Contextualized Vector Embeddings for Malware Detection

Vinay Pandya
San Jose State University

Contextualized Vector Embeddings for Malware Detection

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

Vinay Pandya

May 2022

The Designated Project Committee Approves the Project Titled

Contextualized Vector Embeddings for Malware Detection

by

Vinay Pandya

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2022

ABSTRACT

Contextualized Vector Embeddings for Malware Detection

by Vinay Pandya

Malware classification is a technique to classify different types of malware which form an integral part of system security. The aim of this project is to use context dependant word embeddings to classify malware. Tansformers is a novel architecture which utilizes self attention to handle long range dependencies. They are particularly effective in many complex natural language processing tasks such as Masked Language Modelling(MLM) and Next Sentence Prediction(NSP). Different transfomer architectures such as BERT, DistilBert, Albert, and Roberta are used to generate context dependant word embeddings. These embeddings would help in classifying different malware samples based on their similarity and context.

Apart from using transformer models we also experimented with different bidirectional language models sunch as ELMo which can generate contextualized opcode embeddings.This project also discusses algorithms for generating embeddings for byte level N-grams. We utilize *Word2vec, Glove and Fasttext* algorithms to generate context free embeddings. The classification algorithms employed in this project consist of Resnet-101 CNN, Random forest,Support Vector Machines (SVM), and k nearest neighbours. Transformer models sometimes act as black boxes which makes it difficult to understand their decisions.Various intrepretable models are utilized to explain their inner workings and improve our understanding of the model to explain their results.

Index terms **- Contextualized Embeddings,Transformer models, BERT, Bidirectional Language Models, ELMo, Glove, Word2vec, Fasttext, Optuna**

ACKNOWLEDGMENTS

I would like to thank my advisor Prof. Fabio Di Troia for his constant guidance,support and motivation throughout this project. His advice and encouragement helped in trying out new ventures and perspective in this project.

I would also like to express my gratitude towards my committee members Prof. Katerina Potika and Prof. Mark Stamp for their feedback and support. I have learned a lot through this project expecially the importance of explainability and building simpler tools.

Finally I would like to thank my family for always encouraging me and helping me in my tough times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Malware is a software created with the intent to disrupt or damage a computer systems. Malware developers try to hide malicious code in benign programs making them difficult to detect. Malware developers are constantly looking for new ways to exploit security breaches in order to extract money or gain access to personal information. A malware could encrypt critical information with a ransomware program by masquerading itself as a legitimate program. According to Symantec report more than 669 million new malware variants were created and detected in the last five years [1]. Sophos states that malware attacks consist of 34% of all the data breaches on cloud and remote based infrastructures.According to McAfee around 60 million new malware samples were created in 2019 [2]. In 2017 WannaCry and its variants caused great disturbance nearly costing four billion dollars [3]. With the ever increasing variety of malware attacks, the problem of malware classification becomes critical for information and system security.

Majority of commercial antivirus software use signature-based detection system. They compute the hash of the file and compare it with the hash of known malware files. While this approach works well for detecting known malware, they fails to detect unknown families and even variants of known families. Detection of malware and their variants is not trivial as many attackers use metamorphic and polymorphic code which can generate a vast number of variants. While these techniques are capable of detecting specific malware families it fails to detect new types and new variants of known malware families. Metamorphic malware can morph itself by inserting dead or benign code within their structure. Polymorphic malware pairs a mutation engine with self-propagating code to continually change its “appearance,” and it uses encryption (or other methods) to hide its code. They are adapting and mutating software which hackers use to infiltrate the system. Metamorphic malware such as Evol and zist can change their signature after each new infraction thus rendering the system ineffective. Detecting these malware samples is challenging as they can morph themselves using a combination of substitution,insertion,deletion and transposition [4].

These variants can easily bypass standard signature-based detection systems. Hence the ability of the system to identify these variants can help in creating many risk mitigation strategies for a whole class of programs. Each malware and their variants share certain common characteristics which can be exploited for their classification. Opcode embeddings can be used to generate contextual vector space representation of the malware files [2]. The embeddings can be generated using many transfomer models(BERT-Based models) and ELMo. These representations are dynamically informed by the surrounding opcodes. The embeddings created will then be fed to various multi-class classification models . The overall classification depends upon the ability of the embeddings to recognise the latent features in the opcodes and exploit them accordingly.

Apart from using opcode sequences, N-grams of byte level sequences are one of the most common feature types used in static analysis. We treat a file as a sequence of bytes which are then converted into embeddings using algorithms like Word2Vec,and Fasttext. The N-gram byte level embeddings are easy to generate since they do not require any domain knowledge about the executable.

Many of the transformer models discussed in this report are huge and contain a large number of parameters. Because of their size and attention constraints they often act as black box models which makes it difficult to trust their predictions. Interpretable machine learning techniques can help us in understanding the prediction of these Language models.

The remainder of this report is organized as described in Figure 1. The dataset

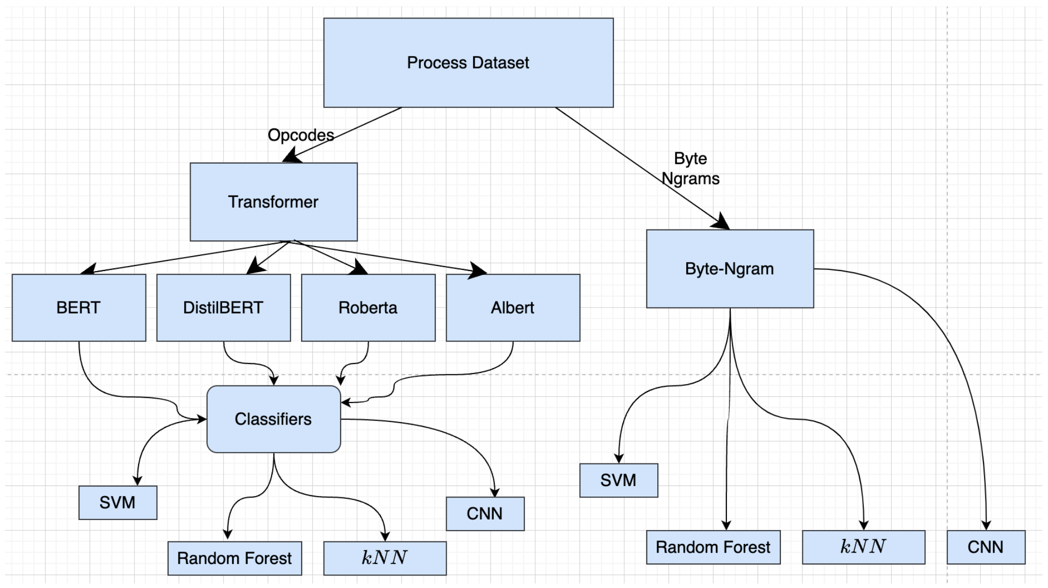


Figure 1: General Layout of the Experiments

is first preprocessed and opcodes are extracted from it using *objdump*. Section 2 discusses previous literature and paradigms used for malware classification and also discusses the drawbacks and limitations of these approaches. Section 3 delves into the background of the architectures mentioned in this research followed by the results of the experiments in Section 4. Section 5 introduces the concept of Explainability where the model agnostic approaches are discussed to interpret the complex CNN and BERT-based models. Section 6 discusses future work and other paradigms which can be applied for malware classification.

CHAPTER 2

Related Work

Malware is a program which is designed with the intention to disrupt or damage the computer systems.Malware developers try to hide malicious code in benign programs making them difficult to detect. A lot of malware based recognition techniques involve signature and behavioural based analysis.

Malware analysis can be broken down into two categories- Static analysis and Dynamic analysis.Static analysis involves disassembling the malware file and extracting important attributes such as opcode sequences and API call graph. Dynamic analysis is used to analyze a software's behavior while executing it in a virtualized environment. Most common dynamic analysis' features include API calls, register changes and intrusion traces and more [4].

In [5], API call sequences from malware files were extracted and experiments with different machine learning algorithms were observed. They achieved 98% accuracy. However,they only focused on separating the malware files from the benign files and failed to classify various variants of other malware.

Authors in [6] discuss various data mining approaches for selecting features from PE files.The features selected are then fed to classification clustering models such as kNN and Naive Bayes classifier for detecting distinct features.

In [7], the authors introduce various data augmentation techniques for better generalization of Convolutional Neural networks.They use a self embedding architecture which uses the networks own opcode embedding layer to apply additive data augmentation which can help in generating more data for the model. The embedding layer learns the semantic relationships between opcodes which can help in generating realistically augmented inputs for training a malware classifier.

In [8], the authors use a genetic algorithm for the selection of optimized feature subset for training machine learning models.They claim that this feature selection can reduce the model complexity while maintaining good accuracy.However they do not discuss the fitness and crossover criteria for different variants and also do not explain the mutation function for generating new features.

Inspired by genetic biology, Yihang Chen *et al* introduces the concept of software gene which can be mapped to a feature vector using machine learning.The vectors are then fed to various classification models and are evaluated using AUC-ROC metrics. They claim that the gene extraction approach achieves better accuracy and is more robust than N-gram based detector [9].

Using static and dynamic data sequences in sequential models is discussed in [4], where the features from both static and dynamic analysis are fed into a recurrent neural network, specifically LSTM. A similar approach is followed by [10], where they use stacked LSTM blocks on API system calls extracted using natural language processing.

Word embeddings are learned representations of text where words which have same meaning appear closer together. They provide a dense representation of a word which can be fed to neural network for further analysis. This dense representation can capture semantic relationships between words.

Developed by Tomas Mikolov and other researchers at Google in 2013, Word2Vec is a word embedding technique for solving advanced NLP problems [11]. Its embeddings learn association and dependencies among words which are then compared using cosine similarity. Glove *i.e* global vectors is another algorithm which generates word embeddings. It uses a large co-occurence matrix which is then reduced using matrix factorization methods such as LSA and skip gram.

These embedding techniques however present a context free representation of the corpus. They do not account for the context in which the word occurs. For example the word "apple" in the sentence "I ate an apple." and in "I like apple products." have different meanings. Since algorithms such as Word2vec and Glove cannot model long term dependencies among tokens in a temporal sequence, they cannot represent contextual representation of the opcodes [12].

Transformer architectures have proven very effective in various language processing tasks such as text classification, named entity recognition and question answering problem. With the help of self attention mechanism they can encode contextual information to generate contextualized word embeddings [12]. Apart from transformer models ELMo(Embeddings for Language model) can also generate context-aware opcode embeddings. ELMo creates deep contextualized word representations using BiLM model. In BiLM the forward LM is trained to predict the next word whereas Backward LM is trained to predict the past words [13].

This project also focuses on creating embeddings for byte level N-grams and evaluates their overall accuracy. The byte N-gram embeddings are then classified using various multi class classifiers.

The experiments conducted in this research consist of testing out various techniques of embedding generation using various transformer models and assessing their overall performance on the accuracy [14]. In order to better interpret these models, various explainable methods are explored which gives us insights into what these large models are predicting.

CHAPTER 3

Background

This Section goes into the details of the transformer models and the classifiers used in this research. It briefly describes the transformer architecture and the models used in this project. The transformer models discussed here are BERT, DistilBERT, Roberta, and Albert while the classifiers discussed are SVM, Random forest, CNN, and kNN. The experiments verifying the results will be discussed in the next Section.

3.1 Transformer models

Prior to the advent of transformers many language processing tasks were handled by complex recurrent neural networks which could integrate well with large scale sequential data like text. One limitation is that it encodes the internal sequence to a fixed length internal vector which degrades its performance while handling long temporal sequences. Attention mechanism introduced by Bahdanau *et al.* alleviates these limitations to certain extent [15].

As seen in Figure 2 the attention mechanism provides an intuitive way to inspect the alignment within words in a sequence. This helps in identifying words which are important during prediction.

$$c_i = \sum_{j=i}^{T_x} \alpha_{ij} h_j \tag{1}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp e_{ik}} \tag{2}$$

$$e_{ij} = LSTM(s_{i-1}, h_j) \tag{3}$$

h_j is the hidden layer of the encoder

here c_i represnts context vector

α_{ij} is the weight vector of the hidden states of the decoder

e_{ij} represents the output from previous timestep.

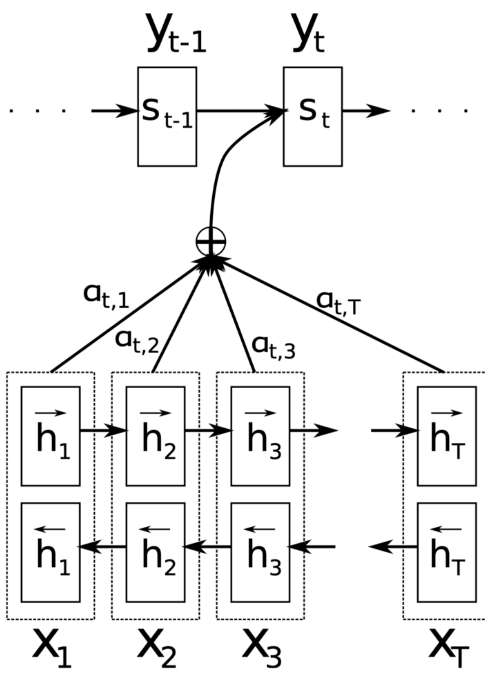


Figure 2: Attention network architecture with encoder decoder model [15]

Going from bottom to top equation 3 calculates the hidden states for the decoder which is the LSTM output at timestep i. Equation 2 calculates the probability softmax for each word in the sentence(encoder sentence) and equation 1 calculates the attention weights for each word at the timestep i. Although the mechanism introduces additional computation overhead (α and c_i context vectors) it is able to outperform vanilla encoder decoder models on various NLP tasks. The context vector helps the decoder in attending every hidden state from the encoder and helps in identifying tokens which give more information at a particular timestep.

Even with the help of attention mechanism the sequential RNN models are unable to handle long term dependencies among the tokens.Since the traditional sequence-to sequence models access the tokens sequentially it precludes parallelization and ultimately cannot handle longer sequences as memory constraints catch limit the batch size across samples.The researches in [16] introduced the concept of self attention which is the central theme of transformer architecture. Self attention sometimes called as intra attention can create a sequence representation by relating different positions in the same sequence. The advantage of self attention is that it can handle larger temporal sequences and also create a contextualized representations of them.

The components of transformers are as follows

$$q = X * W_q \tag{4}$$

$$k = X * W_k \tag{5}$$

$$v = X * W_v \tag{6}$$

X is the Input token(or word)

q is a query vector

W_q is the weight of q

k is key vector

W_k is the matrix representation of k

v is value vector

W_v is the matrix representation of v

The equation for self attention is given as follows

$$Attention(q, k, v) = softmax(\frac{qk^t}{\sqrt{d_k}})v$$

We can consider the attention function as using the query q to find the most similar key k. The closer key value product will have value towards 1 (softmax). The scaling factor $\frac{1}{\sqrt{d_k}}$ helps in preventing the dot products between q and k to grow exponentially large and thereby preventing the vanishing gradient problem.

So the generalised procedure for computing attention is as follows

1. Compute alignment scores by multiplying queries q with keys k

$$qk^t = \begin{bmatrix} e_{11} & e_{12} & \dots & e_{1n} \\ e_{21} & e_{22} & \dots & e_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \dots & e_{mn} \end{bmatrix}$$

2. Scale the alignment scores

$$\frac{qk^t}{\sqrt{d_k}} = \begin{bmatrix} \frac{e_{11}}{\sqrt{d_k}} & \frac{e_{12}}{\sqrt{d_k}} & \dots & \frac{e_{1n}}{\sqrt{d_k}} \\ \frac{e_{21}}{\sqrt{d_k}} & \frac{e_{22}}{\sqrt{d_k}} & \dots & \frac{e_{2n}}{\sqrt{d_k}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{e_{m1}}{\sqrt{d_k}} & \frac{e_{m2}}{\sqrt{d_k}} & \dots & \frac{e_{mn}}{\sqrt{d_k}} \end{bmatrix}$$

3. Apply softmax

$$softmax(\frac{qk^t}{\sqrt{d_k}}) = \begin{bmatrix} softmax(\frac{e_{11}}{\sqrt{d_k}}) & softmax(\frac{e_{12}}{\sqrt{d_k}}) & \dots & softmax(\frac{e_{1n}}{\sqrt{d_k}}) \\ softmax(\frac{e_{21}}{\sqrt{d_k}}) & softmax(\frac{e_{22}}{\sqrt{d_k}}) & \dots & softmax(\frac{e_{2n}}{\sqrt{d_k}}) \\ \vdots & \vdots & \ddots & \vdots \\ softmax(\frac{e_{m1}}{\sqrt{d_k}}) & softmax(\frac{e_{m2}}{\sqrt{d_k}}) & \dots & softmax(\frac{e_{mn}}{\sqrt{d_k}}) \end{bmatrix}$$

   this gives the weight of different words

4. multiply the resulting weights to values in matrix

$$softmax(\frac{qk^t}{\sqrt{d_k}}).v = \begin{bmatrix} softmax(\frac{e_{11}}{\sqrt{d_k}}) & \dots & softmax(\frac{e_{1n}}{\sqrt{d_k}}) \\ softmax(\frac{e_{21}}{\sqrt{d_k}}) & \dots & softmax(\frac{e_{2n}}{\sqrt{d_k}}) \\ \vdots & \ddots & \vdots \\ softmax(\frac{e_{m1}}{\sqrt{d_k}}) & \dots & softmax(\frac{e_{mn}}{\sqrt{d_k}}) \end{bmatrix} \begin{bmatrix} v_{11} & \dots & v_{1d_v} \\ v_{21} & \dots & v_{2d_v} \\ \vdots & \ddots & \vdots \\ v_{m1} & \dots & v_{nd_v} \end{bmatrix}$$

Building on this intuition the researchers in [16] introduce the concept of multi-headed attention.

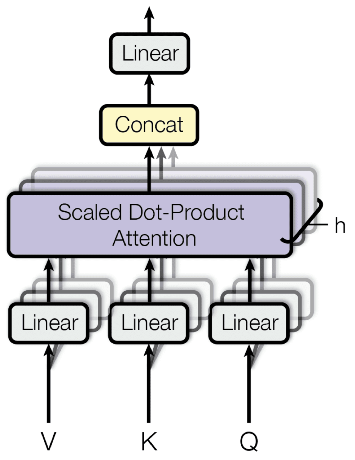


Figure 3: Multi headed attention introduced in "Attention is all you need" [15]

The multi headed attention projects the queries,keys and values h times (Figure 3). The single attention is then applied parallel to produce h outputs which are then concatenated to produce the final result. The multi headed attention mechanism allows the model to focus on different positions. For example if we consider the sentence "Apple products have an issue with memory.They need to fix it." we would like to know what "they" refers to. It also helps in extracting information from multiple representation subspace which would not be possible with a single attention head [17]

The complete encoder decoder architecture is described in fig 4. Since the encoder cannot inherently capture information about the relative positioning of the input sequence a positional encoding vector is added along to the input embeddings. These embeddings inject the necessary positional information for each token in the input sequence.

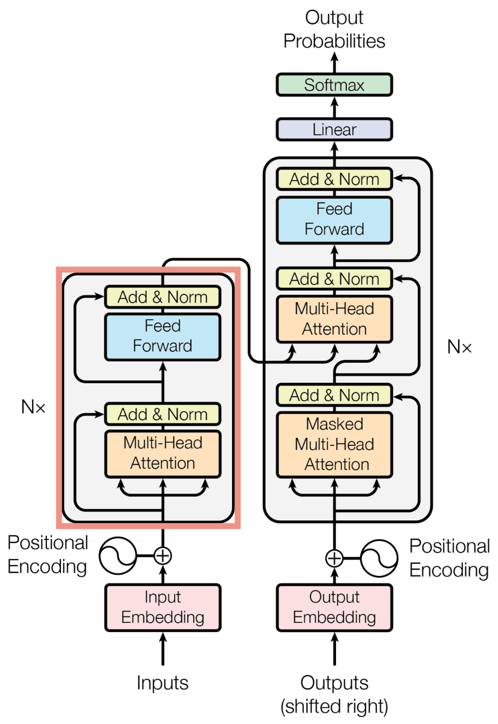


Figure 4: Transformer architecture [18]

According to Figure 4 outputs of the self-attention layer are fed to a feed-forward neural network. The exact same feed-forward network is independently applied to each position. The residual connection between the encoder and the layer normalization step helps in preventing vanishing gradient problem and also helps in stabilizing the network.

The decoder part of the transformer works in the same way as the decoder part of vanilla RNN as the output from the previous timestep is fed back to decoder in the next timestep.The output from the encoder stack are used for encoder-decoder attention as mentioned in Figure 4.

The decoder receives previous output from decoder stack and implements self attention over the tokens.The attention part in decoder is modified to only attend to preceding words. It achieves this with the help of mask attention mechanism. It works as follows

$$mask(qk^t) = \begin{bmatrix} e_{11} & e_{12} & \dots & e_{1n} \\ e_{21} & e_{22} & \dots & e_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \dots & e_{mn} \end{bmatrix} = \begin{bmatrix} e_{11} & -\infty & \dots & -\infty \\ e_{21} & e_{22} & \dots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \dots & e_{mn} \end{bmatrix}$$

the decoder masks over the values obtained by the scalar multiplication of the matrices q and v. This is done by suppressing the values that occur in the future timestep.The complete architecture of the decoder attention is given in Figure 5

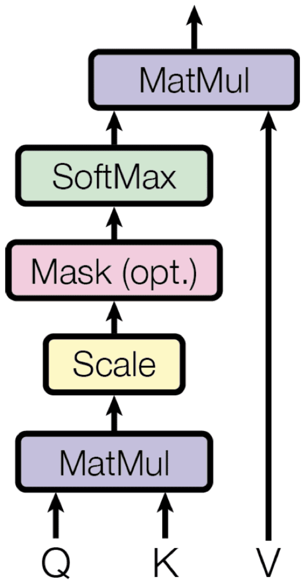


Figure 5: Decoder Network [18]

This ensures that decoder only access words from the preceding timesteps.

3.1.1 BERT

BERT (Bidirectional Encoder Representations for Transformers) was introduced in [12] which is a stack of 12 encoder models of transformer. BERT achieves state of the art in 11 language processing tasks like GLUE, SQuAD v1.1 etc. The BERT model is trained on jointly conditioning the left and right context in all layers which helps in generating context-aware embeddings. Instead of training to predict next word Bert uses masked language model which masks 15% of all the words in the corpus.The objective is then to predict the masked word based solely on the context of the surrounding words. This objective helps in fusing both the left and right context of the sequence and also helps in creating a contextual representation of the input sequence.

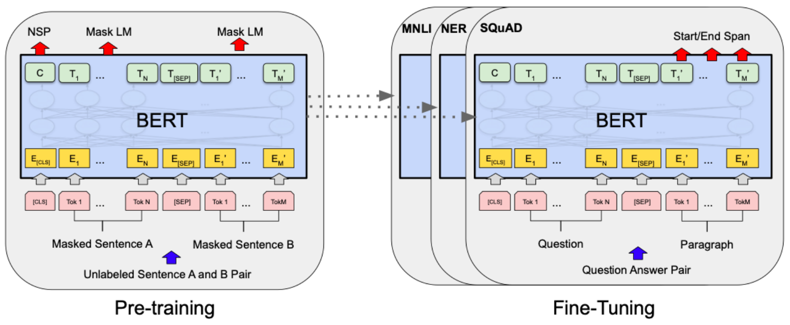


Figure 6: Bert Pre-training [12]

Another task BERT is trained on is NSP(Next Sentence Prediction) where given a pair of sentences the model has to predict whether the sentences come after one another. With this pre-training the model is able to understand the sentence level relationships and give a more coherent understanding of the corpus.

The input structure for pretraining is given in Figure 7

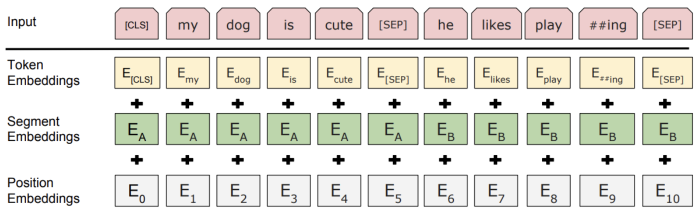


Figure 7: Input sequence for BERT [12]

The contents for the input sequence include

- **CLS**: This is the first token of every sequence. Normally combined with a softmax layer for classification.

- **SEP**: This is a sequence delimiter for next sentence prediction task. For a sequence prediction it is just appended at the end.
- **MASK**: For masking the tokens. Only considered while Pretraining.
- **Token Embeddings**: Pre-trained embeddings for different words
- **Wordpiece Tokenizer**: Subword segmentation algorithm for example word "playing" is tokenized as "play" and "###ing" in the Figure 7
- **Segment Embedding**: Segment embeddings are basically the sentence number that is encoded into a vector. This is important for next sentence prediction where we want to predict the sequence of the next sentence.
- **Positional Embedding**: These embeddings describe the position of the word within a sentence.

3.1.2 DistilBERT

Researchers in [19] introduced a smaller,cheaper and faster version of BERT model called DistilBERT which uses knowledge distillation to reduce the BERT architecture by 40% while retaining 97% of its language understanding capabilities. It is 60% faster in inference and is cheaper to train because of its lightweight architecture.As mentioned in 8 a good language model need not be ridiculously huge to achieve state of the art in language processing tasks.

Knowledge distillation is a compression technique in which a smaller model(student model) is trained to produce the same results as that of teacher model [20]. The student is trained with the distillation loss on the soft targets of the teacher model. The distillation loss can be computed as follows

$$y_i = \frac{\exp(v_i/T)}{\sum_j (v_j/T)} \tag{7}$$

$$\hat{y_j} = \frac{\exp(z_i/T)}{\sum_j (z_j/T)} \tag{8}$$

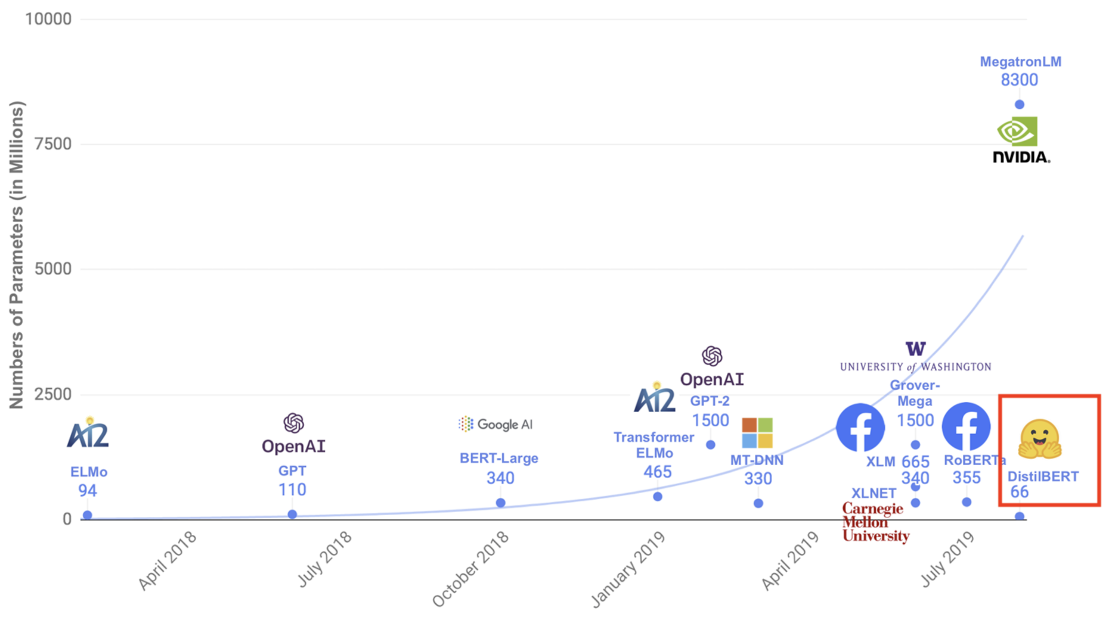


Figure 8: DistilBERT model parameters [19]

$$L1 = \frac{1}{N} * \sum_{j} y_j * \log(\hat{y_j}) \tag{9}$$

$$L2 = \frac{1}{N} * \sum_{j} p_j * \log(\hat{p_j}) \tag{10}$$

$$DistilLoss = \alpha * L1 + (1 - \alpha) * L2 \tag{11}$$

The T is the temperature parameter which is used to reduce difference between class likelihood values. Whole process can be understood from the Figure 9

While training the DistilBERT model the authors combined the distillation loss (equation 11) with MLM(masked language modelling)loss to achieve comparable accuracy while pre-training for masked language modelling. While pre-training they remove the segmented tokens which is present in BERT as DistilBERT is not used for NSP(Next Sentence Prediction) and initialized the parameters from any one of BERT's encoder parameters(BERT contains 13 encoders DistilBERT contains 6) [19].

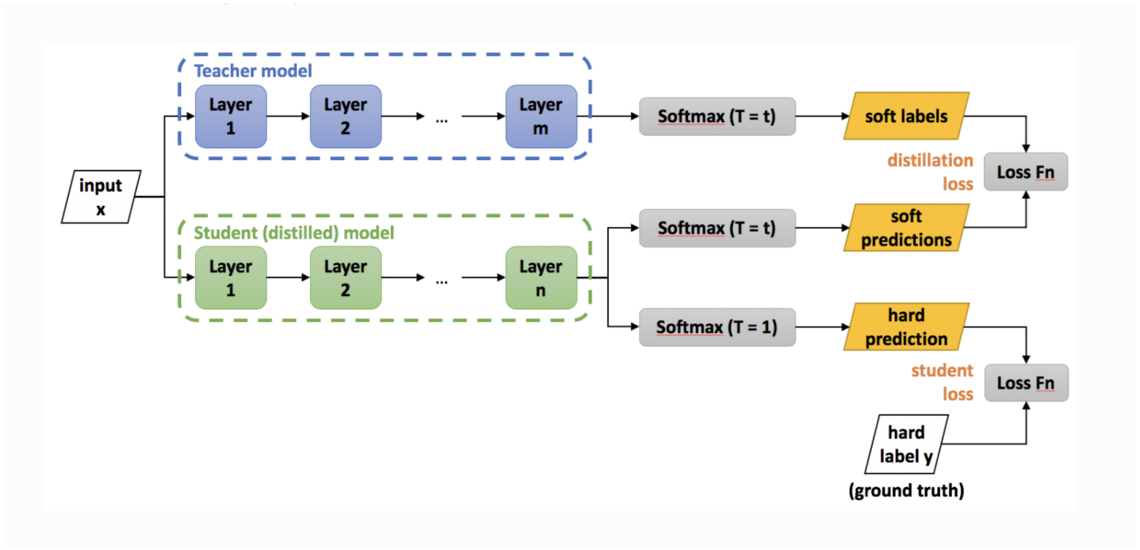


Figure 9: Knowledge distillation model [20]

3.1.3 Roberta

Robustly optimised BERT pre-training approach was introduce by Liu Y. *et al.* which explores how different hyper-parameters and different pre-training objectives coupled with significantly more data can affect the robustness of the model [21].They argue that the BERT model is largely under-trained and therefore not robust or optimal. They train the models for different objectives like

- Segment-Pair+NSP: This is the original NSP pre-training objective for BERT where the models needs to predict whether the segments follow each other.
- Sentence-Pair+NSP: This objective trains the model to predict whether two sentences follow each other.
- Full Sentences: The input consists of full sentences. When the end of document is reached the sentences from the next document are sampled.

The batch size used for pre-training is significantly larger that that of BERT ($2K$). Since WordPiece embedding require larger space to incorporate rules for subword generation changes depending on the data, the authors used Byte-Pair Encoding which can handle these changes and give better representation for input sentences. The dataset used for Roberta is significantly larger than BERT consisting of BookCorpus, CommanCRAWL news dataset, OPENWEBtext and STORIES.

3.1.4 Albert

A Lite BERT model for self supervised learning for language representations (Albert) was introduced by Lan Z *et al.* in [22].They argue that because of the huge size(around 300 million) it is harder to converge and train because of memory limitations. On top of being memory intensive the model also degraded while training for longer time.

The Albert model uses Cross-Layer parameter sharing as shown in 10 where the weights and parameters of the first block are reused for the remaining blocks. This helps in reducing the number of parameters to be learned and achieved comparable accuracy with respect to BERT.

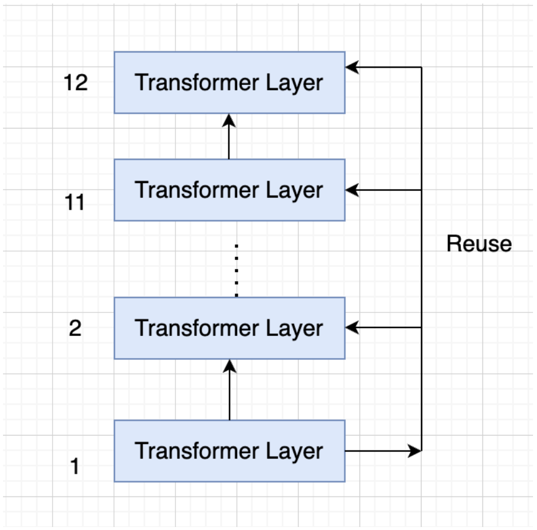


Figure 10: Parameter sharing in the Albert model

In order to reduce the actual number of parameters the authors in [22] utilize matrix factorization. In BERT model the output of the intermediate layers are the

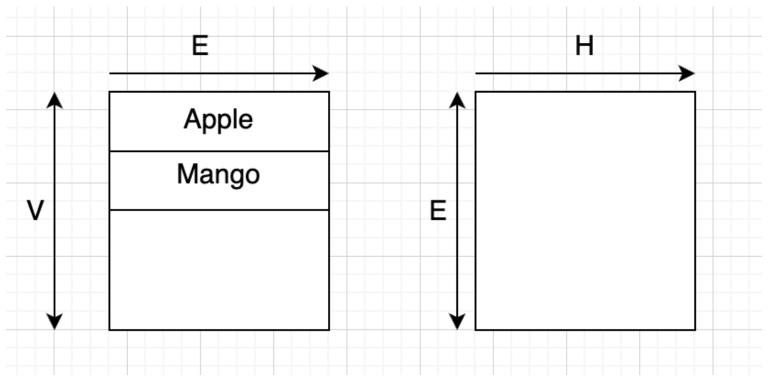


Figure 11: Matrix Factorization technique for reducing the number of parameters [22]

same as that of the vocabulary(768 hidden dimensions) whereas the albert model reduces the embedding size by multiplying the embedding with the matrix which blows up the vector to the same size as that of hidden layers (Figure 11).They claim that helps in reducing the number of BERT parameters by 89%.

Model	**Encoder Type**	**Number of Parameters (In Millions)**	**Corpus trained**
BERT	Transformer	340	BookCorpus
Roberta	Transformer	340	BookCorpus CC-News OpenWebText Stories
DistilBert	Transformer	110	BookCorpus
Albert	Transformer	110	BookCorpus

Table 1: Comparison of different architecture of BERT models

3.2 ELMo architecture

ELMo(Embeddings for Language Model) was first introduced in [13]. The embeddings generated by ELMo capture both the semantics and the syntax of the language and is able to generate a contextualized vector for a particular word. In other words same words will have different embeddings depending on the context.

The forward LM layer of ELMo model is trained on task of next word prediction

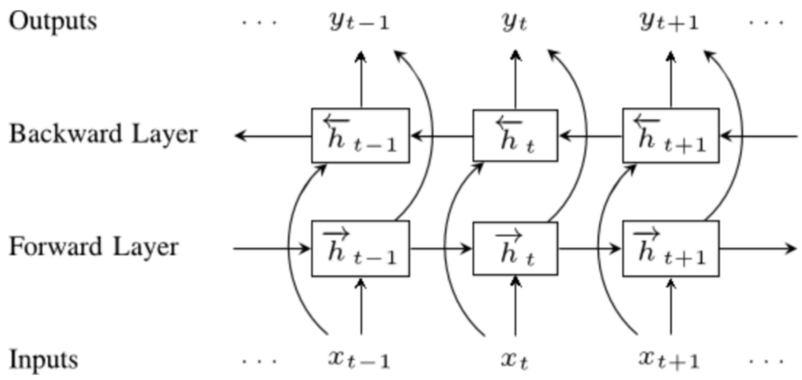


Figure 12: ELMo architecture as described in [13]

and the backward LM layer is trained on the task of Previous Word prediction. The internal states of forward LM and backward LM are concatenated and multiplied with a normalized weight vector which helps in scaling these embeddings. The concatenated layers are then added up to create a deep representation if the sequence. According to Peters .M *et al.* the lower layers of model learn the syntax while the upper layers learn the context of the word. The ELMo model utilizes a character level CNN(Convolutional Neural Network) to generate the character level embedding to handle out of vocabulary words.

3.3 Classifiers

Classification is predictive modelling problem where the model predicts class for a given sample of input data.The input data is mapped to desired class based on its features. The models which enable this facility are called classifiers. This section will cover the background on different types of classifiers.

3.3.1 Random Forest Classifiers

Random forest does classification using an ensemble of decision trees. The decision trees classify the data and the prediction with the highest number of votes is selected as the class of data [23]. The reason for choosing a large number of decision trees is because individually a single decision tree tends to overfit however, when working as an ensemble these errors are minimized and the model generalizes better for the input dataset.

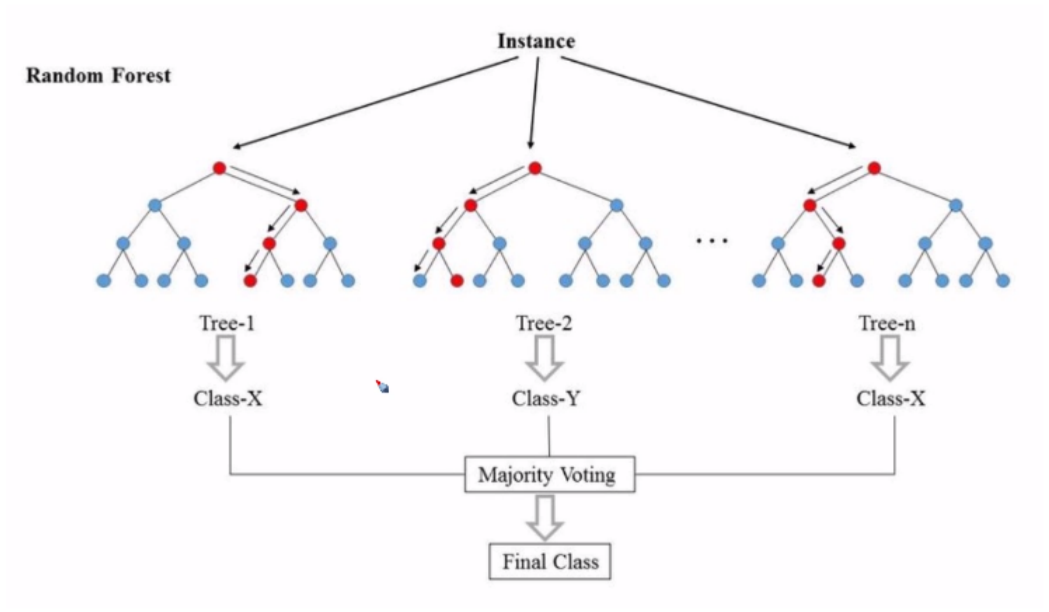


Figure 13: Random forest Trees [23]

Random forest algorithm uses a bootstrapping mechanism to generate a large number of decision trees for ensemble learning (algorithm 1). The model will draw a large number of samples from the subset and train a new classifier on each of the samples (Figure 13). The samples are not exclusive which means samples can be replaced while drawing. Using Bootstrapping with decision trees is called bagging. Bagging prevents random forests from over-fitting since the model now has a holistic view of the subset.

3.3.2 K-Nearest Neighbours

K- nearest neighbhours is a supervised machine learning algorithms which uses information of the neighbouring data points to predict a class label. There is no explicit training phase in this model and while testing the prediction is decided based on the nearest neighbors in the dataset. Small values of K result in unstable values

Algorithm 1 Random Forest Algorithm

```
Require: A training set :=$(x_1, y_1), ....(x_n, y_n)$,features $F$ and number of trees in the
    forest $B$
 1: function RandomForest($S$,$F$)
 2:     $H \leftarrow \phi$
 3:     for $i \in 1...B$ do
 4:         $S^{(i)} \leftarrow$ A bootstrap sample from $S$
 5:         $h_i \leftarrow RandomizedTreeLearn(S^{(i)}, F)$
 6:         $H \leftarrow H \cup \{h_i\}$
 7:     end for
 8: return $H$
 9: end function
10: function RandomizedTreeLearn($S$,$F$)
11:     At each node
12:     $f \leftarrow$ very small subset of $F$
13:     Split on the best feature in $f$
14: return the learned tree
15: end function
```

and tend to overfit quickly. Increasing the K values can result in imporving the accuracy upto a certain point but after that the performance degrades.

3.3.3 SVM

Support Vector Machines(SVM) are a class of supervised machine learning algorithms which use the concept of seperating hyperplane to carry out classification. The Goal of SVM is to create a hyperplane which maximizes the distance between classes. SVM also utlilizes a kernel trick which adds nonlinearity with little computational overhead.

From Figure 14 the hyperplane which causes the maximum separation is chosen. This utility of separating hyperplane can be used to identify subtle changes between the malware families as discussed in [25]. The mathematical proof can be found in [26] Even if the dataset is not linearly separable SVM can map it to a higher dimensional subspace where it is possible to separate the classes.From Figure 15 the nonlinear data is separated into a higher dimension space where it is possible to find a separating

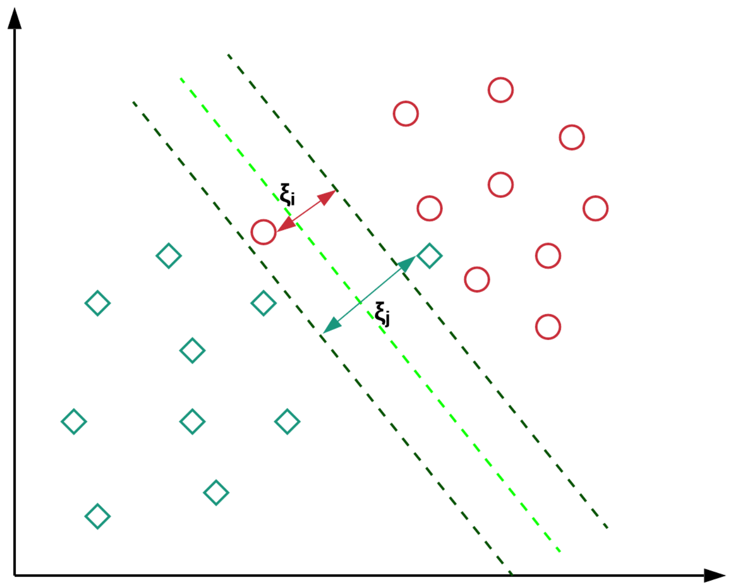


Figure 14: SVM hyperplane separating Green boxes from red circles [24]

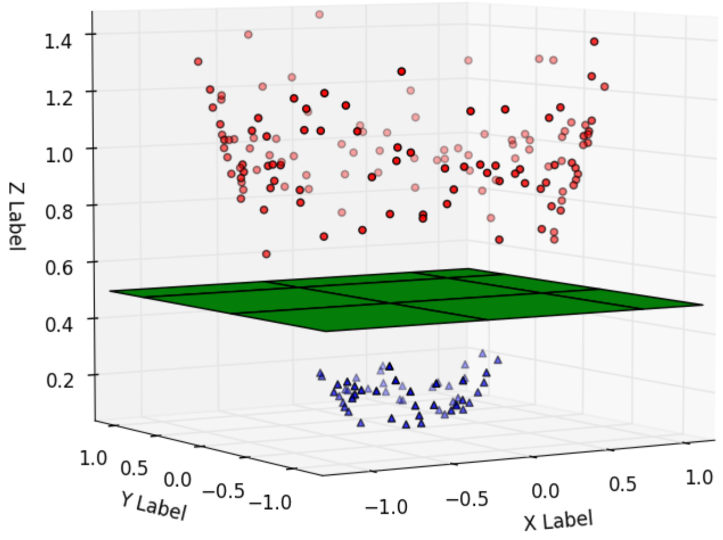


Figure 15: 3D seperating Hyperplane [27]

hyperplane.While training an SVM classifier we need to tune the hyper-paramter C which is a cost or regularization parameter.Thus parameter allows certain classes into

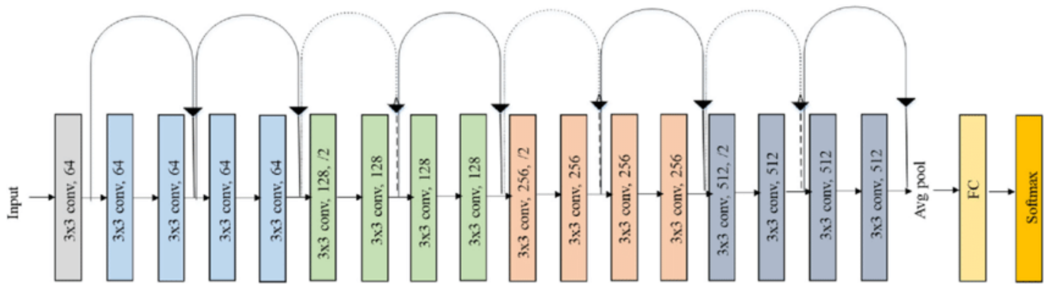


Figure 16: Resnet18 architecture [29]

the separating boundary and thereby preventing overfitting. Several popular nonlinear kernels like RBF and polynomial kernal can be used to deal with nonlinearity.

Algorithm 2 Support Vector Machine Algorithm

```
Require: $X$ and $y$ loaded with training labeled data,$\alpha \leftarrow 0$ or $\alpha$ partially trained
    SVM
1: $C \leftarrow$ some value
2: repeat
3:     for $\{x_i, y_i\}$,$\{x_j, y_j\}$ do
4:         Optimize $\alpha_i$ and $\alpha_j$
5:     end for
6: until no changes in $\alpha$ or other resource criteria is met
Ensure: Retain only the support vectors ($\alpha_i > 0$)
```

3.3.4 CNN

Deep neural networks are a class of neural network algorithms which are loosely modeled on the structure of the human brain. Even though CNN's are heavily used for image analysis, they can be used for malware classification as seen in [28]. The CNN architecture used in this research is Resnet18 which was introduced by researchers of Microsoft in [1]. The general architecture of resnet18 is described in the Figure 16. Resnet18 architecure uses residual connection which helps in mitigating the vanishing gradient problem.The residual operation is described in the Figure 17.The Identity mapping in the residual connection (figure 17 helps in backpropogation to reach from

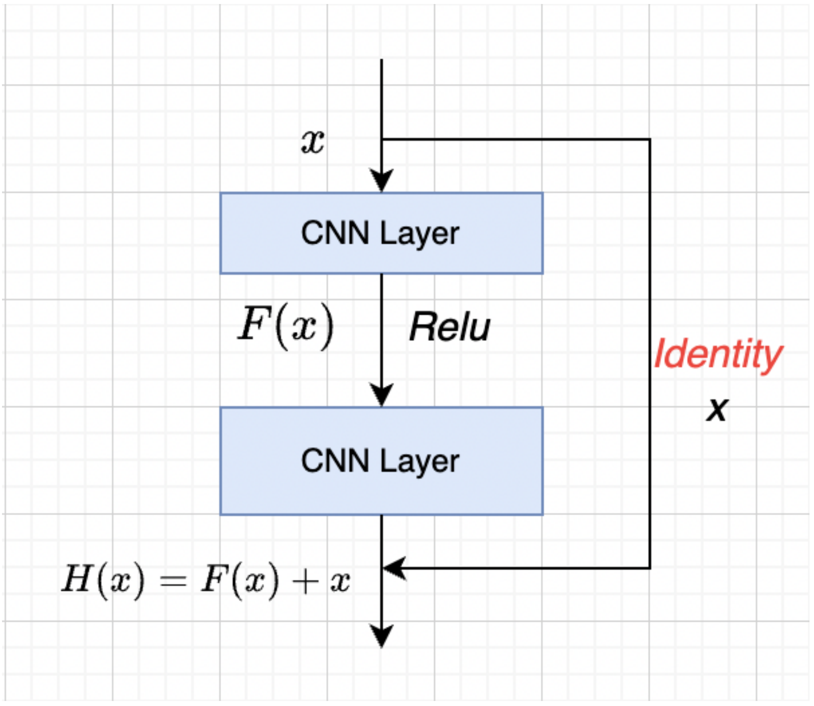


Figure 17: Residual connection introduced in [1]

the last layer of the network to the first layer.

$$H(x) = F(x) + x$$

This helps in approximating the residual function $F(x)$ while training our deep convolutional network.Even if $F(x)$ becomes zero it will at least learn x which will prevent the gradient from vanishing. The model also does not add any additional parameters which need to be trained as the identity function is always present in the network.This helps in stacking more convolutional layers while still be less complex and train better than VGG [1].

3.4 N-grams embeddings

Byte N-grams are one of the most readily available features for static analysis of malware. By treating a file as a sequence of bytes one can extract unique combination of N-gram bytes as a sequence of features. They are particularly attractive since they require no domain knowledge of virus or malware and anyone can easily convert them into features. These embeddings can then be converted into features using various embeddings techniques like Fasttext and Word2vec. In this research we explore using word2vec and fasttext for embedding generation of the byte-level Ngram combinations.

We only consider top 40 bi-grams for this research because of the memory and computational constraints for Converting them into features.

3.4.1 Word2Vec

Word2Vec introduced by Mikolov *et.al* in [11] is an algorithm that can efficiently create word embeddings. Prior to Word2Vec the embeddings were represented using one hot encodings which were computationally inefficient and require a lot of space resuting in "curse of dimensionality". Other than being extremely complex to train these embeddings will be closely coupled with their applications and adding or removing words from the vocabulary would require re-training of the whole model. Word2Vec can be used to map the words in the input sequence into vectors of higher dimension. Given enough data, usage and contexts the algorithm can make highly accurate guesses about its meaning based on its frequency and its association with neighboring words. This helps in creating intuitive word embeddings that can capture distributional semantics of the word.

Word2Vec trains two algorithms for creating the embeddings for the words

- **CBOW or Common Bag Of Words**: In this algorithm model the distributed representations *i.e* context are combined to predict the word in the middle(often called focus word). For *e.g* we want to predict what word comes in the blank "The ____ sat on his throne" (answer king).
- **Skip N-gram**: In this algorithm the model takes the focus word and tries to predict what are the neighboring words which can be associated with it. The

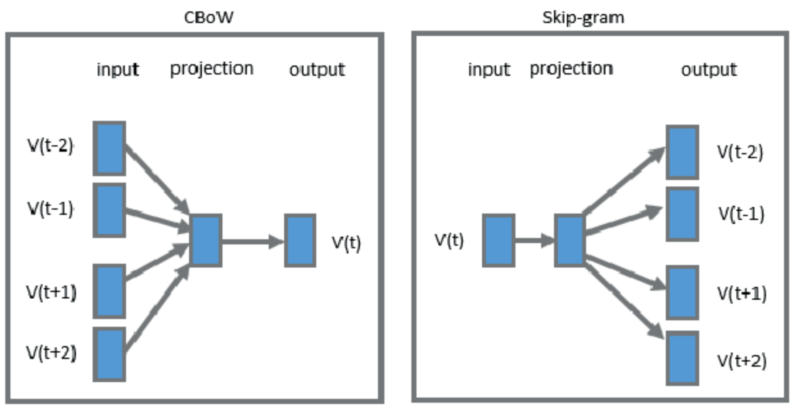


Figure 18: CBOW and Skip gram model [11]

equation for skip gram algorithm is given as follows

$$\log(p(\overset{\text{Context word}}{c} \,|\, \overset{\text{word}}{w}\,; \theta)) = \frac{\exp(\overset{\text{Context vector}}{v_c} \cdot \underset{\text{Word Vector}}{v_w})}{\sum_{c' \in C} \exp(v_{c'} . v_w)} \tag{12}$$

Since the vocabulary can be quite large running the skip gram model for large number of task increases the complexity exponentially. In order to overcome this issue the authores proposed subsampling frequent words to reduce training samples and using negative sampling which only reduces small part of the network while training.

3.4.2 Fasttext

While Word2Vec provides semantic representation of words, it cannot incorporate words which are out the training vocabulary. For *e.g* it there is no word like "tennis" in the vocabulary it will assign a random vector representation and cannot incorporate its complete meaning. Fattext circumvents this problem by using a bag of character level N-grams where each word is represented by sum oof its N-grams [30]. This helps in modelling dataset which is morphologically rich for *e.g* the word "tichtennis" in German and table tennis have same meaning and are represented close to each other using fasttext. Since creating and storing N-gram representation for each word is memory intensive the authors in [31] use a hashtable to store these vectors.Using a

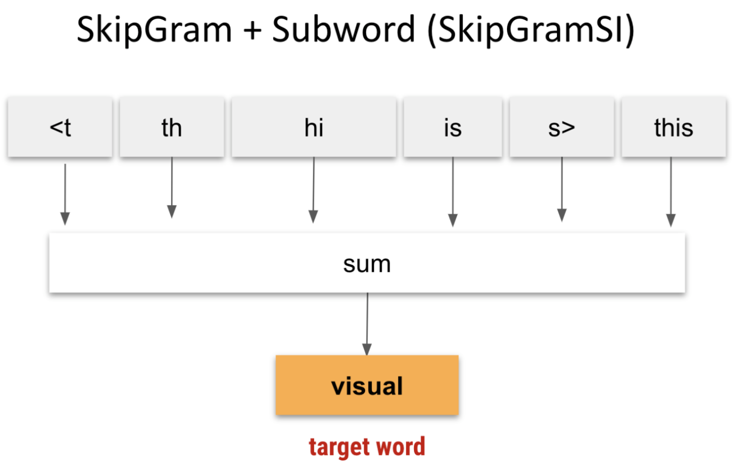


Figure 19: Skip gram model for fasttest [31]

quantization technique they further optimize the model using softmax approximation and keeping only the important N-grams in the hashtable [32].

CHAPTER 4

Datasets and Experiments

The dataset used in the experiments consist of seven types of malware families. The description of each type of malware families are as follows:

1. **CeeInject** : is a malware which can evade detection and hence various families use it for concealing their identities.For *e.g* It can install a spyware on your system by concealing itself as a program from the internet [33]

2. **BHO**: A Browser Helper Object which can redirect the victim to perform malicious actions as intented by the attacker

3. **FakeRean**: Generate and creates alerts for viruses that do not exist in the system and asks user to pay for removing the viruses.

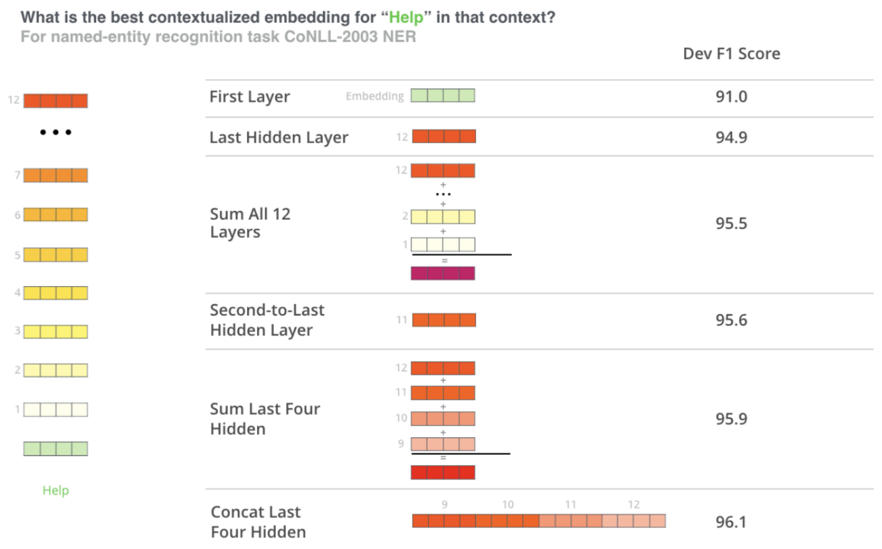
4. **OnlineGames**: Tracks login activity and monitors keystroke activity of online games without consent.

5. **Winwebsec**: Is a trojan malware which masquerades as a legitimate antivirus program and scares the user by creating misleading messages about the system. It then demands money to clean the system.

6. **Renos**: Informs user about fake security warnings by claiming that the system has spyware and then claim payments for removing the nonexistent spyware

7. **Vobfus**: Downloads other malware into the computer and then makes changes to the system which can't be resolved by removing the downloaded malware.

The dataset consists of exe files from which we extract mnemonic opcodes using Objdump. A bash script was run for all the files to convert it into asm format from which we extract opcodes.The opcodes used in this research are in the intel-86 format for compatibility. For each file first 400 tokens of opcodes are extracted and fed to transformer(BERT models) and embeddings are generated from them. The embedding size for all BERT models is 768. Although BERT can handle only upto 512 tokens the accuracy of the model was not impacted at all. From figure 20 we can see that summing up last 4 layers gives us maximum accuracy for text classification. We stack



What is the best contextualized embedding for "Help" in that context?
For named-entity recognition task CoNLL-2003 NER
Dev F1 Score
First Layer
Embedding
91.0
Last Hidden Layer
94.9
Sum All 12 Layers
95.5
Second-to-Last Hidden Layer
95.6
Sum Last Four Hidden
95.9
Concat Last Four Hidden
96.1
Help

Figure 20: Concatenating Last four Layers of Bert Architecture [17]

the tuple and extract the 512x768 embeddings(512 tokens,768 hidden layer size) from each of the layer. Then out of all the layers the token embeddings of last 4 layers are summed up and considered as the representation of all the tokens in the opcode sequence.

4.1 Experiments

This section describes the results of the experiments conduucted on the embeddings generated by different models.

4.1.1 BERT-based-experiments

As discussed in section 3.1.1 BERT is a transformer based encoder model which can generate contextual word embeddings owing to its self attention mechanism.

We use BERT base architecture which contains 110 million parameters. The hyper-parameters utilized for training of BERT are as follows:

- AdamW Optimizer with learning rate of $2^{(e-5)}$ to optimize the training.
- get_linear_schedule_with_warmup which reduces the learning rate to reduce the impact of new data pont on the model. Then the learning rate rate increases slowly if the minima is not found in order to reach convergence

Table 2: Optimal Hyper-Parameters for classifiers for BERT

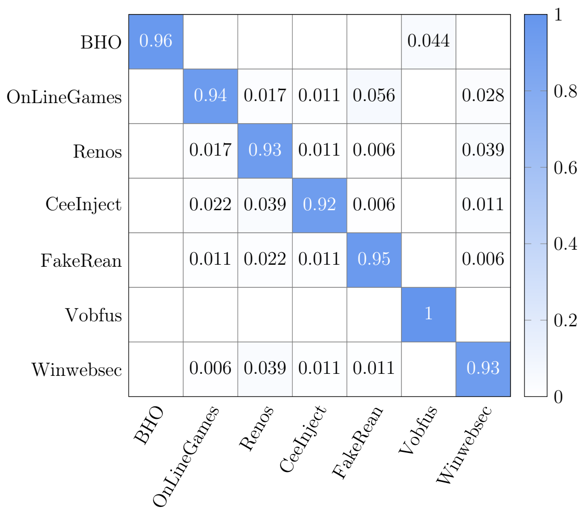
classifier	Parameter	Values
Random Forest	n-estimators	536
	max_depth	50
SVM	C	45572384.436680615
	γ	Auto
	kernel	RBF
kNN	k	25

The BERT model was trained for 20 epochs and the embeddings for the last 4 layers were extracted as shown in 20. The results of the experiments are shown in Figure 21 These experiments suggest BERT-SVM gives the maximum accuracy of all the models.The worst performance is By kNN which gives about 92% accuracy.

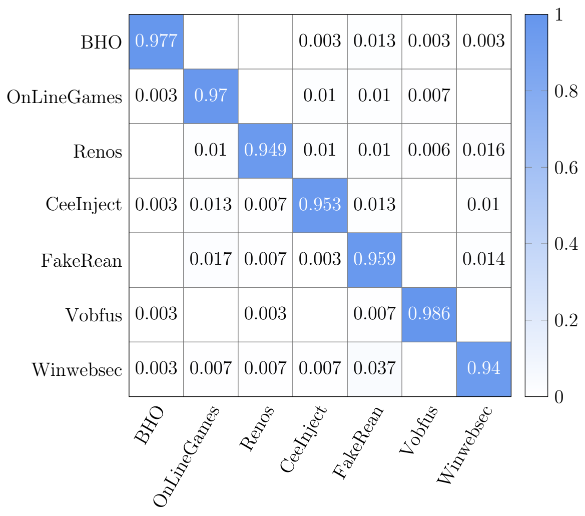
4.1.2 distilBERT based Experiments

As mentioned in section 3.1.2 distilBERT is a distilled or smaller version of BERT architecture which uses know distillation to shrink down the BERT model. As in section 4.1.1 we train the model on the opcodes for 20 epochs with same hyperparameters we used for BERT model and extract the embeddings by summing up the last 4 layers of the model.

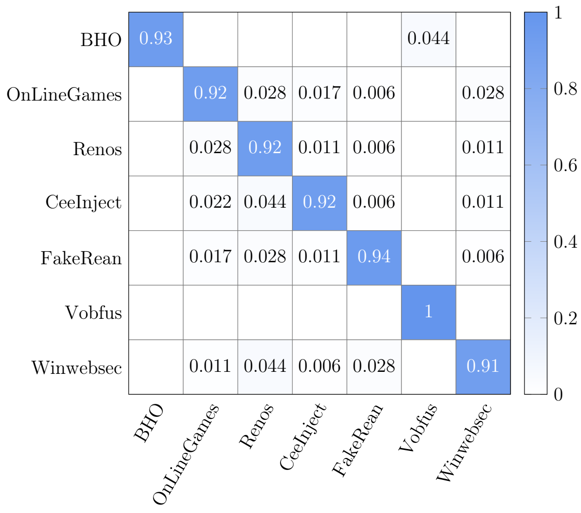
The embeddings are then fed to the classifiers and the results are displayed in figure 22. The distilBERT random forest classifier shows the highest accuracy while the kNN one shows the lowest accuracy.A random forest model uses an ensemble of



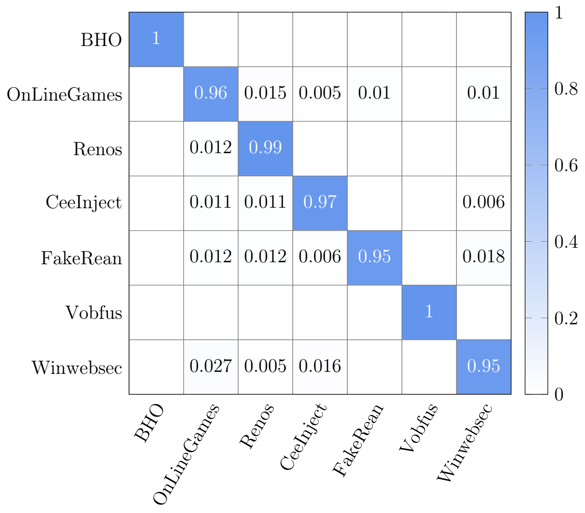
(a)BERT-RandomForest Matrix



(b) Bert-SVM confusion matrix



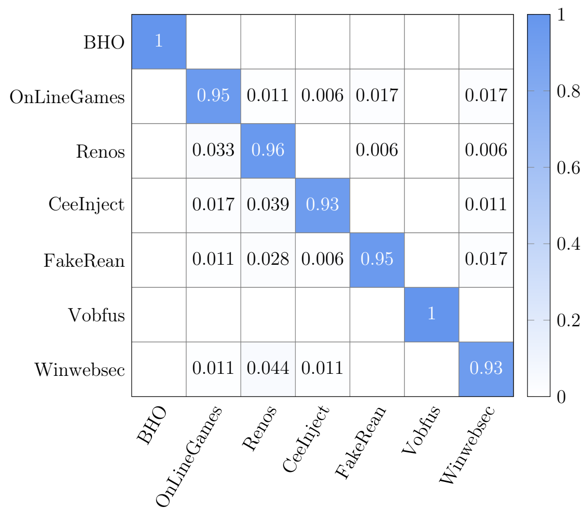
(c) BERT-kNN confusion matrix



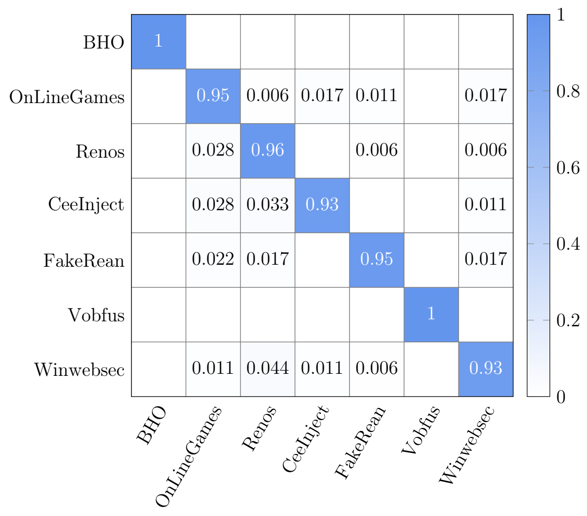
(d) BERT-CNN confusion matrix

Figure 21: BERT-based experiments

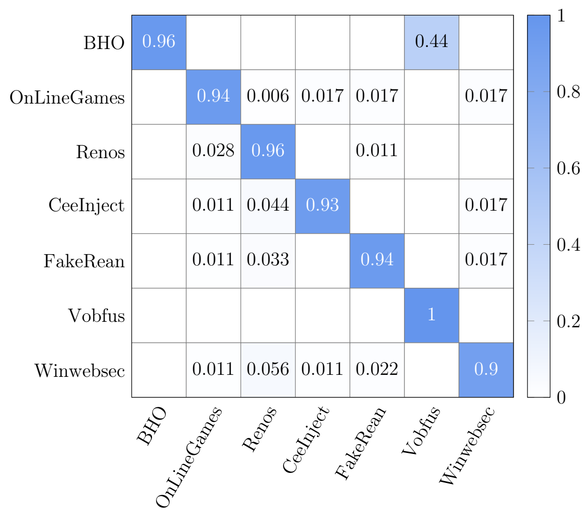
decision tress and uses a voting classifier to predict the class of the input. Because of its bootstrapping mechanism it is able to generalize well to the embeddings generated from the model. SVM classifier gives the best output when the C parameter is 6.8 which means the model tries to regularize simpler weights in the model. Hyperparameters for distilBERT-based classifiers are shown in Figure 3.



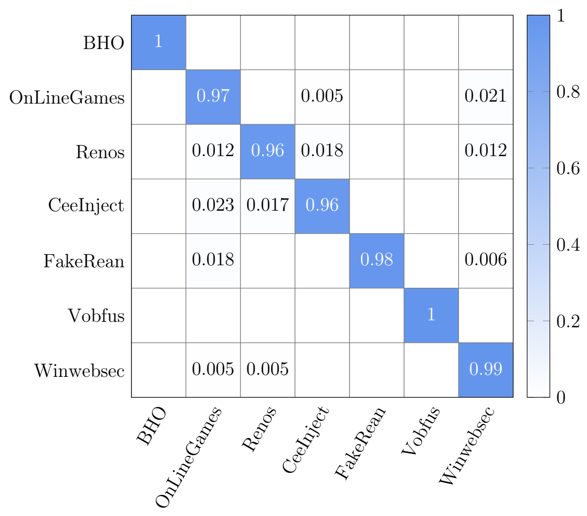
(a)distilBERT-RandomForest Matrix



(b) distilBert-SVM confusion matrix



(c)distilBERT-kNN confusion matrix



(d)distilBERT-CNN confusion matrix

Figure 22: distilBERT-based experiments

4.1.3 Roberta based Experiments

As mentioned in section 4.1.3 Roberta is more robust BERT model which is trained on significantly large dataset and uses a byte level tokenization in order to incorporate words which are not part of the input vocabulary. It also utilizes a large corpus of text to finetune its parameters and achieve state of the art in almost 11 (natural language processing challenges) [21].

Table 3: Optimal Hyper-Parameters for classifiers for distilBERT

classifier	Parameter	Values
Random Forest	n-estimators	266
	max_depth	32
SVM	C	2021198652.6540103
	γ	Auto
	kernel	RBF
kNN	k	25

As with the previous section a Roberta model is trained for 20 epochs and the embeddings are fed to the classifiers. The results of Roberta model are shown in figure 23

The optimal hyperparameters for Roberta based classifiers are as follows

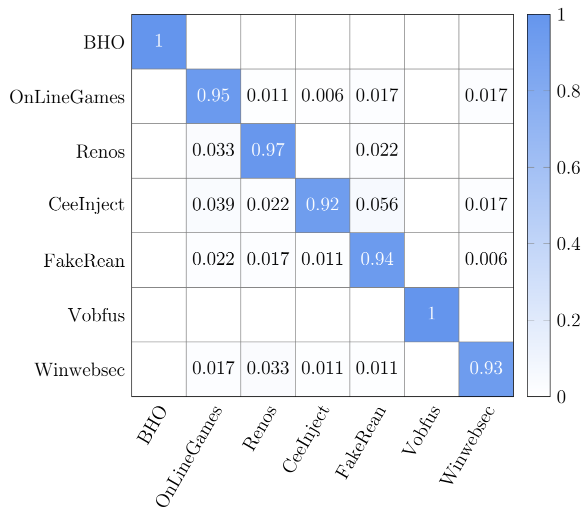
Table 4: Optimal Hyper-Parameters for classifiers for Roberta

classifier	Parameter	Values
Random Forest	n-estimators	292
	max_depth	42
SVM	C 4.430007444654968	
	γ	scale
	kernel	RBF
kNN	k	25

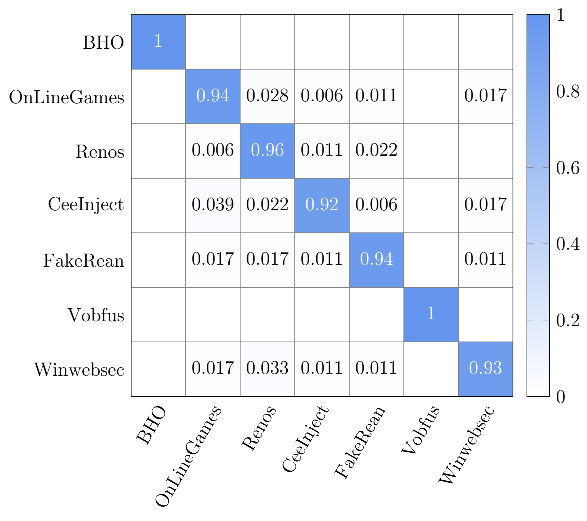
4.1.4 Albert based Experiments

Authors in [22] stated that the albert model has considerably less parameters than BERT(almost 89% parameter reduction) and therefore takes less time for fine-tuning. However even training for smaller batches on the malware dataset takes considerable amount of time and the model actually performs worst among all the BERT based models.

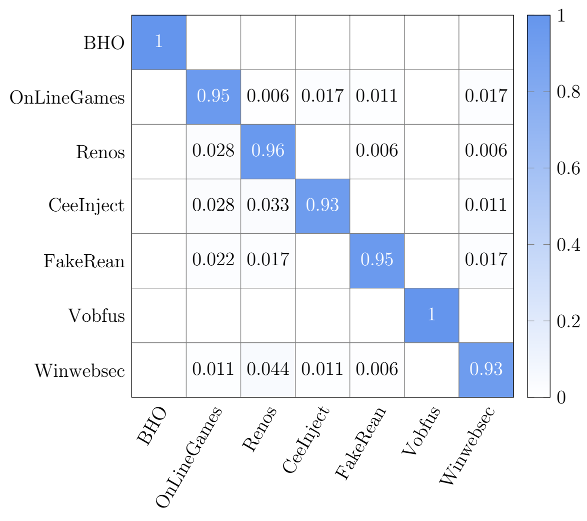
As with the previous section the Albert model is trained for 20 epochs and the embeddings are fed to the classifiers. The results of Albert model are shown in figure 24. Each encoder in the Albert model shares the parameter where the first encoder



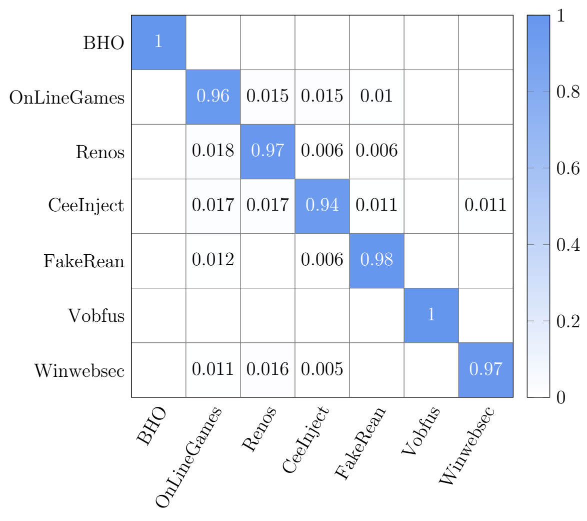
(a)Roberta-RandomForest Matrix



(b) Roberta-SVM confusion matrix



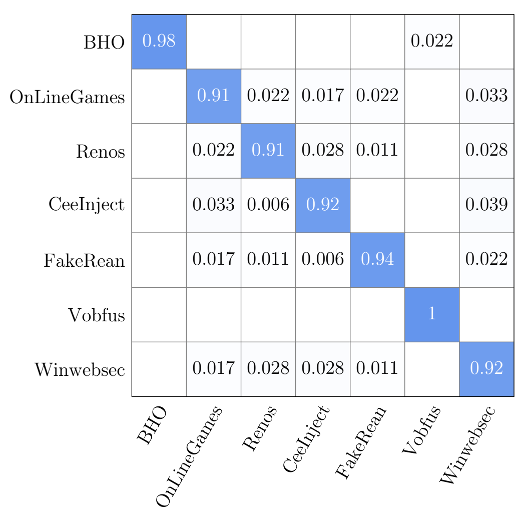
(c)Roberta-kNN confusion matrix



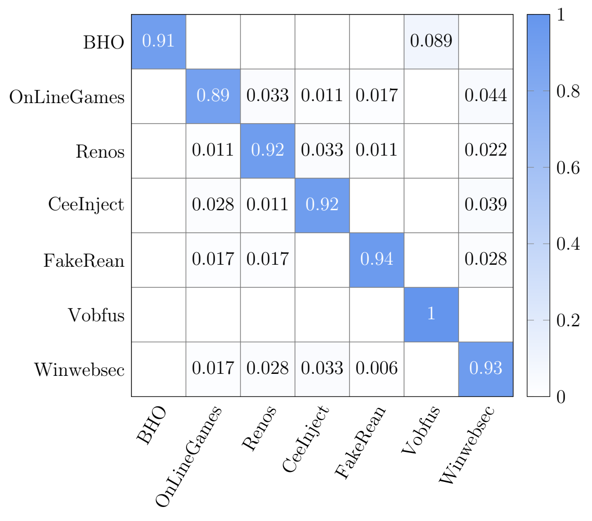
(d)Roberta-CNN confusion matrix

Figure 23: Roberta-based experiments

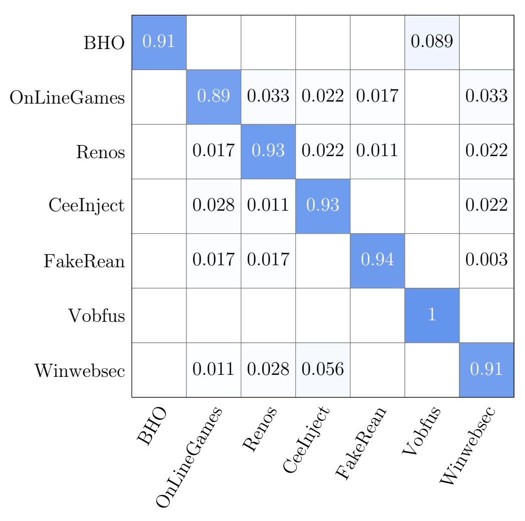
has its weights repeated 12 times which degrades the model performance as compared to BERT. The hyperparameters for Albert-based classifiers are shown in table **??** It is unclear whether embedding factorization in Albert model actually reduces parameters as from the inference on the model it took longer as compared to other BERT models.



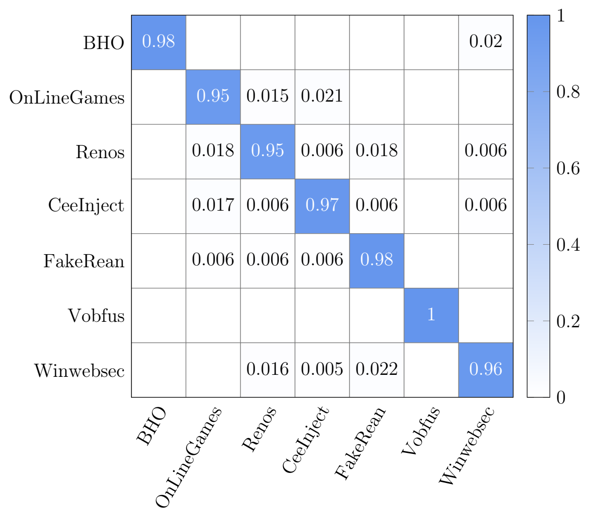
(a)Albert-RandomForest Matrix



(b) Albert-SVM confusion matrix



(c)Albert-*k*NN confusion matrix



(d)Albert-CNN confusion matrix

Figure 24: Albert-based experiments

4.2 ELMo based Experiments

As seen in section 3.2 ELmo uses a biLM model which constructs a context aware representation of input Sequence. The model used in this research is provided by Allennlp which are the main maintainers of this architecture.

We extract only the first 64 tokens from each file because of the computational overhead (64*256) and concatenate each tensor to get the embedding representation of

Table 5: Optimal Hyper-Parameters for classifiers for Albert

classifier	Parameter	Values
Random Forest	n-estimators	346
	max_depth	17
SVM	C	5.815004360060932
	γ	Auto
	kernel	RBF
kNN	k	25

length 256 for each file. The embeddings are then fed to the classifiers and results can be seen in Figure 25. The hyperparameters for the experiments are given in Figure 6

Table 6: Optimal Hyper-Parameters for classifiers for ELMo

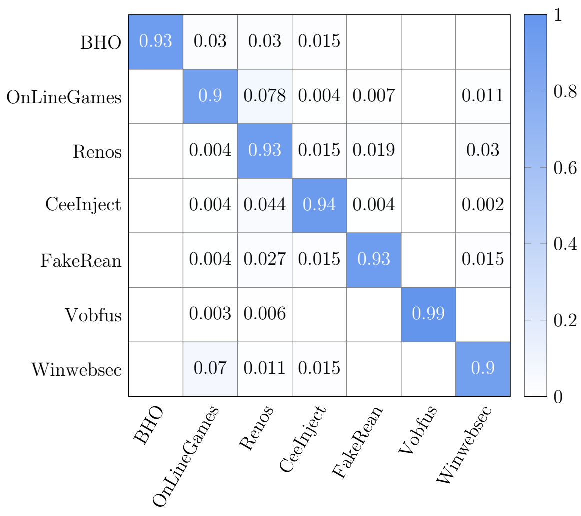
classifier	Parameter	Values
Random Forest	n-estimators	26.9
	max_depth	20
SVM	C	194.45
	γ	Scale
	kernel	RBF
kNN	k	90

The combined accuracies for all the bert models and their classifiers is described in Figure 26

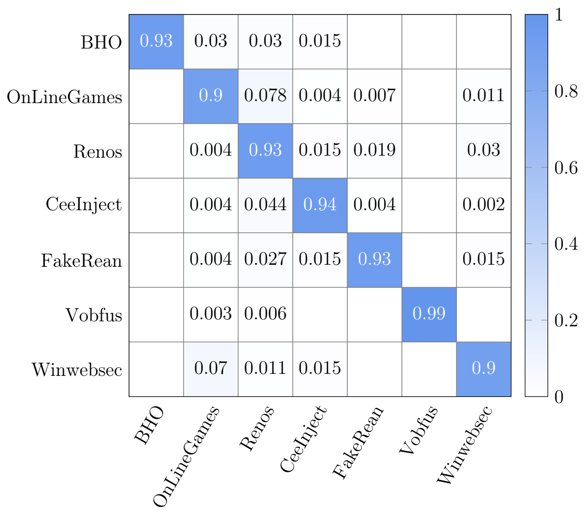
From Figure 27 we can see that the BERT based embeddings consistently outperform the WOrd2Vec and HMM2Vec based embeddings.

4.3 Byte-Level N-gram experiments

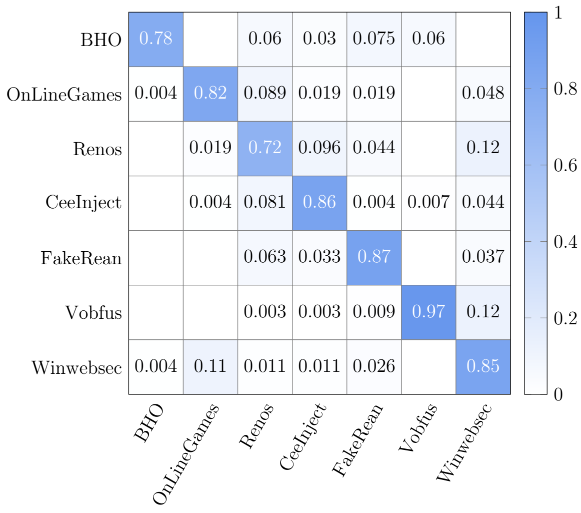
Byte-level N-grams are one of the most common features for static analysis since they require no domain knowledge and are easiest to extract and process. In this research we generate an embedding of byte level N-grams by using Word2vec and fasttext and compare their accuracy (Section 3.4.2).



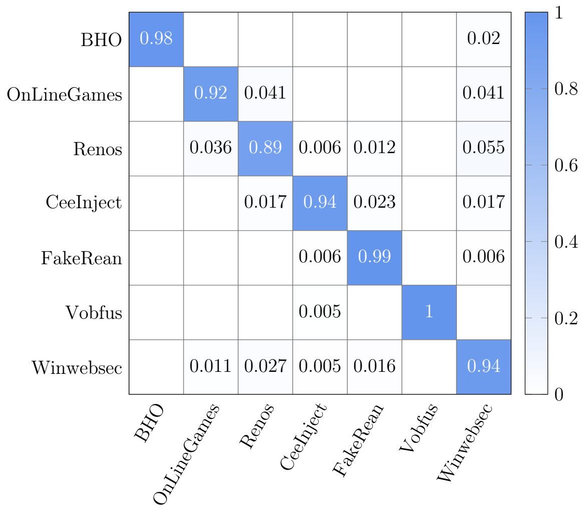
(a)ELMo-RandomForest Matrix



(b) ELMo-SVM confusion matrix



(c) ELMo-*k*NN confusion matrix



(d) ELMo-CNN confusion matrix

Figure 25: ELMo-based experiments

4.3.1 Word2Vec

The byte level sequences are extracted from the malware files and are treated as strings. These sequences are then fed to Word2Vec model with a context window of 10 to generate the byte level embeddings. These embeddings are then fed to various classifiers as mentioned in section 3.3.

The general algorithm is as follows (algorithm 3. We feed the byte level N-grams

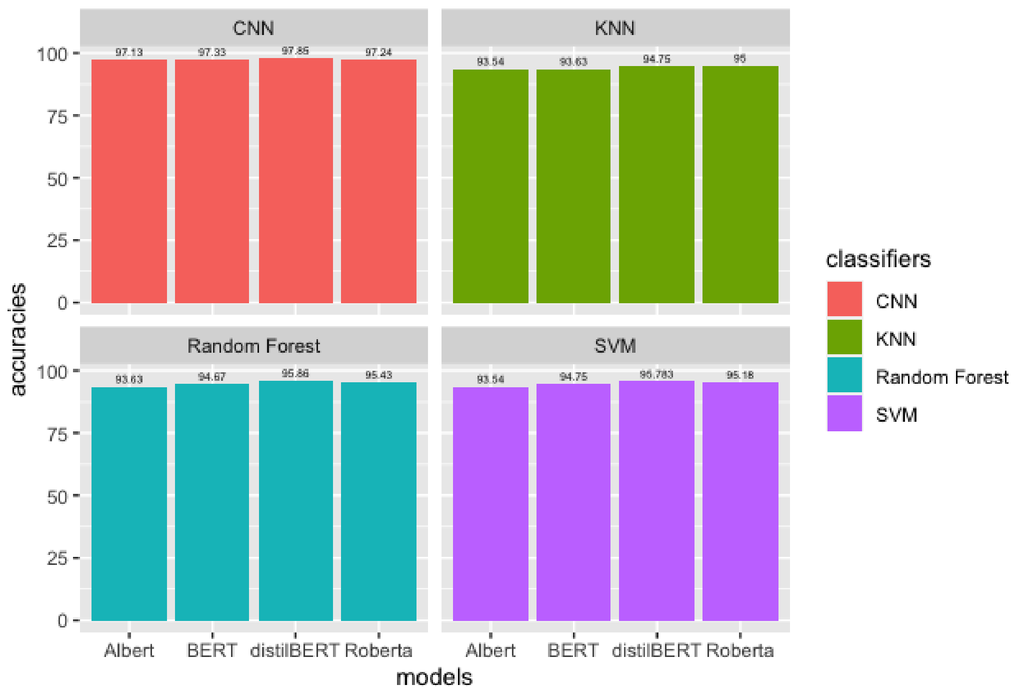


Figure 26: Accuracies for various classifers for BERT model

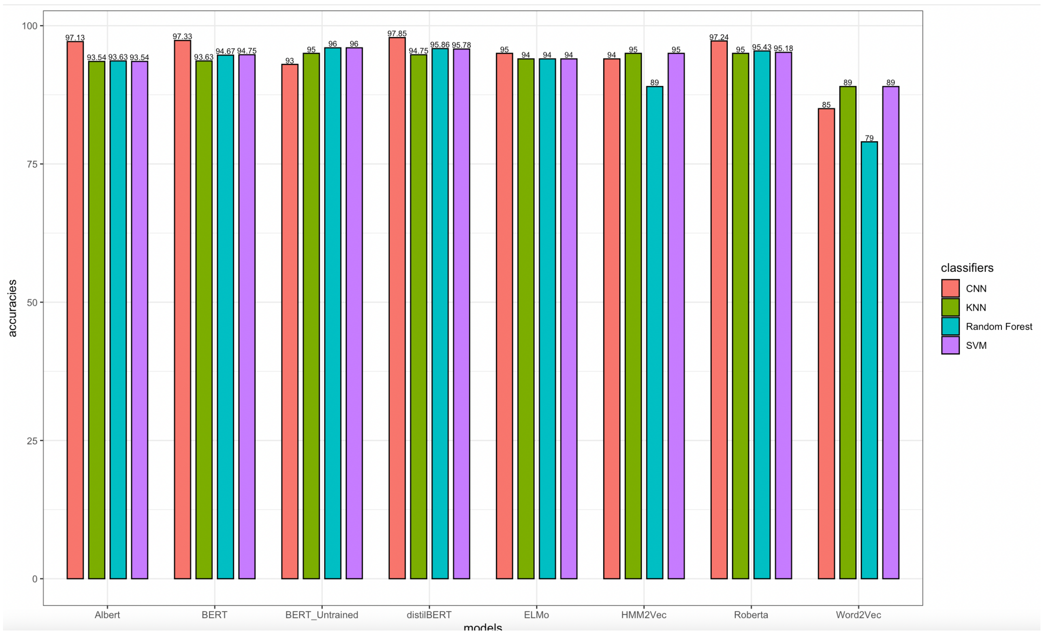


Figure 27: Comparing Transformer based models accuracies with other models

to the word2vec algorithm which generates the embeddings for the malware file. Since there can be a large number N-gram combination of bytes, we only take top 20 most common N-grams and runa Word2Vec algorithm over it . Each byte N-gram will have a vector of size 5 making the vector size of (100) for each malware file.The details of the algorithm are given in the algorithm 3. The embeddings are fed to classifiers and their results are shown in Figure 28

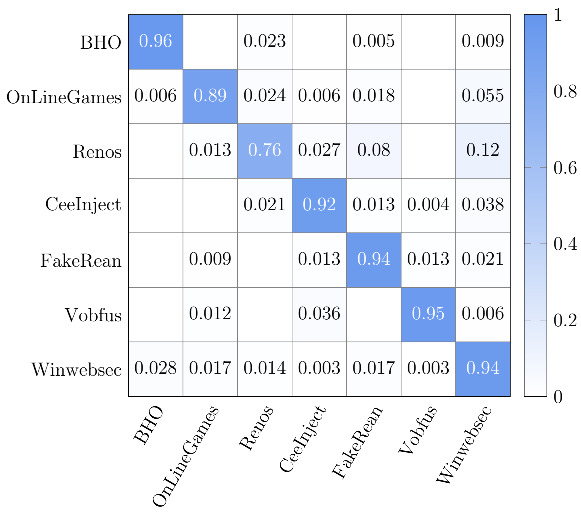
Algorithm 3 Word2Vec Algorithm

```
Require: Byte-Level N-grams for the malware files. Data-Structure Embeddings for
    storing embeddings for malware files
 1:
 2: function Word2VecEmbeddings(File)
 3:     $V_c \leftarrow$ Word2Vec(file,$(Vector\_size) = 5$,$window = 10$,$min\_count = 5$)
 4:     $V_{c'} \leftarrow$ top 20 most common features in $V_c$
 5: return $V_{c'}$
 6: end function
 7: for $File \in Malware\ files$ do
 8:     $features \leftarrow$ Word2VecEmbeddings($File$)
 9: end for
10: Feed the Embeddings to the classifiers
```

The hyperparamters tuning for all the classifiers is done using optuna and their values are mentioned in the table 7. From the confusion matrix we can see that the random forest performs best as compared to other classifiers.

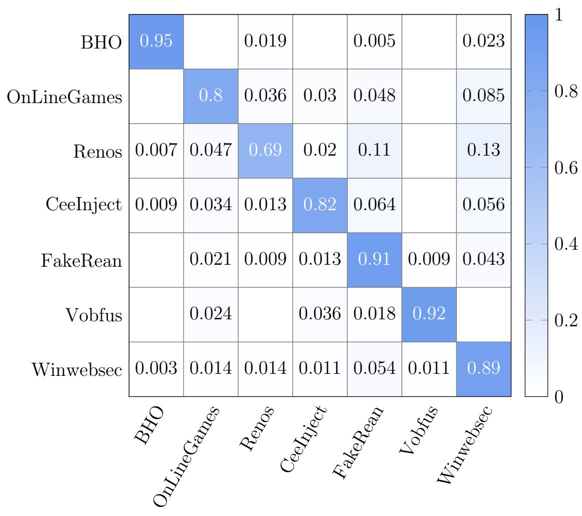
Table 7: Optimal Hyper-Parameters for classifiers for Word2Vec

classifier	Parameter	Values
Random Forest	n-estimators	726
	max_depth	20
SVM	C	1181.45
	γ	Auto
	kernel	RBF
kNN	k	90



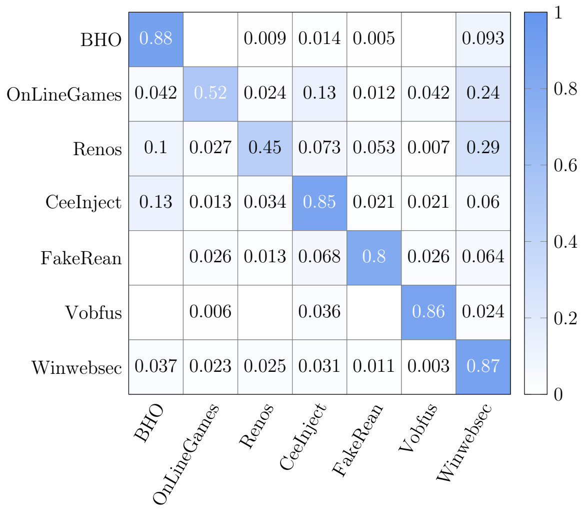
	BHO	OnLineGames	Renos	CeeInject	FakeRean	Vobfus	Winwebsec
BHO	0.96		0.023		0.005		0.009
OnLineGames	0.006	0.89	0.024	0.006	0.018		0.055
Renos		0.013	0.76	0.027	0.08		0.12
CeeInject			0.021	0.92	0.013	0.004	0.038
FakeRean		0.009		0.013	0.94	0.013	0.021
Vobfus		0.012		0.036		0.95	0.006
Winwebsec	0.028	0.017	0.014	0.003	0.017	0.003	0.94

(a)Word2Vec-RandomForest Matrix



	BHO	OnLineGames	Renos	CeeInject	FakeRean	Vobfus	Winwebsec
BHO	0.95		0.019		0.005		0.023
OnLineGames		0.8	0.036	0.03	0.048		0.085
Renos	0.007	0.047	0.69	0.02	0.11		0.13
CeeInject	0.009	0.034	0.013	0.82	0.064		0.056
FakeRean		0.021	0.009	0.013	0.91	0.009	0.043
Vobfus		0.024		0.036	0.018	0.92	
Winwebsec	0.003	0.014	0.014	0.011	0.054	0.011	0.89

(b) Word2Vec-SVM confusion matrix



	BHO	OnLineGames	Renos	CeeInject	FakeRean	Vobfus	Winwebsec
BHO	0.88		0.009	0.014	0.005		0.093
OnLineGames	0.042	0.52	0.024	0.13	0.012	0.042	0.24
Renos	0.1	0.027	0.45	0.073	0.053	0.007	0.29
CeeInject	0.13	0.013	0.034	0.85	0.021	0.021	0.06
FakeRean		0.026	0.013	0.068	0.8	0.026	0.064
Vobfus		0.006		0.036		0.86	0.024
Winwebsec	0.037	0.023	0.025	0.031	0.011	0.003	0.87

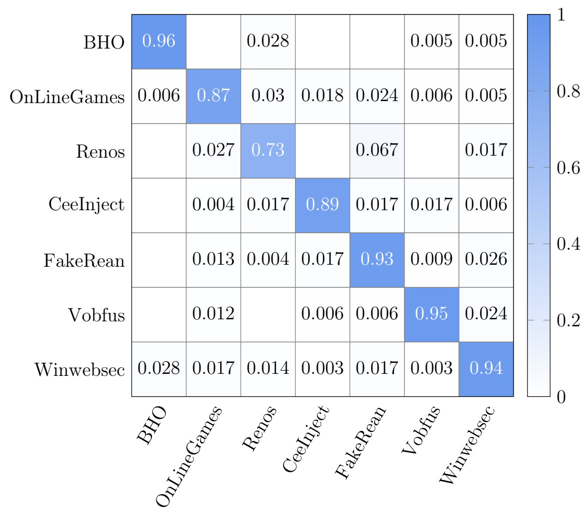
(c) Word2Vec-*k*NN confusion matrix

Figure 28: Word2Vec-based experiments

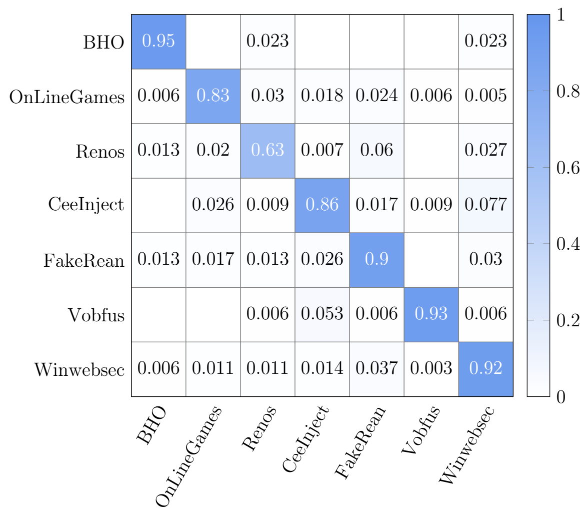
4.3.2 FastText Based Experiments

In this set of experiments we consider the same top 20 combinations of Word2Vec N-grams are fed to an fasttext model. Since FastText os considerably quicker than Word2vec [32] we use the vector size of 400 to represent a file. The embeddings generated are then fed to classifiers and their results are shown as below

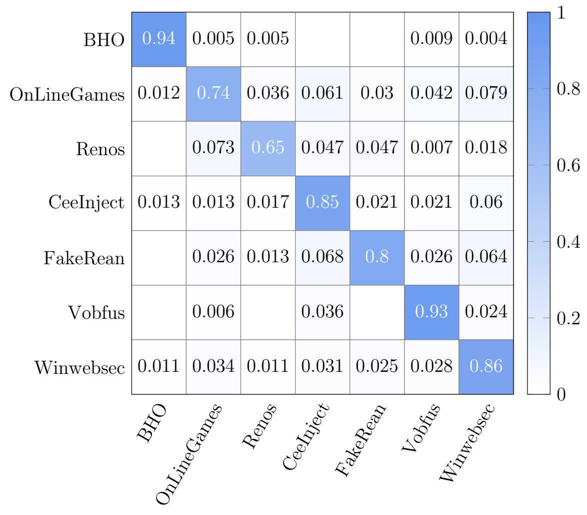
On comparing confusion matrices for Word2Vec (Figure 28) and FastText hybrid



(a)FastText-RandomForest Matrix



(b) FastText-SVM confusion matrix



(c) Fasttext-kNN confusion matrix

Figure 29: FastText-based experiments

classification (Figure 29) FastText is more robust, accurate and generalizes better.

Table 8: Optimal Hyper-Parameters for classifiers for Word2Vec

classifier	Parameter	Values
Random Forest	n-estimators	186
	max_depth	42
SVM	C	170685.38129805663
	γ	Auto
	kernel	RBF
kNN	k	20

CHAPTER 5

Explainable AI and Interpretability of BERT Models

In recent years, machine learning applications have found wide spread use in almost all aspects of life. AI based algorithms have been successfully applied to almost all types of data (sound,image,tabular,forecasting,speech,text) and have become ubiquitous in every kind of industry [34]. A large factor for its progress is the emergence of deep learning algorithms. With the help of these algorithms data can be used in more creative ways to gain analytical insights,recommend movies, facial recognition. With more advances in deep learning, the models have become increasingly complex and hard to interpret. For example the Large Langauge Model (LLM) released by google called Gopher consists of 280 Billion parameters [35]. With their large size and increasingly complex architecture understanding how and what the model predicts has become incomprehensible. Because of their complexity, many AI models are treated as "black boxes" without elaborating how the prediction was obtained.

Since machine learning models are being deployed in various high risk applications, an explanation of how the model predicted an outcome is crucial for ensuring trust and reliability. For example consider a model which predicts pneumonia based on the CT scan and features of the patient, we need to be 100% sure what the model has predicted is correct and how it came to a conclusion. Machine learning models are not perfect and often pick up biases from the training data. This can severely affect machine learning models and discriminate against under-represented groups. Interpreting how models predict can help in detecting and debugging such biases and also help in creating trust in the AI model [36]. If the model is more interpretable it is easier for humans to understand why and how certain decisions are made (Figure 30).

The easiest way of explaining models is using models that are already interpretable such as decision tree, linear regression and random forest [36]. Since the inner workings

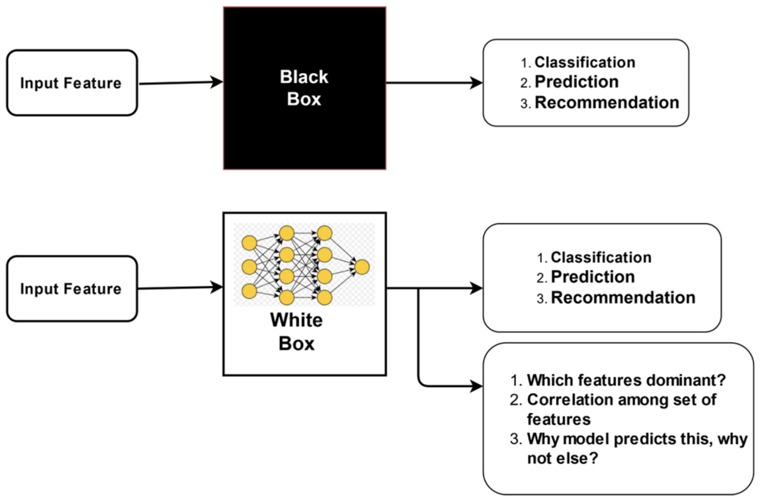


Figure 30: AI vs XAI [34]

and the math behind these models are known and rigorous proofs are given, it is easy to comprehend them. For *e.g* an increase in size for a property mostly results in increase in increase in price value if we are using linear regression. Thus we can say that interpretability for such models is intrinsic since it has mechanisms which are built-in for interpretability. The tools required for interpreting such models always depend on the type of model used [36] and hence they are model specific. Model agnostic methods allow us to interpret any kind of machine learning model after the model has been trained(post-hoc).

5.1 Model Agnostic methods for interpreting BERT model

This section will introduce some model agnostic methods for interpreting BERT models and give some explanations for its predictions.

5.1.1 SHAP values for Explanation

SHAP values or SHapely Additive exPlanations is a method to explain individual predictions [36]. Consider a team taking part in a Kaggle competiton and they get certain payout for their achieved result. For instance they get 10,000 dollars if they win

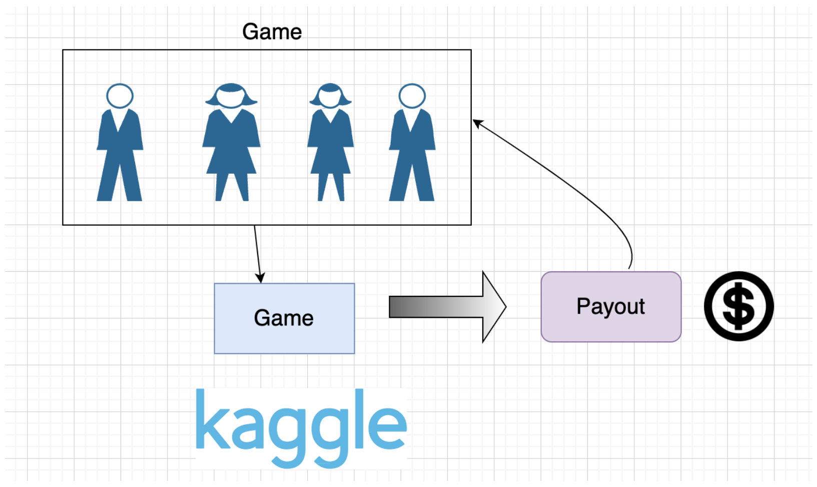


Figure 31: Shapely values for getting the payout

the first prize. The dilemma here is how to distribute money so that the distribution is fair. Since each player contributes differently distributing the money equally doesn't seem fair. The main goal of SHAP is to explain a prediction based on the contribution of features. Each feature can be assumed as a player in the game where the prediction is the payoff. Shapley values introduced in [37] is a method of assigning payouts depending on each player's contribution. Player's operate in coalition and receive some profit from this coalition. A shapely values describes average contribution of the player to the payout.

While explaining machine learning models using shapely values, the features are considered as players and the prediction as payout. Each shapely value of a feature describe its contribution for the prediction.SHAP(shapely additive explanations) use shapely values to explain a particular prediction [38]. The equation for shapely values is given in equation 13.

$$\phi_{\underset{\text{Feature value}}{i}}(\overset{\text{Black Box model}}{f}, x) = \sum_{\underset{\text{Subset}}{z'} \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} \cdot [\overset{\text{Without the feature score}}{f_x(z')} - f_x(z'i)] \tag{13}$$

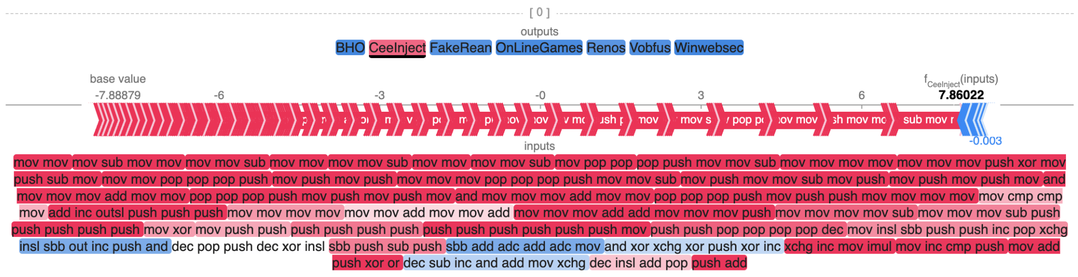


Figure 32: SHAP plot for a CeeInject Instance

The values for which we consider the feature to be absent are filled with random values as they do not contribute to the overall marginal accuracy of the payoff [36]. From 13, there are 2^n possible combinations of subsets which can take up a lot of memory .In order to prevent this and make the calculations somewhat manageable, [38] uses Kernel SHAP that uses a special weighted linear regression to compute the importance of each feature.

Consider an example of CeeInject virus. It would be beneficial to know what features the BERT transformer model considers important. This would not only help in visualizing the importance of each opcode but also help in detecting any defects within the model. Figure 32 indicates what features are important for a BERT based classifier when predicting the instance as CeeInject. It is clear that combinations of opcodes like (mov,mov) and (mov,mov,sub) contribute towards positive direction in the model predicting CeeInject whereas opcode combinations like (sbb,add,adc,add,adc,mov) pull the prediction in opposite direction. Since the number of combinations can be extremely large it becomes difficult and infeasible to calculate shapely values for all possible combinations. In order to counter this situation we only consider 20 unique opcodes in each instance and try to find combinations within them. The Owen values which are based on SHAP values use the concept of coalition and unions in order to make the calculations tractable. The base value of the instance is the average value of the model over the training set. Consider the same instance but for different label for *e.g* ""FakeRean" the opcode combination of (add,mov,mov,mov,add..) significantly push the model to not predict "FakeRean" while (push,xor,or) push the prediction in the opposite direction (Figure 33). They net result is the prediction of the model. The bar plot in Figure 34 shows the importance of each feature in the malware file. It

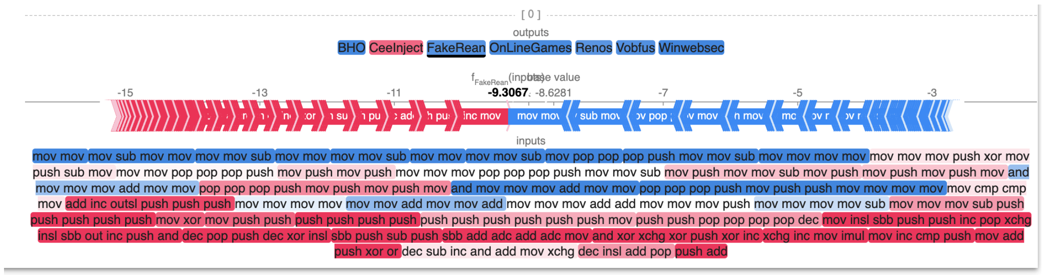


Figure 33: SHAP values for FakeRean

represents the mean absolute contribution of each feature(or combination of opcodes) in the prediction and is sorted from lowest to highest. The idea behind this is simple, the opcodes with largest absolute values are important since they tilt the prediction in either direction. The bar plot is useful since it helps in identifying important features for a prediction but beyond that they do not provide any further information.

The Waterfall plot shown in Figure 35a shows how features of single instance are added up to give the prediction of CeeInject. For the same instance but this time for different class ("BHO") the opcodes which influence the prediction are shown in Figure 35b. The plot reflects the same intuition which was observed from the bar plot(Figure 34) but it represents in a more elegant manner displaying how each feature was added up and their contribution towards their prediction. Shapely values can also be used for visualizing images . In Figure 36 consider an example of CeeInject malware which has been transformed into an image using BERT (Section 4.1.1). The model assigns high scores in the middle-left and slightly higher in the bottom half of

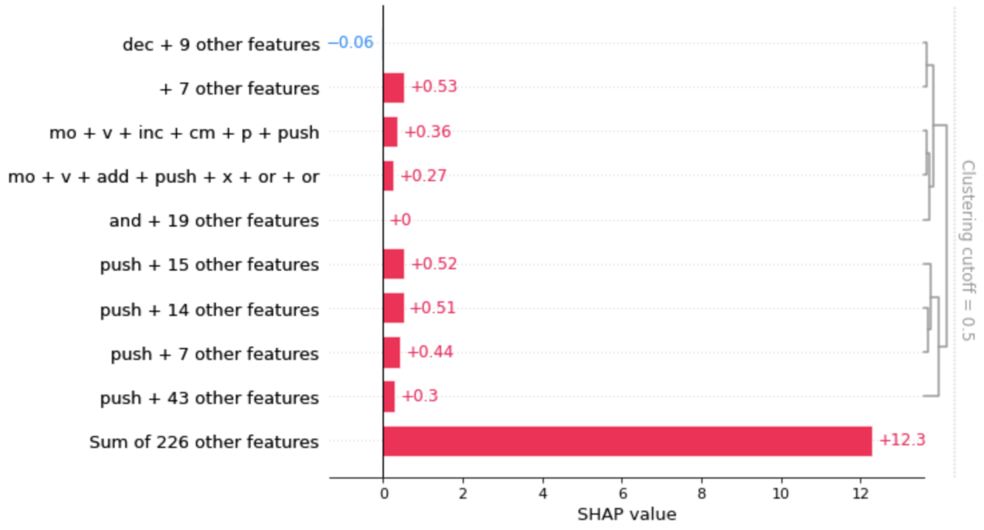
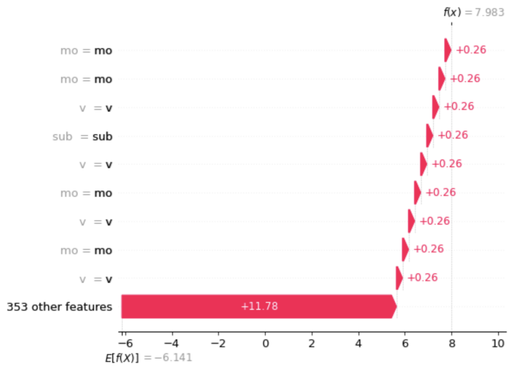
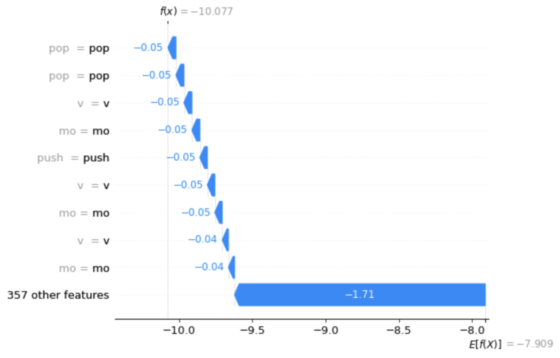


Figure 34: Bar plot for instance of "CeeInject"



(a) CeeInject features



(b) Same instance but plot for different class

Figure 35: Waterfall plot for comparing feature contributions

the image.

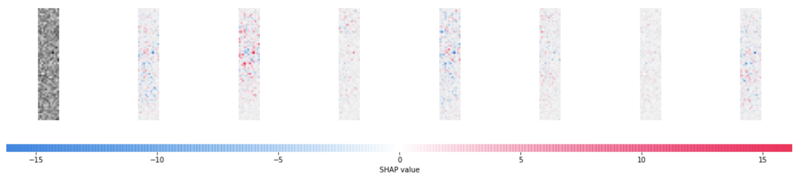


Figure 36: Shap values for CNN images

5.1.1.1 Advantages and disadvantages of Shapely values

Advantages of shapely values are as follows.

- Shapely values provide a uniform methodology to decompose a model's prediction into contributions that can be assigned to different features.
- They provide contrastive explanations which provide details on how the prediction is made.
- They can provide interpretation for almost any kind of model and provide consistent local explanations.

Disadvantages of Shapely plot

- Shap values take considerable amount of time if there are many features and computing feature importance for many instances requires significant computing power
- It is possible to generate misleading plots in order to hide the biases using shapely plots [36]
- When computing feature importance if the model used for prediction is not additive the values can be misleading.

5.1.2 Interpreting with LIME

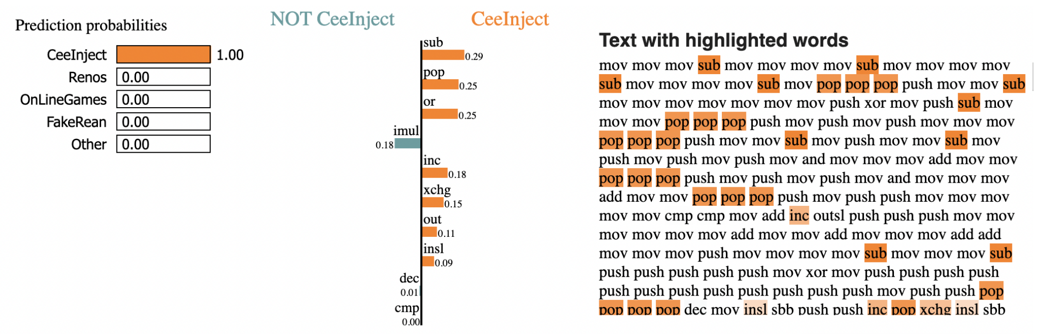
Local Interpretable Model Agnostic Explanation (LIME) is a model agnostic post-hoc method which computes a local approximates of a complex model to explain the prediction of a particular instance. The key idea behind LIME is to use simpler interpretable models like linear regression to approximate the output of complex models like BERT. It works on almost any kind of inputs like text,tabular or images. LIME aims to explain any complex black box model by creating local approximatioins for a single instance of the dataset. Thus the algorithm creates a local surrogate models to approximate an instance [39].

$$explanation(\overset{\text{Instance of dataset}}{x}) = \arg\min_{g\in \underset{\text{Interpretable models}}{G}} \overset{\text{Local Approximation}}{L(f,g,\pi_x)} + \Omega(g) \tag{14}$$

The equation in 14 is a minimizing equation where the goal is to find the find a function g which can provide good approximation of the model f while also staying as simple. Here "simple" means that which can be interpreted by a human. The proximity measure π_x is added to consider how far and wide the neighborhood around instance x should be while considering the model's prediction. The model complexity parameter $\Omega(g)$ is kept low so as to reduce the complexity of the model so that its remains intrinsically interpretable [36]. The general steps for using a surrogate model to explain a black box model are as follows

1. Select an instance from the dataset for explanation
2. Get perturbations for that instance and get black box predictions for the particular instance
3. Weight the models according to their proximity and points of interest
4. Train an interpretable model with its variations
5. Explain the predictions using the local simpler model

The perturbations for the text data are created by randomly removing certain words from the original text data. From the Figure 37 the features which are important and move towards positive direction for CeeInject are highlighted in yellow whereas the features which tilt the model away from predicting CeeInject are shown in green. The presence of opcodes like "sub","pop" and "or" are more prominent in CeeInject which gives an indication on how the model will predict given an unseen data.



Text with highlighted words

mov mov mov sub mov mov mov mov sub mov mov mov mov sub mov mov mov mov sub mov pop pop pop push mov mov sub mov mov mov mov mov mov mov push xor mov push sub mov mov mov pop pop pop push mov push mov push mov mov mov pop pop pop push mov mov sub mov push mov mov sub mov push mov push mov push mov and mov mov mov add mov mov pop pop pop push mov push mov push mov and mov mov mov add mov mov pop pop pop push mov push push mov mov mov mov mov cmp cmp mov add inc outsl push push push mov mov mov mov mov mov add mov mov add mov mov mov add add mov mov mov push mov mov mov mov sub mov mov mov sub push push push push push mov xor mov push push push push push push push push push push push push mov push push pop pop pop pop dec mov insl sbb push push inc pop xchg insl sbb

Figure 37: Importance of an instance of CeeInject class with Lime

CHAPTER 6

Conclusions and Future Work

As part of the experiments conducted in this research, the distilBERT-CNN gave the maximum accuracy of 97.24%. It was observed that both Roberta and distilBERT generate much better representation of opcodes than malware. The reasons for Roberta performing much better can attributed to the fact that it was trained or significantly large amounts of data and its pre-training tasks like SEGMENT-PAIR NSP helps in generating more robust embeddings. The performance of distilBERT was surprising because even though being much smaller than BERT and Roberta it was able to capture much of the information about the malware file and create good embeddings for them. It was observed that almost all transformer models gave good results when coupled with Resnet18 CNN classifier. The explanatory analysis introduced in chapter 5 explores various explanatory techniques to interpret the complex BERT model.

6.1 Future work

Since all the BERT models are trained on natural language, it will be more interesting if the pre-training of BERT is applied to malware samples such as MLM for masking and predicting opcodes and next opcode prediction. Other research which can be applied is discussed in the following sections.

6.1.1 Quantization of models

Section 3.1.2 introduced the concept of knowledge distillation where a smaller student model was taught to handle the tasks under the supervision of a larger teacher(BERT) model. This methodology helped in creating a much smaller model without affecting the accuracy of the original model. Another approach to downsize a large complex network is to use quantization where a approximate a large neural network which use floating point number by using a simpler smaller network of low bit width numbers. There are 2 approaches while applying quantization of neural

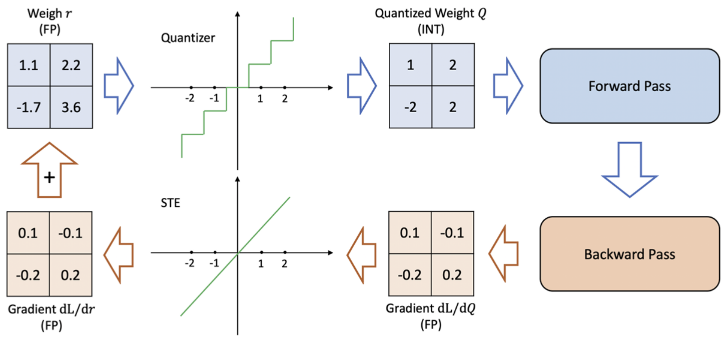


Figure 38: Quantization aware Training [40]

network.

1. **Applying quantization on pre-trained models**: In this approach, parameters of a large pre-trained model such as Resnet18 are converted to int8. This reduces the size of the model but also reduces the accuracy of the model. There are two types of "Post -Training Quantization" techniques which can be applied for quantizing a pre-trained model

   - **Weight Quantization**: Since weights of the model are not dependent on input, they can be quantized during training. When batch processing is used, it is important to quantize per channel or else the loss will explode [40]

2. **Quantization Aware Training**: In this method all the activations and weights are "fake quantized" *i.e* floating point values are rounded to mimic int8 values but the computations are done with floating point numbers. The forward pass of the neural network uses a scheme for rounding float-precision parameters to discrete levels, and in the backwards pass, the float-precision parameters are updated using gradients calculated during the forward pass .

6.1.2 Graph Neural Networks for API call sequences

API call sequences can be extracted from malware files using dynamic analysis. As these types of data are unstructured,complex neural networks may not be able to handle these representations. Graph neural networks provide a unified view of these data types ranging from images,text to unstructured graphs [41]. Representations like Node2vec and other paradigms like Grpah transformer networks can be explored to provide a holistic and intuitive represntation of API call sequences.

6.1.3 Fourier Neural Networks

As observed in Section 4.1.1 BERT based embeddings provide a good representations of opcodes. Since the transformer architecture used in BERT(Section 3.1.1 is computationally expensive, researchers in [42] introduced the idea of using fourier transforms mixed with input tokens as an encoder part of BERT. Since the parameters of fourier transform remain constant, the computational overhead during inference can be reduced significantly although it may slightly reduce the accuracy. From the Figure 39 it can be observed that only the transformer part of the encoder is changed.

6.1.4 More Explainability tools

Chapter 5 introduces tools and techniques for interpreting machine learning models like BERT. However both techniques do not explain the attention layers of BERT and the influential examples which help in prediction of BERT model. Techniques like Integrated gradients can help in looking inside the architecture of these complex models and help us in visualizing the attention mechanism for the malware instance. Other paradigms such as adversarial learning and and the perspective of counterfactual examples can also be explored.

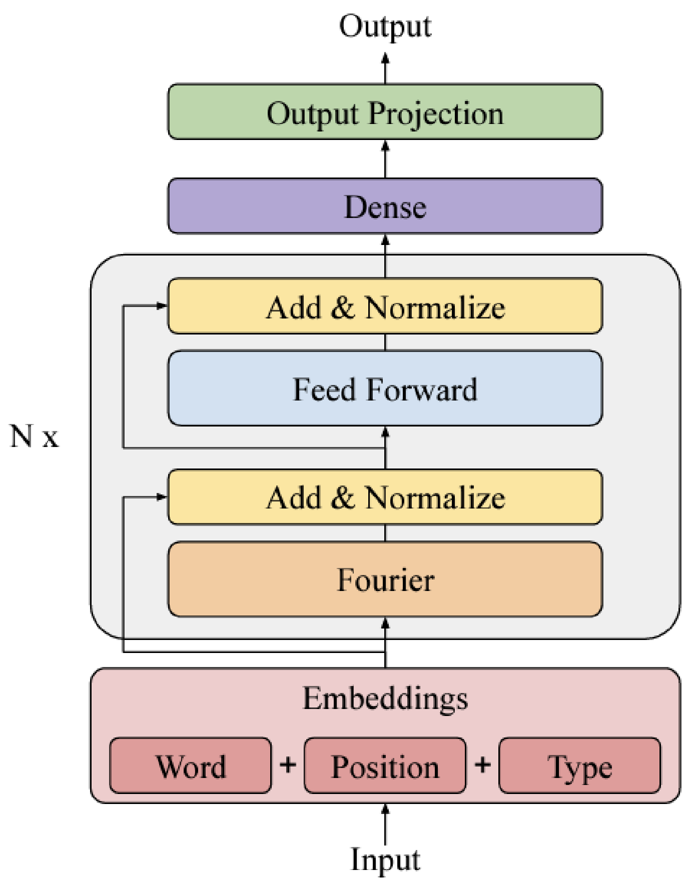


Figure 39: Fourier Encoder Block in FNet [42]

6.1.5 Different Transformer model

The transformers used in this project consist of only encoder based models. It would be interesting to see if decoder based transformers like GPT could match the accuracy of BERT based models. Other encodr based models like Xlnet can overcome the limitations of BERT based models by incorporating more tokens and using MLM(masked language modelling) on the opcodes itself.

LIST OF REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, ''Deep residual learning for image recognition,'' in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770--778.

[2] J. L. Alvares, ''Malware classification with bert,'' 2021.

[3] F. Leder, B. Steinbock, and P. Martini, ''Classification and detection of metamorphic malware using value set analysis,'' in *2009 4th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2009, pp. 39--46.

[4] W. B. Andreopoulos, *Malware Detection with Sequence-Based Machine Learning and Deep Learning*, M. Stamp, M. Alazab, and A. Shalaginov, Eds. Springer International Publishing, 2021.

[5] M. Goyal and R. Kumar, ''Machine learning for malware detection on balanced and imbalanced datasets,'' in *2020 International Conference on Decision Aid Sciences and Application (DASA)*, 2020, pp. 867--871.

[6] 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.libaccess.sjlibrary.org/10.1145/1593105.1593239

[7] N. McLaughlin and J. M. del Rincon, ''Data augmentation for opcode sequence based malware detection,'' 2021. [Online]. Available: https://arxiv.org/abs/2106.11821

[8] A. Fatima, R. Maurya, M. K. Dutta, R. Burget, and J. Masek, ''Android malware detection using genetic algorithm based optimized feature selection and machine learning,'' in *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, 2019, pp. 220--223.

[9] Y. Chen, Z. Shan, F. Liu, G. Liang, B. Zhao, X. Li, and M. Qiao, ''A gene-inspired malware detection approach,'' *Journal of Physics: Conference Series*, vol. 1168, no. 6, p. 062004, feb 2019. [Online]. Available: https://doi.org/10.1088/1742-6596/1168/6/062004

[10] M. A. Mathew, J.and Ajay Kumara, ''Api call based malware detection approach using recurrent neural network---lstm,'' A. Abraham, A. K. Cherukuri, P. Melin, and N. Gandhi, Eds. Cham: Springer International Publishing, 2020, pp. 87--99.

[11] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, ‘‘Distributed representations of words and phrases and their compositionality,’’ in *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., vol. 26. Curran Associates, Inc., 2013. [Online]. Available: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ‘‘Bert: Pre-training of deep bidirectional transformers for language understanding,’’ 2018. [Online]. Available: https://arxiv.org/abs/1810.04805

[13] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, ‘‘Deep contextualized word representations,’’ 2018. [Online]. Available: https://arxiv.org/abs/1802.05365

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

[15] D. Bahdanau, K. Cho, and Y. Bengio, ‘‘Neural machine translation by jointly learning to align and translate,’’ 2014. [Online]. Available: https://arxiv.org/abs/1409.0473

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, ‘‘Attention is all you need,’’ 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[17] J. Alammar. ‘‘The illustrated transformer.’’ [Online]. Available: https://jalammar.github.io/illustrated-transformer/

[18] C.Stefania. ‘‘The transformer model.’’ 2021. [Online]. Available: https://machinelearningmastery.com/the-transformer-model/

[19] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, ‘‘Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter,’’ 2019. [Online]. Available: https://arxiv.org/abs/1910.01108

[20] C. Buciluundefined, R. Caruana, and A. Niculescu-Mizil, ‘‘Model compression.’’ New York, NY, USA: Association for Computing Machinery, 2006. [Online]. Available: https://doi.org/10.1145/1150402.1150464

[21] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, ‘‘Roberta: A robustly optimized bert pretraining approach,’’ 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[22] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, ‘‘Albert: A lite bert for self-supervised learning of language representations,’’ 2019. [Online]. Available: https://arxiv.org/abs/1909.11942

[23] T.You. ‘‘Understanding random forest.’’ 2019. [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[24] R. Misra. ‘‘Support vector machines — soft margin formulation and kernel trick.’’ 2019. [Online]. Available: https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe

[25] M. Wadkar, F. Di Troia, and M. Stamp, ‘‘Detecting malware evolution using support vector machines,’’ *Expert Systems with Applications*, vol. 143, p. 113022, 10 2019.

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

[27] S. Sharma. ‘‘Svm: What makes it superior to the maximal-margin and support vector classifiers.’’ 2021. [Online]. Available: https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe

[28] W. W. Lo, X. Yang, and Y. Wang, ‘‘An xception convolutional neural network for malware classification with transfer learning,’’ in *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2019, pp. 1--5.

[29] F. Ramzan, M. U. Khan, A. Rehmat, S. Iqbal, T. Saba, A. Rehman, and Z. Mehmood, ‘‘A deep learning approach for automated diagnosis and multi-class classification of alzheimer’s disease stages using resting-state fmri and residual neural networks,’’ *Journal of Medical Systems*, vol. 44, 12 2019.

[30] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, ‘‘Enriching word vectors with subword information,’’ *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135--146, 2017.

[31] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, ‘‘Bag of tricks for efficient text classification,’’ in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers.* Association for Computational Linguistics, April 2017, pp. 427--431.

[32] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, ‘‘Fasttext.zip: Compressing text classification models,’’ 2016. [Online]. Available: https://arxiv.org/abs/1612.03651

[33] ‘‘Microsoft security intelligence. virtool:win32/ceeinject,’’ 2007. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool%3AWin32%2FCeeInject

[34] P. Gohel, P. Singh, and M. Mohanty, ‘‘Explainable ai: current status and future directions,’’ 2021. [Online]. Available: https://arxiv.org/abs/2107.07045

[35] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, E. Rutherford, T. Hennigan, J. Menick, A. Cassirer, R. Powell, G. v. d. Driessche, L. A. Hendricks, M. Rauh, P.-S. Huang, A. Glaese, J. Welbl, S. Dathathri, S. Huang, J. Uesato, J. Mellor, I. Higgins, A. Creswell, N. McAleese, A. Wu, E. Elsen, S. Jayakumar, E. Buchatskaya, D. Budden, E. Sutherland, K. Simonyan, M. Paganini, L. Sifre, L. Martens, X. L. Li, A. Kuncoro, A. Nematzadeh, E. Gribovskaya, D. Donato, A. Lazaridou, A. Mensch, J.-B. Lespiau, M. Tsimpoukelli, N. Grigorev, D. Fritz, T. Sottiaux, M. Pajarskas, T. Pohlen, Z. Gong, D. Toyama, C. d. M. d’Autume, Y. Li, T. Terzi, V. Mikulik, I. Babuschkin, A. Clark, D. d. L. Casas, A. Guy, C. Jones, J. Bradbury, M. Johnson, B. Hechtman, L. Weidinger, I. Gabriel, W. Isaac, E. Lockhart, S. Osindero, L. Rimell, C. Dyer, O. Vinyals, K. Ayoub, J. Stanway, L. Bennett, D. Hassabis, K. Kavukcuoglu, and G. Irving, ‘‘Scaling language models: Methods, analysis amp; insights from training gopher,’’ 2021. [Online]. Available: https://arxiv.org/abs/2112.11446

[36] C. Molnar, *Interpretable Machine Learning*, 2nd ed., 2022. [Online]. Available: https://christophm.github.io/interpretable-ml-book

[37] L. Shapley, ‘‘Quota solutions op n-person games1,’’ *Edited by Emil Artin and Marston Morse*, p. 343, 1953.

[38] S. Lundberg and S.-I. Lee, ‘‘A unified approach to interpreting model predictions,’’ 2017. [Online]. Available: https://arxiv.org/abs/1705.07874

[39] M. T. Ribeiro, S. Singh, and C. Guestrin, ‘‘" why should i trust you?" explaining the predictions of any classifier,’’ in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135--1144.

[40] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, ‘‘A survey of quantization methods for efficient neural network inference,’’ in *Low-Power Computer Vision.* Chapman and Hall/CRC, pp. 291--326.

[41] I. R. Ward, J. Joyner, C. Lickfold, Y. Guo, and M. Bennamoun, ‘‘A practical tutorial on graph neural networks,’’ 2020. [Online]. Available: https://arxiv.org/abs/2010.05234

[42] J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontanon, “Fnet: Mixing tokens with fourier transforms,” 2021. [Online]. Available: https://arxiv.org/abs/2105.03824