Bimodal Deep Learning Architecture for Malware Classification

Deniz Aytemiz

Committee Members: Creed Jones, Angelos Stavrou, Kendall Giles

COLLEGE OF ENGINEERING
BRADLEY DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
VIRGINIA TECH™

6 May 2023

1. Abstract

In recent years, there has been an increase in the spread of malicious software, as reported by AV vendors. Conventional detection methods have traditionally relied on signature-based approaches, which have become inadequate currently since malware authors utilize the polymorphic and metamorphic techniques that can bypass conventional detection methods. This prompted people working on malware identification and detection to search for better techniques. In the last decade, one approach is the use of machine learning techniques for malware detection and identification, which has shown promising results.

This research project involves the classification of malware samples from BIG2015 dataset into their corresponding families. To address this challenge, two different modalities are implemented and combined in one final Artificial Neural Network. The approach of first modality involves the use of a CNN model to learn feature hierarchy from malware samples represented as grayscale images. This modality is referred to as the Image Classification modality. The second modality's approach is feeding the x86 mnemonics sequences of malware samples into CNN architecture for obtaining a set of features of opcode patterns.This modality is referred as Natural Language Model modality. Two different CNN architectures are tested in this modality. In order to achieve better performance, a final CNN architecture is implemented that uses the features obtained from both modalities as input and performs the classification. The complete architecture concatenates these high level features obtained from both modalities and feeds them into an Artificial Neural Network model in order to classify the data.

Both modalities are evaluated individually. First modality performs a test accuracy of 0.8120 and second one performs test accuracies of 0.6771 and 0.5801. The full architecture performs test accuracy up to 0.8728. The F1 scores, precisions, and recalls are obtained for each modality and compared and discussed, along with possible further improvements.

2. Introduction

Malware is a term used to describe software applications that have been created with the intention of causing harm or carrying out unwanted actions on a computer system. Such actions may include disrupting the normal operations of the computer, collecting confidential information, bypassing security controls, gaining unauthorized access to private systems, and displaying unwanted advertisements. Malware can be categorized into different groups based on their intended purpose broadly as; adware, spyware, virus, worm, trojan, rootkit, backdoors, ransomware, command & control bots.

The detection of malicious software is currently achieved using signature-based and heuristic methods, which struggle to keep up with malware evolution. While signature-based methods have been widely used for antivirus software, identifying a specific virus is faster through a generic signature that identifies a virus family's common behaviors. However, malware creators try to evade detection by creating polymorphic and metamorphic malware that does not match virus signatures. AV vendors rely on heuristic-based methods that use rules based on expert analysis of malicious behavior, but this approach generates more false positives than signature-based methods. Therefore, AV vendors use a hybrid analysis approach that combines signature-based and heuristic-based methods to address unknown malware.

The analysis of malware can be categorized into two titles which are statistics and dynamic analysis. Static analysis involves analyzing a program without executing it, using techniques such as identifying string signatures, byte sequences, and API calls. Before conducting static analysis, malware is typically unpacked and decrypted using tools like IDA Pro or OllyDbg to display the code as a sequence of Intel x86 assembly instructions. Dynamic analysis, on the other hand, involves analyzing the behavior of a malicious program while it is being executed in a controlled environment such as a virtual machine or sandbox. Tools such as Process Monitor, Process Explorer, and Wireshark are used to monitor function and API calls, as well as the network and flow of information. While dynamic analysis is more effective than static analysis, it takes more time and resources and may not detect certain behaviors that are triggered only under specific conditions in the real environment.

In recent years, machine learning based approaches in the scope of malware detection and classification have increased. The traditional machine learning approaches rely mainly on feature extraction of a set of discriminative features. Deep learning methods facilitate this process even further since they automize the feature extraction from raw data at once. In this work, CNNs as deep learning models are implemented for feature extraction.

3. Literature Review

a. Convolutional Neural Networks for Malware Classification

This thesis serves as the foundation for my project, as it encompasses the implementations of the discussed methods. It provides a comprehensive overview of the background information related to the concepts discussed, such as malware types, neural networks, and the dataset used. Additionally, it covers the two CNN modalities utilized for classifying malware with image classification and NLP techniques. The first CNN modality employs .bytes files as input and performs image classification, while the other modality takes .asm files as input and retrieves opcodes from the files, classifying based on textual input data using CNN. The results presented in this thesis provide a benchmark for comparison, since my work mimics the methodologies implemented in this thesis.[1]

b. Applying NLP techniques to malware detection in a practical environment

This paper gives background information on static and dynamic malware detection techniques and their shortcomings in detection. The static methods are bypassed by obfuscation and dynamic methods take too much time and execution. This paper implements NLP for malware identification and discusses how NLP methods can be effective in detection of the zero day malware. It describes the NLP methods related to the study such as bag of words, latent semantic indexing and W2vec. The model structure authors use has a similar structure to my project which is shown in Figure 1.

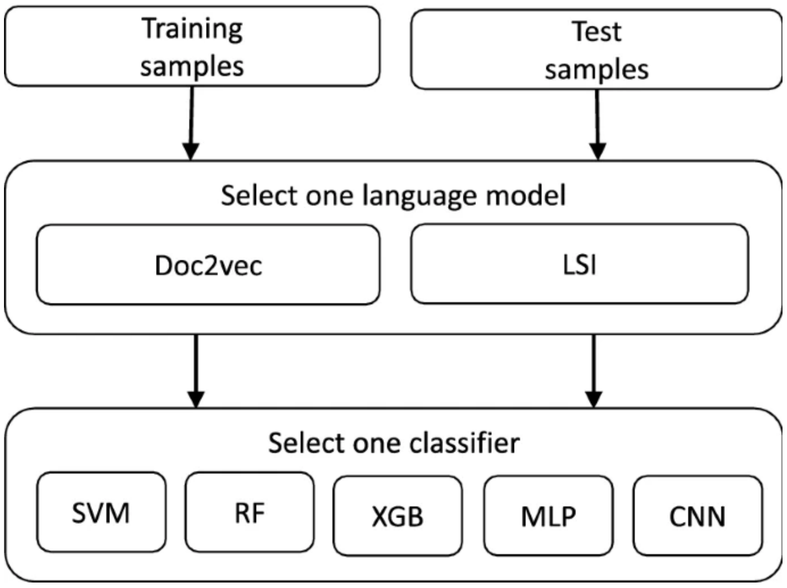


Figure 1: The model structure

This work uses the model for identifying the malware rather than classifying it. The dataset consists of benign and malicious samples. The logic is to retrieve printable strings from both classes samples and find out the most frequent words from them. Based on these words a language model is constructed. These words are converted into lexical features with the selected language model. Thus, the labeled feature vectors are derived. Thereafter, the selected classifier is trained with the both labeled feature vectors.

Overall this part has helped me to understand the NLP based techniques and the model structures they can be utilized in detection schemes. The NLP modality part in my project is the same except from the chosen language model and the classifier model.[2]

c. Malware Classification with Deep Convolutional Neural Networks

In this work, the authors use CNN based malware classification. Malware binaries are converted to grayscale images and subsequently train a CNN for classification. This work is very insightful in terms of how advantageous to tackle the problem of malware classification as an image classification problem. It is mentioned the images of samples in the same malware families are very similar to each other along with the variants of a malware. Hence the image classification can effectively determine which sample belongs to which class. And this type of classification is effective in terms of obfuscation techniques used by malware authors in addition to zero-day malware classification.

The proposed approach is data independent and learns the discriminative representation from the data itself rather than depending on hand-crafted feature descriptors which is another significant advantage of the model. The proposed method for converting byte files into grayscale images is given in Figure 2. The idea is to retrieve 8 bit vectors from the binary file and convert those vectors to decimal values which will represent one pixel in the image of malware. In Figure 3, it displays the samples of malware images from the same families and one can notice the resemblance among them.

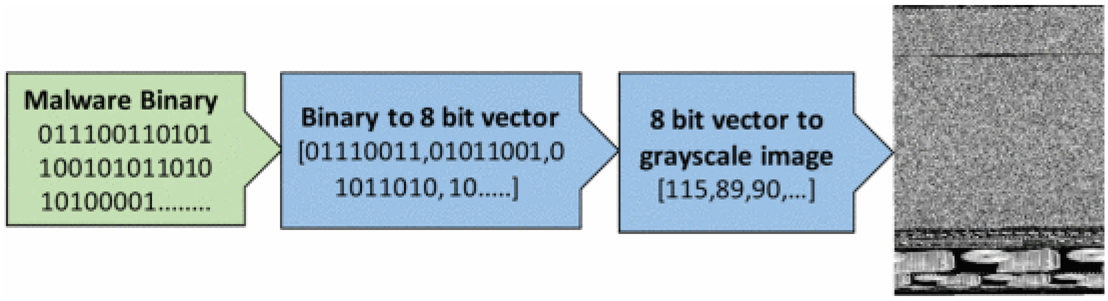


Figure 2: Malware byte file into gray scale image

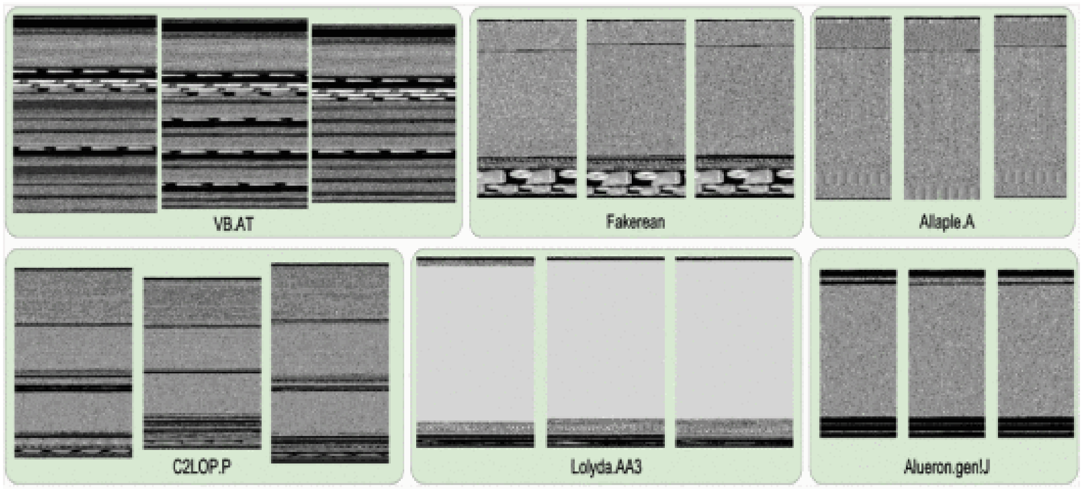


Figure 3: Examples of malware samples as images

In this work the authors used Microsoft Malware Classification Challenge (BIG 2015) dataset they achieved an accuracy of 99.87% on the dataset.[3]

d. Orthrus: A Bimodal Learning Architecture for Malware Classification

In this work, authors perform malware classification on BIG2015 dataset and they use two modalities. In the first modality features are learned from the hexadecimal sequence from byte files and in the second modality n-grams features are retrieved from opcodes sequence. The logic is similar to the work mentioned in (a) however instead of converting byte files to images, the authors preferred to take the hexadecimals in the byte files as sequence too. The model structure is given in Figure 4. Both modalities take raw bytes and raw mnemonics as inputs and feed them through convolutional layers with varying kernel sizes and are then passed through filters to obtain the most discriminative features. These features obtained from byte and asm files are then are concatenated and fed into softmax in order to classy the malware.[4]

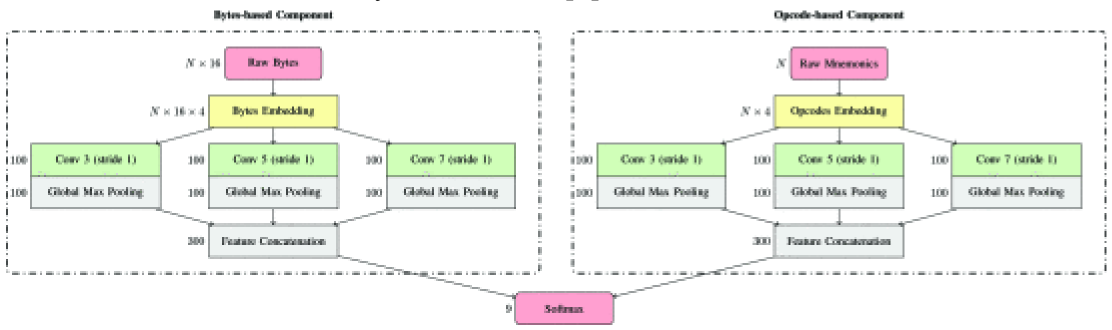


Figure 4: The model structure

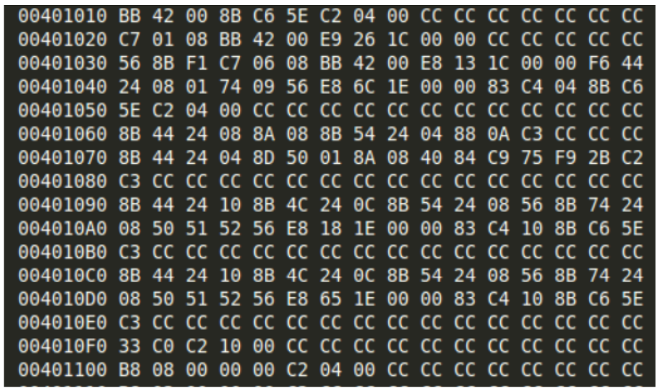
This work is informative since it explains each layer and their function in the model in detail in addition to discussing best evaluation metrics for this model. I have utilized these in my own implementation. It is also useful since, different from other methods discussed so far, the work merges output of two modalities into one layer in order to use the information coming from different sources for classification. They mention both modalities of information as input is suboptimal because it leads to overfitting one subset of features belonging to one modality and underfitting the features belonging to the other. To address this issue they have separately trained each subnetwork and optimized their hyper parameters for each subtask.

They mention instead of accuracy it is better to use F1 score since, in the dataset there is a large class imbalance. Hence the model can correctly predict the value of the majority class for all predictions and do poorly on the other ones yet still the accuracy can be good. It is also noted that the embedding size did not change the performance of the model significantly.

4. BIG 2015 Dataset

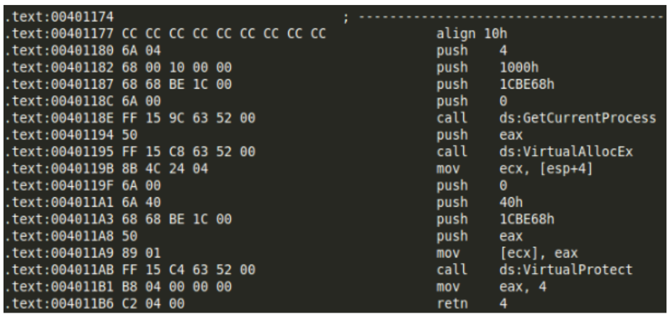
The dataset consists of multiple .asm files and .byte files of each malware sample. The byte file is the raw hexadecimal representation of the malware file and .asm files are log containing various metadata such as rudimentary function calls, memory allocation, and variable manipulation.

In Figure 5 and Figure 6, an example of .byte file and .asm is displayed



```
00401010 BB 42 00 8B C6 5E C2 04 00 CC CC CC CC CC CC CC
00401020 C7 01 08 BB 42 00 E9 26 1C 00 00 CC CC CC CC CC
00401030 56 8B F1 C7 06 08 BB 42 00 E8 13 1C 00 00 F6 44
00401040 24 08 01 74 09 56 E8 6C 1E 00 00 83 C4 04 8B C6
00401050 5E C2 04 00 CC CC CC CC CC CC CC CC CC CC CC CC
00401060 8B 44 24 08 8A 08 8B 54 24 04 88 0A C3 CC CC CC
00401070 8B 44 24 04 8D 50 01 8A 08 40 84 C9 75 F9 2B C2
00401080 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
00401090 8B 44 24 10 8B 4C 24 0C 8B 54 24 08 56 8B 74 24
004010A0 08 50 51 52 56 E8 18 1E 00 00 83 C4 10 8B C6 5E
004010B0 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
004010C0 8B 44 24 10 8B 4C 24 0C 8B 54 24 08 56 8B 74 24
004010D0 08 50 51 52 56 E8 65 1E 00 00 83 C4 10 8B C6 5E
004010E0 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
004010F0 33 C0 C2 10 00 CC CC CC CC CC CC CC CC CC CC CC
00401100 B8 08 00 00 00 C2 04 00 CC CC CC CC CC CC CC CC
```

Figure 5: .byte file



```
.text:00401174                               ; ---------------------------------------
.text:00401177 CC CC CC CC CC CC CC CC CC                align 10h
.text:00401180 6A 04                                     push    4
.text:00401182 68 00 10 00 00                            push    1000h
.text:00401187 68 68 BE 1C 00                            push    1CBE68h
.text:0040118C 6A 00                                     push    0
.text:0040118E FF 15 9C 63 52 00                         call    ds:GetCurrentProcess
.text:00401194 50                                        push    eax
.text:00401195 FF 15 C8 63 52 00                         call    ds:VirtualAllocEx
.text:0040119B 8B 4C 24 04                               mov     ecx, [esp+4]
.text:0040119F 6A 00                                     push    0
.text:004011A1 6A 40                                     push    40h
.text:004011A3 68 68 BE 1C 00                            push    1CBE68h
.text:004011A8 50                                        push    eax
.text:004011A9 89 01                                     mov     [ecx], eax
.text:004011AB FF 15 C4 63 52 00                         call    ds:VirtualProtect
.text:004011B1 B8 04 00 00 00                            mov     eax, 4
.text:004011B6 C2 04 00                                  retn    4
```

Figure 6: .asm file

The information regarding malware families in this dataset are given in the Table 1 below.

index	name	# of instance in the dataset	type of malware
1	Ramnit	1541	Worm
2	Lollipop	2478	Adware
3	Kelihos_ver3	2942	Backdoor
4	Vundo	475	Trojan
5	Simda	42	Backdoor
6	Tracur	751	TrojanDownloader
7	Kelihos_ver1	398	Backdoor
8	Obfuscator.ACY	1228	Obfuscated malware
9	Gatak	1013	Backdoor

Table 1: The summary of BIG 2015 dataset

Since the BIG2015 dataset is approximately 120 GB, I have encountered RAM and memory issues trying to use the whole dataset provided. Hence instead, I have used a subset of the original dataset. There is much less data imbalance than the original dataset however there is some still imbalance especially because of the class simda which has 42 samples in the original dataset.
The data distribution trained and tested in each modalities are displayed in the following sections.

6. The Model

a. Image Classification Modality

In this modality, the raw bytes files are used as input. The binaries in raw byte files are retrieved and they are grouped into 8 bit sized vectors. Each group is mapped to an unsigned integer which will be the pixel value in the image. The 1D array of the vectors of unsigned integers are then reshaped into 2D arrays of shape 64x64 so that we would obtain a grayscale representation of the raw bytes file.

Visualizing malware as an image has a key advantage, such that it enables the distinct sections of a binary file to be easily distinguished. Furthermore, since malware creators only modify a small part of the code to create new variations, images provide a useful means to identify even small changes while retaining the overall structure. This causes the malware variants belonging to the same family to be very similar in appearance as images, yet are distinct from images of other families.

After images for each file are obtained, there are 1962 images in total belonging to 9 classes. They are split into train and test data. Then the CNN A model is trained. The architecture of CNN A is displayed in Figure 7. The structure for whole image classification modality is shown in Figure 8.

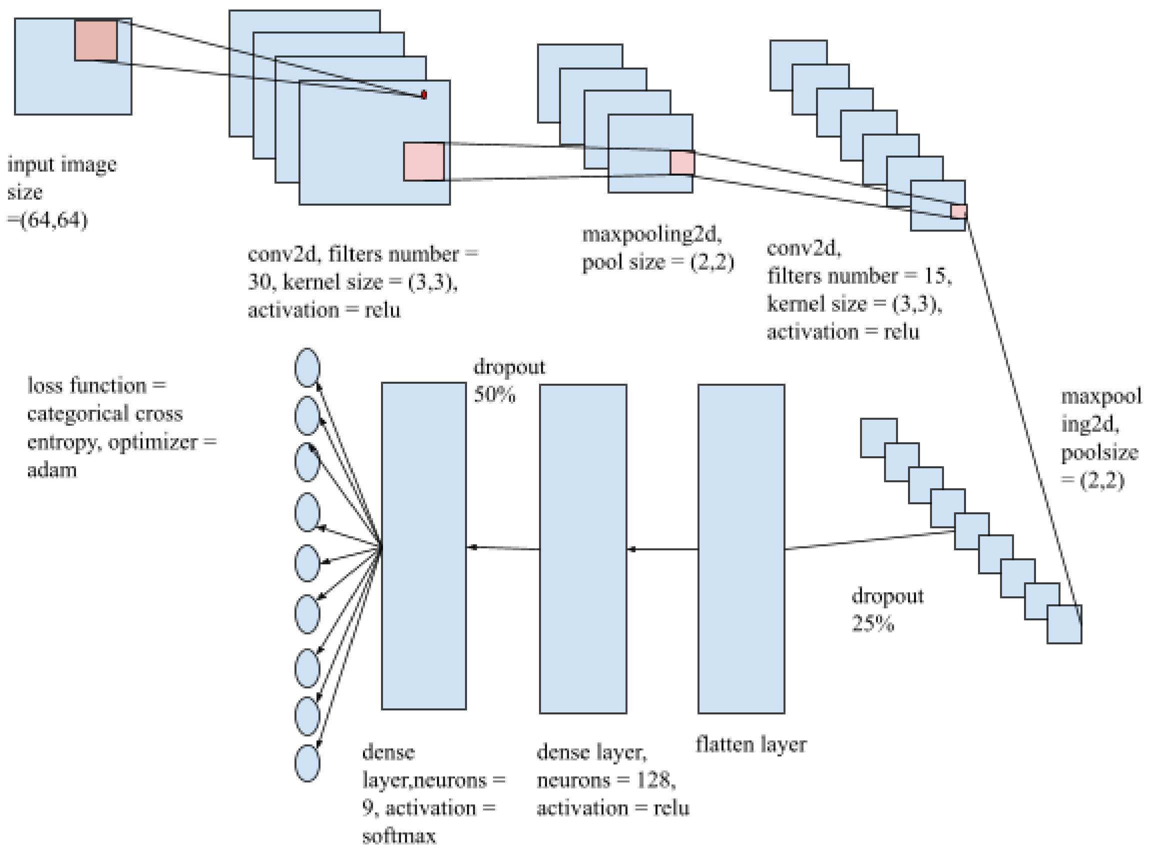

input image size =(64,64)
conv2d, filters number = 30, kernel size = (3,3), activation = relu
maxpooling2d, pool size = (2,2)
conv2d, filters number = 15, kernel size = (3,3), activation = relu
maxpooling2d, poolsize = (2,2)
dropout 25%
flatten layer
dense layer, neurons = 128, activation = relu
dropout 50%
dense layer,neurons = 9, activation = softmax
loss function = categorical cross entropy, optimizer = adam

Figure 7: The CNN A architecture

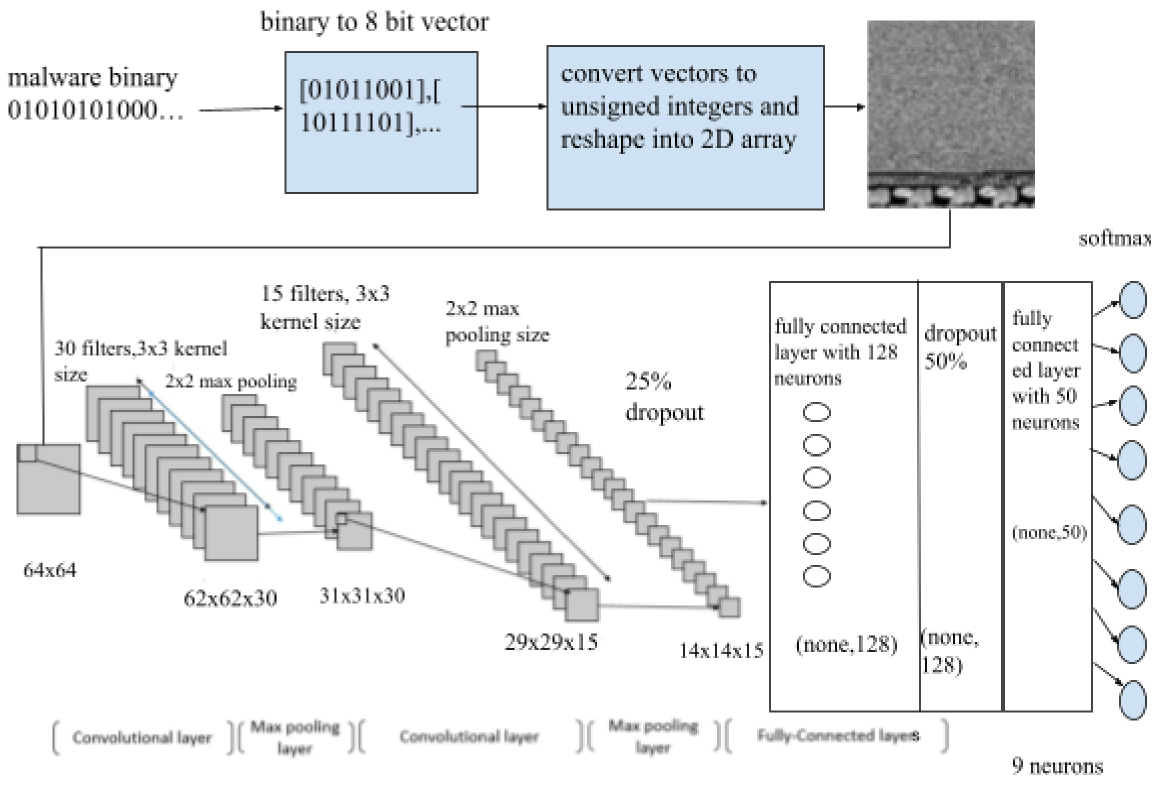


Figure 8: Image Classification Modality

In Figure 9, there is the display of the grayscale images obtained from .bytes files. It can be observed there are similar patterns in the images of malware from the same families which makes them distinguishable from other malware families' grayscale representation.

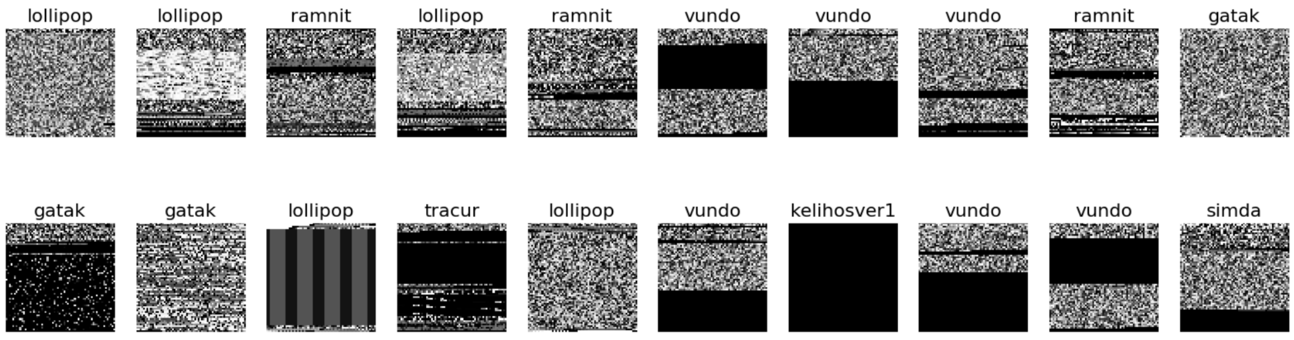


Figure 9: Images of malwares/ inputs to CNN modules

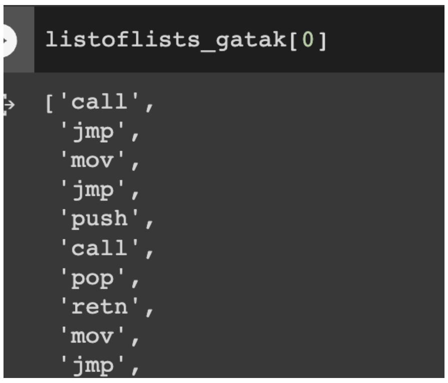
This model is evaluated individually and the performance of this model is mentioned in the Experimental Results part.

b. Natural Language Processing Modality

This modality aims to use natural language processing on .asm files by converting them into sequences of opcodes and embedding them as numerical vectors. The resulting vectors are then inputted into CNN B and CNN C models, which extract distinguishing features for classification. The procedure for achieving this is detailed below.

To sort the .asm files into their respective malware classes, I referenced a csv file that maps filenames to their corresponding labels. Once the files were sorted into the appropriate destination folders, they were converted to text files to facilitate data preprocessing.

I have found a dictionary of x86 mnemonics that includes all the possible commands in .asm files from online. Then, each .asm file is read line by line and split tokens. I check each token to see if it's a x86 mnemonic, and if it is, I add it to a list. When the entire .asm file is read, I have a list that contains all the mnemonics in that file in order. I repeat this process for every file in the malware folder. At the end of this step, I end up with a list of lists for each malware folder. Each list contains the mnemonics extracted from the respective file. Figure 10 displays an example from Gatak class. listoflists_gatak[0] is the mnemonic sequence list from the first .asm file in the Gatak folder and listoflists_gatak[1] is the sequence list of the second .asm file in the Gatak folder and so on.



```
listoflists_gatak[0]

['call',
 'jmp',
 'mov',
 'jmp',
 'push',
 'call',
 'pop',
 'retn',
 'mov',
 'jmp',
```

Figure 10: The list of lists for Gatak folder storing mnemonics

After obtaining these lists for each class, the next step is word embedding. We need a fixed length for each list in order to use them as inputs to the word embedding layer. In order to choose the fixed length, I have analyzed the average length of opcodes in the files of each class. This is displayed in Figure 11.

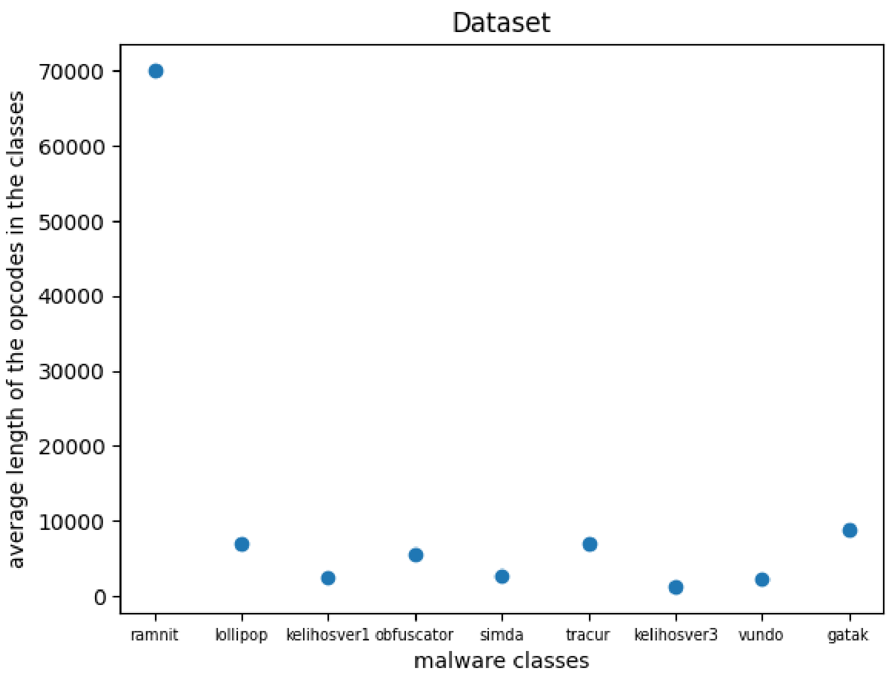
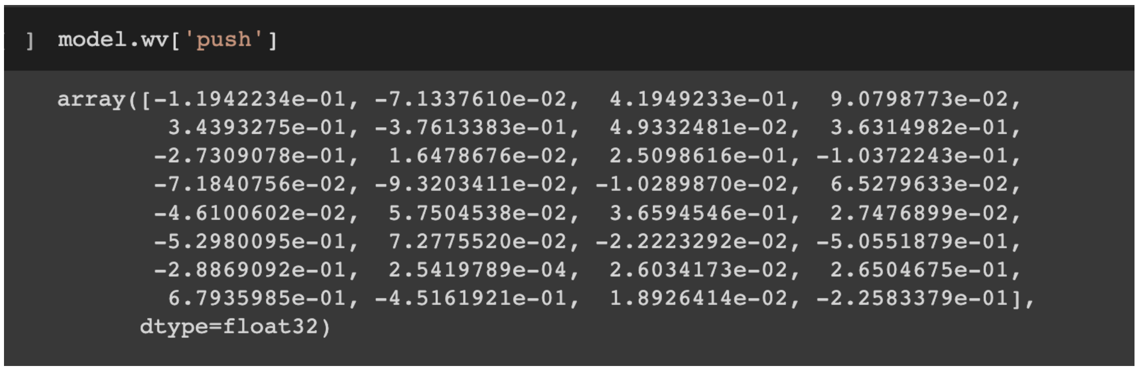


Figure 11: The average length of opcodes in each classes

The next step after obtaining the lists of mnemonics for each malware class is to use word embedding. To do this, we need a fixed length for each list so that they can be used as inputs for the word embedding layer. In order to determine this fixed length, I analyzed the average length of opcodes in each class and found that a fixed length of 10000 was suitable for all classes except Ramnit, which has a larger average opcode length. Therefore, I retrieved the first 10000 mnemonics from each file and padded any files with less than 10000 mnemonics with an 'unk' token until they reached the desired length. The input to the word embedding layer is a vector of length 10000.

To perform word embedding, the skip gram language model of word2vec is used. The embedding size is 32, and the window size is 5. This means that each mnemonic is mapped to a representative vector of size 32 once the model is trained. The window size of 5 means that the skip gram model is trained to predict the 5 words to the left and right of the input word. The weight matrix of the trained model is the desired word embeddings for all the unique input words. The matrix has a size of VxE, where V is the vocabulary size and E is the embedding size. Each row of the matrix represents a 32-sized vector for all the words in the vocabulary. The word embedding for the mnemonic 'push' is shown in Figure 12.



```
] model.wv['push']

  array([-1.1942234e-01, -7.1337610e-02,  4.1949233e-01,  9.0798773e-02,
          3.4393275e-01, -3.7613383e-01,  4.9332481e-02,  3.6314982e-01,
         -2.7309078e-01,  1.6478676e-02,  2.5098616e-01, -1.0372243e-01,
         -7.1840756e-02, -9.3203411e-02, -1.0289870e-02,  6.5279633e-02,
         -4.6100602e-02,  5.7504538e-02,  3.6594546e-01,  2.7476899e-02,
         -5.2980095e-01,  7.2775520e-02, -2.2223292e-02, -5.0551879e-01,
         -2.8869092e-01,  2.5419789e-04,  2.6034173e-02,  2.6504675e-01,
          6.7935985e-01, -4.5161921e-01,  1.8926414e-02, -2.2583379e-01],
        dtype=float32)
```

Figure 12: Vector representation of ‘push’

In order to train the CNN B and CNN C models in this approach, we must assign each mnemonic in the training samples a corresponding low-dimensional vector representation. This is accomplished by utilizing the weight matrix from the skip gram model as a reference table. The resulting weight vectors are stored in rows within an embedding matrix in a particular order that corresponds to the vocabulary dictionary of all files. For instance, if the vocabulary dictionary contains an entry for 'push':122, then the vector representation for the mnemonic 'push' will be saved in the 122nd row of the embedding matrix. Likewise, if there is an entry for 'add':15 in the vocabulary dictionary, then the vector representation for the mnemonic 'add' is saved in the 15th row of the embedding matrix. The vocabulary dictionary has a total of 443 keys, which includes the 'unk' key.

Figure 13 shows a t-SNE visualization of word vector representations, which can be used to determine the proximity of word vectors to one another in terms of Euclidean distance.

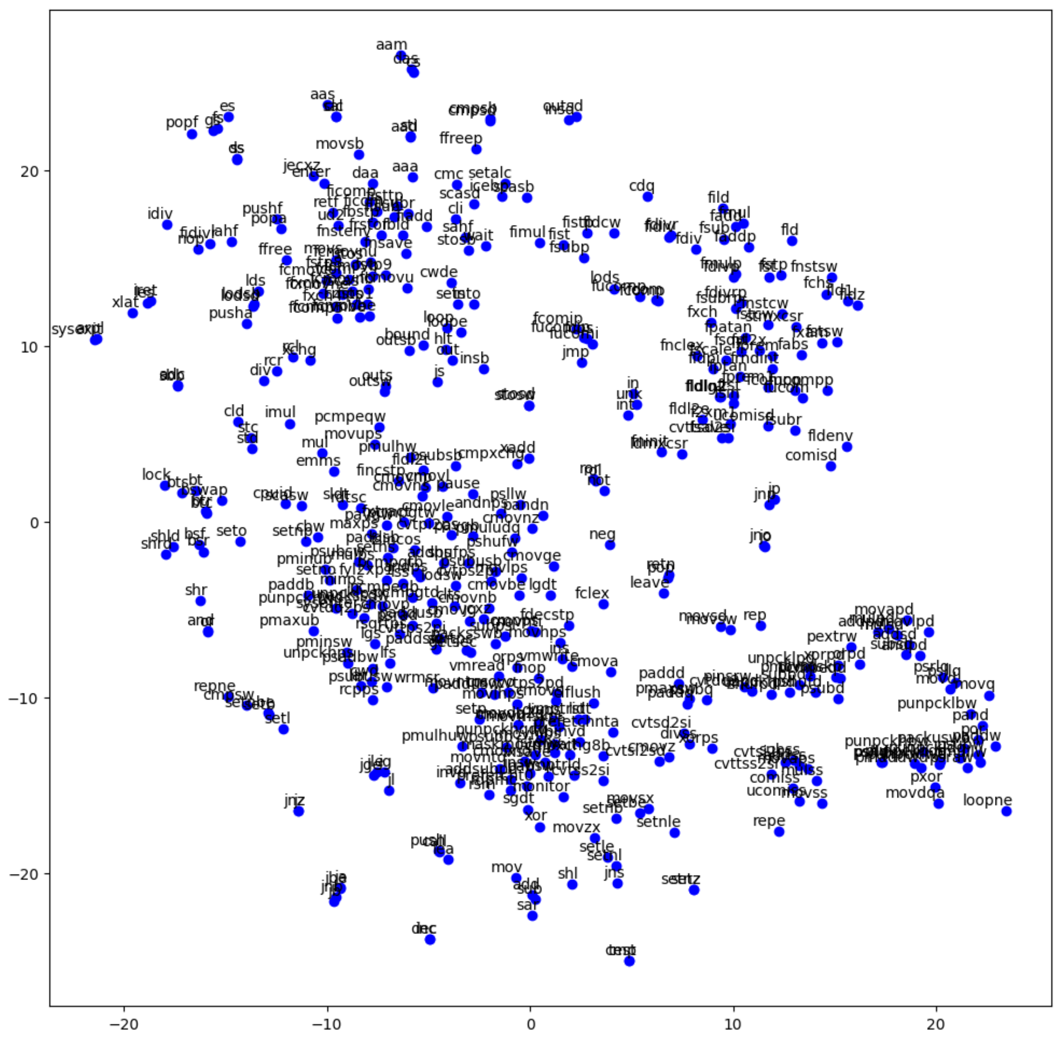
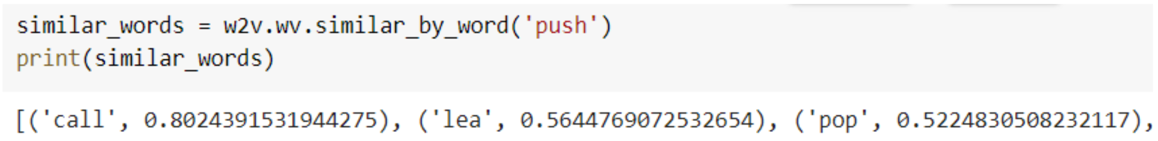


Figure 13: t-SNE representation of the word embeddings



```
similar_words = w2v.wv.similar_by_word('push')
print(similar_words)
```

```
[('call', 0.8024391531944275), ('lea', 0.5644769072532654), ('pop', 0.5224830508232117),
```

Figure 14: The most similar words to opcode 'push'

After the embedding matrix is obtained as discussed, the next step is to define a function that maps the training samples to their vector representations.

The function takes input of size 10000x1 string array and outputs an integer matrix of size 10000x32. Firstly, the function maps the opcode sequence of size 10000x1 to a one-hot encoded vector representation. The opcodes are one hot encoded based on their values in the vocabulary dictionary. For instance, if the key word 'push' has value 122 in the dictionary, the one hot encoded vector for that push contains a 1 in the position of 122 which corresponds to the opcodes' value in the dictionary. This function's objective is displayed in Figure 15.

V is the vocabulary size which is 443 and N is the fixed length of the lists which is 10000, E is the embedding size which is 32.

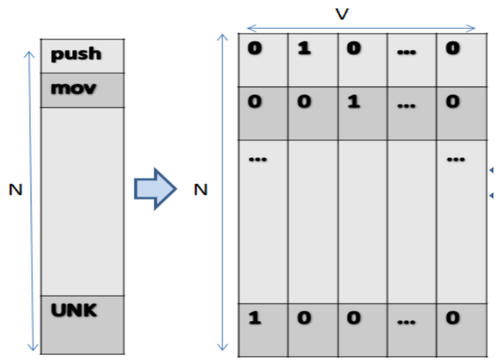


Figure 15: Training samples mapped to their one hot encoded representations

The function uses the obtained one hot encoded representation and multiplies it with the embedding matrix, as shown in Figure 16. This process results in the embedded training matrix for the opcode sequence, which is then outputted by the function.

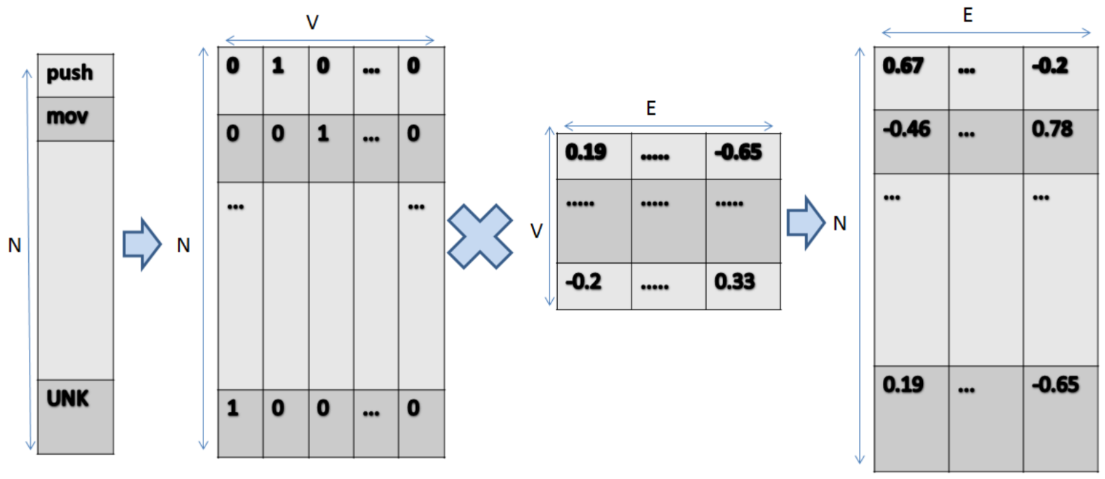


Figure 16: Each training sample is mapped to their corresponding word embeddings

After this function is implemented as described, all the training samples with size (10000,1) are embedded and we obtain a train set with 1440 samples with size (10000,32). This will be the input to the CNN B and CNN C models. The same operation is realized for the test set. Then 2 different CNNs are trained with these datasets.

I. CNN B
In Figure 17, the CNN B model's architecture is displayed.

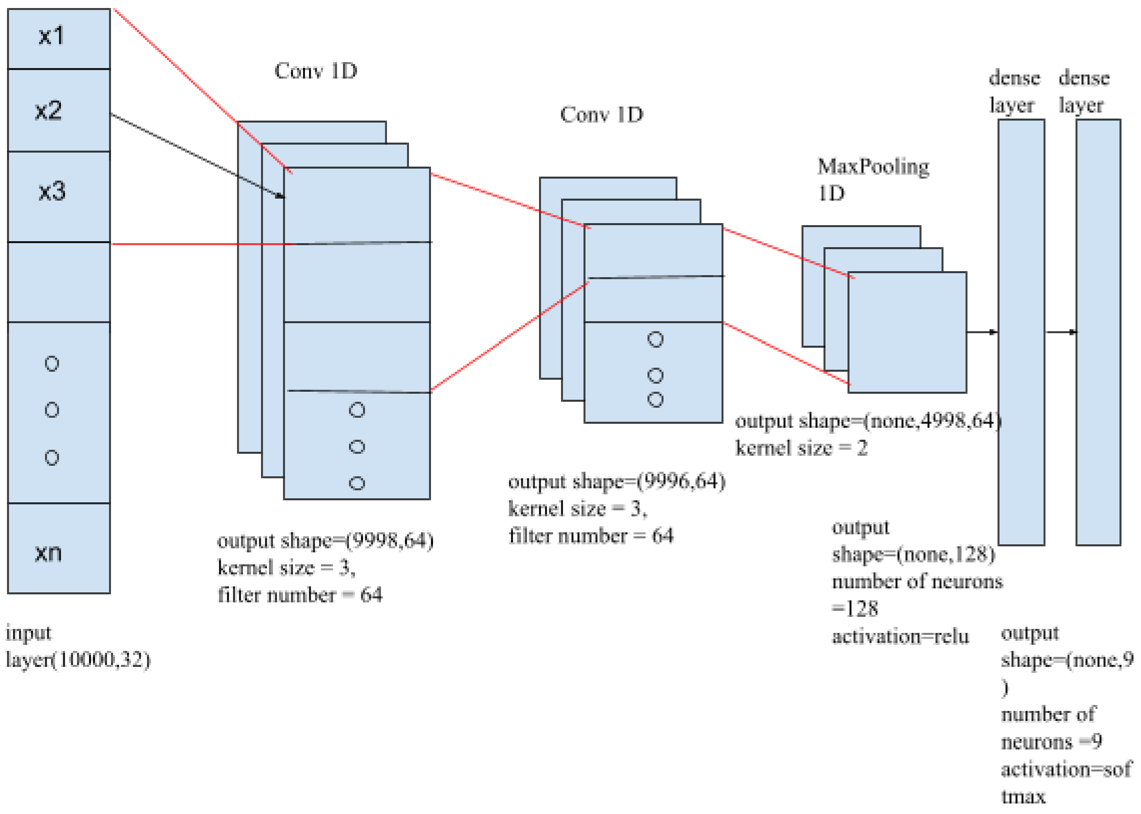


Figure 17: CNN B model architecture

II. CNN C
In Figure 18, the CNN C model's architecture is displayed. In this model, 3 convolution layers are applied in parallel and the features from each are concatenated and inputted into dense layers.

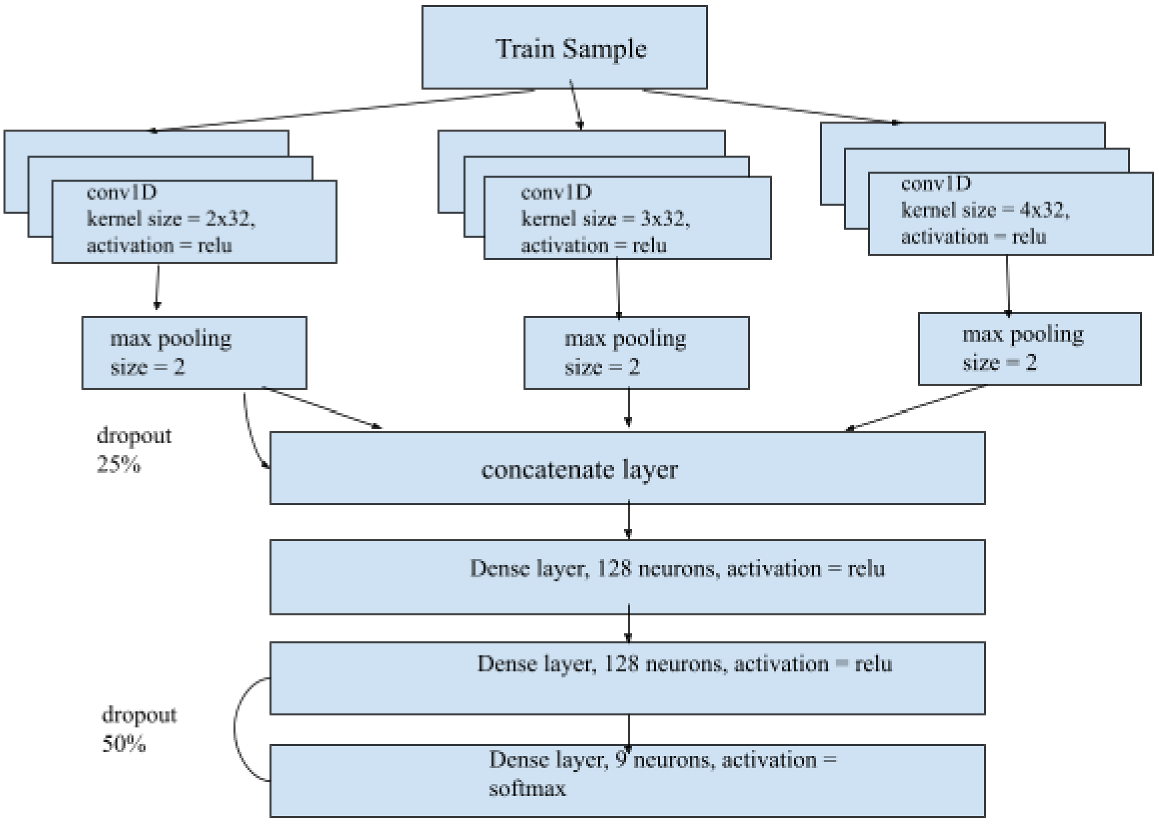


Figure 18: CNN C model architecture

c. The Bimodal Modality

The architecture of this model is given in Figure 19. In order to improve the performance of classification, the approach involves merging the output of the last fully-connected layer of CNN A with the output of the concatenate layer from CNN C. It's important to note that this combined CNN requires two different types of input data: grayscale images of the malware representation and opcodes from the disassembled files. Initially, CNN A and CNN C are trained separately, with CNN A learning low- and high-level image features while CNN C learns opcode patterns from the disassembled files. Next, the filters learned by each model are loaded into the combined CNN, which is then retrained to learn the weight matrix of the output layer. The main advantage of this approach is that the resulting CNN has the potential to solve problems that neither individual CNN alone can solve.

The concatenated layer takes 2 inputs. First is the output of the Dense 128 layer of the CNN A and second is the output of the concatenate layer of the CNN C. We obtain feature vectors with size (128,),(1792,) respectively for each training sample. These training samples are concatenated according to their file names such that for the same file, the feature vectors obtained from Dense 128 layer of the CNN A and concatenate layer of the CNN C. These are concatenated and we obtain a feature vector of size (1920,). This is fed to a sequential model that classifies the samples into 9 classes.

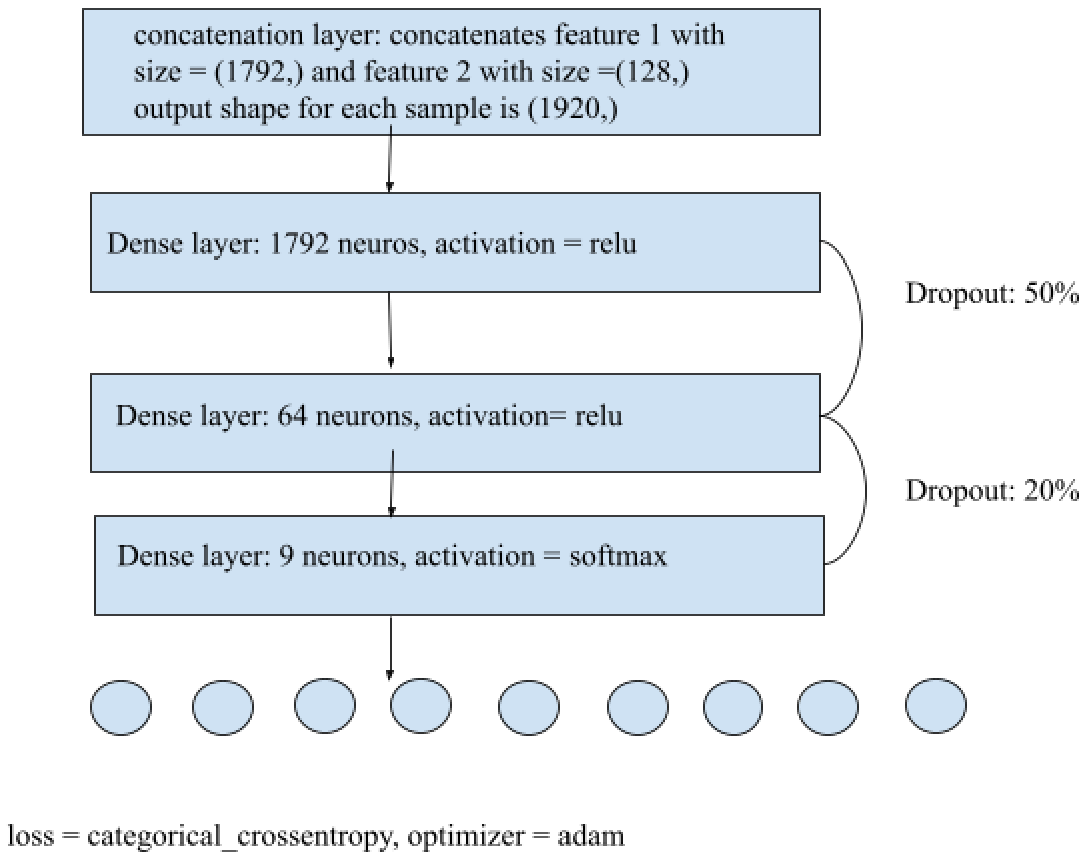

concatenation layer: concatenates feature 1 with size = (1792,) and feature 2 with size =(128,) output shape for each sample is (1920,)
Dense layer: 1792 neuros, activation = relu
Dropout: 50%
Dense layer: 64 neurons, activation= relu
Dropout: 20%
Dense layer: 9 neurons, activation = softmax
loss = categorical_crossentropy, optimizer = adam

Figure 19: The architecture of bimodal modality

7. Experimental Results

In this work, the overall model is implemented in the Google Collab environment with 25 RAM and 166.77 GB disk capacity. GPU and TPU runtimes are up to 12 hours and the GPUs are K80, P100, T4.

a. Image Classification Modality

The train and set distributions used for this modality are given in Figure 20 and 21.

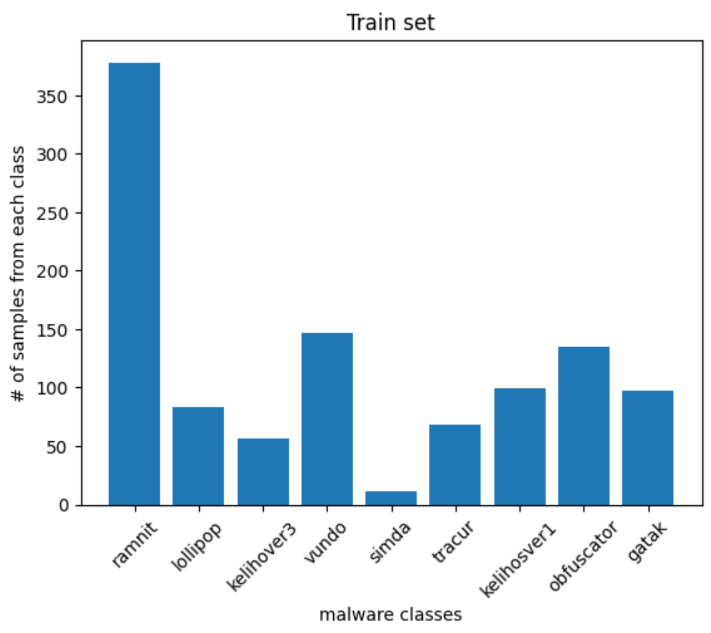


Figure 20: Train set data distribution

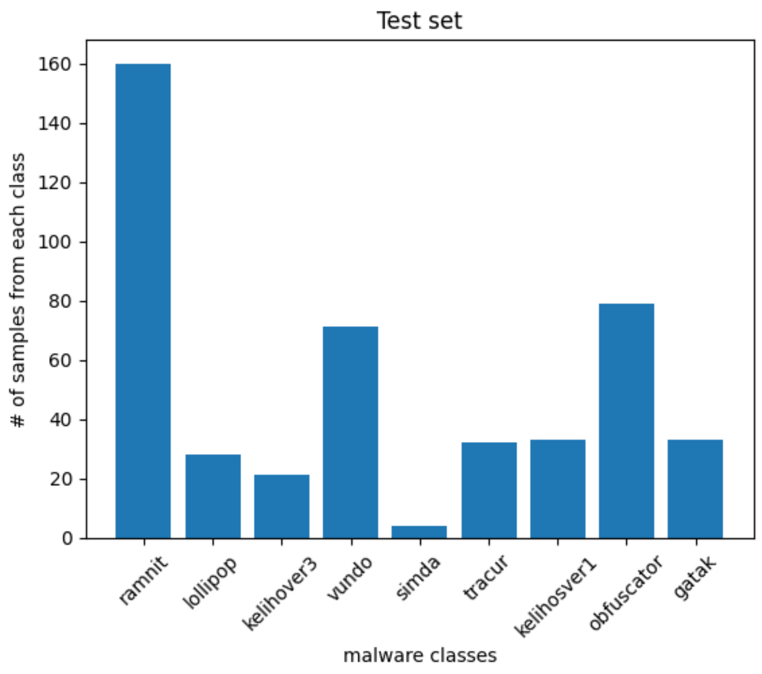


Figure 21: Test set data distribution

The accuracy and loss in each epoch is shown in Figure 22.

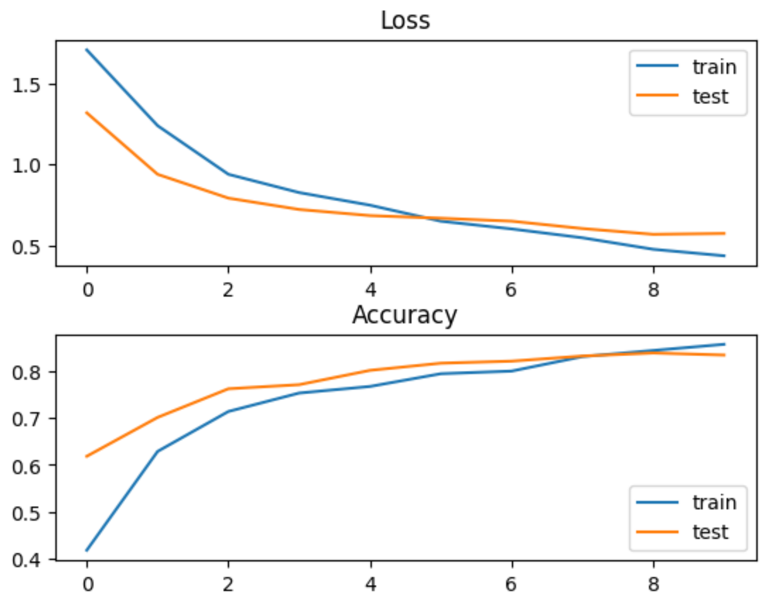
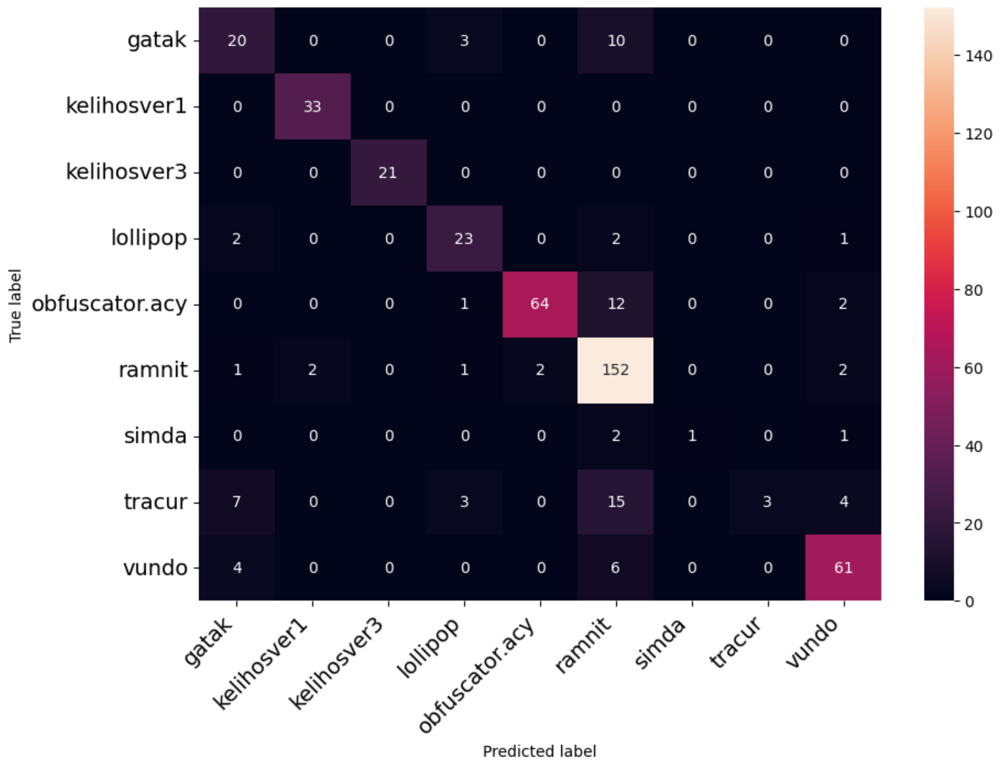


Figure 22: The loss and accuracy in each epoch

Since the train accuracy increases in each epoch yet test sets accuracy plateau, the model may be overfitting, therefore I decreased the epoch to 5 and trained again. The accuracy result became 0.8120 which is slightly better. In Figure 23,the confusion matrix is shown for this model.



	gatak	kelihosver1	kelihosver3	lollipop	obfuscator.acy	ramnit	simda	tracur	vundo
gatak	20	0	0	3	0	10	0	0	0
kelihosver1	0	33	0	0	0	0	0	0	0
kelihosver3	0	0	21	0	0	0	0	0	0
lollipop	2	0	0	23	0	2	0	0	1
obfuscator.acy	0	0	0	1	64	12	0	0	2
ramnit	1	2	0	1	2	152	0	0	2
simda	0	0	0	0	0	2	1	0	1
tracur	7	0	0	3	0	15	0	3	4
vundo	4	0	0	0	0	6	0	0	61

Figure 23: The confusion matrix for CNN A

CNN A performs well in most of the classes, but there is a noticeable issue where a significant number of tracur samples are being misclassified as ramnit, while

ramnit samples are being correctly identified. This suggests a possible bias towards ramnit files, as they are overrepresented in the training set. Additionally, the poor classification of simda can likely be attributed to the scarcity of samples from that class.
The precision for each class indicates how well the model performed to correctly predict the sample when it actually belongs to that class and recall indicates how well the model identifies all the samples from a particular class.The F1 score is the overall performance of a classification. It combines precision and recall into a single score making it a useful indicator of the model's overall effectiveness in identifying positive samples.The Table 2 shows the precision, recall, and F1 scores for each class.

Class	**Precision**	**Recall**	**F1**
gatak	0.6060	0.5882	0.5970
kelihosver1	1	0.9428	0.9706
kelihosver3	1	1	1
lollipop	0.8214	0.7419	0.7796
obfuscator	0.8101	0.9697	0.8827
ramnit	0.9500	0.7716	0.8516
simda	0.2500	1	0.4
tracur	0.0937	1	0.1713
vundo	0.8591	0.8592	0.8591

Table 2: Precision, recall and F1 scores of CNN A

It is observed that CNN A gives low precision but very high recall for classes tracur and simda. The model is correctly identifying all the positive samples, but it is also incorrectly labeling many negative samples as positive. In other words, the model is achieving high recall by capturing all the positive samples, but at the cost of low precision due to the high number of false positives for these classes. This indicates not enough discriminative patterns are found in these classes. On the other hand for class kelihosver3, the model is perfectly identifying all the positive samples and not making any false positives. In other words, the model is achieving a perfect balance between precision and recall.

b. Natural Language Processing modality

The data distribution of each class in the dataset used in this modality is shown in Figure 24.

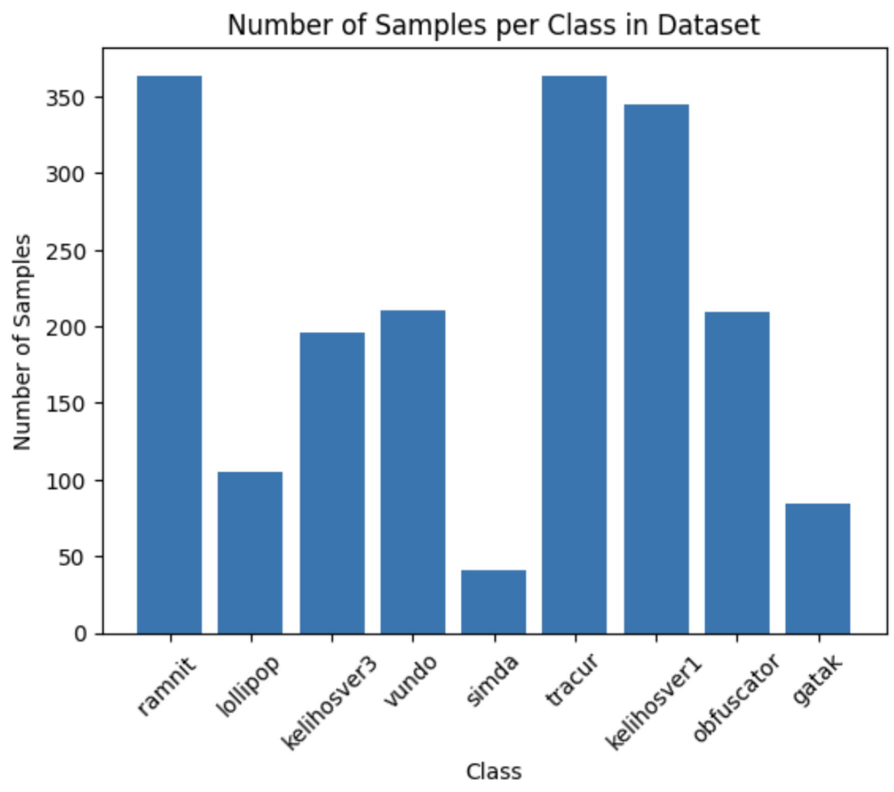


Figure 24: Dataset used in this modality

I. CNN B model

For the CNN B model the test accuracy achieved is 0.6771. The confusion matrix is displayed in Figure 25.

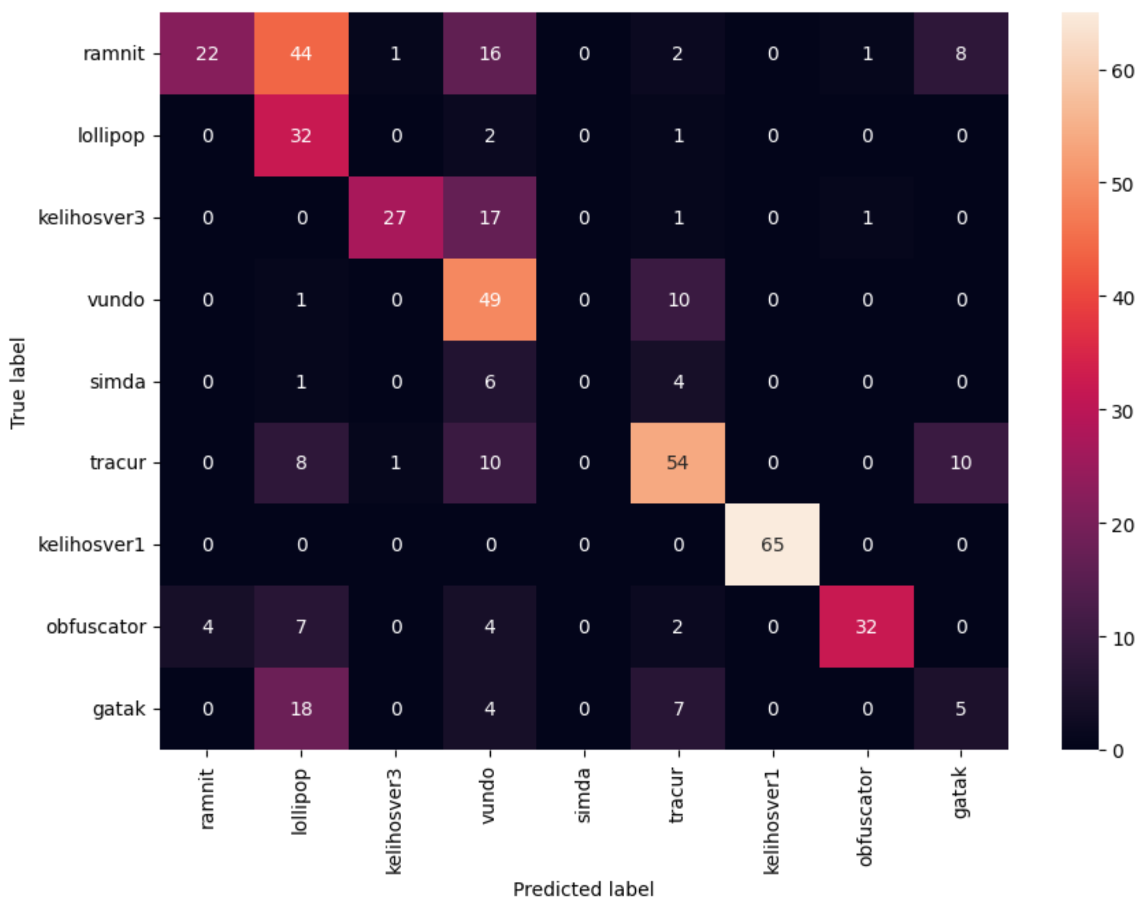


Figure 25: Confusion matrix for CNN B

The model encountered difficulties in accurately categorizing samples from the ramnit class, as numerous samples were misclassified as other classes. Nonetheless, when the model did identify a sample as ramnit, it was typically correct, indicating a high recall but low precision for this class. Conversely, the model excelled in identifying samples from the lollipop and kelihosver1 classes, but many samples from other classes were also misclassified as lollipop, leading to a high precision but low recall for this class, similar to Vundo. The model's performance in classifying simda was inadequate, as no samples were predicted as simda, regardless of whether they were from that class or not. Furthermore, the model struggled to accurately classify samples from the gatak class, resulting in low precision and recall.

The presion, recall and F1 score for CNN B is given in Table 3.

class	precision	recall	F1
ramnit	0.2340	0.8461	0.3666
lollipop	0.9142	0.2883	0.4384
kelihosver3	0.5869	0.9310	0.7199
vundo	0.8167	0.4537	0.5833

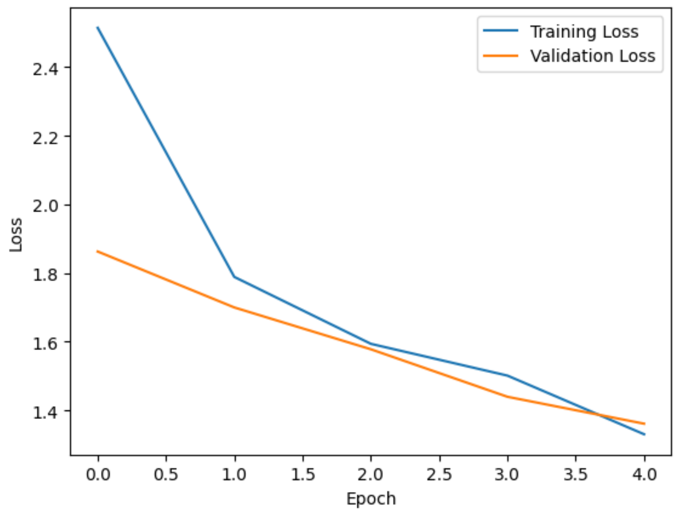
simda	0	undefined	undefined
tracur	0.6506	0.6667	0.6585
kelihosver1	1	1	1
obfuscator	0.6531	0.9412	0.7711
gatak	0.1470	0.2174	0.1754

Table 3: Precision, recall and F1 scores for CNN B

Apart from that this model performed average overall. This could be improved as more data is fed to the CNN model. Since this model processes text sequence data, either a larger dataset or more feature extraction is needed since the model may fail to extract discriminative patterns in data especially for class simda.

II. CNN C

This CNN model performed a test accuracy of 0.5801 which is slightly worse than CNN B. CNN C extracts more features from the data since there are more convolutional layers running in parallel with different filter sizes. Each filter encapsulates different lengths of sequences. The training, validation loss and accuracy plots are given in Figure 26.



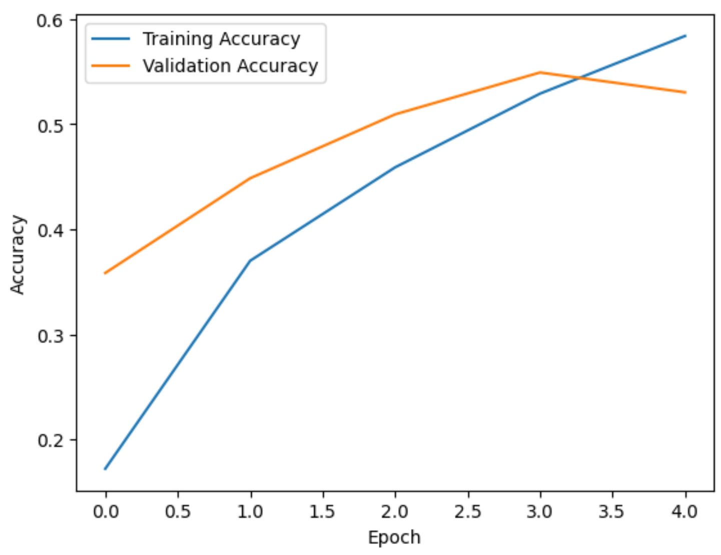


Figure 26: Train/test loss and train/test accuracy in each epoch

The plots show that the model is not overfitting, as the training stops just after the validation loss begins to stabilize while the training accuracy plateaus and validation accuracy continues to improve.

Accuracy alone cannot accurately represent the performance of a model, as it may appear high even if only one class is correctly classified while the rest are misclassified. Therefore, it is necessary to examine the confusion matrix, which is shown in Figure 27, to get a more comprehensive understanding of the model's performance across all classes.

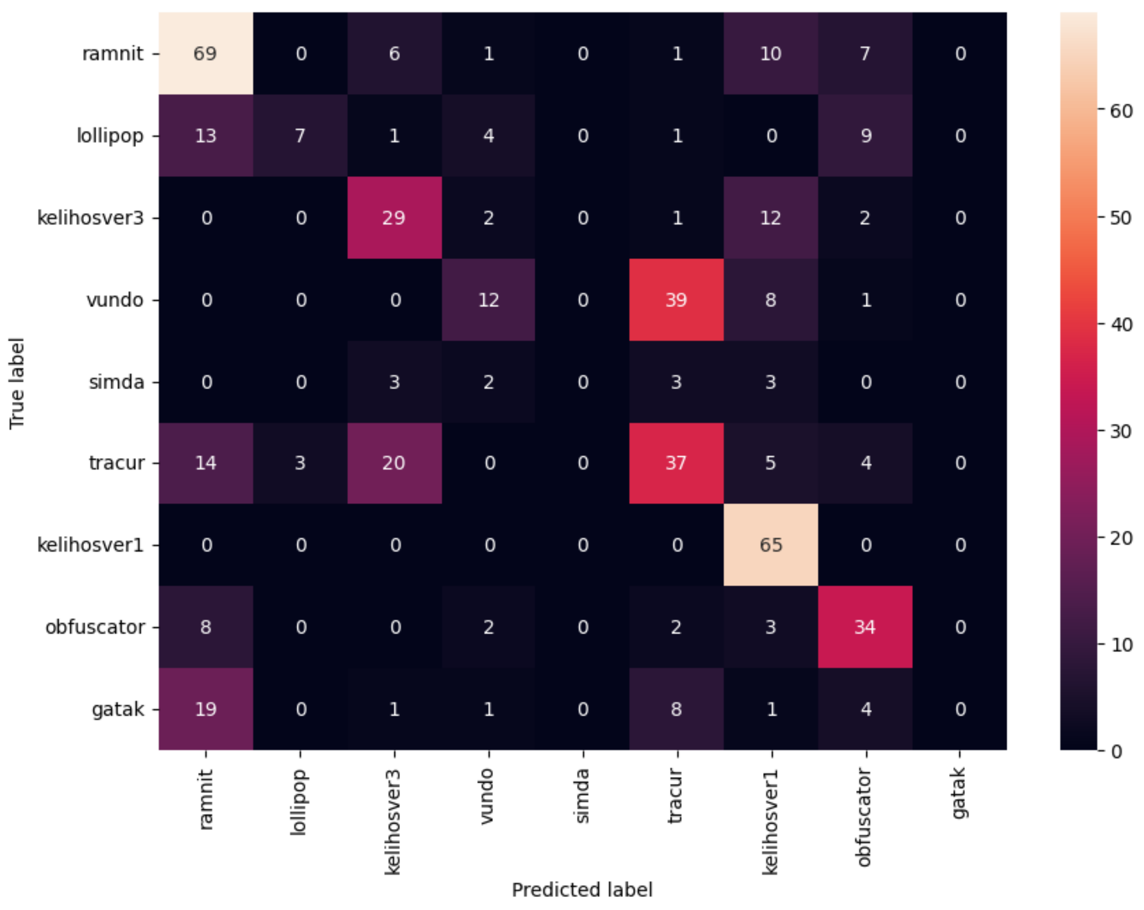


Figure 27: Confusion matrix for CNN C

The model showed significant improvement in classifying the ramnit class. As expected, the misclassifications were mostly observed in classes with fewer samples. Overall, the CNN C did not perform better than CNN B in fact analyzing F1 scores. To improve the classification for each class, the dataset can be expanded and balanced to prevent the model from overfitting to one set of features. This could lead to a significant improvement in performance.Moreover, adding additional convolutional layers to CNN C to enhance the extraction of discriminative features could potentially improve the model's performance. In table 4 precision, recall and F1 scores for CNN C are shown.

class	precision	recall	F1
ramnit	0.7340	0.5609	0.6359
lollipop	0.2	0.7	0.3111
kelihosver3	0.6304	0.5088	0.56311
vundo	0.2	0.5	0.2857
simda	0	undefined	undefined

tracur	0.4458	0.4022	0.4229
kelihosver1	1	0.6075	0.7558
obfuscator	0.6939	0.5574	0.6182
gatak	0	undefined	undefined

Table 4: Precision, F1 score and recall for CNN C

c. The Bimodal Architecture

The distribution of data used in this modality is shown in Figure 30.

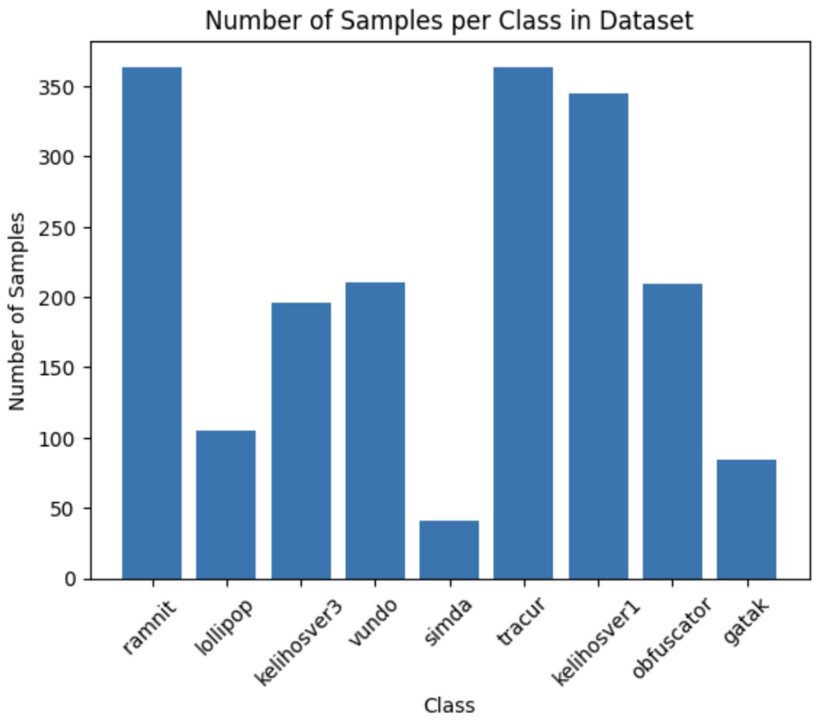


Figure 28: The distribution of samples in dataset

The training and validation loss and accuracies are displayed in Figure 29 and Figure 30.

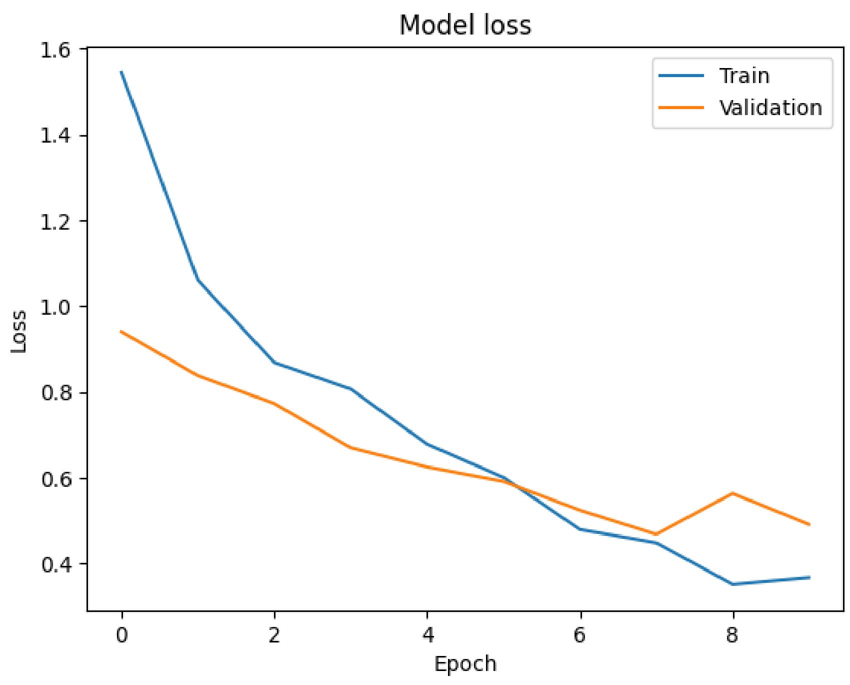


Figure 29: The loss in each epoch in train and validation set

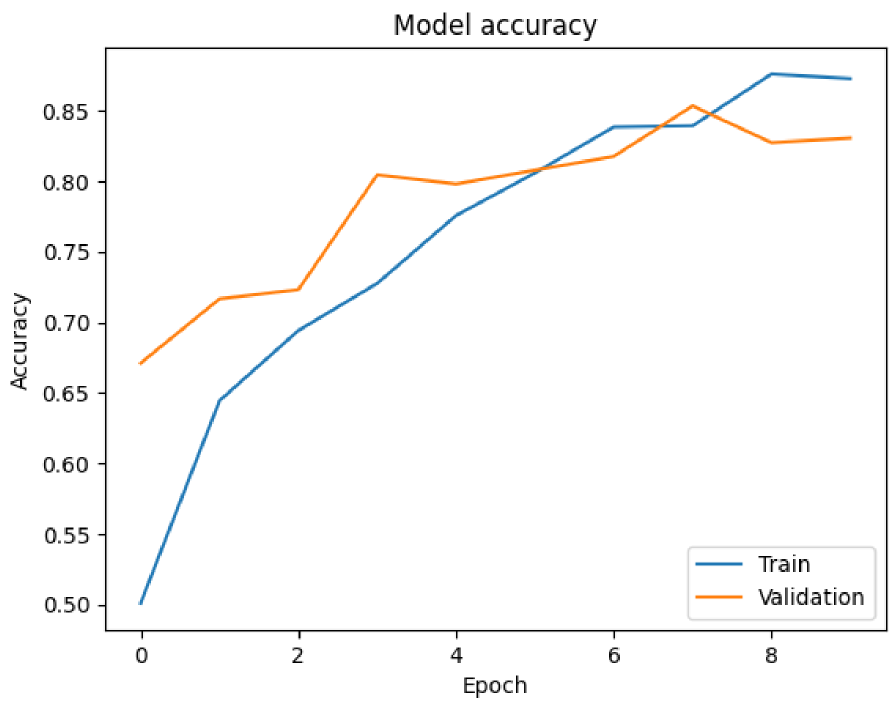


Figure 30: The accuracy in each epoch in train and validation set

This model demonstrates a test accuracy of 0.8788, which is notably higher than the other models previously discussed.The corresponding confusion matrix is shown in Figure 31.

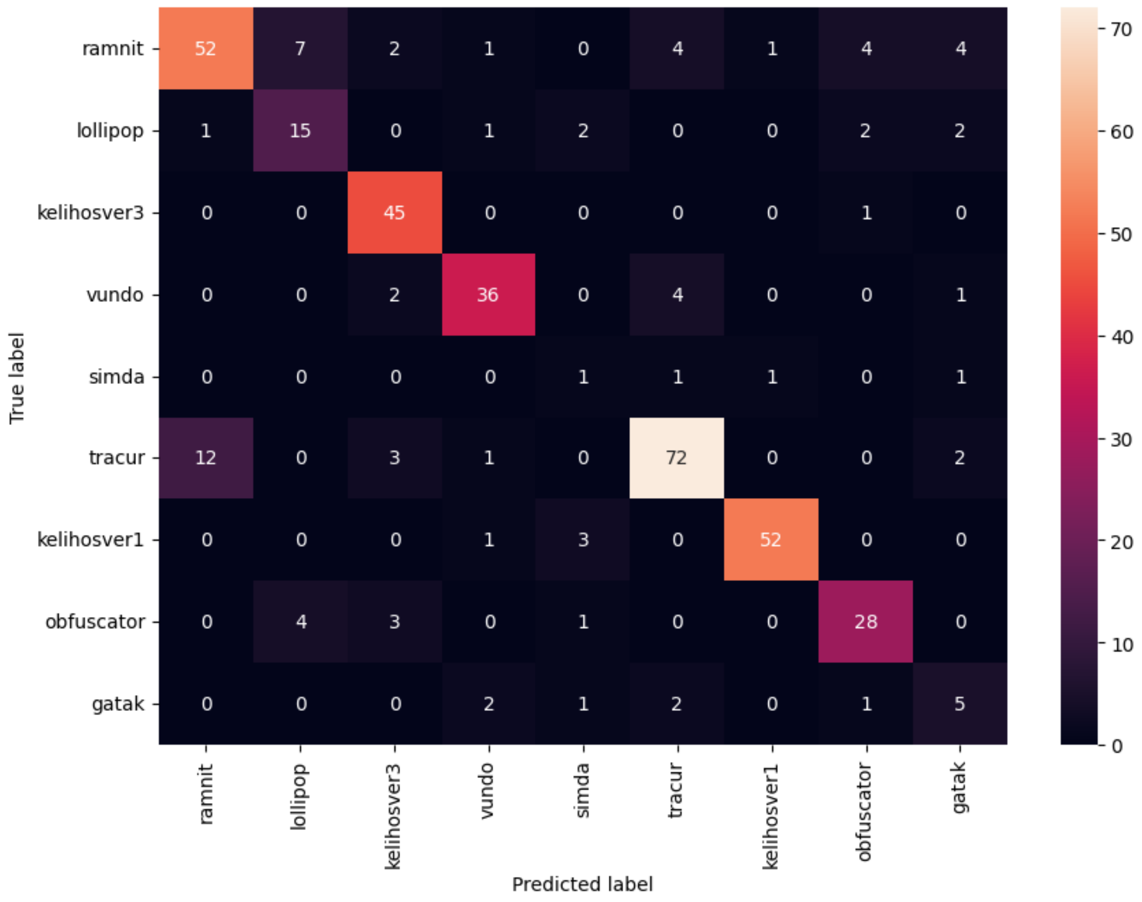


Figure 31: Confusion Matrix for bimodal structure

In Table 5, there is the precision, recall and F1 scores for this modality.

class	precision	recall	F1
ramnit	0.6933	0.8	0.7428
lollipop	0.6522	0.5769	0.6122
kelihosver3	0.9782	0.8182	0.8911
vundo	0.8372	0.8571	0..8470
simda	0.25	0.125	0.1667
tracur	0.8	0.8675	0.8324
kelihosver1	0.9286	0.9630	0.9455
obfuscator	0.7778	0.7778	0.7778
gatak	0.4545	0.3333	0.3845

Table 5: Precision, recall and F1 scores for bimodal architecture

This approach shows significantly improved results for all classes compared to the other models discussed. The F1 scores for each class are higher than those obtained by other models. Remarkably, this model is capable of identifying samples from the simda class, indicating that CNN A and CNN C were successful in extracting sufficient features.

8. Future Work

Expanding the scope of the project could involve incorporating detection and classification capabilities into the model. In addition to identifying whether a sample is malicious or benign, the model could also assign it to the appropriate class, which would provide more insights into the malware landscape.

To potentially improve the model's accuracy, one possible enhancement would be to expand the allowable input opcodes to include the maximum number found in any file within the dataset, thus enabling the model to capture all relevant information about each malware sample.

Another approach to enhancing performance would be to use grid search to optimize hyperparameters, identifying the best parameters for the model.

To address the imbalanced data, another improvement would be to use class weights, which could enhance the model's ability to accurately classify samples from underrepresented classes.

Another possible improvement to the bimodal architecture would be to replace CNN C with a more effective feature extraction model, concatenate the outputs, and then feed them into another machine learning classifier, which could potentially improve performance.

9. References

[1]. D. Gibert, “Convolutional Neural Networks for Malware Classification.”

[2]. M. Mimura and R. Ito, “Applying NLP techniques to malware detection in a practical environment - International Journal of Information Security,” *SpringerLink*, 06-Jun-2021. [Online]. Available: https://link.springer.com/article/10.1007/s10207-021-00553-8.

[3]. M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang and F. Iqbal, "Malware Classification with Deep Convolutional Neural Networks," 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 2018, pp. 1-5, doi: 10.1109/NTMS.2018.8328749.

[4]. D. Gibert, C. Mateu and J. Planes, "Orthrus: A Bimodal Learning Architecture for Malware Classification," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 2020, pp. 1-8, doi: 10.1109/IJCNN48605.2020.9206671.