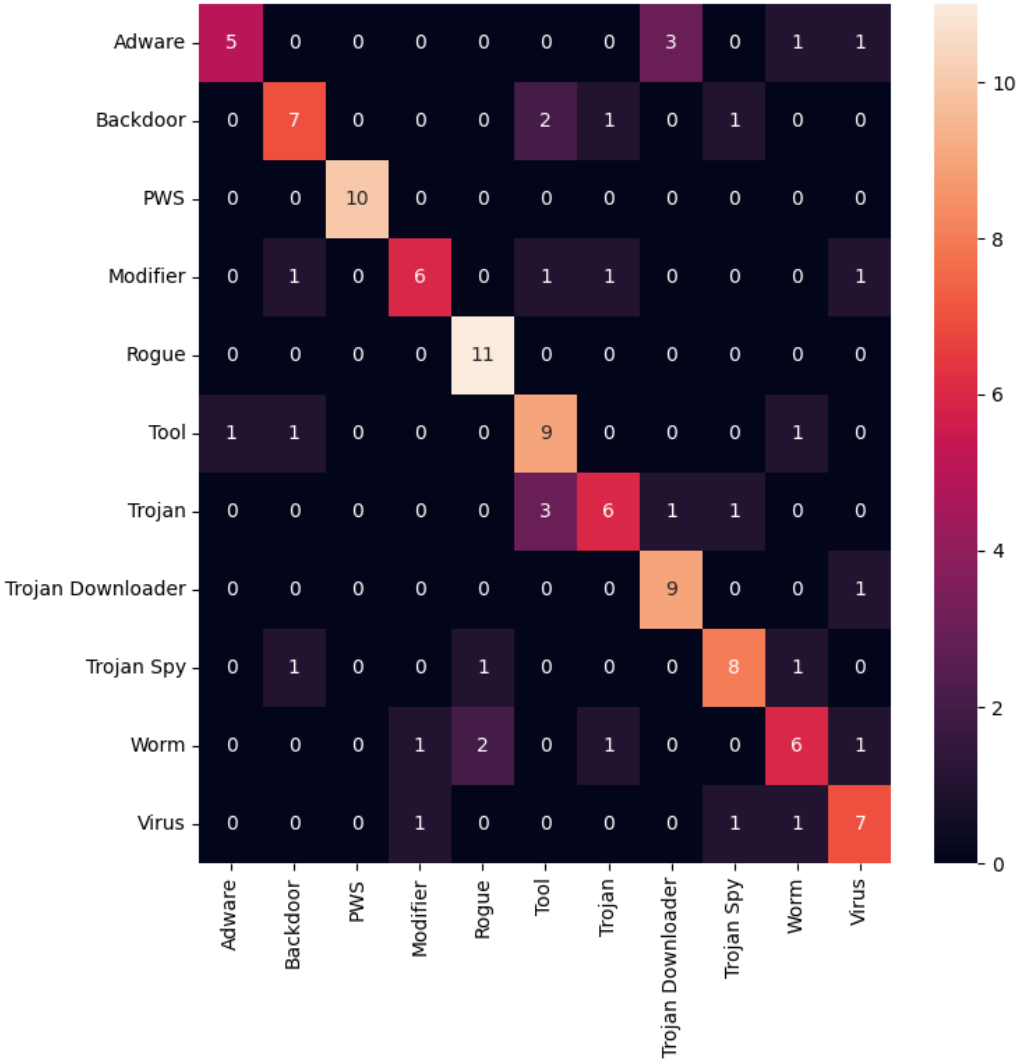


Figure 20: Confusion Matrix for Word2Vec-KNN Category Classification with calls in all categories

The best results achieved with Hmm2Vec and Word2Vec experiments came with Random Forest and KNN at 0.67 and 0.72 respectively. The figures below compare the scores achieved here to that from the experiments performed using Top 40 Calls.

Table 11: Classification Report for Word2Vec-KNN Category Classification with calls in all categories

Category	Precision	Recall	F1-Score
Adware	0.83	0.50	0.62
Backdoor	0.70	0.64	0.67
PWS	1.00	1.00	1.00
Modifier	0.75	0.60	0.67
Rogue	0.79	1.00	0.88
Tool	0.60	0.75	0.67
Trojan	0.67	0.55	0.60
Trojan Downloader	0.69	0.90	0.78
Trojan Spy	0.73	0.73	0.73
Worm	0.60	0.55	0.57
Virus	0.64	0.70	0.67

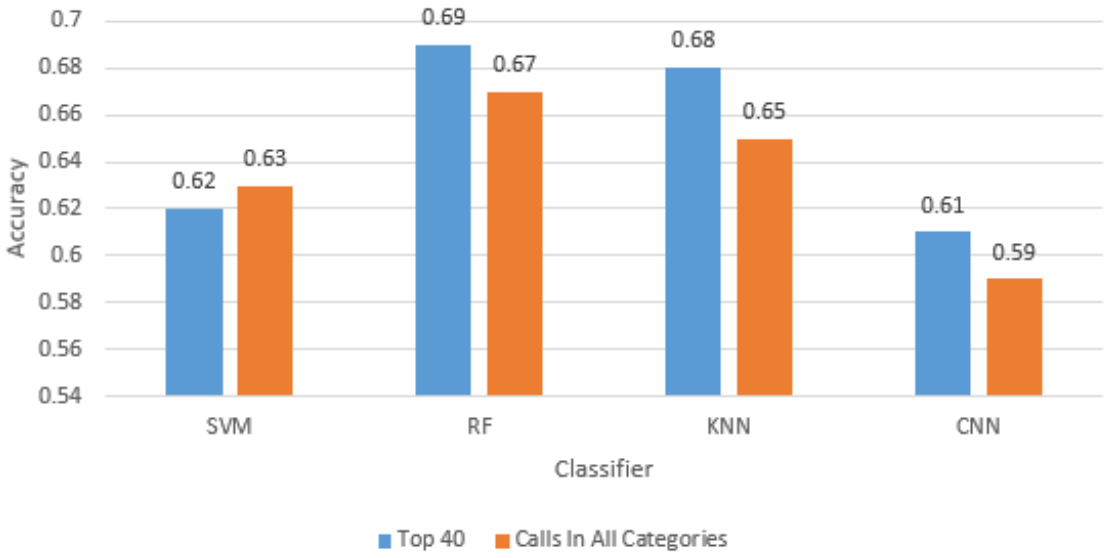


Figure 21: Comparison of HMM2Vec scores for categories with Top 40 calls and calls in all categories (31 calls)

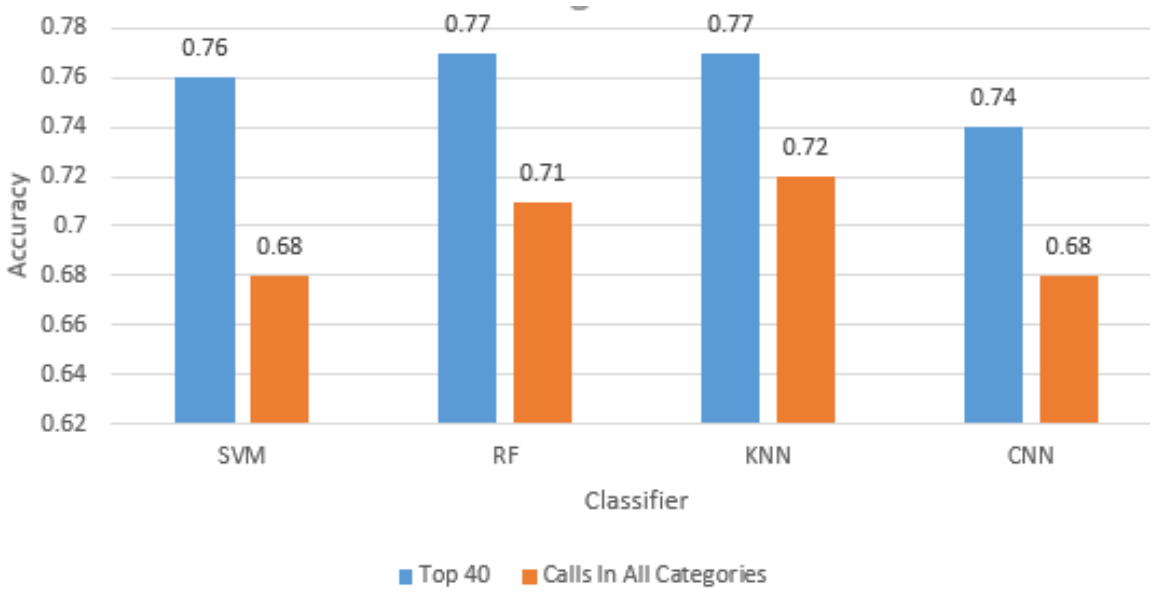


Figure 22: Comparison of Word2Vec scores for categories with Top 40 calls and calls in all categories (31 calls)

These results indicate that the Top 40 Calls give us a better representation of the malware sample than the calls that are present across all categories. This may be because the number of these calls is 31 and is mostly a subset of the Top 40 Calls which would mean that we are actually not gaining any insight. We also tried to do some experiments with a set of calls that were common across all samples, however since the number of calls was too low at 5 we concluded that there will be too much information lost. However, selecting a different set using other methods such as specifically tagging suspicious or rarer calls may still produce better results as seen here [27].

4.3.3 Malware Family Classification

The experiments here consider seven malware families that were introduced in 4.1. Since the Top 40 calls performed consistently better than the Top 20 calls for all our hybrid techniques, the features with Top 20 Calls are not considered for this part of the research.

4.3.3.1 HMM2Vec

For HMM2Vec experiments, the best accuracy achieved came with Random Forest at 0.85. The other classifiers performed very close achieving 0.84, 0.84 and 0.82 for SVM, kNN and CNN respectively. Fig 23 shows the confusion matrix for these set of experiments.

Table 12: Classification Report for best HMM2Vec Family Classification

Category	Precision	Recall	F1-Score
Adload	0.93	0.93	0.93
Bancos	0.69	0.73	0.71
Onlinegames	0.81	0.93	0.87
Vbinject	0.92	0.79	0.85
Vundo	0.75	0.86	0.80
Winwebsec	1.00	0.79	0.88
Zwangi	0.93	0.93	0.93

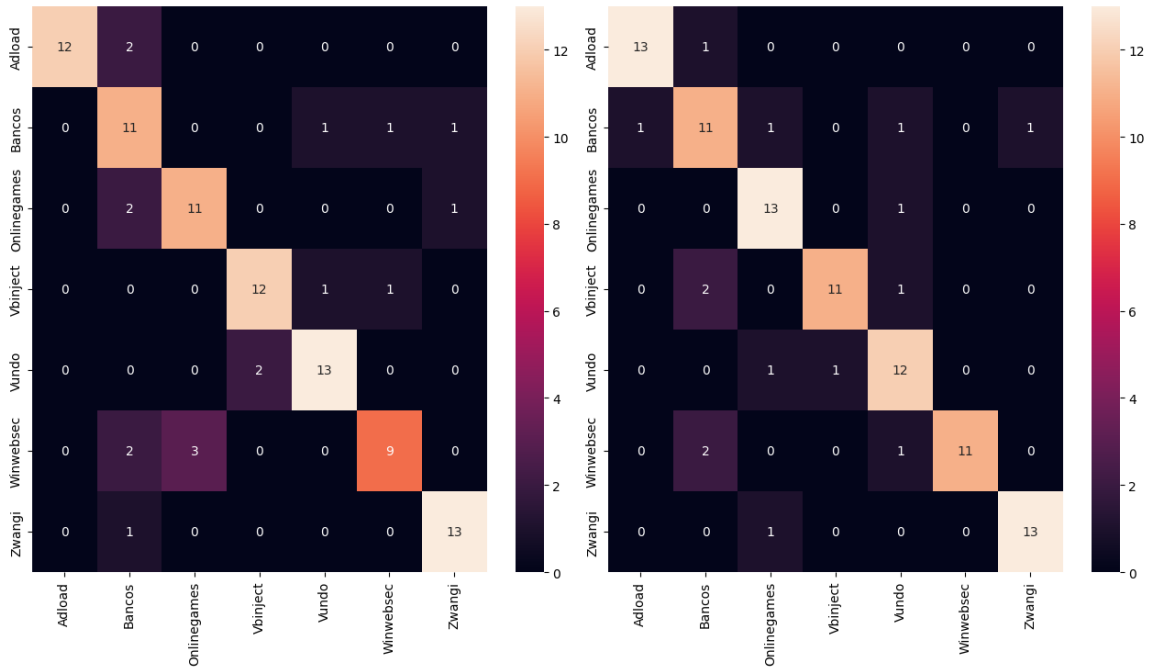
Note that a lot of the samples from distinct malware families are being misclassified into the Bancos and Onlinegames families. This situation is apparent in other experiments as well and may indicate a need for a more robust feature selection technique.

4.3.3.2 Word2Vec

Word2Vec-RF received the best results overall for family classification, even among all the word embedding techniques, at 0.93. This is a particularly good result as this approach is able to predict a high number of samples. Other classifiers also performed quite well with SVM, kNN and CNN achieving 0.9, 0.92 and 0.89 respectively.

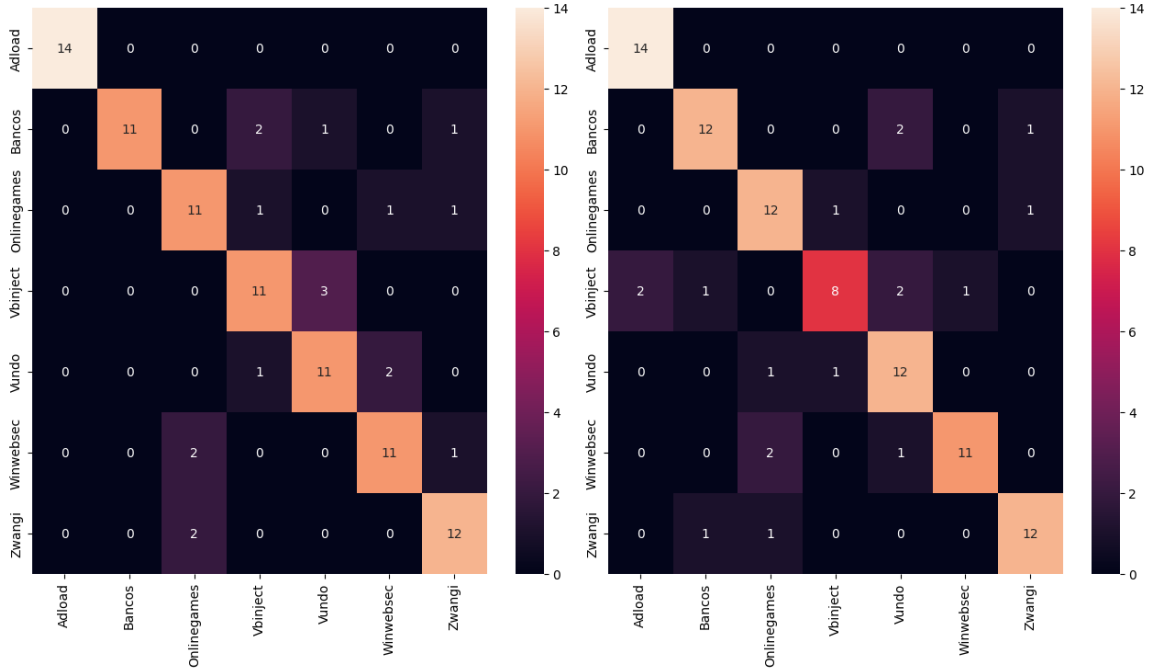
4.3.3.3 ELMo

Fig 27 shows the confusion matrix for our hybrid ELMo experiments.



(a) HMM2Vec-SVM

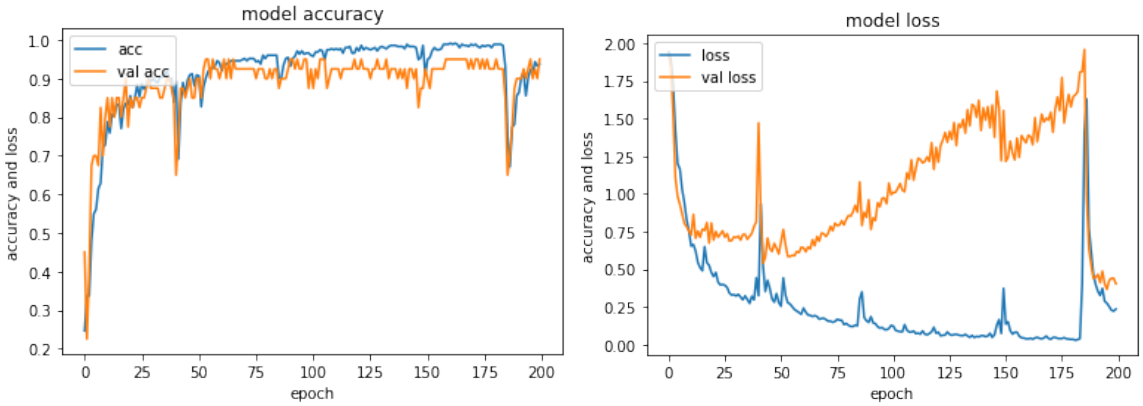
(b) HMM2Vec-RF



(c) HMM2Vec-kNN

(d) HMM2Vec-CNN

Figure 23: Confusion Matrix for HMM2Vec Family Experiments



(a) Accuracy (b) Loss

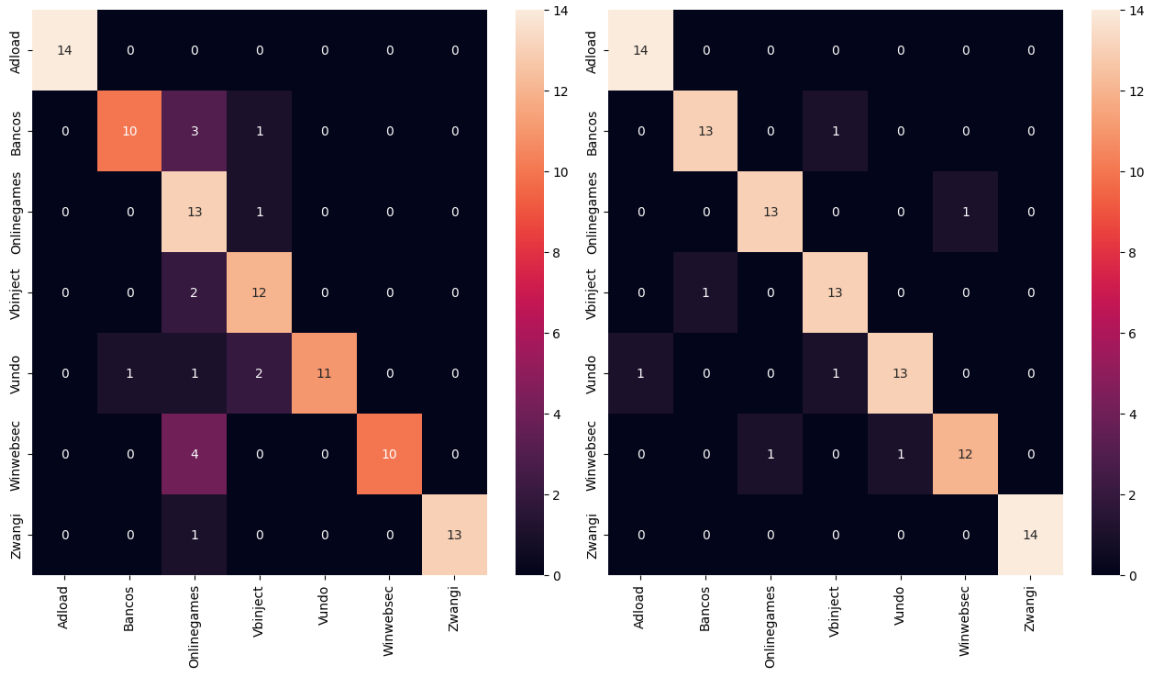
Figure 24: Training Model Accuracy vs Loss For HMM2Vec-CNN

Table 13: Classification Report for best Word2Vec Family Classification

Category	Precision	Recall	F1-Score
Adload	0.93	1.00	0.97
Bancos	0.93	0.93	0.93
Onlinegames	0.93	0.93	0.93
Vbinject	0.87	0.93	0.90
Vundo	0.93	0.87	0.90
Winwebsec	0.92	0.86	0.89
Zwangi	1.00	0.93	0.97

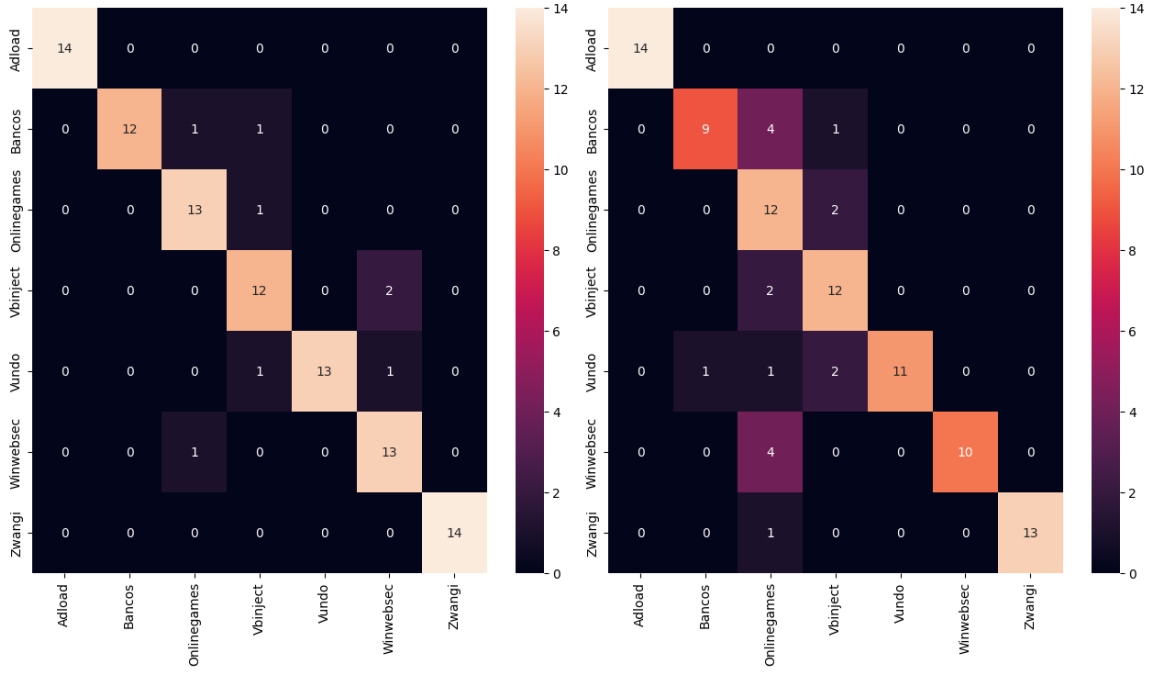
Table 14: Classification Report for best ELMo Family Classification

Category	Precision	Recall	F1-Score
Adload	1.00	1.00	1.00
Bancos	0.82	0.93	0.87
Onlinegames	0.83	0.71	0.77
Vbinject	0.87	0.93	0.90
Vundo	1.00	1.00	1.00
Winwebsec	0.97	0.93	0.90
Zwangi	1.00	0.86	0.92



(a) Word2Vec-SVM

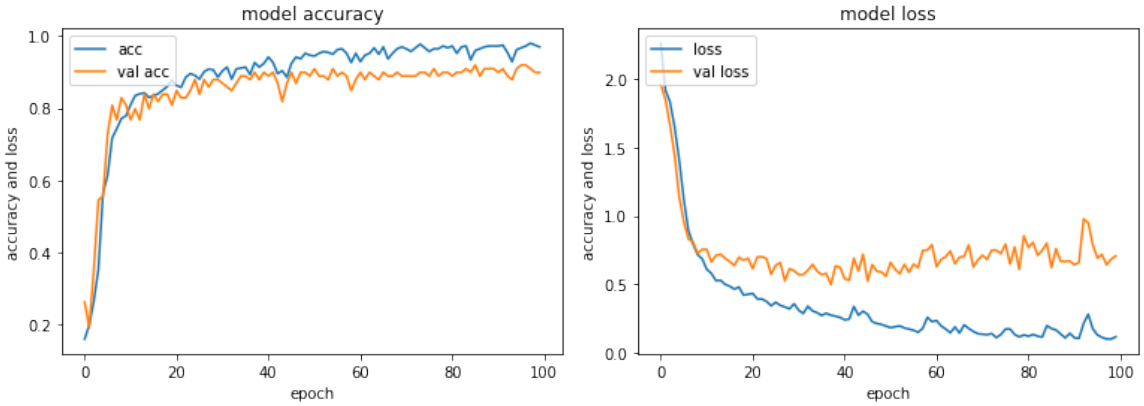
(b) Word2Vec-RF



(c) Word2Vec-kNN

(d) Word2Vec-CNN

Figure 25: Confusion Matrix for Word2Vec Family Experiments



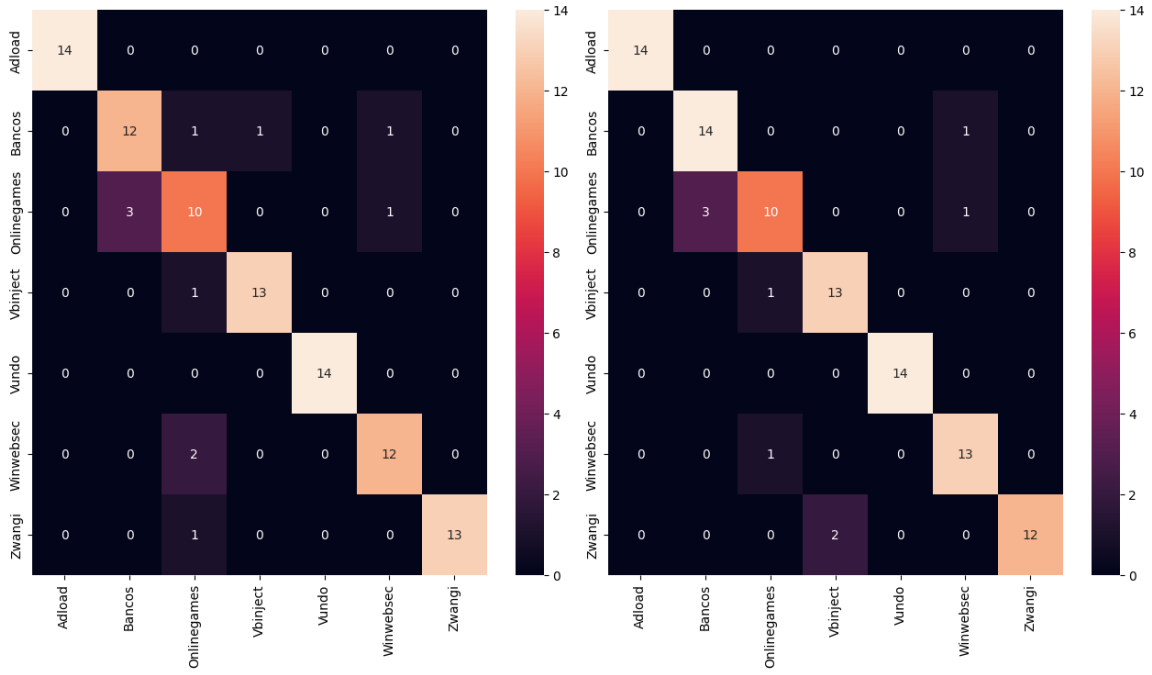
(a) Accuracy (b) Loss

Figure 26: Training Model Accuracy vs Loss For Word2Vec-CNN

The accuracies, in increasing order, were 0.86 for CNN, 0.9 for SVM and kNN and finally Random forest with 0.91.

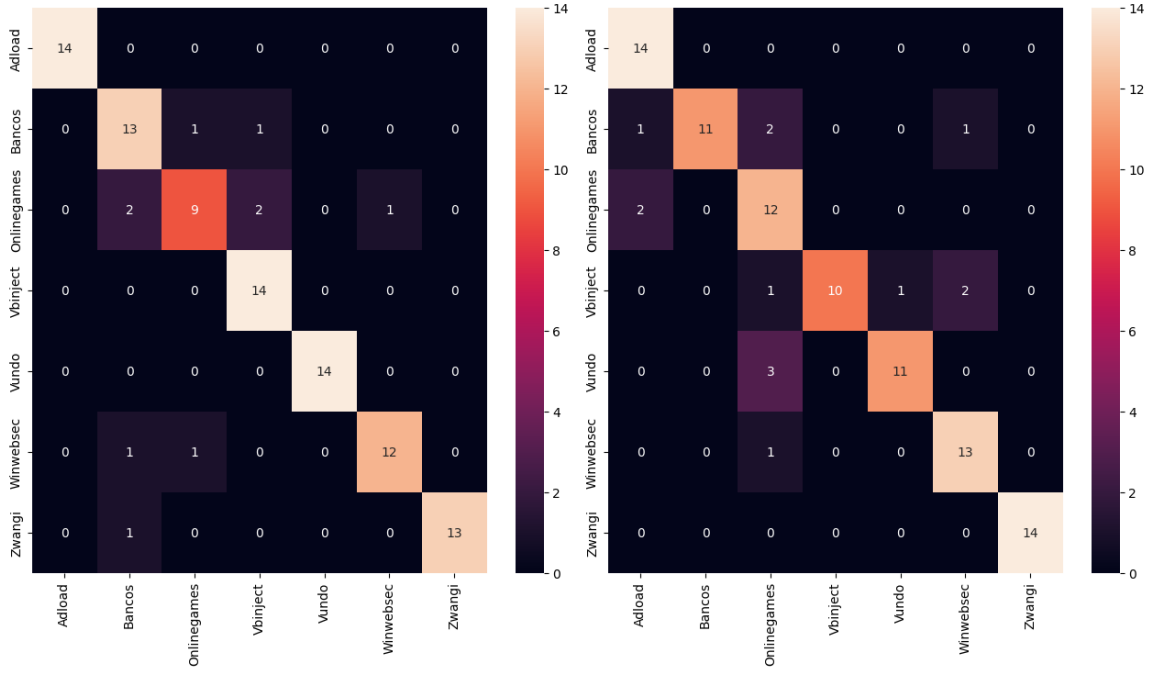
4.3.3.4 BERT

The highest accuracy achieved for BERT was 0.92 with Random forest. Similar to the previous category experiments, we saw that SVM performed better than kNN getting 0.9 compared to the latter's 0.88. Finally, CNN came last with about 0.88.



(a) ELMo-SVM

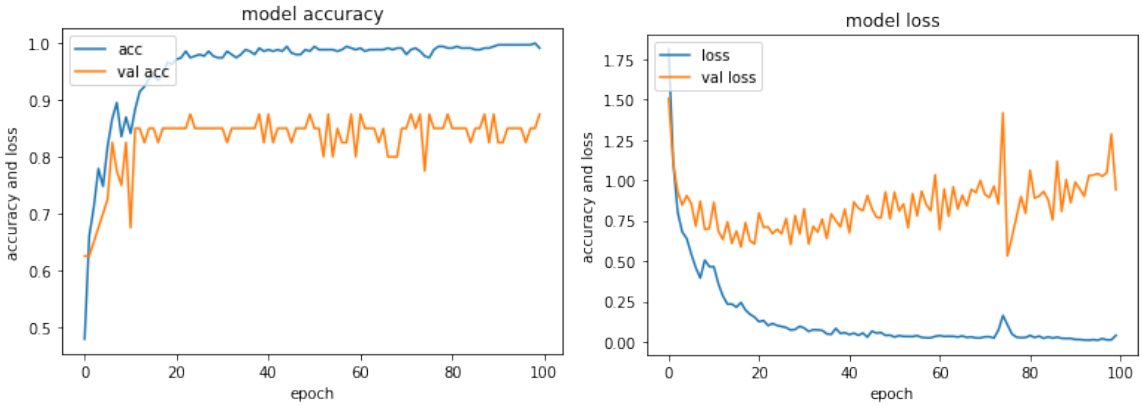
(b) ELMo-RF



(c) ELMo-kNN

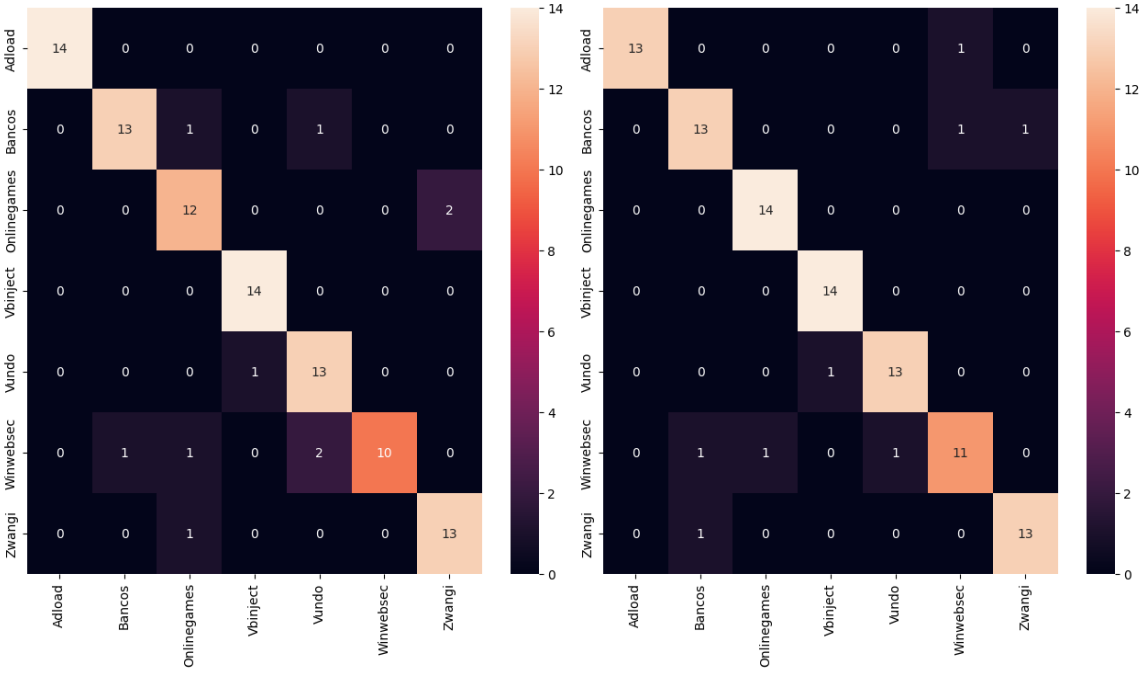
(d) ELMo-CNN

Figure 27: Confusion Matrix for ELMo Family Experiments

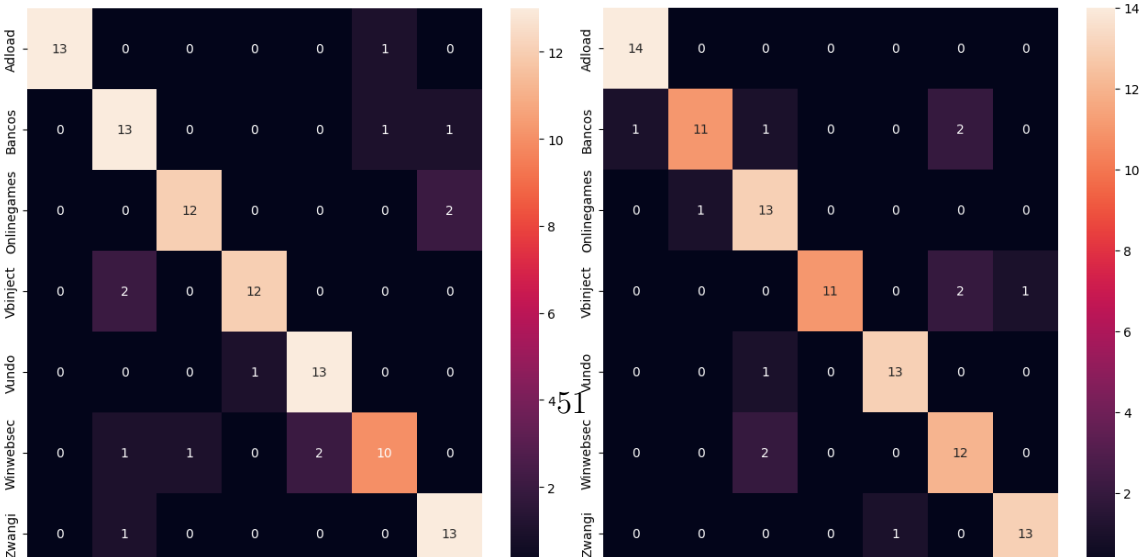


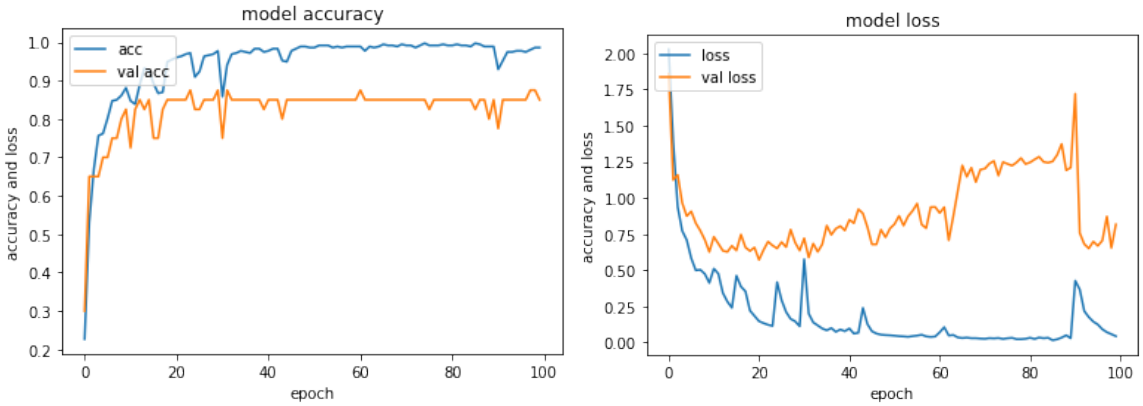
(a) Accuracy (b) Loss

Figure 28: Training Model Accuracy vs Loss For ELMo-CNN



(a) BERT-SVM (b) BERT-RF





(a) Accuracy (b) Loss

Figure 30: Training Model Accuracy vs Loss For BERT-CNN

Table 15: Classification Report for best BERT Family Classification

Category	Precision	Recall	F1-Score
Adload	1.00	0.93	0.97
Bancos	0.87	0.87	0.87
Onlinegames	0.93	1.00	0.97
Vbinject	0.93	1.00	0.97
Vundo	0.93	0.93	0.93
Winwebsec	0.85	0.79	0.82
Zwangi	0.93	0.93	0.93

4.3.3.5 Results on Malware Family Classification

Table 16 shows the optimal values we got for the hyperparameters of all classifiers.

Table 16: Hyperparameters Selected for Family Classification

Classifier	Hyperparameter	HMM2Vec	Word2Vec	ELMo	BERT
SVM	Kernel	linear	linear	rbf	rbf
	C	1000	1000	100	100
	gamma	-	-	0.0001	0.0001
RF	n_estimators	200	200	400	400
	max_depth	40	40	10	40
	min_samples_split	2	2	2	3
	min_samples_leaf	2	1	1	2
kNN	n_neighbors	3	3	3	3
	metric	manhattan	euclidean	euclidean	euclidean
	weights	distance	distance	distance	distance
CNN	epochs	200	200	200	200
	learning_rate	0.0001	0.0001	0.0001	0.0001
	activation	relu	relu	relu	relu
	optimizer	adam	adam	adam	adam

Using the hybrid techniques, we were able to achieve high accuracy for classification among the seven families peaking at 93% with Word2Vec-RF. Fig 31 shows the accuracies achieved for the different techniques. The maximum accuracy with HMM2Vec came at about 85 percent with Random forest, which is the least among all our word embedding techniques. We observed that in samples where the number of distinct API calls was not very high, the feature vector obtained had much less information compared to the other techniques. This could attribute to the reason why we see the low scores. It was observed that most of the misclassifications came from Winwebsec being predicted as Bancos or Onlinegames.

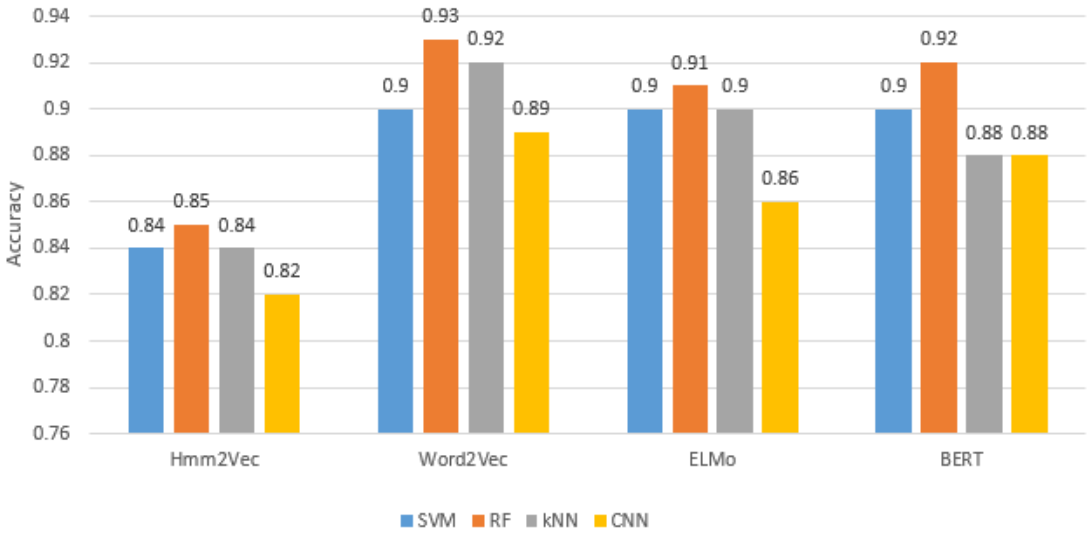


Figure 31: Comparison of hybrid machine learning approaches for families

CHAPTER 5

Conclusion and Future Work

In this work, we conducted several experiments to understand the effectiveness of API call information as a feature for malware category and family classification. We tested hybrid machine learning techniques wherein we used different word embedding techniques for engineering features. We tested the performance of four embedding techniques, namely Hmm2Vec, Word2Vec, ELMo, and BERT, in conjunction with four different classifiers SVM, RF, kNN, and CNN. From our results, it is clear that Word2Vec outperforms all other embedding techniques with the highest accuracy in the case of families reaching 93% and 77% in the case of categories. This is good as Word2Vec training also takes the least amount of time among all our techniques however it was expected for BERT to perform better. One of the reasons for this may be the size of the dataset used. BERT requires a large amount of training data to learn meaningful relationships and it is possible that with an increase in the number of samples, we see an increase in performance. Another possible reason may be the limitation on the length of the input. For BERT, this length is set at 512 and we make use of the first 512 tokens in our sequence. However, since our sequences are quite long in some cases, it is possible that we are losing essential information and a different method to yield the tokens such as the last 512 or a custom selection may produce better results. In terms of classifiers, we observe that Random Forest consistently outperforms all other classifiers. The results documented show that API calls can be very useful for classifying malware.

In the future, the experiments performed here can be extended to more malware categories and families. Moreover, it would be interesting to see similar experiments performed with other features such as byte n-grams or some combinations of features. Another possibly promising approach would be to try more enhanced techniques for

feature selection. In this research, we select features based on their frequency but other techniques such as TF-IDF and Fisher score have proved effective for this task [8] [28]. These techniques could boost the scores specifically in the case of HMM2Vec as they would remove some irrelevant features.

We have used many word embedding techniques in this project but there exist many more that could provide fruitful results. Firstly, experiments done here can be extended to include more parameters such as the skip-gram algorithm for Word2Vec. Since Word2Vec with CBOW gave us the best results, the performance could potentially improve with skip-gram. In addition, we could also try to use other embedding techniques. Recently, GPT-based models have gained a lot of popularity and it would be interesting to see how they perform compared to the techniques we tried here. BERT performed quite well in most of the experiments and as mentioned previously, with more exploration the results could potentially improve. This warrants a further look into other models related to it as well such as distilBERT or Roberta. Another possible approach would be to use more complex architectures for CNN such as ResNet rather than building sequential layers.

LIST OF REFERENCES

[1] M. Stamp, ‘‘A revealing introduction to hidden markov models,’’ *Science*, pp. 1--20, 01 2004.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, ‘‘Efficient estimation of word representations in vector space,’’ 2013.

[3] D. A. Nguyen, ‘‘Create a strong text classification with the help from elmo,’’ 2019. [Online]. Available: https://andy-nguyen.medium.com/create-a-strong-text-classification-with-the-help-from-elmo-e90809ba29da

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ‘‘Bert: Pre-training of deep bidirectional transformers for language understanding,’’ 2019.

[5] ‘‘Sonicwall cyber threat report,’’ p. 21, 2023. [Online]. Available: https://www.sonicwall.com/medialibrary/en/white-paper/2023-cyber-threat-report.pdf

[6] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, ‘‘A survey on automated dynamic malware-analysis techniques and tools,’’ *ACM Comput. Surv.*, vol. 44, no. 2, mar 2008. [Online]. Available: https://doi.org/10.1145/2089125.2089126

[7] D. Uppal, R. Sinha, V. Mehra, and V. Jain, ‘‘Exploring behavioral aspects of api calls for malware identification and categorization,’’ in *2014 International Conference on Computational Intelligence and Communication Networks*, 2014, pp. 824--828.

[8] Namita, Prachi, and P. Sharma, ‘‘Windows malware detection using machine learning and tf-idf enriched api calls information,’’ in *2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA)*, 2022, pp. 1--6.

[9] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal, ‘‘Malware classification with deep convolutional neural networks,’’ in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2018, pp. 1--5.

[10] M. Siddiqui, M. C. Wang, and J. Lee, ‘‘A survey of data mining techniques for malware detection using file features,’’ in *Proceedings of the 46th Annual Southeast Regional Conference on XX*, ser. ACM-SE 46. New York, NY, USA: Association for Computing Machinery, 2008, p. 509–510. [Online]. Available: https://doi.org/10.1145/1593105.1593239

[11] T. K. Tran and H. Sato, ‘‘Nlp-based approaches for malware classification from api sequences,’’ in *2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES)*, 2017, pp. 101--105.

[12] I. Popov, ‘‘Malware detection using machine learning based on word2vec embeddings of machine code instructions,’’ in *2017 Siberian Symposium on Data Science and Engineering (SSDSE)*, 2017, pp. 1--4.

[13] S. Alqurashi and O. Batarfi, ‘‘A comparison between api call sequences and opcode sequences as reflectors of malware behavior,’’ in *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2017, pp. 105--110.

[14] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, ‘‘A comparison of static, dynamic, and hybrid analysis for malware detection,’’ *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 1--12, dec 2015. [Online]. Available: https://doi.org/10.1007%2Fs11416-015-0261-z

[15] A. Chandak, W. Lee, and M. Stamp, ‘‘A comparison of word2vec, hmm2vec, and pca2vec for malware classification,’’ 2021.

[16] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, ‘‘Deep contextualized word representations,’’ 2018.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, ‘‘Attention is all you need,’’ 2017.

[18] A. Kale, V. Pandya, F. Di Troia, and M. Stamp, ‘‘Malware classification with word2vec, hmm2vec, bert, and elmo,’’ *Journal of Computer Virology and Hacking Techniques*, vol. 19, pp. 1--16, 04 2022.

[19] ‘‘Buster sanbox analyzer.’’ [Online]. Available: https://bsa.isoftware.nl/

[20] ‘‘Sandboxie.’’ [Online]. Available: https://sandboxie-plus.com/sandboxie/

[21] M. Stamp, *A Reassuring Introduction to Support Vector Machines*, 09 2017, pp. 95--132.

[22] L. Breiman, ‘‘Random forests,’’ *Machine Learning*, vol. 45, pp. 5--32, 10 2001.

[23] A. Liaw and M. Wiener, ‘‘Classification and regression by randomforest,’’ *Forest*, vol. 23, 11 2001.

[24] K. O’Shea and R. Nash, ‘‘An introduction to convolutional neural networks,’’ 2015.

[25] M. Stamp, ‘‘Reference implementation of hmm in c.’’ [Online]. Available: https://www.cs.sjsu.edu/~stamp/RUA/HMM_ref.zip

[26] ‘‘Gensim.’’ [Online]. Available: https://radimrehurek.com/gensim/

[27] S. Gupta, H. Sharma, and S. Kaur, ‘‘Malware characterization using windows api call sequences,’’ in *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6*. Springer, 2016, pp. 271--280.

[28] S. Sharma, C. R. Krishna, and S. K. Sahay, ‘‘Detection of advanced malware by machine learning techniques,’’ 2019.

APPENDIX

Appendix

A.1 Additional Results

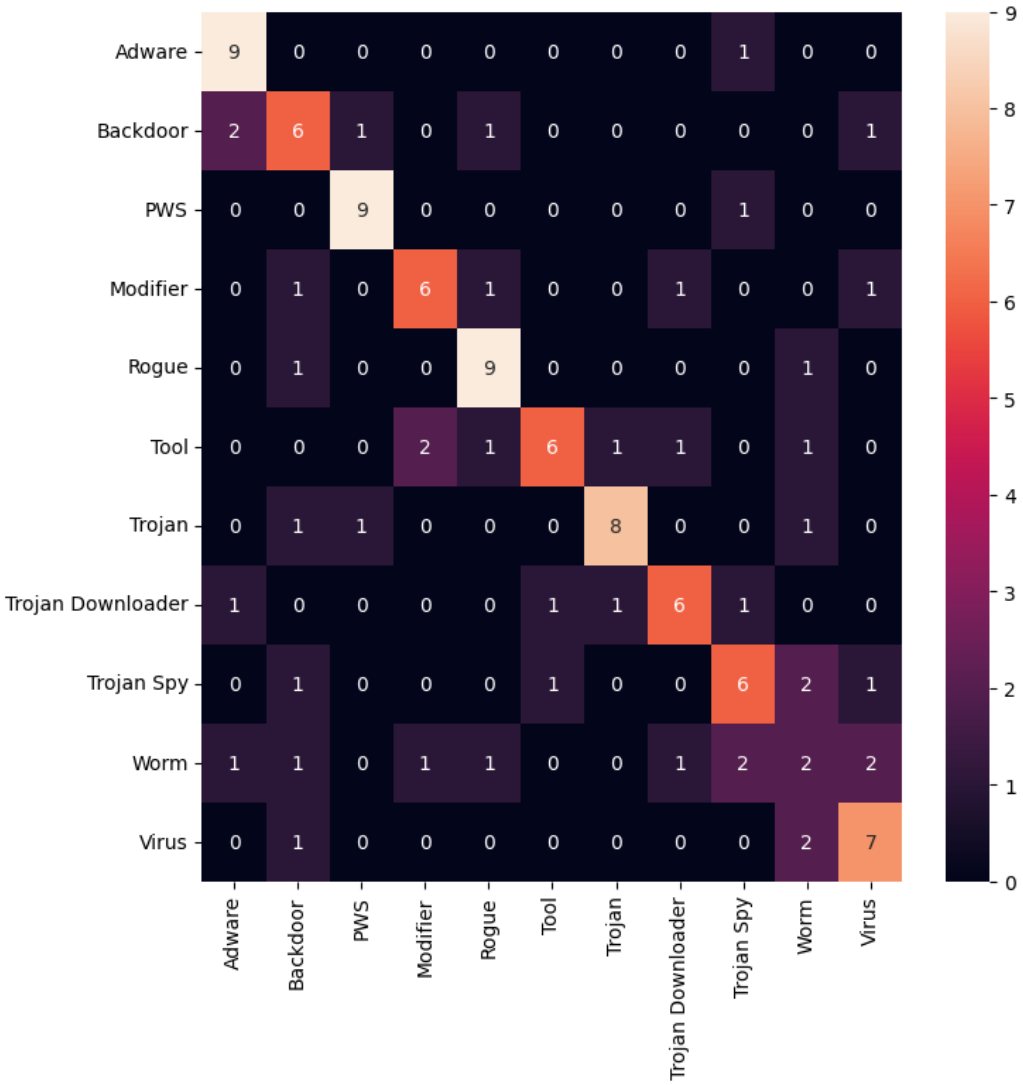


Figure A.32: Confusion Matrix for HMM2Vec-SVM Category Classification

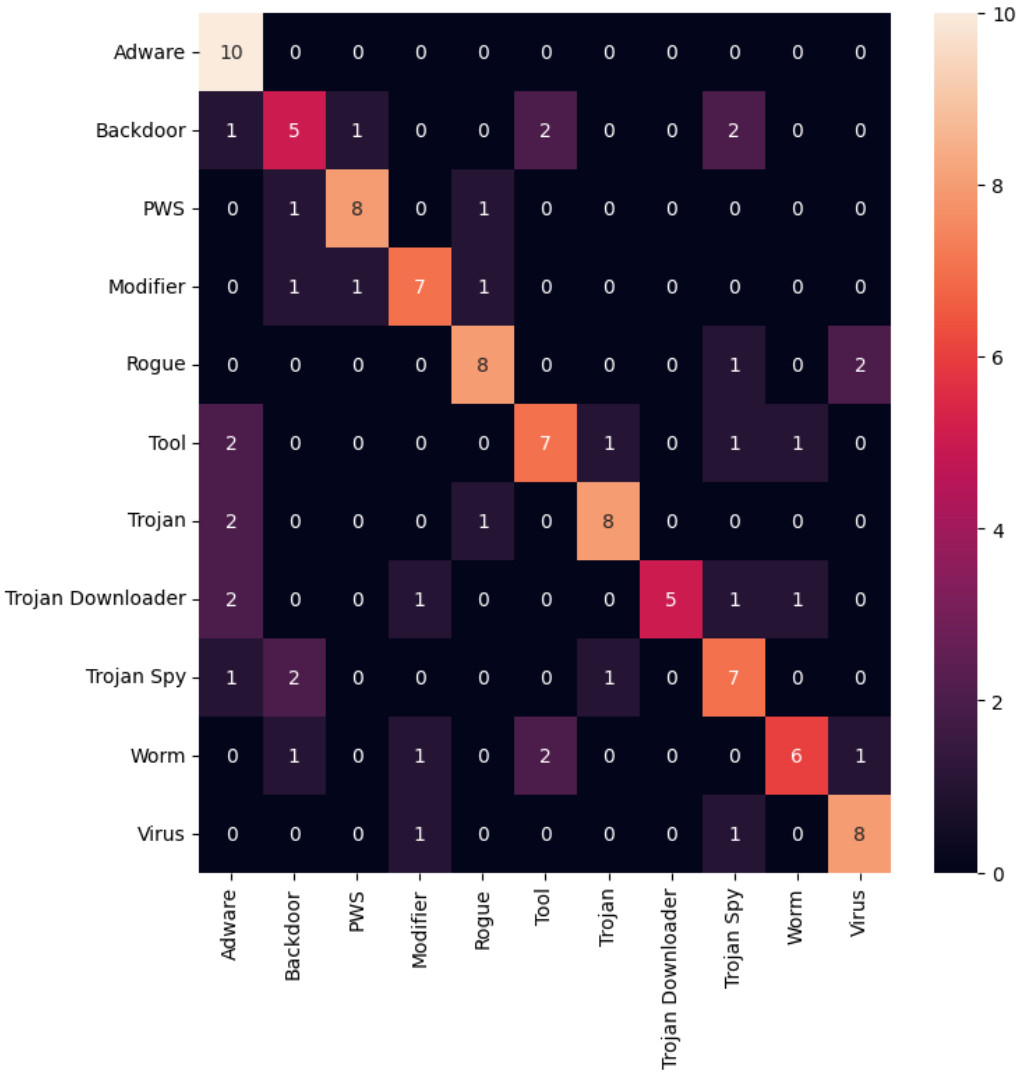


Figure A.33: Confusion Matrix for HMM2Vec-KNN Category Classification

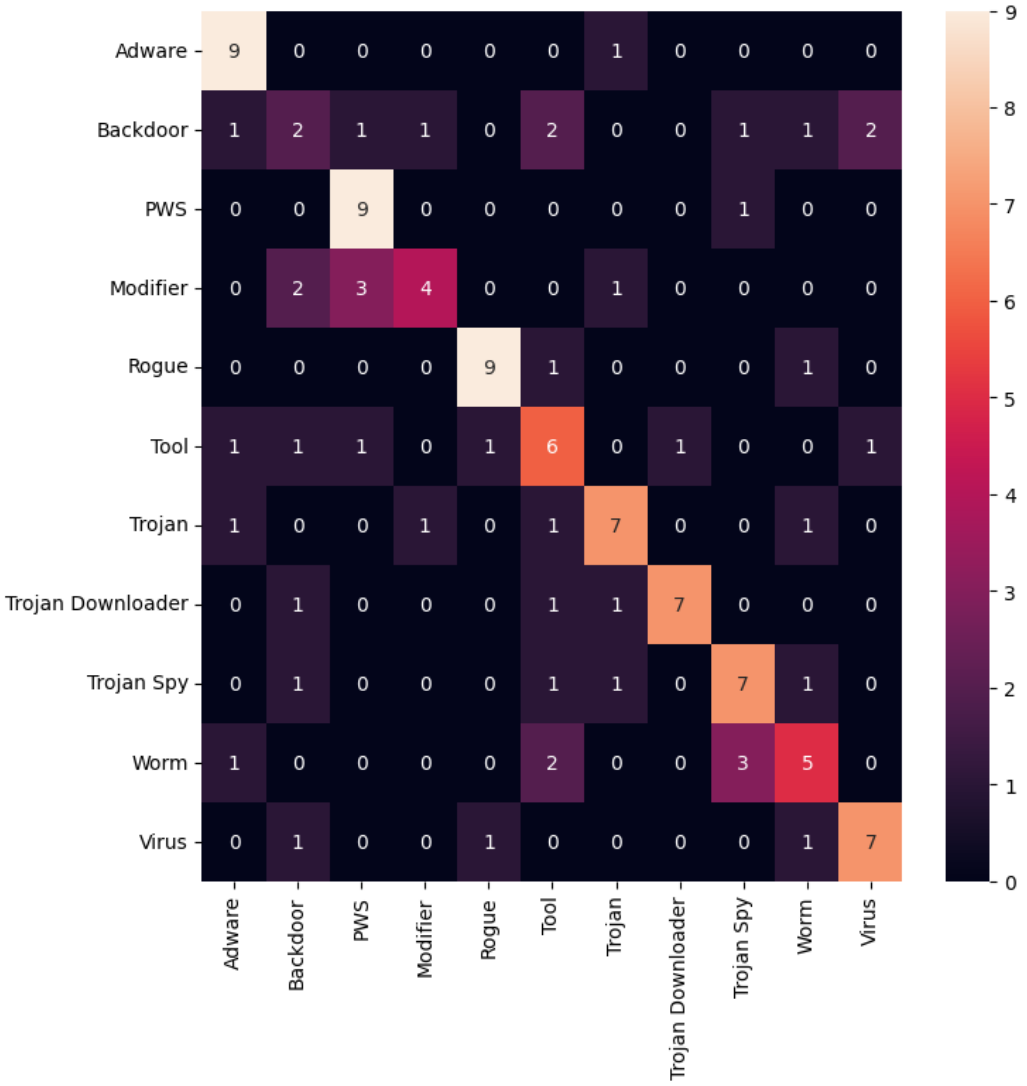
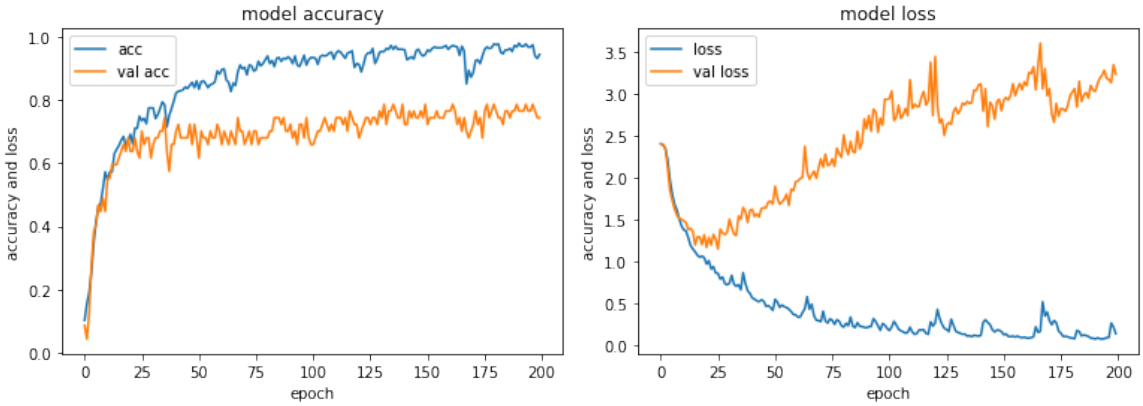


Figure A.34: Confusion Matrix for HMM2Vec-CNN Category Classification



(a) Accuracy (b) Loss

Figure A.35: Training Model Accuracy vs Loss For Hmm2Vec-CNN For Category

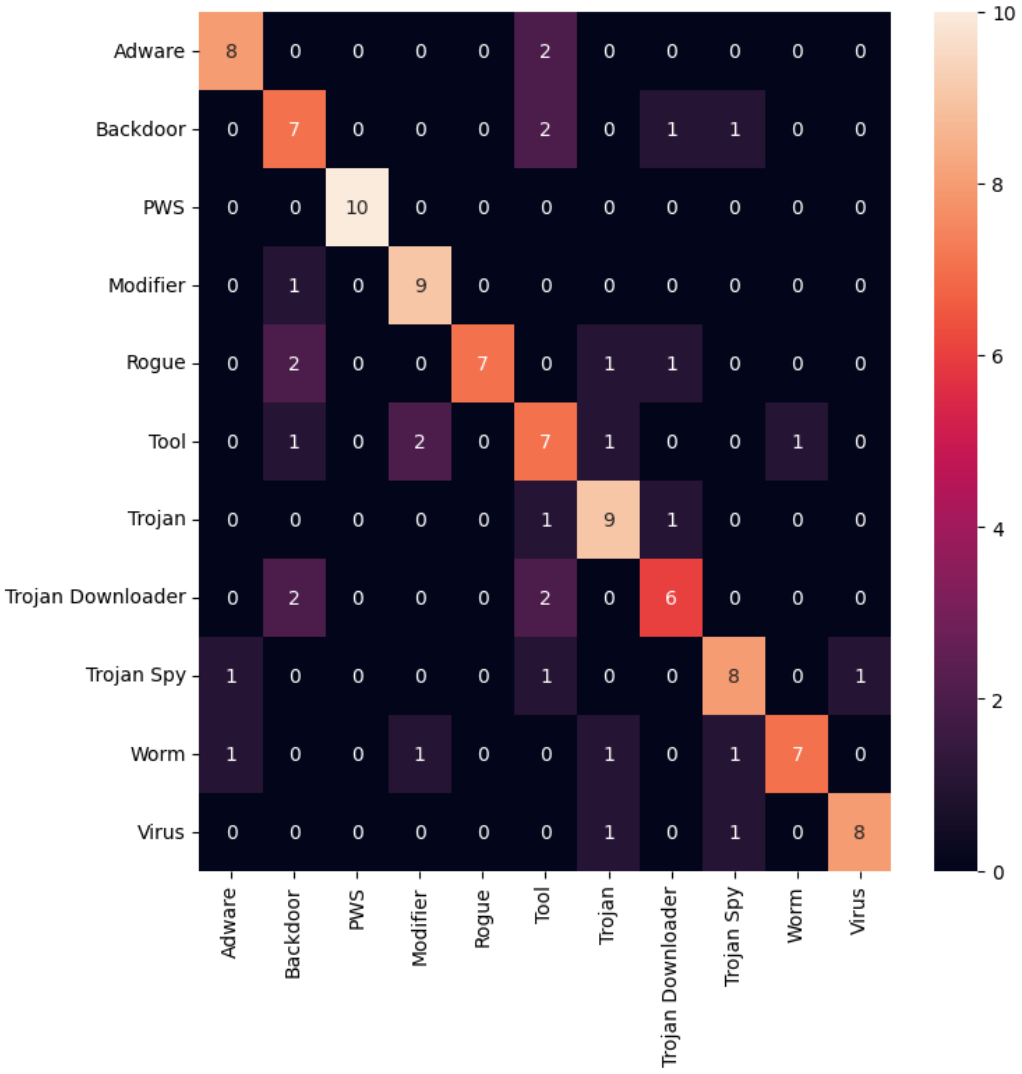


Figure A.36: Confusion Matrix for Word2Vec-SVM Category Classification

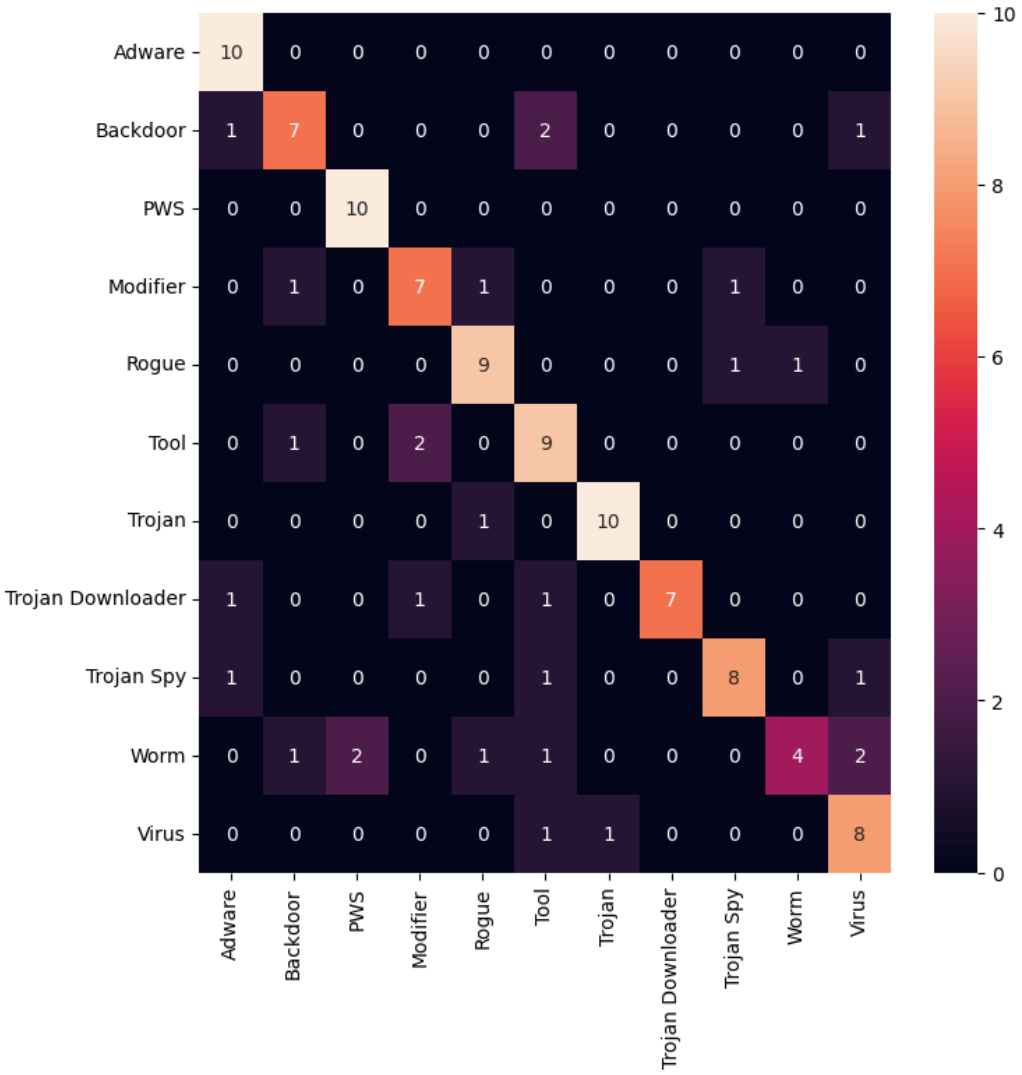


Figure A.37: Confusion Matrix for Word2Vec-KNN Category Classification

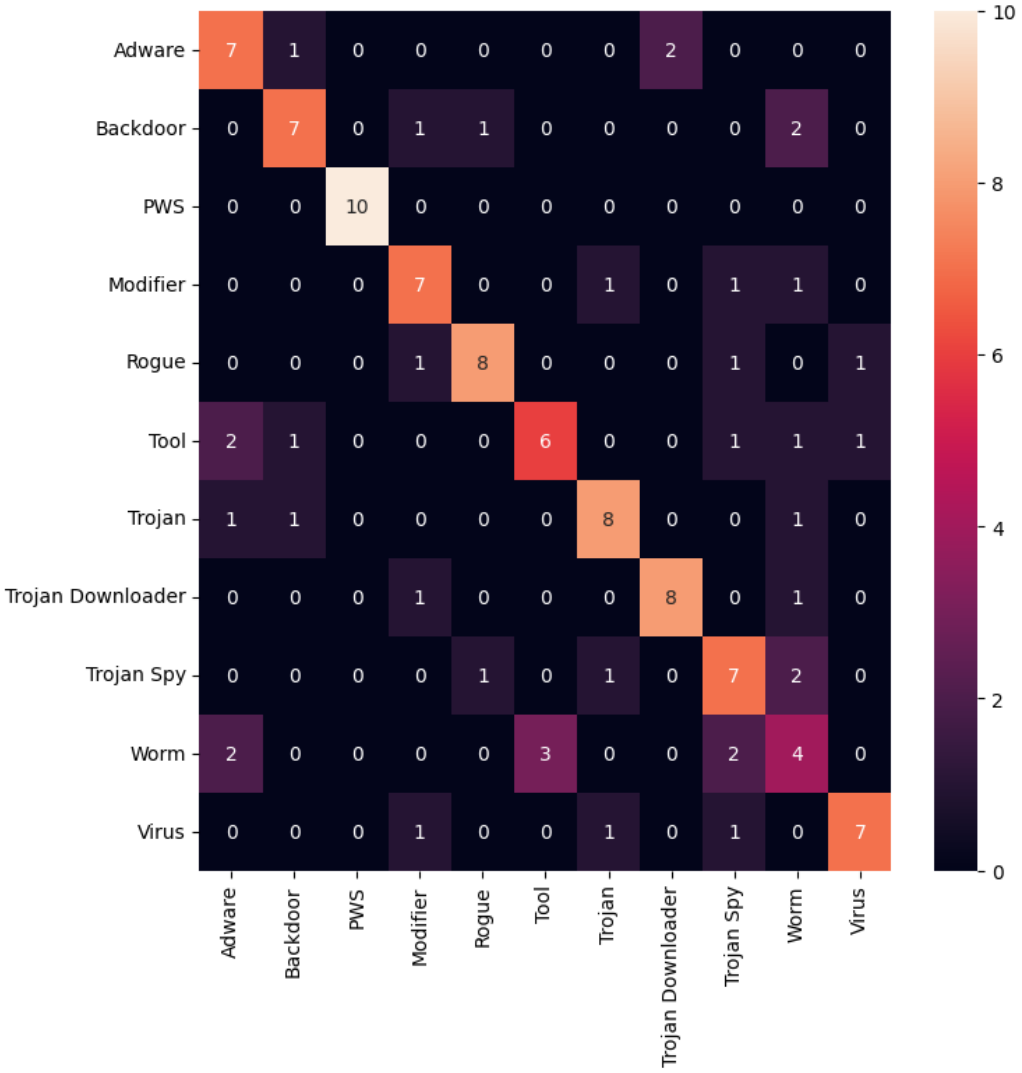
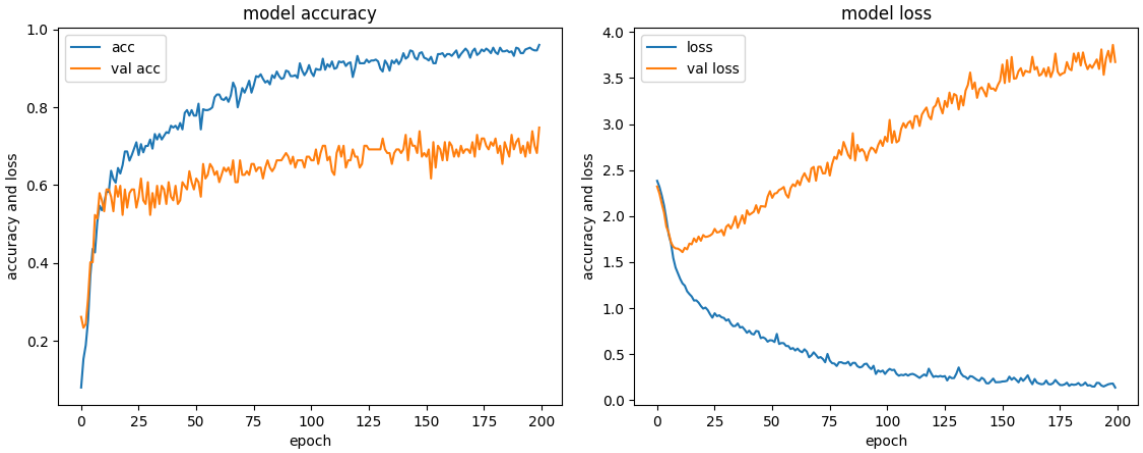


Figure A.38: Confusion Matrix for Word2Vec-CNN Category Classification



(a) Accuracy (b) Loss

Figure A.39: Training Model Accuracy vs Loss For Word2Vec-CNN For Category

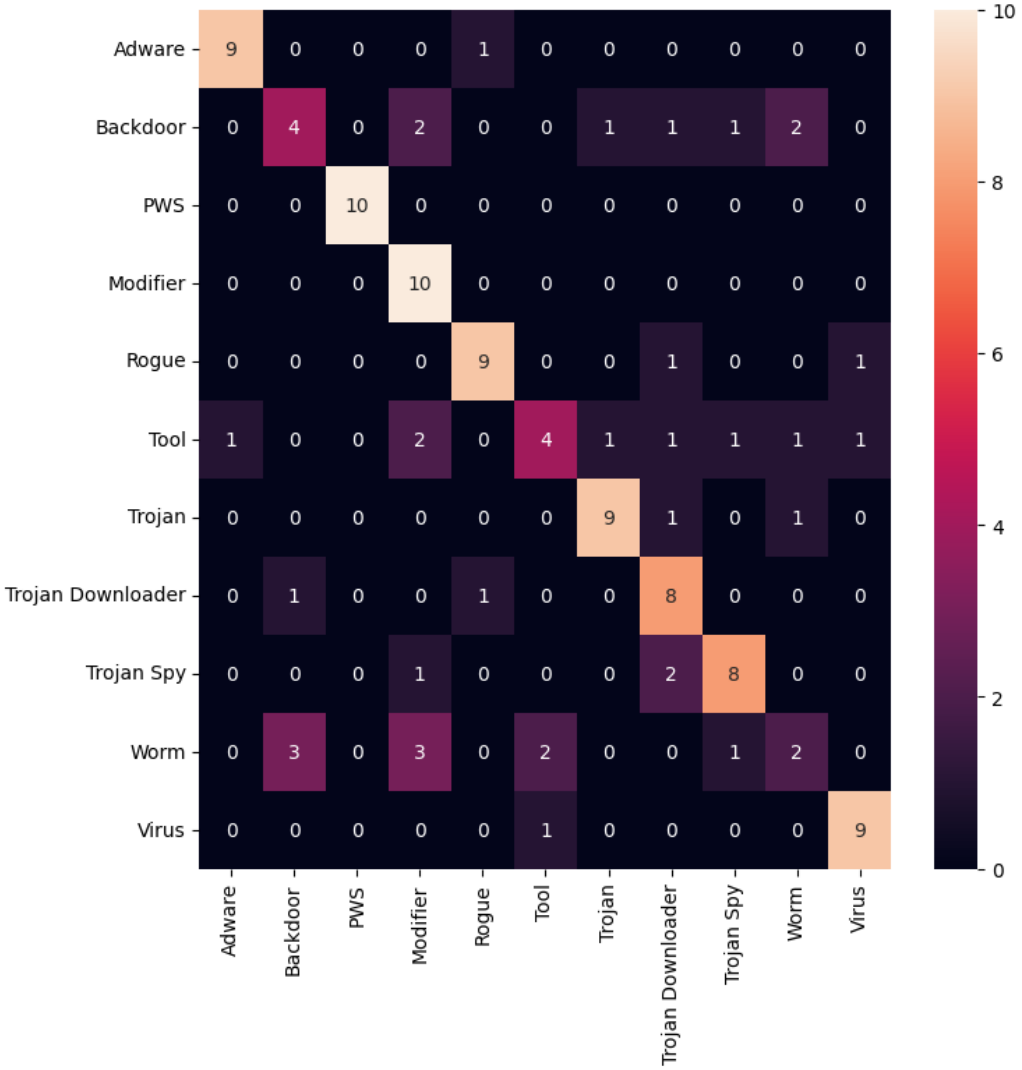


Figure A.40: Confusion Matrix for ELMo-SVM Category Classification

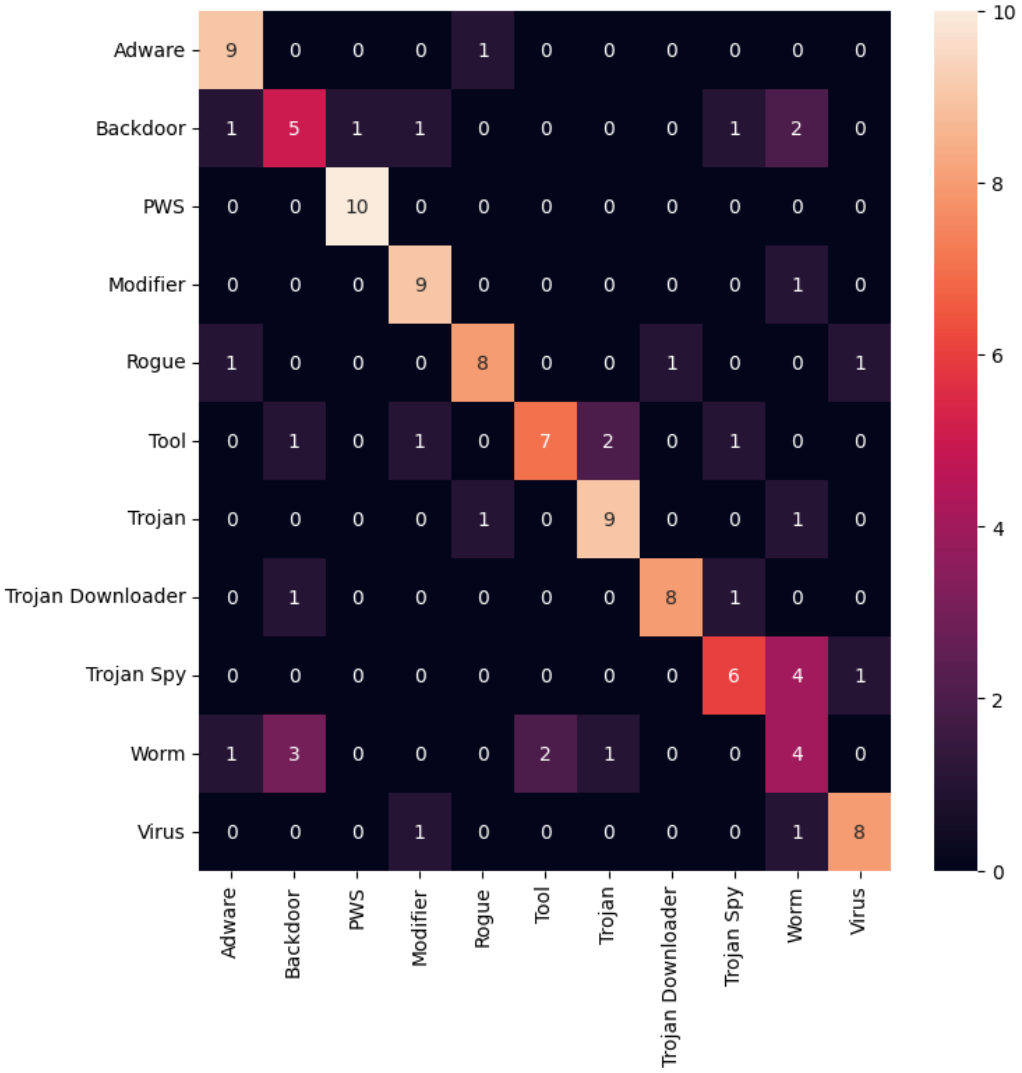


Figure A.41: Confusion Matrix for ELMo-KNN Category Classification

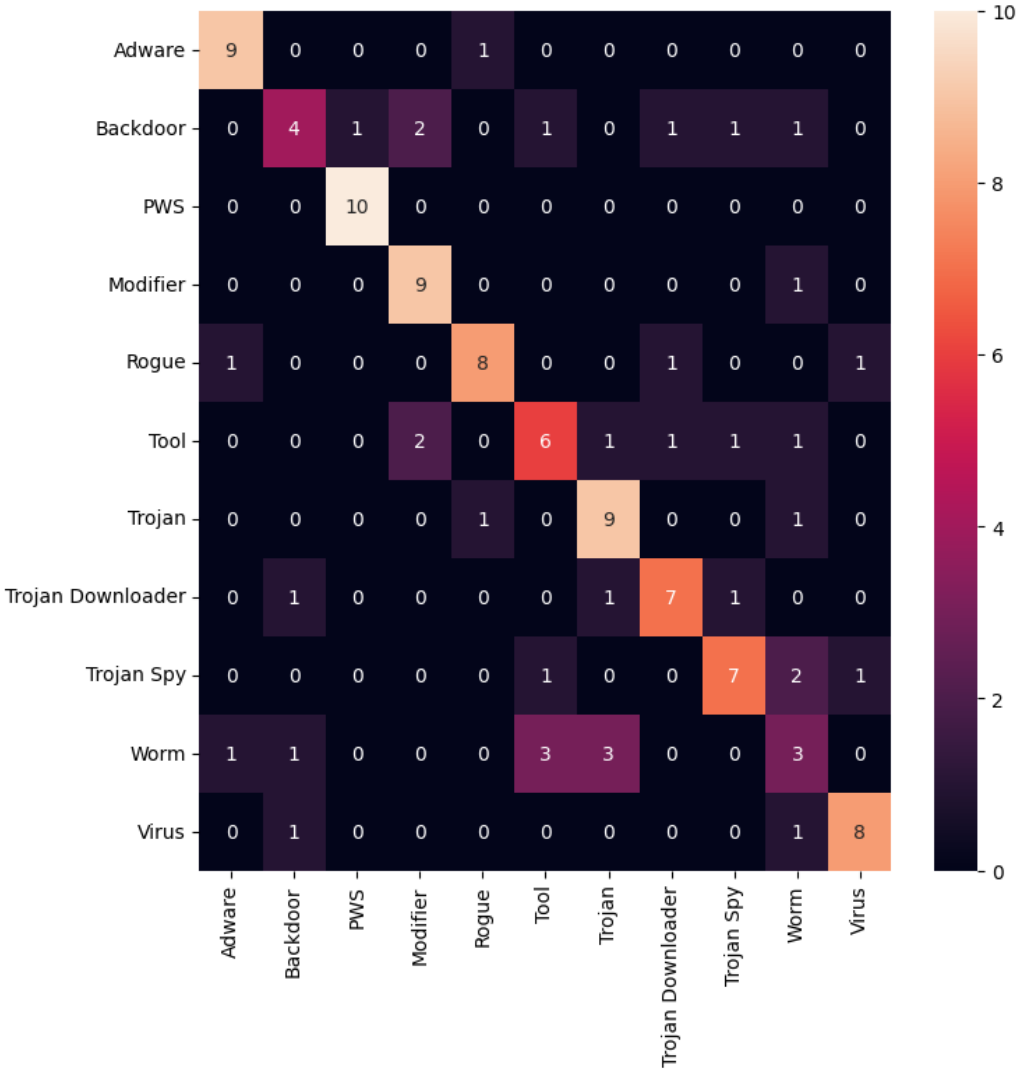
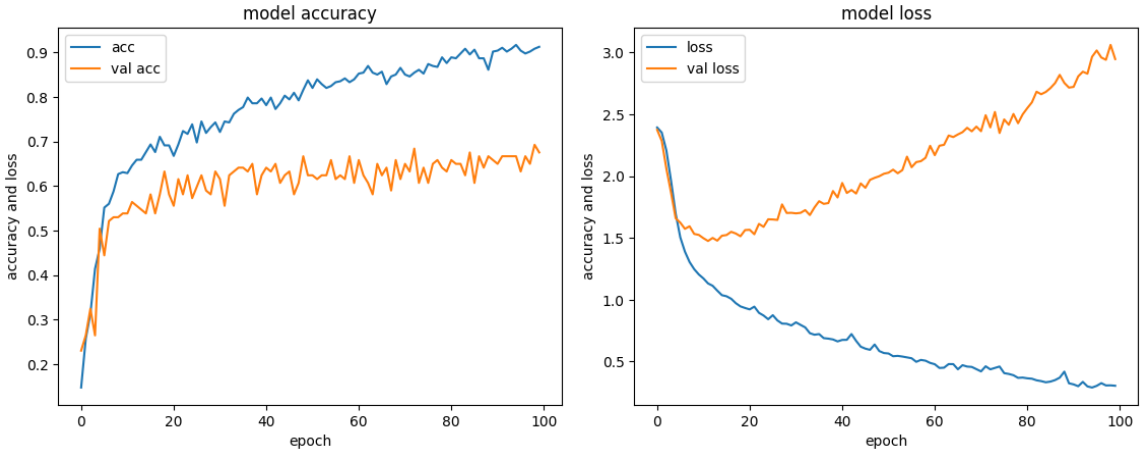


Figure A.42: Confusion Matrix for ELMo-CNN Category Classification



(a) Accuracy (b) Loss

Figure A.43: Training Model Accuracy vs Loss For ELMo-CNN For Category

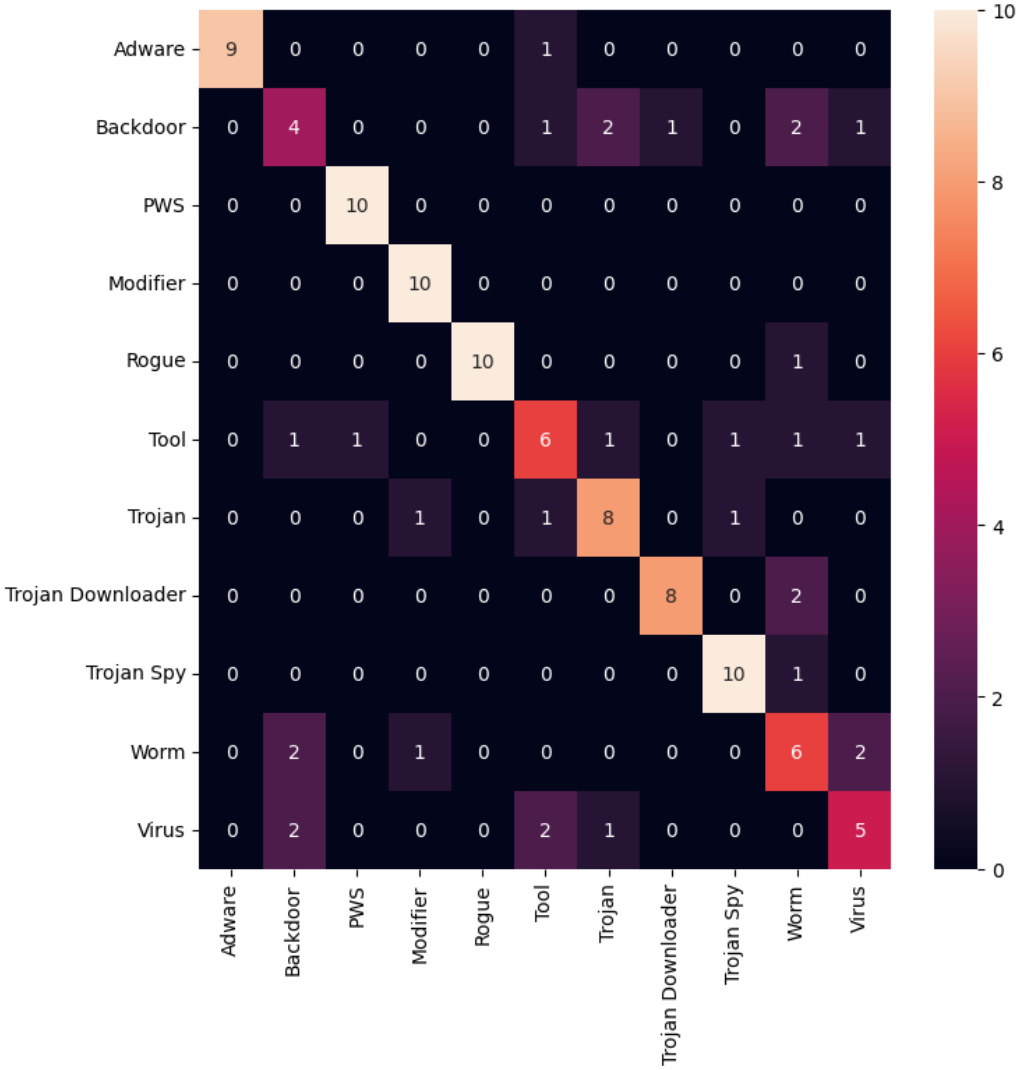


Figure A.44: Confusion Matrix for BERT-SVM Category Classification

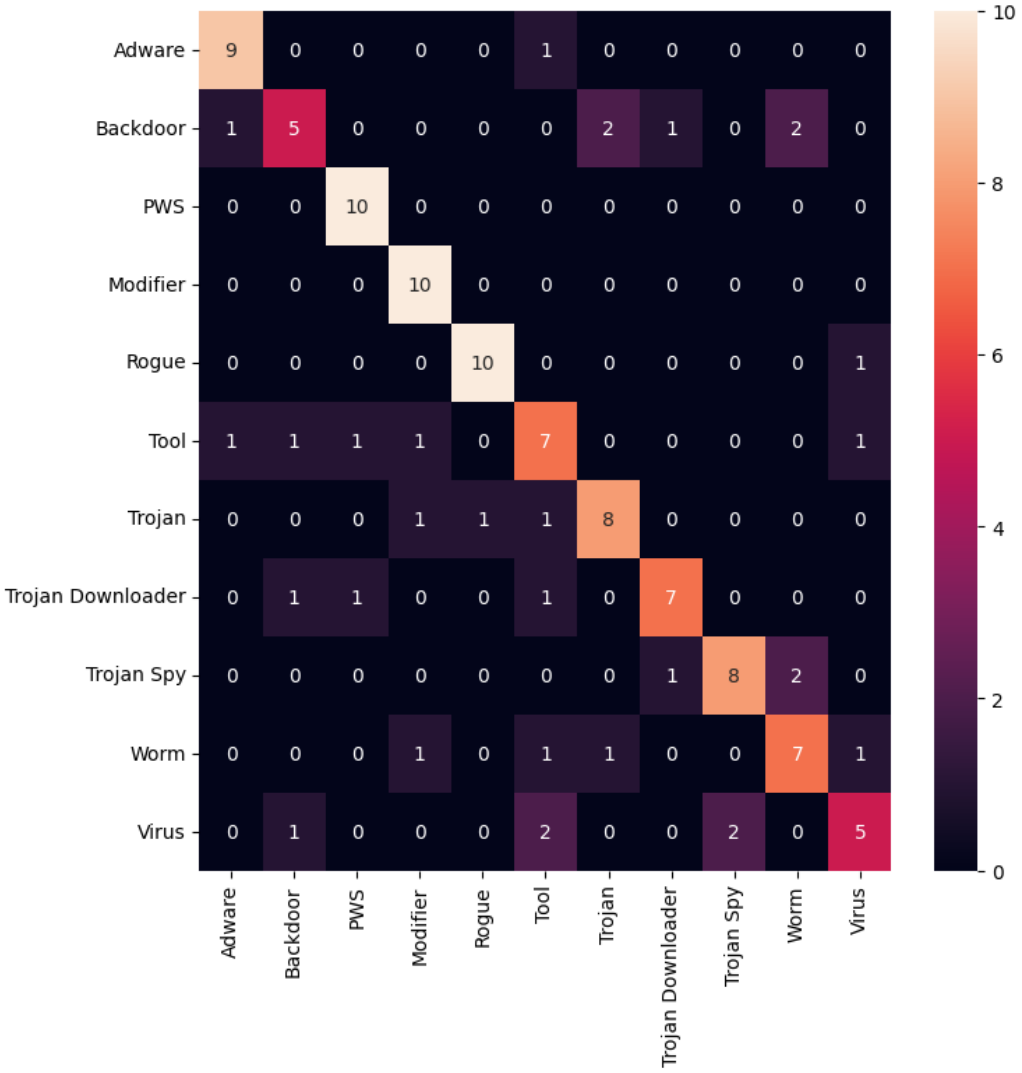


Figure A.45: Confusion Matrix for BERT-KNN Category Classification

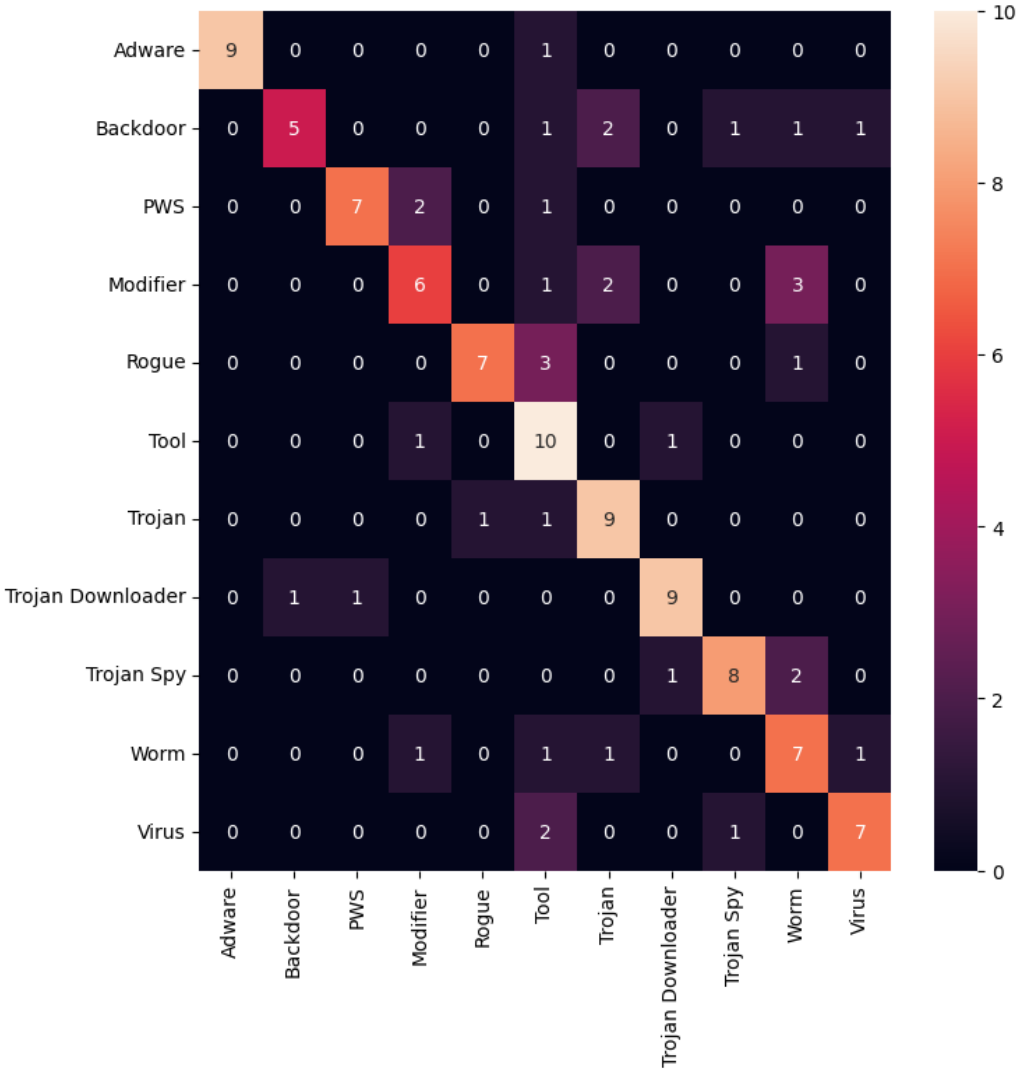
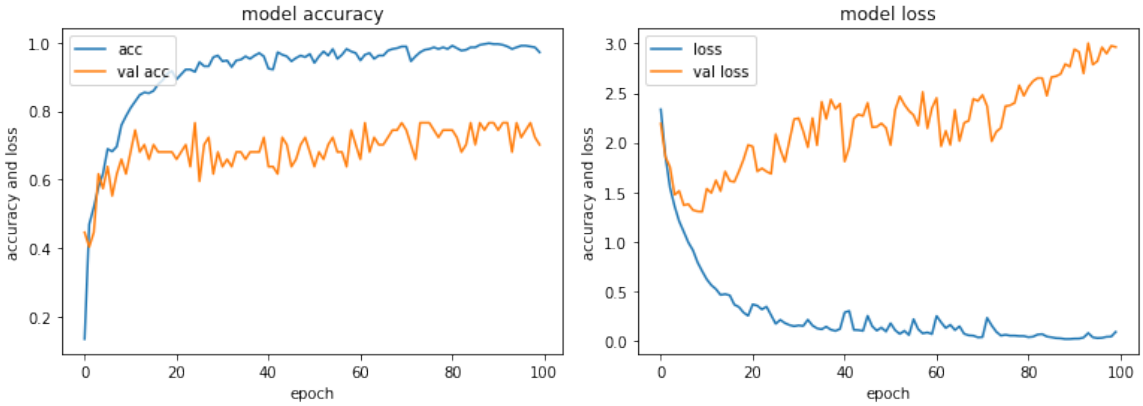


Figure A.46: Confusion Matrix for BERT-CNN Category Classification



(a) Accuracy (b) Loss

Figure A.47: Training Model Accuracy vs Loss For BERT-CNN For Category

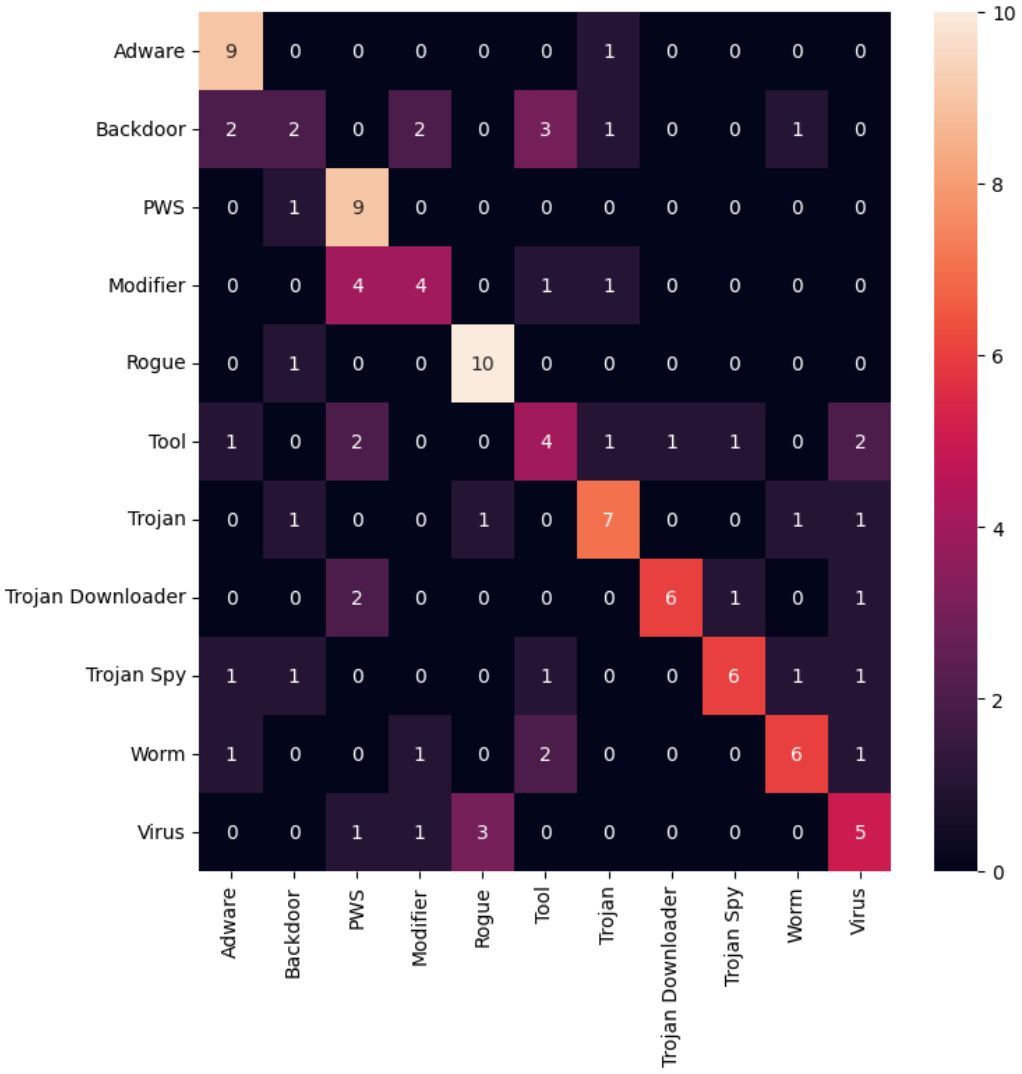


Figure A.48: Confusion Matrix for HMM2Vec-SVM Category Classification with Other Symbol (41 Calls)

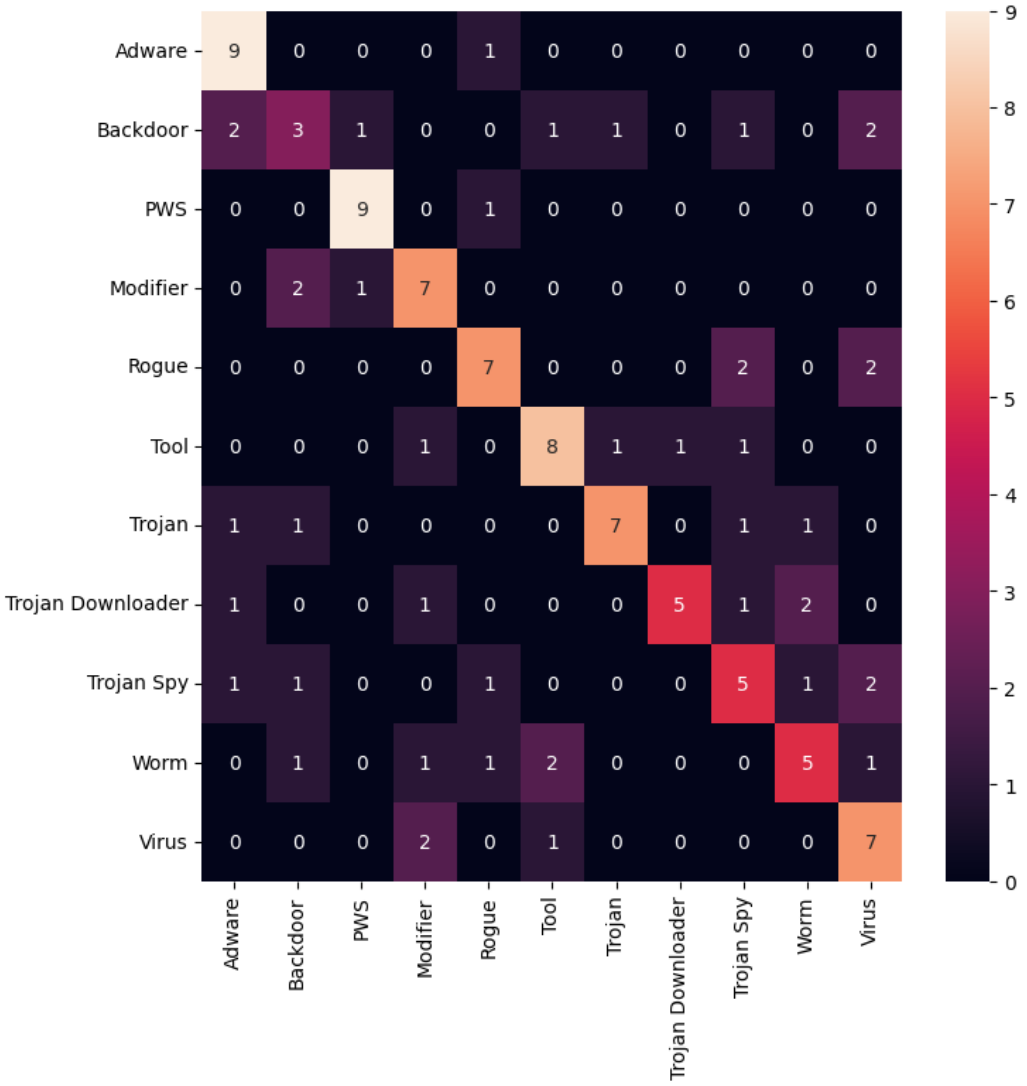


Figure A.49: Confusion Matrix for HMM2Vec-KNN Category Classification with Other Symbol (41 Calls)

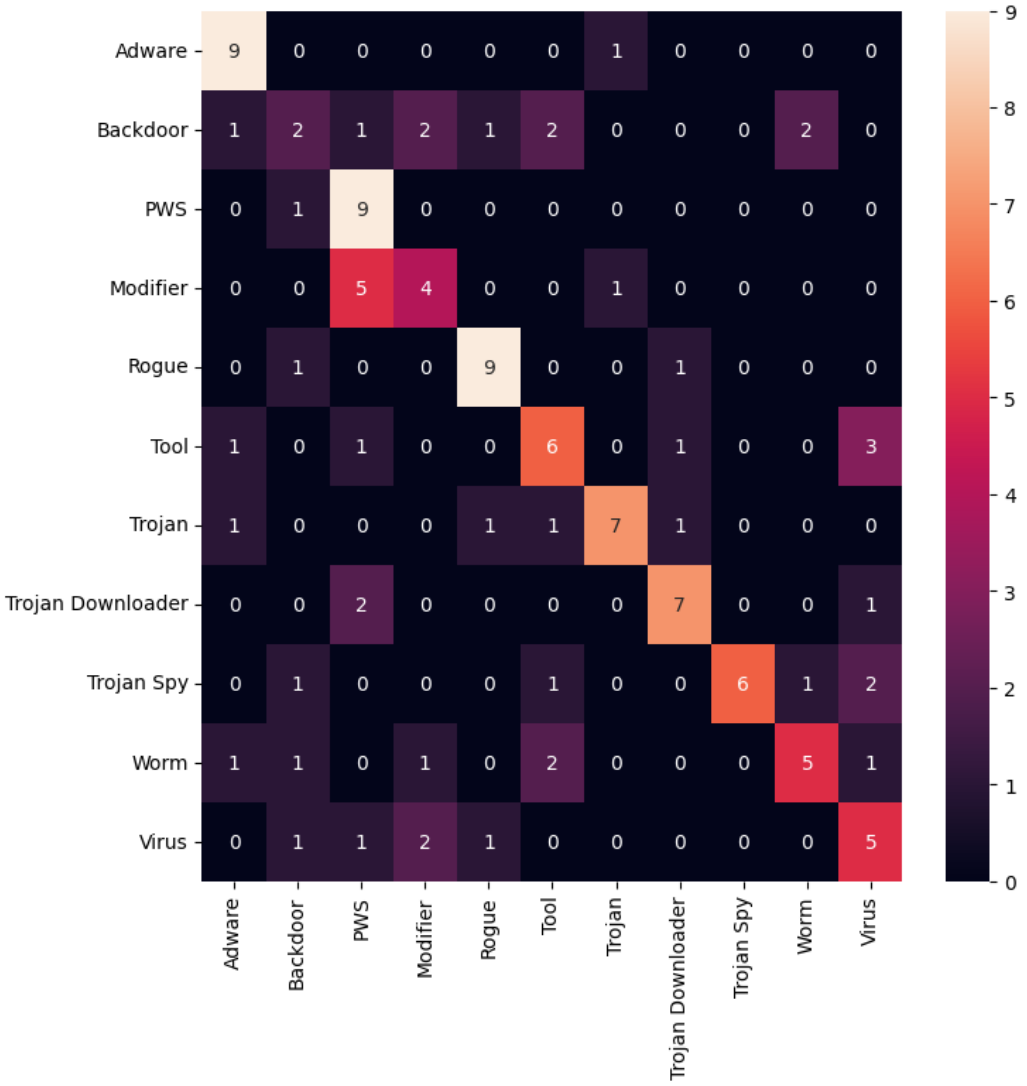


Figure A.50: Confusion Matrix for HMM2Vec-CNN Category Classification with Other Symbol (41 Calls)

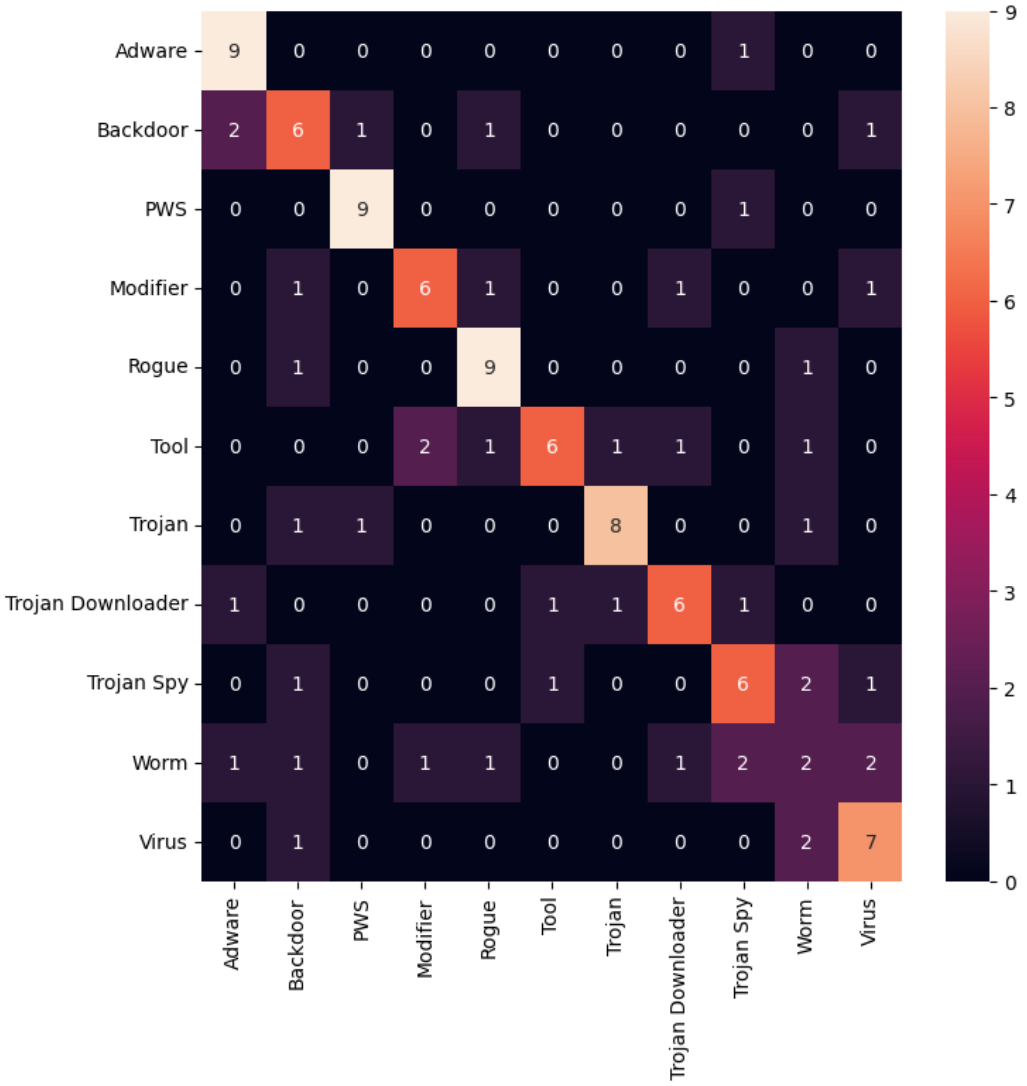


Figure A.51: Confusion Matrix for HMM2Vec-SVM Category Classification with calls in all categories

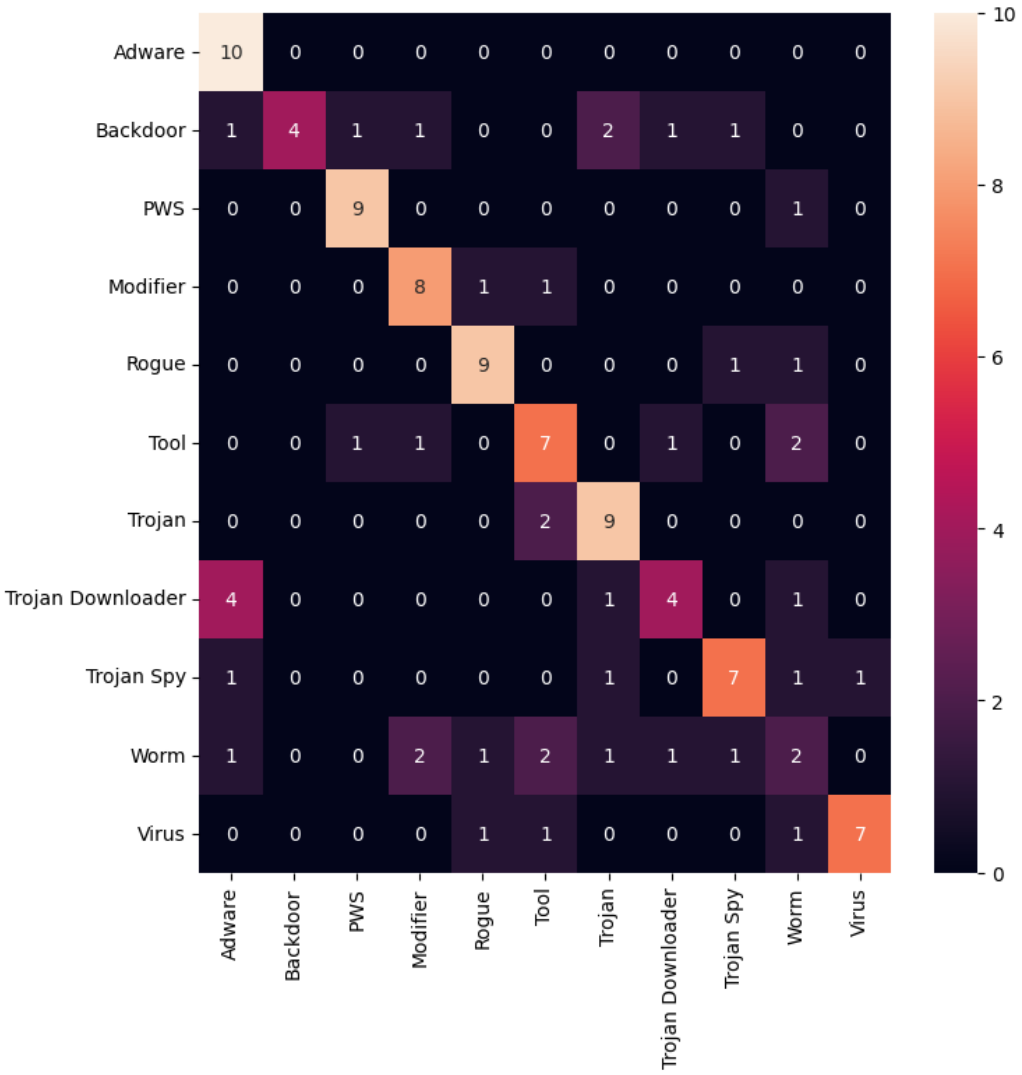


Figure A.52: Confusion Matrix for HMM2Vec-KNN Category Classification with calls in all categories

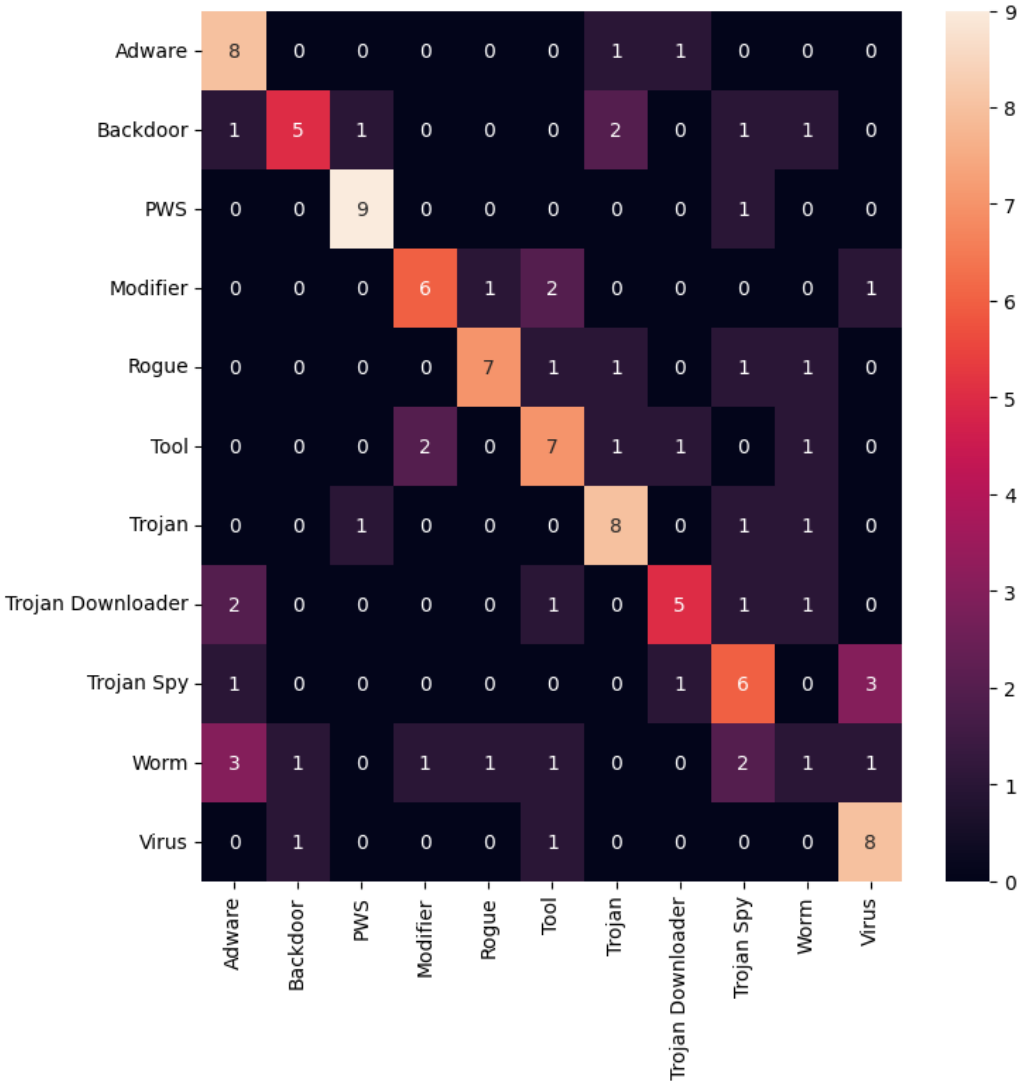


Figure A.53: Confusion Matrix for HMM2Vec-CNN Category Classification with calls in all categories

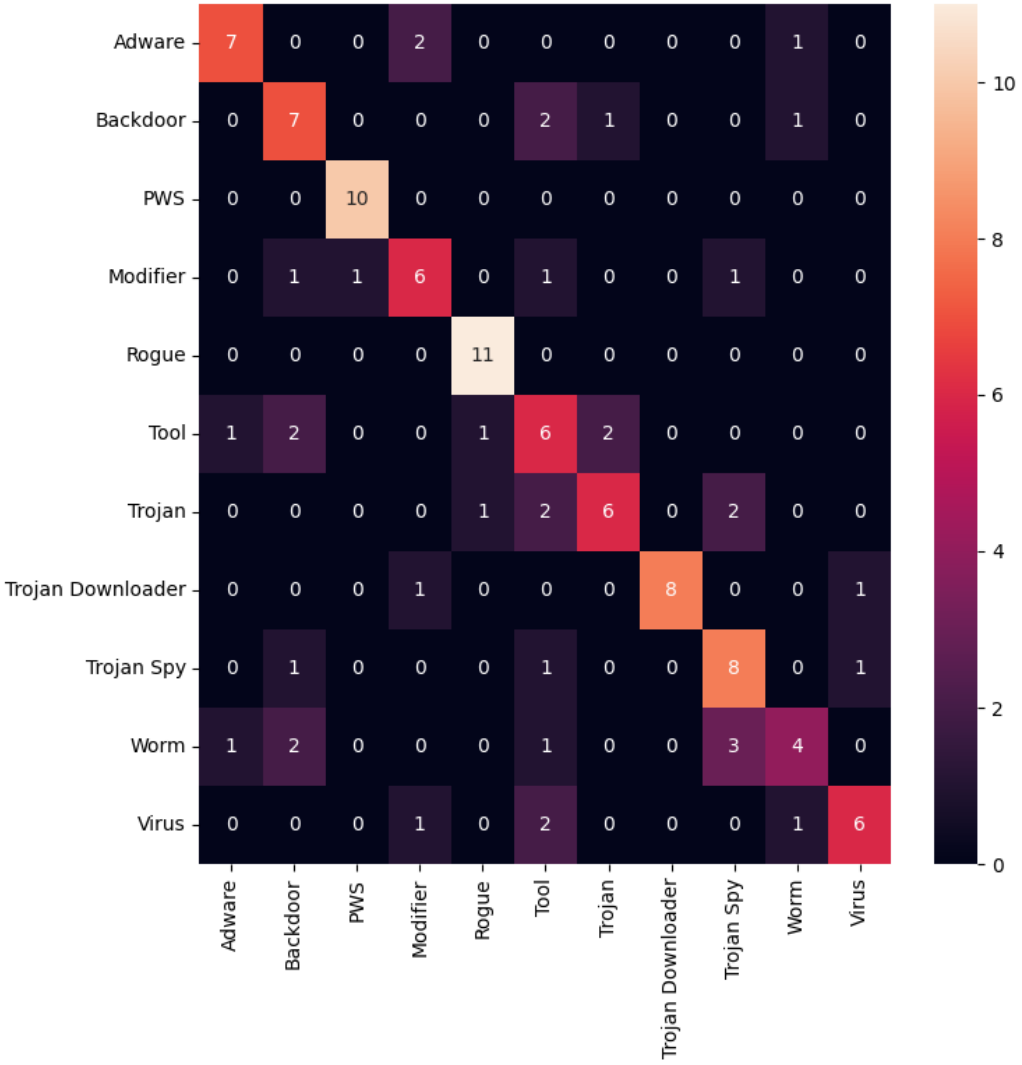


Figure A.54: Confusion Matrix for Word2Vec-SVM Category Classification with calls in all categories

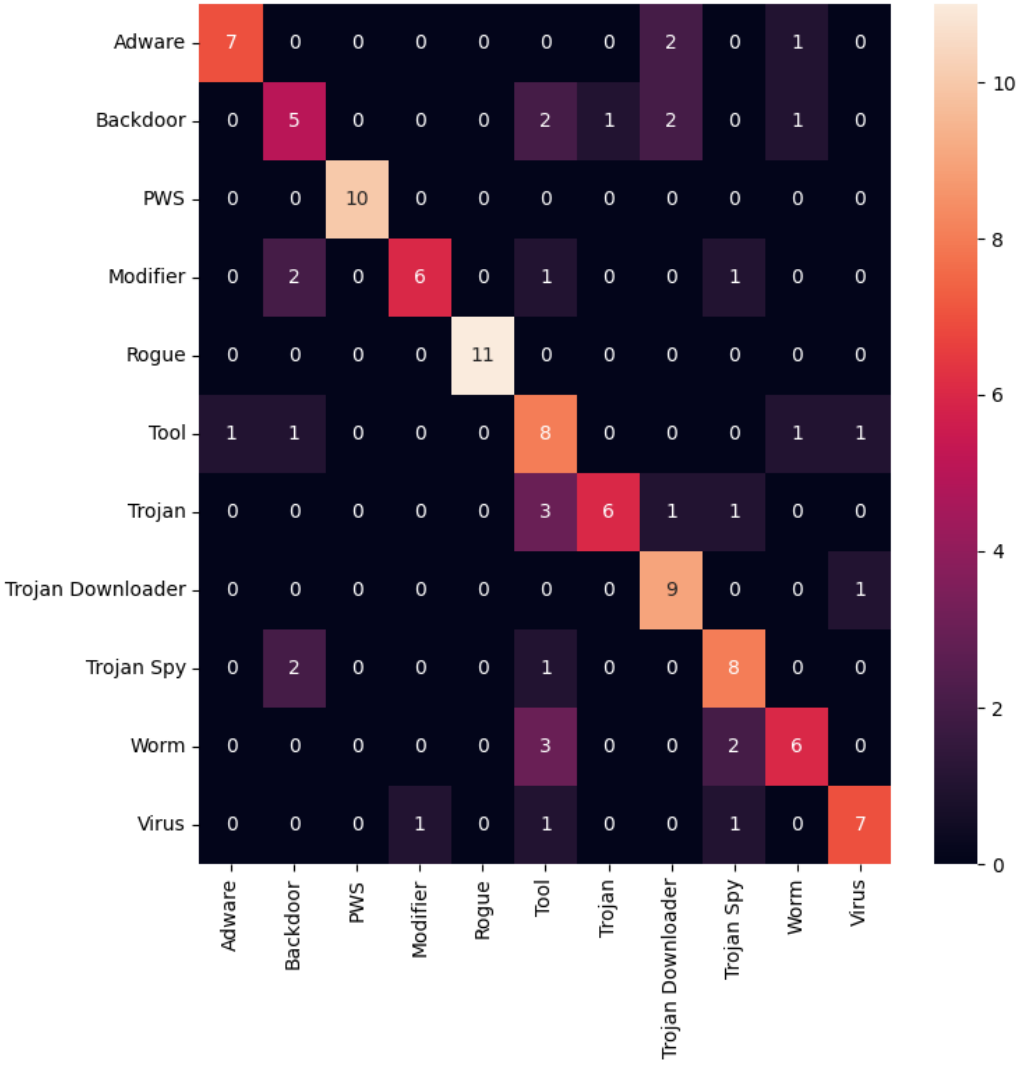


Figure A.55: Confusion Matrix for Word2Vec-RF Category Classification with calls in all categories

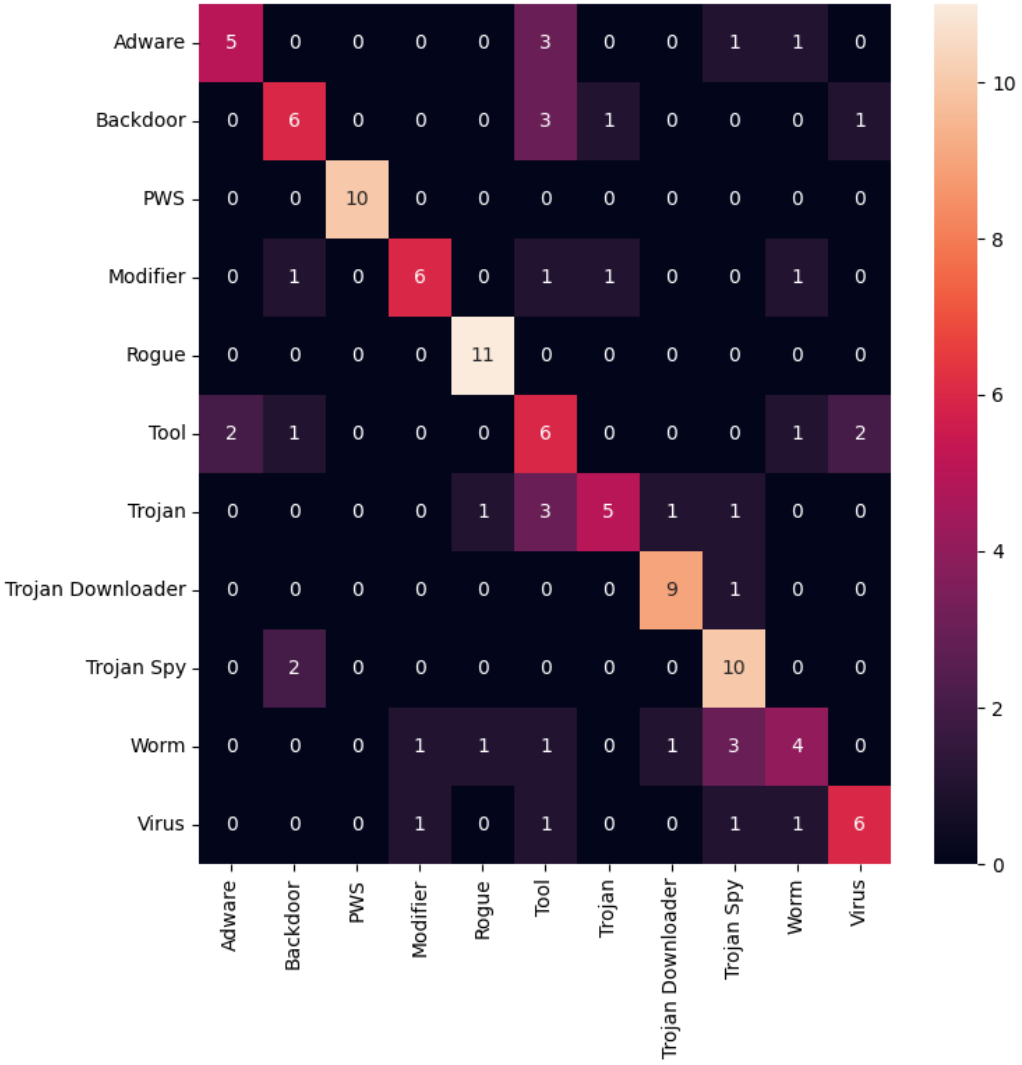


Figure A.56: Confusion Matrix for Word2Vec-CNN Category Classification with calls in all categories