CS 171: Introduction to Machine Learning

Midterm 2

Chee Jun Wong (013844545)

Introduction

Malicious software, sometimes known as malware, is software that is meant to do things like steal private information, gain root authority, disable a targeted host, and so on. Yet, as the Internet and software industries continue to grow, more and more malware varieties appear practically everywhere. A great number of studies on how to identify malware have been published. Malware detection is essentially a binary classification problem, and most antivirus software depends on a static signature-based detection mechanism. For this particular project, I was given a dataset containing files from 3 malware families which are Winwebsec, Zbot and ZeroAccess. The files contain all the opcodes that form the binary file. Hence, I am going to classify malware using machine learning method based on the opcodes of the malicious files given.

Data preprocessing

Before we can feed our data to a model, it must first be converted into a format that the model can understand. The malicious files have been already disassembled and are now in innocuous *txt* files. Furthermore, these files are stripped for everything but the opcodes. This is because of machine learning algorithms take numbers as inputs. This means that I have to convert the opcodes into numerical vectors.

In order to feed the data in an appropriate format, I have created a tokenization-based sequence matrix as the input dataset. Tokenization is the process of breaking down large pieces of text into smaller ones. Tokenization divides the raw text into words and sentences, which are referred to as tokens. These tokens aid in the comprehension of the context or the development of the NLP model. By evaluating the sequence of words, tokenization aids in interpreting the meaning of the text. Tokenization can be accomplished using a variety of methods and libraries. For this project, I have done tokenization with the library of Keras from Tensorflow. I have used texts_to_sequences to tokenize the opcodes. On top of that, before the tokenization of the data, I have created multiple datasets by combining different families of malware for different experiments. Such datasets I have created are as follows:

- Winwebsec with Zbot
- Winwebsec with ZeroAccess

- Zbot with Winwebsec
- Zbot with ZeroAccess
- ZeroAccess with Winwebsec
- ZeroAccess with Zbot
- A combination of Winwebsec, Zbot, and ZeroAccess

After tokenization of the data, I have split the data into training and validation sets. During the split, I used 80 percent of the samples for training and the remaining 20 percent for validation before the data is fed into the model.

Tuning

For the purpose of this project, using a mix of embedding, LSTM, and dense is the best architecture. Because I am dealing with a sequence analysis problem in a space of moderate dimension, utilizing Long short-term memory (LSTM) models is a natural choice. LSTM is a special kind of RNN, capable of learning long-term dependencies. LSTM models are powerful when it comes to sequence analysis but they are also tricky to use. To make sure I have chosen the right parameters for our network, I varied multiple parameters of the network and projected the average performance overall values.

Apart from that, the reason why I have chosen embedding with LSTM in this project is because an embedding is a relatively low-dimensional space into which you can translate high-dimensional vectors. Machine learning on big inputs, such as sparse vectors representing words, is made easier via embeddings. An embedding should, in theory, capture some of the input's semantics by clustering semantically related inputs together in the embedding space. On top of that, the reason why I used a dense layer after the LSTM layer is due to the fact that the dimensionality of LSTM output is equal to the number of units, which is probably not the dimensionality of the desired target. That is why I have to specify a dense layer in the model.

Below is the table containing the optimized hyperparameters that I used for the model from my grid search results. Due to the fact that my gird search results are quite large, I will not be included in the report. It perhaps might not be the best hyperparameters but I will try to use a hyperparameters optimizer like Optuna if time permits.

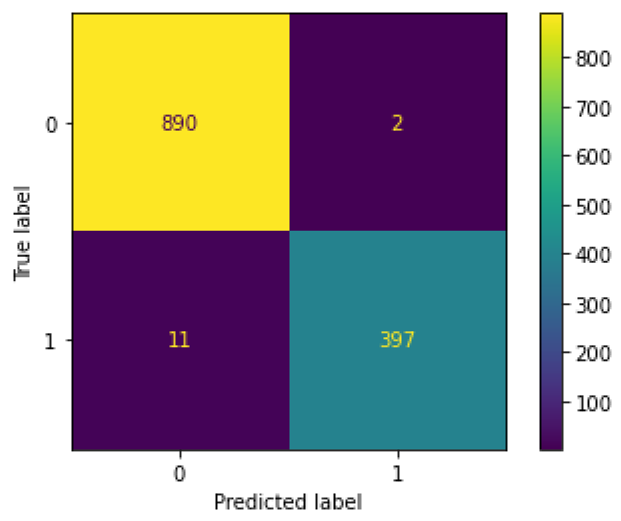
Embedding	Yes	
LSTM	units	32
	dropout	0.1
Dense	units	1
	activation	sigmoid

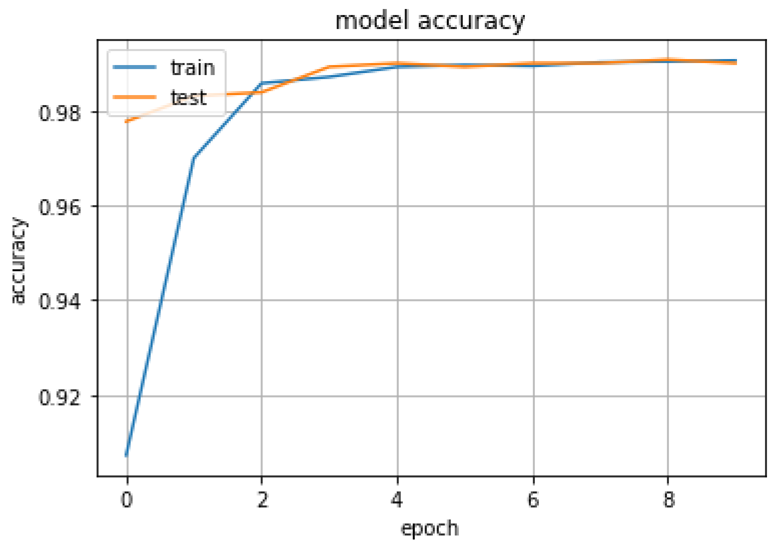
Experiments

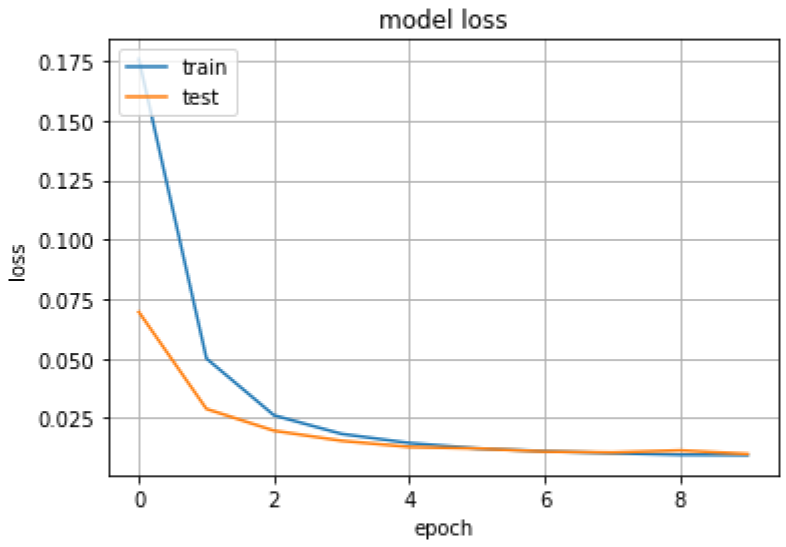
There are a total of 7 experiments with the model. All the results are presented in a confusion matrix and graph. Below are the results of each experiment I have done with the model.

Test 1: Winwebsec (0) with Zbot (1)

```
Epoch 10/10
6/6 [==============================] - 3s 483ms/step - loss: 0.0094 - accuracy: 0.9906 -
val_loss: 0.0099 - val_accuracy: 0.9900
```

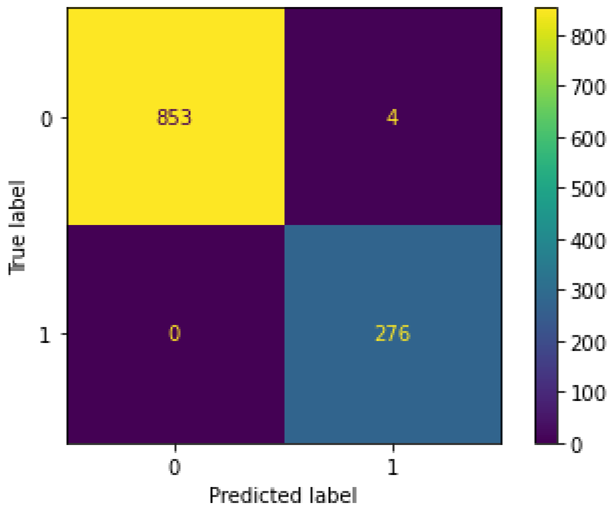


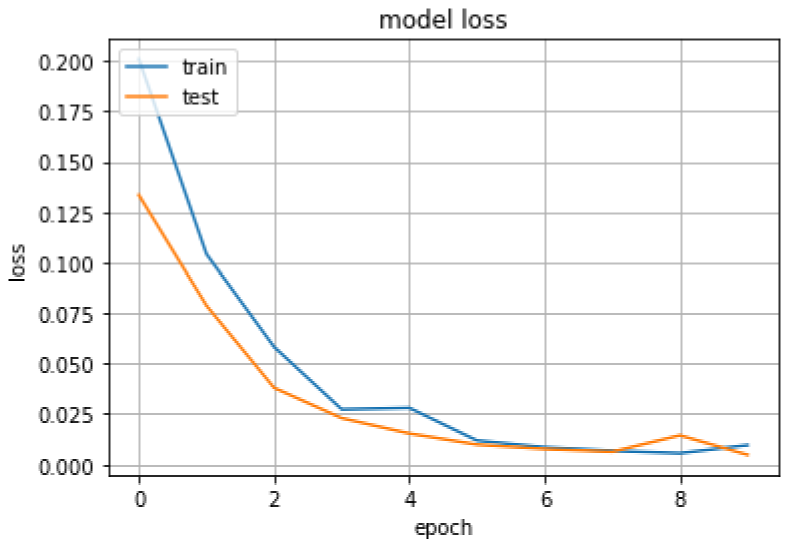




<u>Test 2: Winwebsec (0) with ZeroAccess (1)</u>

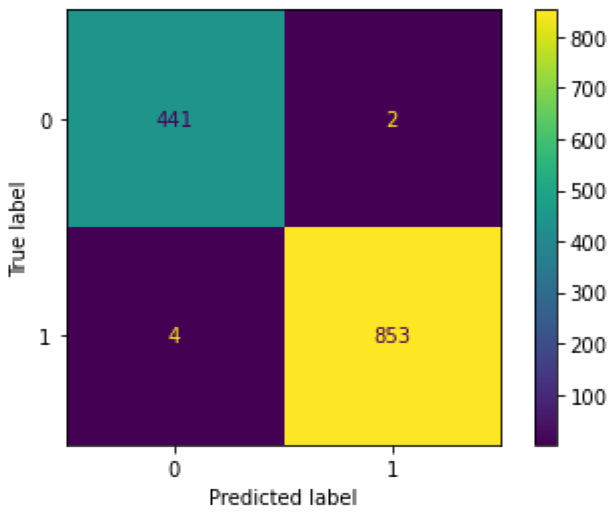
```
Epoch 10/10
5/5 [==============================] - 3s 580ms/step - loss: 0.0095 - accuracy: 0.9907 -
val_loss: 0.0047 - val_accuracy: 0.9965
```

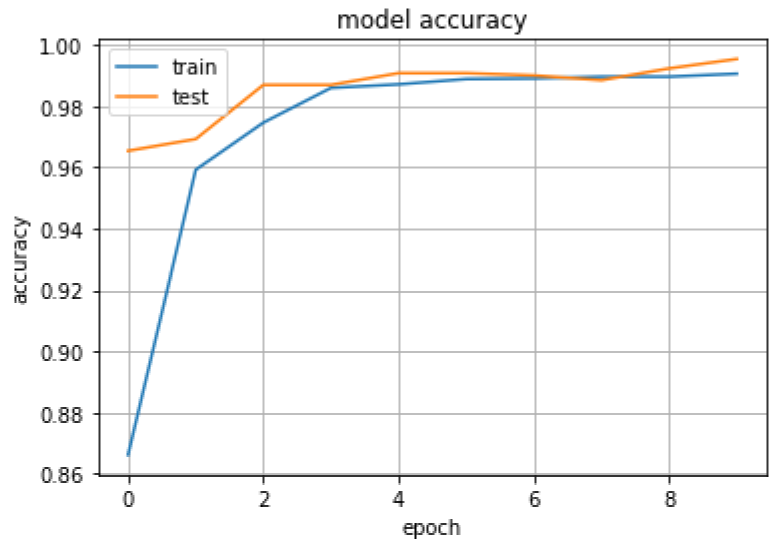


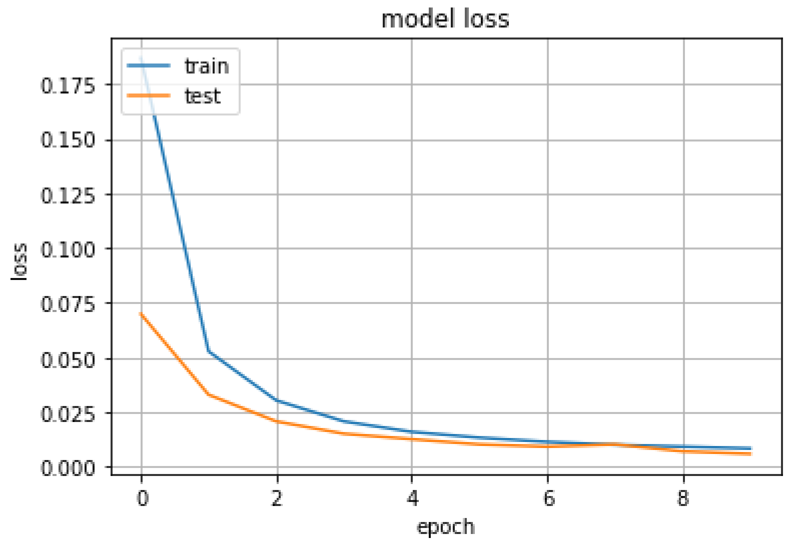


Test 3: Zbot (0) with Winwebsec (1)

```
Epoch 10/10
6/6 [==============================] - 3s 496ms/step - loss: 0.0085 - accuracy: 0.9906 -
val_loss: 0.0060 - val_accuracy: 0.9954
```

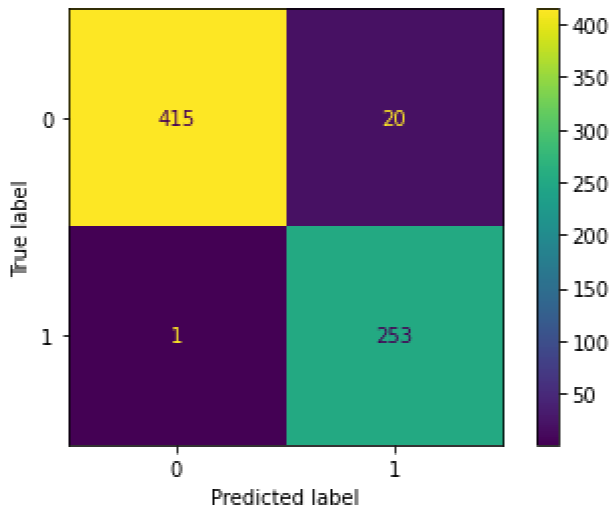

441
2
4
853
0
1
True label
0
1
Predicted label
100
200
300
400
500
600
700
800

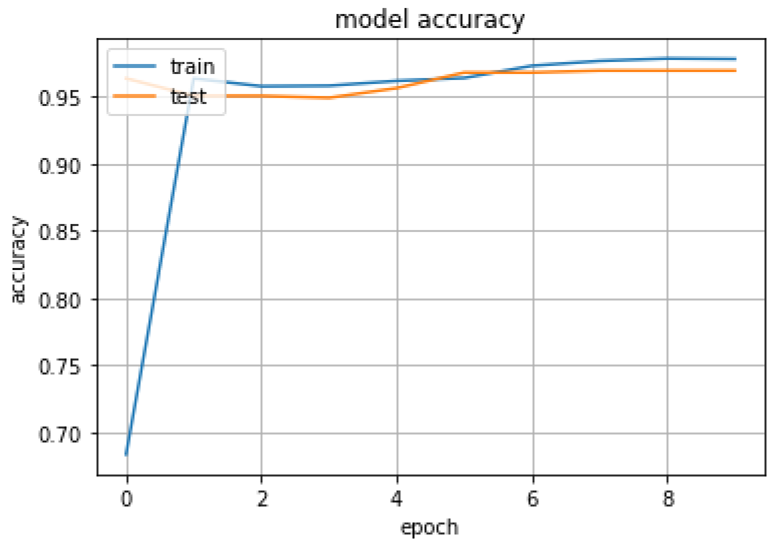

model accuracy
train
test
accuracy
epoch

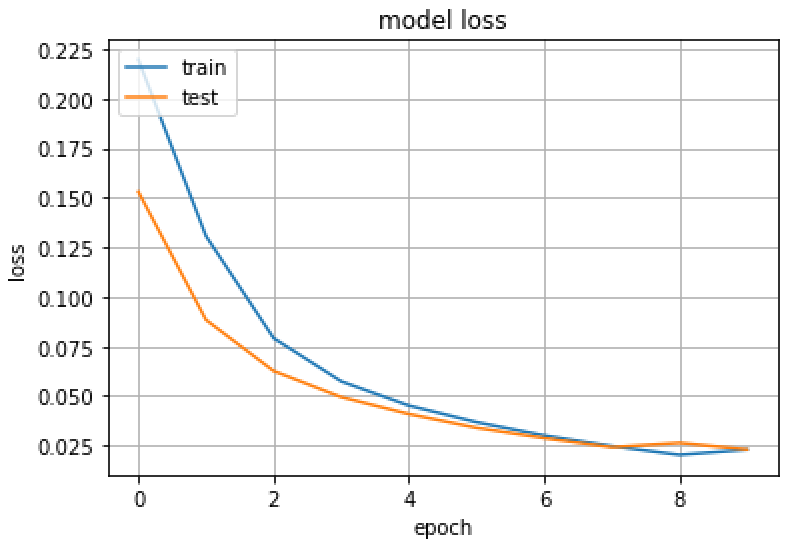

model loss
train
test
loss
epoch

Test 4: Zbot (0) with ZeroAccess (1)

```
Epoch 10/10
3/3 [==============================] - 2s 543ms/step - loss: 0.0229 - accuracy: 0.9782 -
val_loss: 0.0230 - val_accuracy: 0.9695
```

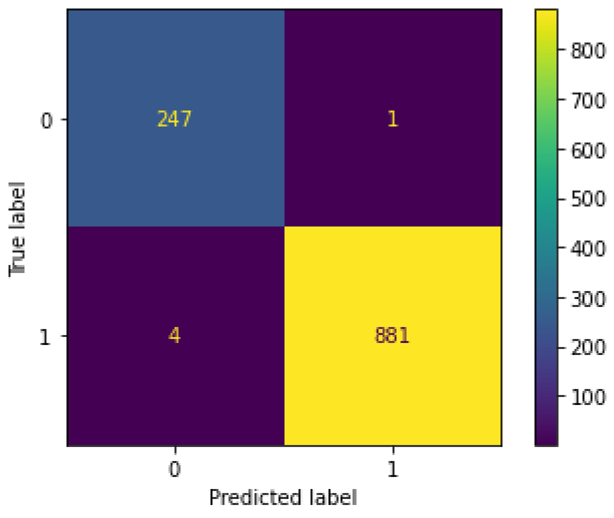


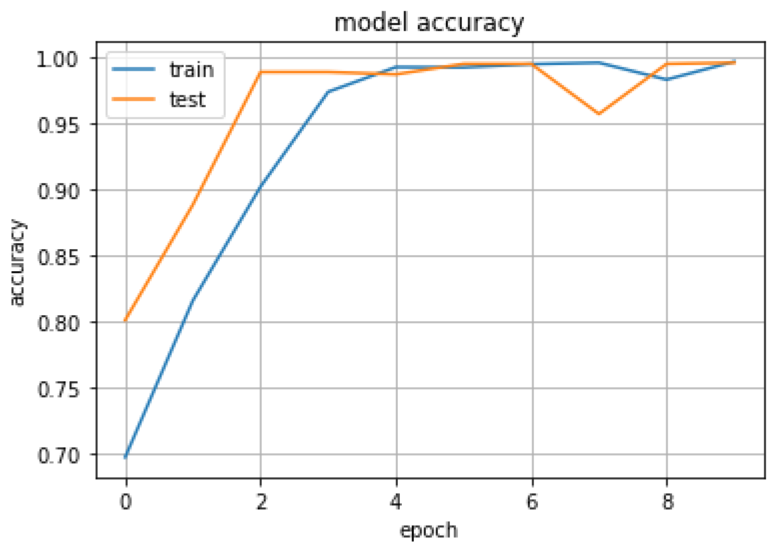


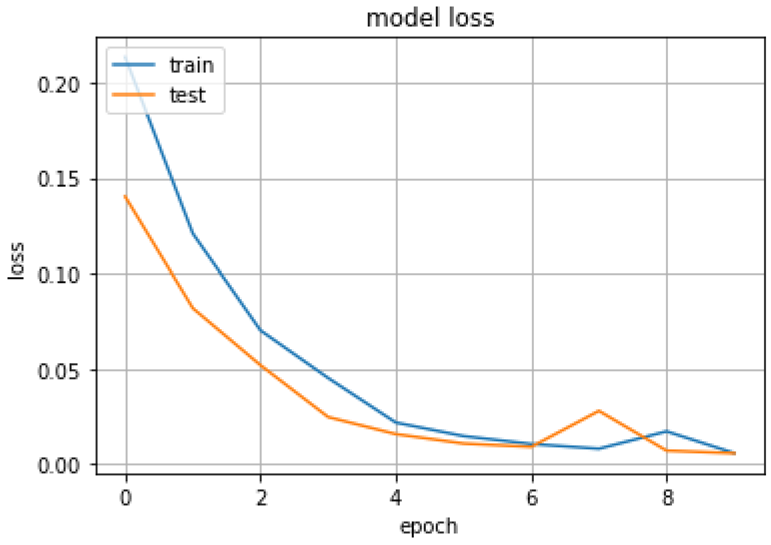


<u>Test 5: ZeroAccess(0) with Winwebsec (1)</u>

```
Epoch 10/10
5/5 [==============================] - 3s 591ms/step - loss: 0.0058 - accuracy: 0.9965 -
val_loss: 0.0058 - val_accuracy: 0.9956
```

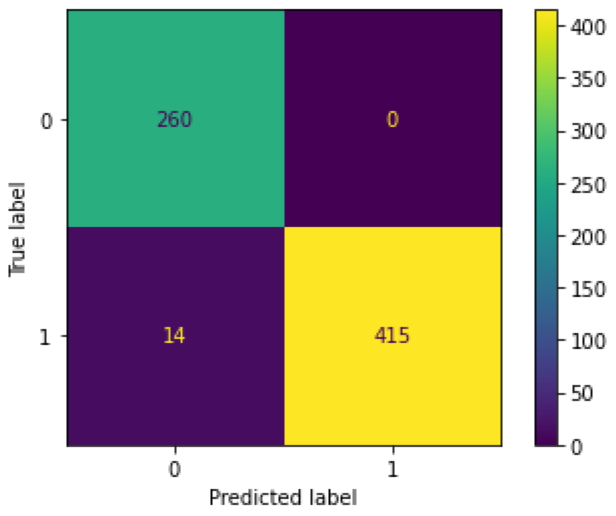




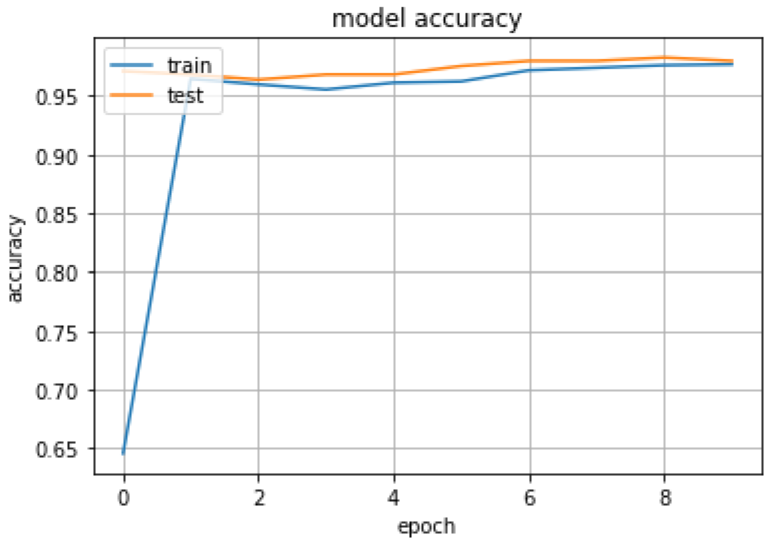


<u>Test 6: ZeroAccess (0) with Zbot (1)</u>

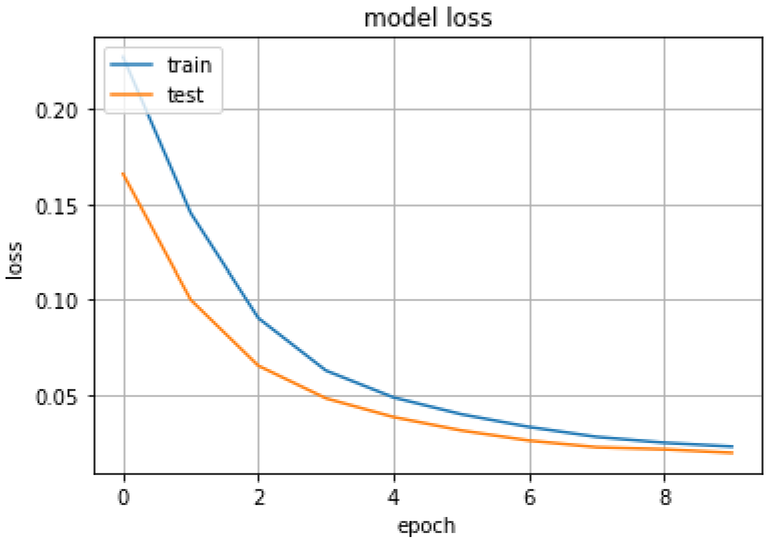
```
Epoch 10/10
3/3 [==============================] - 2s 755ms/step - loss: 0.0228 - accuracy: 0.9767 -
val_loss: 0.0195 - val_accuracy: 0.9797
```



260
0
14
415
True label
Predicted label
0
1
400
350
300
250
200
150
100
50
0



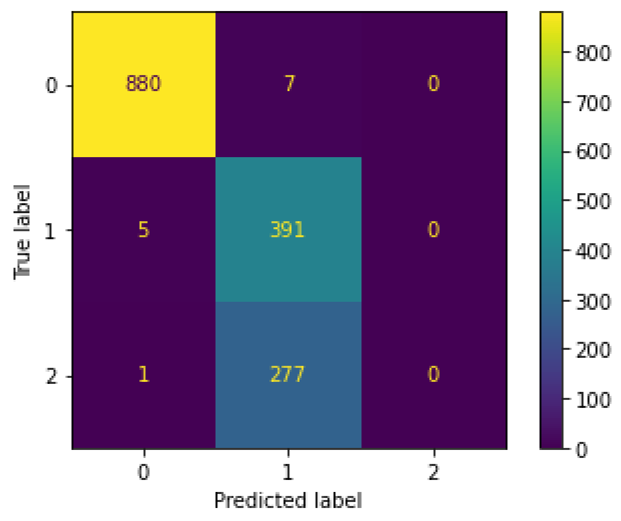
model accuracy
train
test
accuracy
epoch
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
2
4
6
8

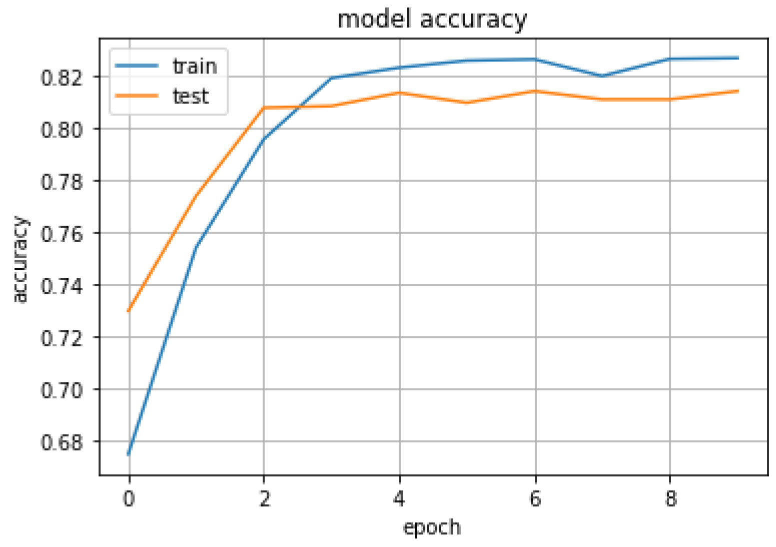


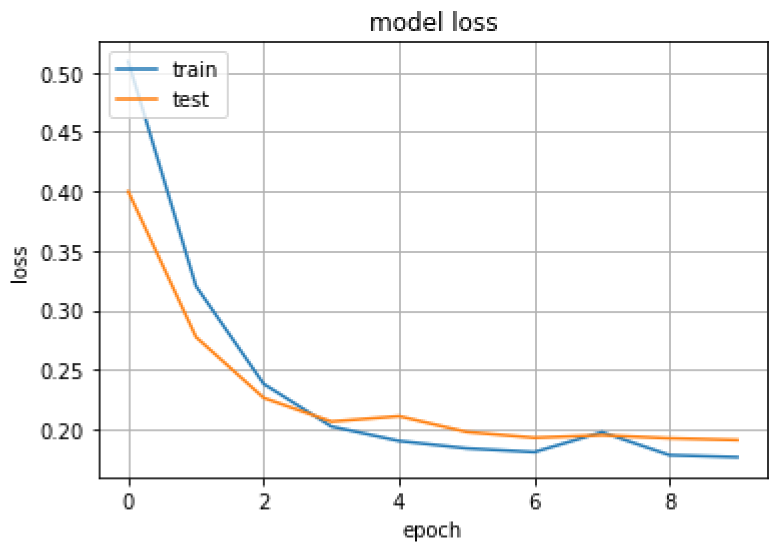
model loss
train
test
loss
epoch
0.20
0.15
0.10
0.05
0
2
4
6
8

Test 7: Combination of Winwebsec (0), Zbot (1), ZeroAccess (2)

```
Epoch 10/10
3/3 [==============================] - 4s 278ms/step - loss: 0.173 - accuracy: 0.827 -
val_loss: 0.1836 - val_accuracy: 0.8164
```







Test	**Validation accuracy (%)**
#1 (Winwebsec with Zbot)	99.00
#2 (Winwebsec with ZeroAccess)	99.65
#3 (Zbot with Winwebsec)	99.54
#4 (Zbot with ZeroAccess)	96.95
#5 (ZeroAccess with Winwebsec)	99.56
#6 (ZeroAccess with Zbot)	97.97
#7 (Combination of Winwebsec, Zbot and ZeroAccess)	81.64

Test	**Average validation accuracy (%)**
#1 (Winwebsec with Zbot)	98.74
#2 (Winwebsec with ZeroAccess)	99.43
#3 (Zbot with Winwebsec)	99.19
#4 (Zbot with ZeroAccess)	97.21

#5 (ZeroAccess with Winwebsec)	99.15
#6 (ZeroAccess with Zbot)	98.44
#7 (Combination of Winwebsec, Zbot and ZeroAccess)	82.97

Conclusions

In the nutshell, from the obtained results, all the validation accuracies are quite close to the train accuracies. It is true to say that test 2, Winwebsec with ZeroAccess, has the highest average validation accuracy, i.e. 99.43 percent. Despite the fact that the malware families have an imbalanced distribution in the dataset, it has been demonstrated that this does not influence classification performance. Apart from that, it is true to say that the overfitting of the model is prevented.

Hence, for future work, I plan to use artificial neural networks other than LSTM as it might give better results. On the other hand, I also would like to implement n-gram for data pre-processing with LSTM and other artificial neural networks such as Bidirectional Encoder Representations from Transformers (BERT) and so on. If time allows, I think that Support Vector Machine is probably a good idea to try out with this dataset. Moreover, I would like to use a hyperparameter optimizer such as Optuna for future work.

Appendix

https://github.com/cheejunwong/Malware-Classification-using-LSTM