Packer Detection using visualisation

MSc Research Project

Cyber Security

Norman Kolarikkal

Student ID: x19226365

School of Computing

National College of Ireland

Supervisor: Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Norman Kolarikkal ………………………………………………………………………………

Student ID: X19226365 ……………………………………………………………………………

Programme: Cyber Security ……………………………………………… **Year:** 2020-2021 ………………………

Module: Msc Research Project ………………………………………………………………………………

Supervisor: Imran Khan ………………………………………………………………………………

Submission Due Date: 16-August-2021 ………………………………………………………………………………

Project Title: Packer detection using visualisation ………………………………………………………………………………

Word Count: 5137 ……………………**Page Count**……………19……………………………….…

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Norman Kolarikkal ………………………………………………………………………………

Date: 18-09-2021 ………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Packer classification using visualization methods.

Norman Kolarikkal

X19226365

Abstract

Malware is software for harming a computer system. Current methods for detecting malware heavily use signatures such as hashes. However, these methods can easily be deceived using methods such as packing. We, therefore, proposed use of visualization and Convolutional Neural network (CNN) model to detect and classify packers as well to detect if a packed sample is malicious or benign. We would be converting image to a RGB image and then use CNN on the images to classify packed samples. Our model was able to work on multiple types of files with us testing our algorithm on exe files and apk files.

1 Introduction

The main goal of malware is to gain unauthorized access or to deny access to systems. As per (Johnson, n.d.), by March 2020, there were about 677 million reports of new malware samples being found. Malware analysts face a great challenge due to the increased number of malware samples.

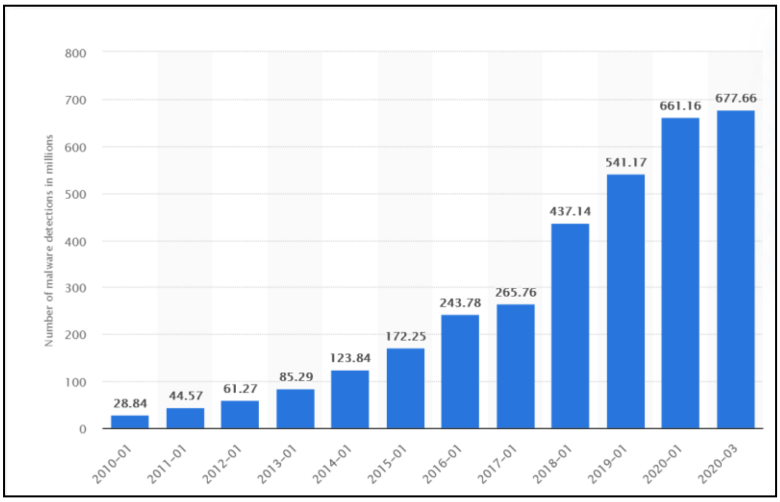

Figure 1: Cumulative detection of malware worldwide from 2015-2020 (Johnson, n.d.)

Static and dynamic analysis are the traditional methods to detect malware. Static analysis is the method of collecting information such as file size of the malware, its hash signature, etc. without running the malware. Malware authors/ attackers have, however, started to hide the malicious code using tools such as packers, crypters, or protectors (Arntz, n.d.). When file is executed, packers will unpack/extract the code from the file. Crypters use code obfuscation and encryption to fool detection methods (Arntz, n.d.). Attackers usually use both packers and crypters to prevent reverse engineering of the malicious code (Arntz, n.d.). Automated packer tools such as UPX are easily available and made it easy for attackers to create malware requiring a complex skillset to reverse engineer the malware. Static analysis is disturbed by code obfuscation through methods such as compression and encryption.

Information collected by running malware in sandbox is called dynamic analysis. Information such as network calls and registry changes can be easily identified using dynamic analysis. Dynamic analysis is more robust against code obfuscation methods; however, they usually require more resources than static analysis.

Recently, visualization methods have been proposed to analyze malware binaries. The visualization methods have proven to be effective because most variants of malware binaries are generated using automated technology or reusing some modules (Fu et al., 2018).

Visualization algorithm also face a similar issue that packed samples usually make it difficult to get all the bytes.

As the malware variants increases, it is becoming more important that research is done on the evasion techniques such as packers. (Rahbarinia et al., 2017) surveyed both malware and benign software samples and found that about 58% of malware samples and 54% of benign samples used well-known packers. Around 35% of the malware samples used custom packers. Therefore, we propose to use the visualization method to first convert the file to an RGB-coloured image and then use CNN to train the model to recognize the patterns of well-known packers. This would help in analyzing the malware samples as it can detect if a particular sample requires unpacking. In case it requires unpacking, it could also determine which tool was used for packing, if a well-known packer were used.

1.1 Research Hypothesis

We would be trying to find out if our visualisation model would be able to find out if a file is packed and detect which packer was used. Therefore, our null hypothesis (H_{10}) and alternative hypothesis ($H_{1\alpha}$) would be as follows:

H_{10}: Visualisation algorithm would not be able to detect packer.

$H_{1\alpha}$: Visualisation algorithm would be able to detect packers.

We also want to check if our algorithm would be able to detect if a packed sample is a malware. Therefore, our second hypothesis is as follows:

H_2: Proposed visualisation algorithm would not be able to detect malware even if the sample is packed.

$H_{2\alpha}$: Proposed visualisation algorithm would not be able to detect malware even if the sample is packed.

1.2 Research Question

From our research hypothesis, the following are the research questions we are trying to investigate:

Q1: How effective will using a CNN algorithm with visualization be in detecting and classifying packers?

Q2: How effective will the proposed algorithm be in classifying malware from benign samples even if they are packed?

1.3 Conclusion

The remaining sections of the research paper are as follows: Section 2 discusses the previous research carried out. The method followed by us for the research is discussed in section 3. The design is described in Section 4, while Section 5 covers the implementation of the artefact. Section 6 covers the evaluation of our model while we discuss the areas of future research in section 7.

2 Related Work

We discuss the recent works on visualisation, malware and packer detection in this section. Sub-section A contains the related works on malware detection. We discuss recent methods researched on packer detection in sub-section B. Finally, we discuss visualisation methods researched in sub-section C. Sub-section C also describes the use of visualisation method in malware classification and packer classification as well.

A. *Malware detection*

Traditional malware detection depends on signature and behaviour analysis. However, using packers, polymorphic and encoding techniques the malware can fool signature detection. Dynamic analysis is more robust compared to static analysis; however, the cost poses an issue. Moreover, newer variants of malware are able to remain dormant in case the malware detects sandbox environment (Baker, 2020).

(Saurabh, 2018) did malware analysis using advanced static analysis (analysis for strings and use of disassembler is used to load linked libraries and imported functions) and advanced dynamic analysis (Advance debugging on malware along with registry analysis). They used PEid to determine the packer used. However, PEid can only determine common/well-known packers. As mentioned before, in the study by (Rahbarinia et al., 2017), they found that 35% of the attackers have started using custom packers.

As malware variant have started to remain dormant when a sandbox is detected, (Ijaz et al., 2019), found that dynamic detection had given lower accuracy than static. They also explain the limitation of static analysis is packed files. Similarly, (Murali et al., 2020) were able to show how newer malware variants can avoid dynamic analysis by stopping the execution of malicious code if it detects a debugging attempt. While static analysis would be able to overcome this issue, the difficulty of reverse engineering often reduces the effectiveness of static analysis.

Recent research is being done on hybrid analysis to detect malware where both static and dynamic analysis is used along with the use of machine learning models (Hadiprakoso et al., 2020; Kuo et al., 2019). (Kuo et al., 2019) discovered about 5% increase in the detection of android malware using hybrid analysis. (Kuo et al., 2019) similarly found an accuracy of 88% in malware detection using their model. However, as discussed earlier, the presence of packers has caused the accuracy of the models to drop (Fu et al., 2018).

B. *Packer detection methods*

Signature detection is one of the main methods to detect malware. However, detection using signature is no longer viable due to the use of custom packers. (Omachi & Murakami, 2020) using k-nearest neighbor algorithm of entropies were able to identify 125 out of 253 packers. However, the research was done on double-packed samples. As discussed in (Kim et al., 2020), current packers are capable of using more than 2 layers. (Kim et al., 2020) also introduces a taxonomy to measure the complexities of the packers based on the layers of packing done by the packers. (Alkhateeb & Stamp, 2019) suggested using Levenshtein distance and naïve Bayes classifier to classify packers. They were able to get a higher accuracy compared to commercially available packer detections. (Hua et al., 2020) tried classifying packed malware based on control flow graphs with them able to achieve 96.4% accuracy in classifying packed files in their tests. They only used the function call graph for their model.

(Sun et al., 2020) suggested running all methods with forged arguments to gather information about the code. However, if the malware is able to detect presence of sandbox, this method may not work. (Korczynski, 2017) unpacked the packed file and reconstructed the file in a format that was easier to analyze. However, the tools used for the research did not allow their use on multilayer packers. (Lim & Nicsen, 2016) used static analysis to detect if a file is packed or not with them being able to achieve an accuracy of 98.16%. They extracted the features of files and scored them according to a predetermined risk and weight.

C. *Visualization methods*

There have been a number of recent research where visualization methods have been used to detect malware and packers effectively. (Corum et al., 2019) were able to distinguish benign pdf samples from malicious pdf samples by converting the file to an image using byte plot and Markov plot. They were able to achieve an F1-score of 99.48% in classifying malicious pdf from benign pdfs. (Kartel et al.,

2020) used visualization and image processing using machine learning to classify malware. They raise an interesting argument against using machine learning in tha tit is possible to misclassify malware files as benign files if they are trained with the wrong data. (Kartel et al., 2020) raised an issue with machine learning that there is a lack of visibility on why the model classified a sample as malware or benign after comparing existing models. (Venkatraman & Alazab, 2017) used feature extraction on known malware and used a threshold-based in the extracted features to then classify a sample as malware. However, the method would be difficult to be implemented in the case of many extracted features due to the limitation on processing power.

(Fu et al., 2018) used an RGB image of the malware to get more information than the grey-scaled images. They populated the red, blue, and green channels with more information (such as entropy and relative size of file) rather than converting a grey-scale image to an RGB image. However, the method suggested was not able to classify packed malware. We would be building our model based on the visualization method suggested in this paper.

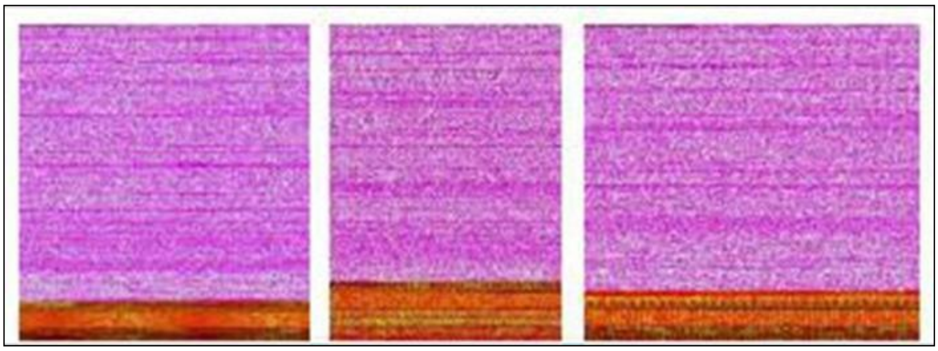

Figure 2: Output of the model proposed by (Fu et al., 2018) on samples belonging to Backdoor.Win32.Hupigon

(Li et al., 2019) use a graph-based approach to classify packers. They also include an updater in their model which goes through the result of the model to improve the system after the training phase. (Donahue et al., 2013) used Markov byte plot to visualize malware samples. They also noted some similarities between packed and unpacked image samples. As shown in Figure 2, both images showed a red line in the middle of the image. We believe that our model could show similar similarities, and this could be a potential route to detect malware without executing the file.

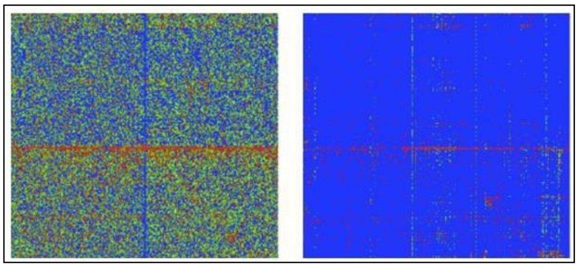

Figure 3: Comparison of packed and unpacked sample images by (Donahue et al., 2013)

However, one of the issues with the existing visualization research done on packed files is that there is no research done on multi-level packers. Our work will also be done on lower-layer level packers.

D. *Research Niche*

As discussed in the previous sections, traditional static analysis methods such as opcode analysis and hash signatures cannot be used for malware that employs tactics such as code obfuscation or polymorphism. Packers are one of the well-known methods of obfuscation used by malware authors. We hope that our model would be able to detect and classify the packers.

Many algorithms using machine learning have been implemented including ones using K-nearest neighbor and naïve Bayes. However, they have been tested for well-known packers such as UPX. Our proposed model will use CNN model along with visualization. We hypothesize that our model would be able to detect and classify well-known packers. We also hypothesize that our model would be able to find similarities between packed and unpacked malware samples. This would help in malware detection and reduce the effort in detection and classification.

The below table contains a summary of the main research papers we went through for our model.

Table 1: Summary of research papers

Research paper	**Summary**	**Comments**
(Fu et al., 2018)	Used GLCM to extract features from a RGB converted image.	We would be using a similar approach for our visualization model. We would be using CNN algorithm instead of using GLCM to extract features from the image.
(Donahue et al., 2013)	Used Markov Byte plot to convert the file to an image. They were able to spot similarities between packed and unpacked samples of the same malware	From the research, we think that using visualization, it would be possible to detect malware even without unpacking sample.
(Li et al., 2019)	They added an updater model to improve the model.	-

Our Contribution as compared to (Fu et al., 2018) is that with our algorithm we should be able to classify files that are not PE files which was the main focus of their research. We would be running our algorithm against android malware and benign samples as a proof of concept.

3 Research Methodology

As per our understanding from the related works, we would be using a visualisation approach similar to the one taken by (Fu et al., 2018). However, the main difference would be that we would be using the byte probability occurring in the file as compared to the file sections and that we would be using file size instead of relative section size. The following figure shows a high-level methodology used in the research:

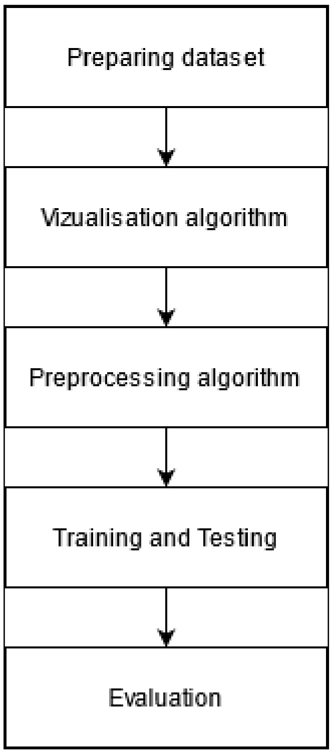

Figure 4: Methodology

3.1 <u>Preparing datasets:</u>

Our dataset mainly consists of malware samples downloaded from virusshare. We mainly focused on malware samples as it would help with our second hypothesis as well. We created a set of 11591 samples. These samples consisted of 3,893 samples of files packed with UPX, 3,893 samples of unpacked files and 3,807 samples of zip files.

We have also downloaded approximately 2000 samples of apk files consisting of malware and benign samples. As checking whether our model would work with files other than PE files is not the main focus of our research, we would be working with a considerably smaller dataset. We would be using the same dataset to evaluate our second hypothesis.

3.2 <u>Visualisation algorithm:</u>

As discussed before, we would be using visualisation algorithm similar to (Fu et al., 2018). Our algorithm is as follows:

Our goal with the visualisation algorithm is to get a RGB image with each channel/colour having a different information such as the occurrence of byte in the file and the size of file. In our algorithm, we convert the byte values in the file and use the values for green colour matrix. The occurrence of byte in the file is passed on the red channel. The size of the file is passed on the blue channel. The process is as shown below:

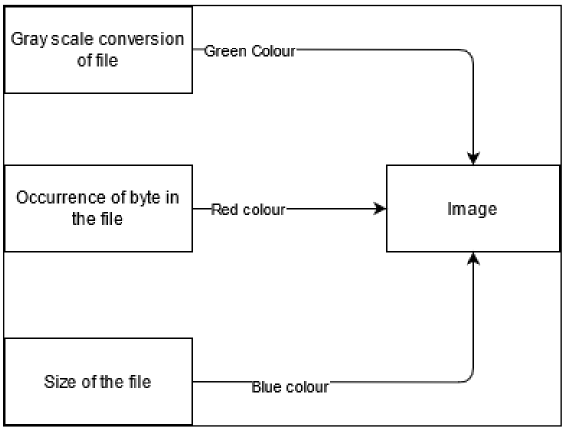

Figure 5: Visualization algorithm

3.3 Pre-processing Image:

For CNN algorithm, we need images of same sizes. To do this, we would be using de-sampling the image to reduce its dimension to a maximum of 128. For images having dimensions less than 128, we resize the images, which is basically to zoom smaller images to the preferred dimension.

3.4 Training and testing

We would be using CNN model with 14 layers. Our dataset would be passed into training and testing with 90% of the images in the dataset being used for training and the rest for validation.

3.5 Evaluation

We would be running our trained model against an evaluation dataset and using accuracy and precision to evaluate our model's performance.

4 Design Specification

A. Models used

For the purpose of this research, we have used a visualisation model and a CNN model for classifying output images from the visualisation model.

Visualisation model:

As mentioned before, we use a similar approach to visualisation as proposed by (Fu et al., 2018). However, as their algorithm was used on PE files, we have done the following changes:

1. We have used the same method for green channel. We use the gray-scale converted image matrix for the green channel.
2. For the Red channel, we have created a matrix having the value of the number of occurrences of that particular byte.
3. For the blue channel, as compared to the method by (Fu et al., 2018), we have used the file size. (Fu et al., 2018) used relative section sizes in the PE file for the blue channel.

CNN model:

We have implemented CNN model using tensorflow keras. It has many packages which help in building a CNN model. We have the following layers:

Table 2: CNN Layers

Layer	Output Shape	Paramameter number
rescaling_1 (Rescaling)	(None, 128, 128, 3)	0
conv2d (Conv2D)	(None, 128, 128, 16)	448
max_pooling2d (MaxPooling2D)	(None, 64, 64, 16)	0
conv2d_1 (Conv2D)	(None, 64, 64, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 64)	0
flatten (Flatten)	(None, 16384)	0
dense (Dense)	(None, 128)	2097280
dense_1 (Dense)	(None, 3)	387

B. Hardware and Software Specifications

Hardware details: We have used a 10^{th} Generation i5 Processor laptop with 8 GB RAM.

Software Details: Below are the list of software used in the research.

1. Python: We have coded our implementation in python. In our research, we use python version **3.8.10.**
2. Jupyter: We ran our code using jupyter. We used version **7.22.0** in our research
3. Python packages: We use multiple available python packages including numpy, tensorflow and PIL. The versions are as below:

Table 3: Package versions

Package	Version
Numpy	1.19.5
PIL	8.2.0
Tensorflow	2.5.0
Matplot	3.3.4

5 Implementation

The implementation for our research is as shown in the below diagram:

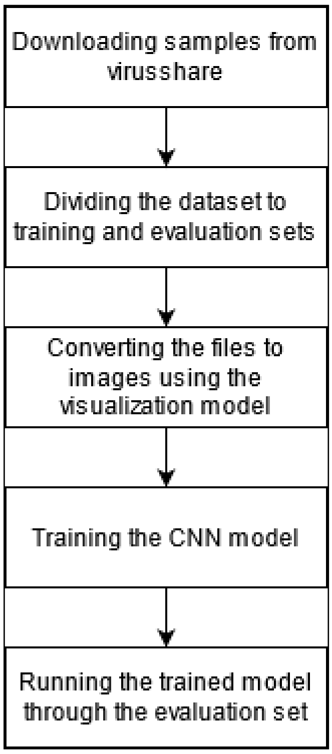

Figure 6: Implementation steps

1. We started by downloading malware samples from virusshare.com and downloaded the set into training and evaluation dataset.
2. We then run visualization model on both the datasets.
3. We used the training dataset to train the CNN model. For the CNN model, the training dataset was further divided into training and validation datasets with a 9:1 split.
4. Finally, once the CNN model is trained, we run the trained model on the evaluation set.

6 Evaluation

A. Hypothesis 1: Would our model be able to detect packers

a. Summary

For the research, we would be focusing on classifying upx packed files and zipped files. Our model had a validation accuracy of 95.34%. The details loss and accuracy of the model is as follows:

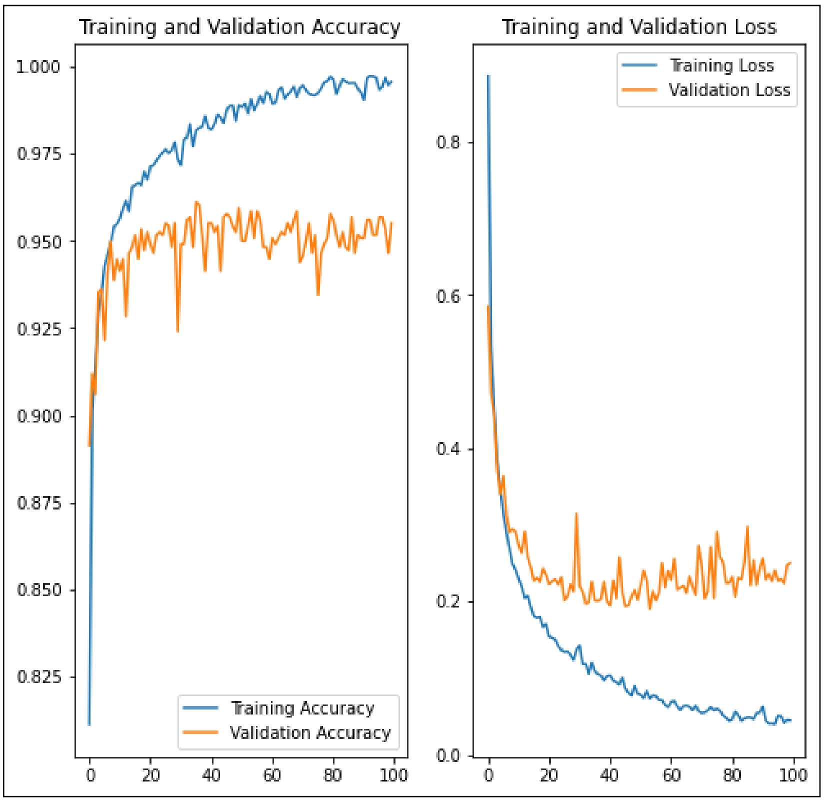

Figure 7: Accuracy and loss of the model

b. Evaluation

For evaluation, we passed a set of 11578 images with around 3000 images of unpacked, upx packed and zip files passed through our visualisation algorithm. Below are the results of evaluation:

Table 4: Evaluation matrix

Class	**Precision**	**Recall**	**Accuracy**	**F1 - Score**
Original	51.564	93.58	68.0473	66.4902
UPX Packed	69.4785	23.27	70.5447	34.8662
Zip	94.1082	79.3	91.7247	86.0741

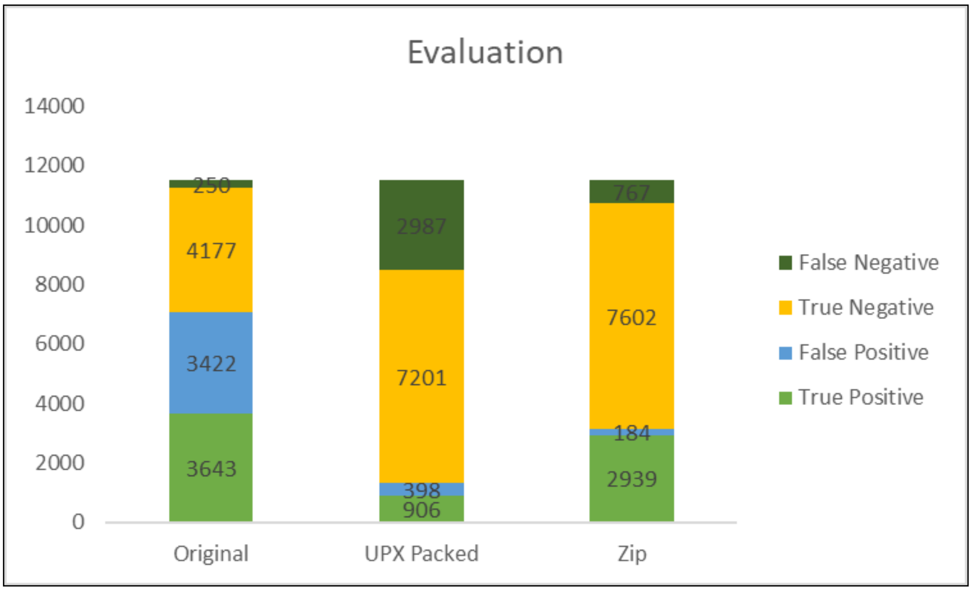

Figure 8: Results of evaluation

From figure 8, our proposed algorithm was able to classify unpacked samples from upx packed and zipped files. Out of 3893 unpacked files, our algorithm was able to correctly detect 3643 samples as unpacked, while only 906 samples out of 3893 were correctly detected as packed samples and 2939 samples of 3792 were successfully detected as zipped samples.

We compare our results on packer classification with the research by (Omachi and Murakami, 2020). They used k-nearest neighbour on the entropy of the files. With the method, they were able to successfully identify 125 out of 253 samples. Our trained model was able to correctly classify 7488 samples correctly out of 11578 samples.

B. Hypothesis 2: Would our model be able to classify malware samples from benign samples

a. Summary

As discussed before, we would be running apk file samples to check for this hypothesis. We got a validation accuracy of 81% with our model. Details are as follows:

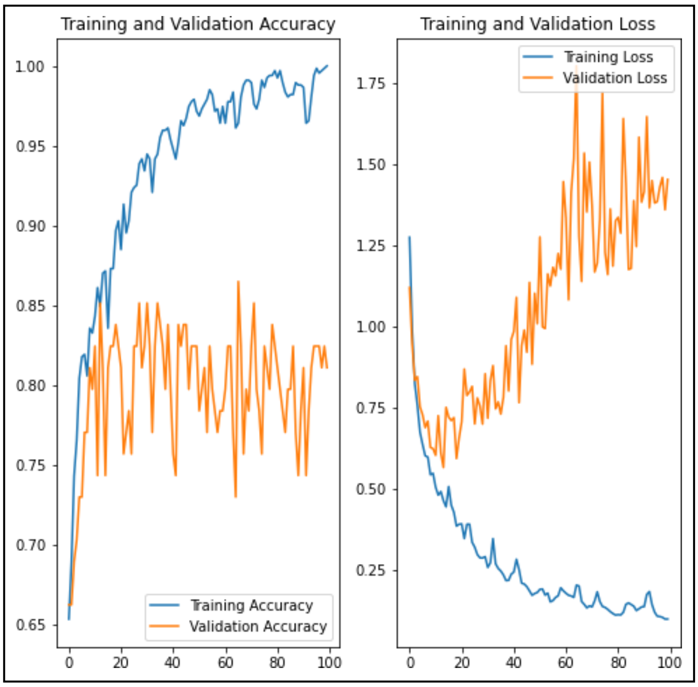

Figure 9: Hypothesis 2 result

b. Evaluation

For evaluation purpose, we passed a set consisting of 10 malware apk samples, 10 zipped malware apk samples and 10 benign apk samples. The details of the samples used are as follows:

Table 5: Evaluation dataset description

Filename	File type	Predicted	Actual
a7ddc91af4d63163ec942d4297cb254519dfe38a8bafceafafbf9630ff7f2d32.8c32a989e1f5eed4ef6ed4a0dc3fd6ae.zip.png	Zipped apk	malware	malware
a8054925585483e6dbc0ca0366a5460d3f9f5909f491e465f479adb112e5db54.8f47cf382b9c48c4c08eca5caa5b9e1c.zip.png	Zipped apk	malware	malware
a8726d22a5480651c395c88d9cc24c0cc7d8b2d2d626526454ef96daf0bcd999.09dc1c81c827f274e2dbf99a542af83d.zip.png	Zipped apk	malware	malware
a8e029ea800433fe6fc6ebcc677f922387d8fff07871c5097ba5d7bed70ac15f.2bf4fe977c684fc019a3b21a53999bf7.zip.png	Zipped apk	malware	malware

a96c8c2977aae28bf37a5576c45af326c50e6684c5191116c4862fcffb33aea5.2bb4d95bc46c30644a82263eeffe4b31.zip.png	Zipped apk	malware	malware
a975260826bb19a9f4735b77b2e02a947f94f9785c8b30d73410b67d73678090.9f82a19cd754b944b490c5ac6d432477.zip.png	Zipped apk	malware	malware
aa0fb35aad2b6beb77cbdacd07828806513dae1975be030060fe703cce1d9054.0d1366528bf2276fdc686b1f3deb38e5.zip.png	Zipped apk	malware	malware
aa2be0ac79028bf59168218894a61c9990630bac3054416d4185714f6aa33eff.37f7e5e37f40f05e371903ec76b2d01a.zip.png	Zipped apk	malware	malware
ab0a824f00e4aee68a17dad86182b6ffe83d6d7d07d572d31183cf4a8c1723da.13efad96f27cd583f37b4c22f3af4e6a.zip.png	Zipped apk	malware	malware

com.jb.gosms.pctheme.venus.apk.png	apk	malware	benign
com.jetcost.jetcost.apk.png	apk	benign	benign
com.jk.dailytext.apk.png	apk	malware	benign
com.joris.uokay.apk.png	apk	malware	benign
com.josegd.monthcalwidget.apk.png	apk	malware	benign
com.joshbegley.dronestream.apk.png	apk	malware	benign
com.jrummy.apps.google.play.api.apk.png	apk	malware	benign
com.mercury.wpad.apk.png	apk	malware	benign
com.metro.bangalore.apk.png	apk	malware	benign
com.mg.ola.shortcut.apk.png	apk	malware	benign
com.mikeperrow.spiral.apk.png	apk	malware	benign
com.miteksystems.android.mobiledeposit.brandable.rbcgeorgia.apk.png	apk	malware	benign

com.mobilerise.mystreetview.apk.png	apk	malware	benign
com.moneris.pos.apk.png	apk	malware	benign
com.muhanov.apk.png	apk	malware	benign
fec9aa964070c3044edd7ab8d2ac819a799e99feff7d058b2befc07f97cc67e5.apk.png	apk	malware	malware
fef1b6eb8d7285c53176b64119cacc879d9c634436fea81738d973695bd45711.apk.png	apk	benign	malware
ff09fd183d2f7a1e94951852c01c5e2303f6f24e80587879bc6f5f7b81a2afbd.apk.png	apk	malware	malware
ff21b2149e5f4e915bbf3f94ddd12555f2a8dd8f5ec0a666410c6eb7c87513c4.apk.png	apk	benign	malware
ff82cc0c97dc6588f4b26533c0fa8a88752f6795b6d51fff38fdeb85c82f2051.apk.png	apk	malware	malware
ffc588993173d8b4a19a9ee87888d53f1b13c957e47a89027439deb73ad3ba4d.apk.png	apk	malware	malware
ffd5efd46b2b861b8b11e71c6ba00ea891c854029ebb407500c38aed86e9e880.apk.png	apk	malware	malware
ffdae5b124392d6c22806f1126487c219b360c444072ea62110edc16e109f5a4.apk.png	apk	malware	malware
ffe64c04d97cf22ce30a3d43df9bfff084834f1b2cbaf4890f96153264a9997e.apk.png	apk	malware	malware
fff8e36e72ca18a049929c7f2f584f57b6fa2a03dfbe40fbc491b2e06e58edac.apk.png	apk	malware	malware

The result of the run is as follows:

Table 6: Evaluation matrix

Class	True Positive	False Positive	True negative	False negative	Precision	Recall	F1-score
malware	17	14	1	2	54.83871	89.47368	68
benign	1	2	17	14	33.33333	6.666667	11.11111

From the evaluation dataset, we could see that the algorithm had a lower accuracy for benign programs. This is to be expected, as the number of samples used were considerably lower than the dataset used for packer classification. However, we were able to classify zipped files as malware which supports our second hypothesis.

(Donahue, Paturi and Mukkamala, 2013) used a Markov Byte plot to convert PE file to an image. They were able to give an example where they found similarities in packed and unpacked malicious samples. We were able to use our method to classify samples as malicious or benign. Our method was able to detect zipped files as malicious as well. Moreover, we were able to classify apk files as well instead of just PE files.
Similarly, the method proposed by (Fu et al., 2018) classified samples as malicious or benign. One of their limitations was that their algorithm only classified unpacked PE files. With our method, we were able to classify both packed PE files and non-PE files as well. However, we had a lower accuracy at 81% on classifying malicious files as compared to the 97.47% accuracy obtained by (Fu et al., 2018). We believe that increasing the dataset would help in increasing the accuracy in classifying malicious samples as we used a smaller dataset of 2000 samples as compared to 7000 samples used by (Fu et al., 2018).

C. Discussion

As compared to the method in [3], our model has a lower accuracy. This is to be expected as we generalized the visualisation method to fit multiple types of files instead of just PE files. We were able to run our algorithm against a smaller dataset of apk files consisting of malicious and benign samples. We were able to get a validation accuracy of 81% with our model. We believe that increasing the number of samples in the dataset would help increasing the accuracy of our model comparable to the accuracy we received for exe files.

After running the model against evaluation dataset, we could see that our model was able to correctly predict zipped files more precisely compared to packed files. However, original files had more than both zipped and packed files.

We were also able to classify zipped malware apk supporting our second hypothesis that our algorithm would be able to classify packed malware samples.

From our understanding, using a fully convolutional network (FCN) would give a better result as we would be able to use the images without reshaping or downsampling. Downsampling and reshaping images could result in information getting lost. However, as our images had high resolution, the computing power required for running FCN model increased as well.

7 Conclusion and Future Work

From the results of our tests, we were able to support both our hypothesis that our model was able to classify packed files as well as classify packed malicious files as malware. We were able to detect the packers used by using CNN and visualisation. From the output of our model, we were able to observe an accuracy of around 95% while classifying the exe files and around 81% while classifying apk files.

Future research could be done on using the image without reshaping the image from the output of visualisation model. Another area of interest would be the information used for visualisation algorithm. Future research could be done on entropy of the file instead of size or the occurrence of byte in the file. We also believe that the method used could be able to classify malware samples without unpacking samples. We believe this is also a good area for future research.

8 References

Alkhateeb, E. M., & Stamp, M. (2019). A Dynamic Heuristic Method for Detecting Packed MalwareUsing Naive Bayes. *2019 International Conference on Electrical and Computing Technologiesand Applications, ICECTA 2019*. https://doi.org/10.1109/ICECTA48151.2019.8959765

Arntz, P. (n.d.). *Explained: Packer, Crypter, and Protector - Malwarebytes Labs | MalwarebytesLabs*. Retrieved April 13, 2021, from https://blog.malwarebytes.com/cybercrime/malware/2017/03/explained-packer-crypter-and- protector/

Baker, K. (2020). *What is Malware Analysis?* https://www.crowdstrike.com/cybersecurity-101/malware/malware-analysis/

Corum, A., Jenkins, D., & Zheng, J. (2019). Robust PDF Malware Detection with Image Visualization and Processing Techniques. *Proceedings - 2019 2nd International Conference onData Intelligence and Security, ICDIS 2019*, 108–114. https://doi.org/10.1109/ICDIS.2019.00024

Donahue, J., Paturi, A., & Mukkamala, S. (2013). Visualization techniques for efficient malware detection. *IEEE ISI 2013 - 2013 IEEE International Conference on Intelligence and Security Informatics: Big Data, Emergent Threats, and Decision-Making in Security Informatics*, 289–291. https://doi.org/10.1109/ISI.2013.6578845

Fu, J., Xue, J., Wang, Y., Liu, Z., & Shan, C. (2018). Malware Visualization for Fine-Grained Classification. *IEEE Access*, *6*, 14510–14523. https://doi.org/10.1109/ACCESS.2018.2805301

Hadiprakoso, R. B., Kabetta, H., & Buana, I. K. S. (2020). Hybrid-Based Malware Analysis for Effective and Efficiency Android Malware Detection. *Proceedings - 2nd International Conference on Informatics, Multimedia, Cyber, and Information System, ICIMCIS 2020*, 8–12. https://doi.org/10.1109/ICIMCIS51567.2020.9354315

Hua, Y., Du, Y., & He, D. (2020). Classifying Packed Malware Represented as Control Flow Graphs using Deep Graph Convolutional Neural Network. *Proceedings - 2020 International Conferenceon Computer Engineering and Application, ICCEA 2020*, 254–258. https://doi.org/10.1109/ICCEA50009.2020.00062

Ijaz, M., Durad, M. H., & Ismail, M. (2019). Static and Dynamic Malware Analysis Using Machine Learning. *Proceedings of 2019 16th International Bhurban Conference on Applied Sciences and*

Technology, IBCAST 2019, 687–691. https://doi.org/10.1109/IBCAST.2019.8667136

Johnson, J. (n.d.). • *Global new malware volume 2020 | Statista*. Retrieved April 13, 2021, fromhttps://www.statista.com/statistics/680953/global-malware-volume/

Kartel, A., Novikova, E., & Volosiuk, A. (2020). Analysis of Visualization Techniques for Malware Detection. *Proceedings of the 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering, EIConRus 2020*, 337–340. https://doi.org/10.1109/EIConRus49466.2020.9038910

Kim, J. W., Namgung, J., Moon, Y. S., & Choi, M. J. (2020). Experimental comparison of machine learning models in malware packing detection. *APNOMS 2020 - 2020 21st Asia-Pacific Network Operations and Management Symposium: Towards Service and Networking Intelligence for Humanity*, 377–380. https://doi.org/10.23919/APNOMS50412.2020.9237007

Korczynski, D. (2017). RePEconstruct: Reconstructing binaries with self-modifying code and import address table destruction. *2016 11th International Conference on Malicious and Unwanted Software, MALWARE 2016*, *anno 2008*, 31–38. https://doi.org/10.1109/MALWARE.2016.7888727

Kuo, W. C., Liu, T. P., & Wang, C. C. (2019). Study on android hybrid malware detection based on machine learning. *2019 IEEE 4th International Conference on Computer and Communication Systems, ICCCS 2019*, 31–35. https://doi.org/10.1109/CCOMS.2019.8821665

Li, X., Shan, Z., Liu, F., Chen, Y., & Hou, Y. (2019). A consistently-executing graph-based approachfor malware packer identification. *IEEE Access*, *7*, 51620–51629. https://doi.org/10.1109/ACCESS.2019.2910268

Lim, C., & Nicsen. (2016). Mal-Eve: Static detection model for evasive malware. *Proceedings of the 2015 10th International Conference on Communications and Networking in China, CHINACOM 2015*, 283–288. https://doi.org/10.1109/CHINACOM.2015.7497952

Murali, R., Ravi, A., & Agarwal, H. (2020, February 1). A Malware Variant Resistant to Traditional Analysis Techniques. *International Conference on Emerging Trends in Information Technologyand Engineering, Ic-ETITE 2020*. https://doi.org/10.1109/ic-ETITE47903.2020.264

Omachi, R., & Murakami, Y. (2020). Packer Identification Method for Multi-layer Executables with k-Nearest Neighbor of Entropies. *Proceedings of 2020 International Symposium on InformationTheory and Its Applications, ISITA 2020*, *C*, 504–508.

Rahbarinia, B., Balduzzi, M., & Perdisci, R. (2017). Exploring the Long Tail of (Malicious) Software Downloads. *Proceedings - 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017*, 391–402. https://doi.org/10.1109/DSN.2017.19

Saurabh. (2018, December 1). Advance Malware Analysis Using Static and Dynamic Methodology. *2018 International Conference on Advanced Computation and Telecommunication, ICACAT 2018*. https://doi.org/10.1109/ICACAT.2018.8933769

Sun, C., Zhang, H., Qin, S., Qin, J., Shi, Y., & Wen, Q. (2020). DroidPDF: The Obfuscation ResilientPacker Detection Framework for Android Apps. *IEEE Access*, *8*, 167460–167474. https://doi.org/10.1109/access.2020.3010588

Venkatraman, S., & Alazab, M. (2017). Classification of Malware Using Visualisation of Similarity Matrices. *Proceedings - 2017 Cybersecurity and Cyberforensics Conference, CCC 2017*, *2018-Septe*, 3–8. https://doi.org/10.1109/CCC.2017.11