Ransomware Detection & Identification Using AI

by
Leon Wiskie

To obtain the degree of Master of Science at
The Hague Graduate School.

Ransomware Detection & Identification Using AI

This page is intentionally left blank

Ransomware Detection & Identification Using AI

By
Leon Wiskie

To obtain the degree of Master of Science at
The Hague Graduate School.

STUDENTNUMBER - 18110177
PROGRAM - CYBER SECURITY ENGINEERING
SUPERVISOR- DR. S.C.A PETERS
THE HAGUE GRADUATE SCHOOL
DATE - 11-11-2020

This page is intentionally left blank

The Hague Graduate School

Acknowledgements

We gratefully acknowledge the support of VirusTotal for providing the malware samples used in this research.

Table of Contents

List of Figures

List of Tables

Abstract

Malware is a constantly evolving and rising threat, especially ransomware, a form of malware. The rise of ransomware as a service platform adds to this surge, and malware researchers need options to swiftly and reliably identify a family of ransomware to protect the data of individuals and vital infrastructures.

In this study we provide an image-based detection and classification method that can aid researchers in identifying the origins of ransomware by comparing it to known ransomware families. We aimed to reach a high level of accuracy and a low false positive rate on a given ransomware sample using a limited-size training dataset and COTS hardware.

We used a dataset of 347,307 Windows executable malware samples obtained from VirusTotal (VT). These samples were collected by VT between 2017 and 2020. From this dataset we selected samples positively identified as known ransomware.

We applied a novel AI-driven approach to classify ransomware based on an image representation of the binary file. This approach has been used by security practitioners and academics on malware in general but not on particular types of malware like ransomware.

We used a naïve approach to selecting the best-performing convolutional neural networks based on 16 of the available applications in Keras, a Python API for the TensorFlow opensource machine learning platform. These applications were pre-trained on the ImageNet dataset of natural images.

The proposed method achieves above **90%** accuracy and a high recall rate for a high probability of detection of ransomware based on three channel (RGB) images. The best-scoring models on our dataset were *MobileNet* and *MobileNetV2*.

Keywords: Ransomware, Computer vision, Deep learning, CNN, Machine learning

1 Introduction

This chapter introduces the background to the study, defines the problem, and articulates the research questions.

1.1 Background of the study

Ransomware is a form of malware (Hassan, 2019) that is increasingly used by criminals to extort large sums of money from individuals and especially companies. Worldwide, it is seen by researchers as a major threat in an increasingly digitizing society (Kaspersky, 2020; McAfee, 2019; Trend Micro, 2020).

Recently a large university in the Netherlands fell victim to a ransomware infection that resulted in the unavailability of IT systems for students and employees and caused great social unrest ("Maastricht University paid ransomware attackers ransom," 2020). In 2017, WannaCry had a disruptive effect on the non-virtual world, including the operations of the British National Health Service (NHS); people in the United Kingdom may have died as a result of the unavailability of data necessary to provide care (Ghafur et al., 2019).

1.2 Problem field analysis

Various researchers (Chen, 2018; Ganapathi and Shanmugapriya, 2020) argue that artificial intelligence (AI) can make a positive contribution in protecting society against the increasing threat posed by ransomware. We consider ransomware detection and family identification as two separate tasks, following the classification of malware in the literature. We define "families" as instances of ransomware with the same origin or a likely common origin and a strong similarity in code.

1.3 Research aim and objectives

The purpose of this research is to (1) contribute to existing knowledge in the field of identifying unknown ransomware using AI and (2) propose a practical model that can be used by malware analysts when detecting and initially identifying ransomware. We aim to achieve a high level of accuracy and a low false positive rate in identifying ransomware and classifying a given sample into a known ransomware family.

1.4 Central research questions and associated questions

To reach the aforementioned objectives, we have formulated research questions. The main research question of this study is:

How can deep learning be used to detect and identify ransomware?

To answer the main research question, two theoretical sub-questions and one empirical sub-question have been formulated:

1. *What are the current methods for detecting and identifying ransomware?*
2. *What is deep learning and deep transfer learning?*
3.
    a. *How can a pre-trained model be implemented for detecting and identifying ransomware?*
    b. *How effective is this implemented model?*

1.5 Limitations of the study

The scope of this research is limited to developing a prototype of a detector. The implementation of the detector and the measures that may be taken to limit the effects and consequences of ransomware are beyond the scope of this study. This study is focused on detecting and classifying Microsoft Windows–based malware because the Windows operating system is one of the most common targets of ransomware (Kaspersky, 2020; McAfee, 2019; Trend Micro, 2020).

1.6 Structure of thesis

This report begins by summarizing current literature on malware analysis and similarity analysis among malware and ransomware binaries in Chapter 2; from this past research, the research gap for present study is identified. In Chapter 3 we introduce our approach to bridge the gap. Then in Chapter 4 we discuss the method for scientific research used on this subject. In Chapter 5 we describe the experimental environment and the fine-tuning of the convolutional neural networks (CNNs) before they can be applied on our use case. In Chapter 6 we evaluate models for detection and identification of ransomware using image-based analysis, and a novel method based on deep transfer learning and MobileNet CNNs to answer the main research question. In Chapter 7 the conclusion of the thesis is summarized and answers to the research questions are formulated. Finally, in Chapter 8, we discuss the literature and methodology used in our study and the overall outcome. This document has three appendices; the first contains an explanation of the files in the dataset used in our research, the second contains the confusion matrices of the MobileNet models applied on the test data and the last contains a description of the system used during the experiments, an overview of the software and scripts used and the link to the data repository.

2 Literature review

In this chapter, a literature review is conducted covering the key concepts from the research questions formulated in Chapter 1 and discussing their relevance to this research.

2.1 Ransomware detection & identification

In this study, ransomware is defined according to the description used by the Dutch Nation Cyber Security Center (NCSC): Ransomware is software that encrypts computer files so that users no longer have access to them. Only after payment of the ransom will data or documents be made accessible again (MITRE, 2020; NCSC, 2020).

However, there are no guarantees that files will actually be accessible again after payment. Systems are infected with ransomware through files opened by an end user via email or by visiting infected websites. The infection and further spread occur because the malware exploits vulnerabilities in the operating system, as the well-known WannaCry ransomware did (Talos, 2017). Malicious actors can deploy ransomware against many different targets. Any system that contains valuable data can be an interesting target, and attackers often demand more ransom from large companies, government agencies or sensitive data (Loman, 2019).

Vendors and developers of anti-malware products use detection methods such as signature, heuristic or behaviour-based detection (Amro and Alkhalifah, 2015; Sikorski and Honig, 2012). In practice, these methods often turn out to be insufficiently effective to detect new ransomware or variants of existing ransomware. Gilbert et al. (2019) have conducted an extensive literature review of the existing methods of ransomware detection.

2.1.1 Similarity between families

Looking for similarity between files-based entropy and hashing for detection is a well-known technique applied by all AV vendors. However, the rapid increase in malware makes this kind of signatures-matching process decreasingly efficient and poorly scalable (Chen, 2018). Developers of malware makes changes to it so that the signatures change, aiding in evading detection.

Researchers (Ganapathi and Shanmugapriya, 2020; Kaspersky, n.d.; Nataraj et al., 2011) argue that different families have functional similarities that can be found in decompiled code, making it possible, in principal, to detect them based on the similar features across a family. Algorithms can be taught to recognize the similarities when binaries of malware are converted into images, and we predict this can be done with ransomware, given that it is a type of malware and that families of ransomware also have similarities in code. Figure 1 shows samples of benign Microsoft Windows executables that are not related.

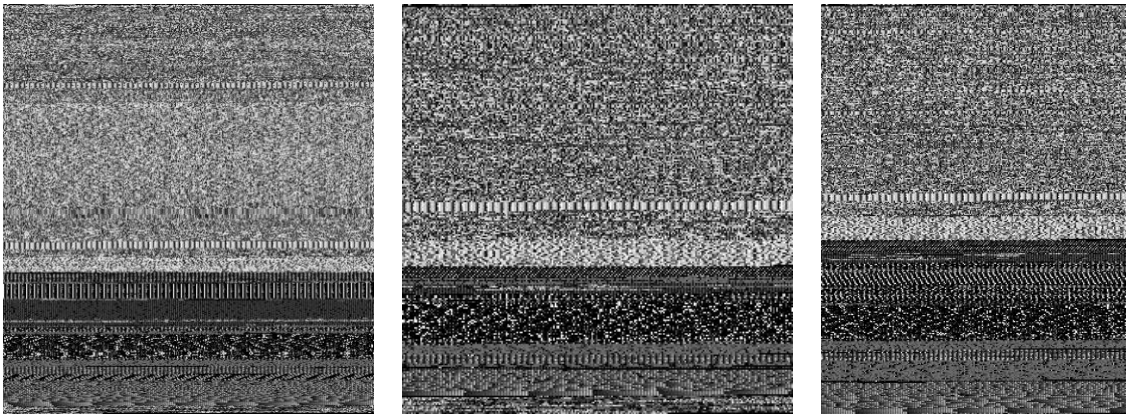


Figure 1 Images of unrelated benign samples

Figure 2 shows images of samples of the Cerber ransomware family.

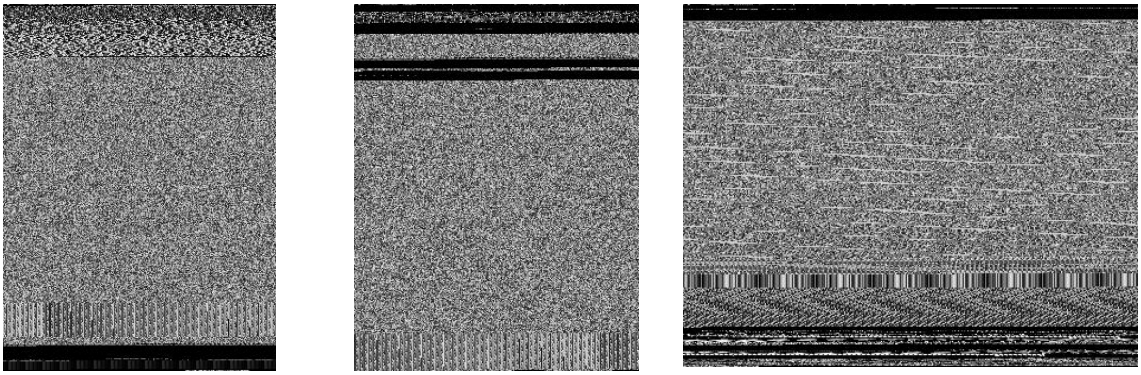


Figure 2 Images of related Cerber samples

2.1.2 Machine Learning

Several studies have shown that the malware responsible for file encryption (referred to in this study as ransomware) can be reliably detected and identified using machine learning (ML) algorithms and techniques (Al-rimy et al., 2018; Nataraj et al., 2011).

There are two aspects that make detecting ransomware with ML and further investigation relevant. First, ransomware is becoming more sophisticated and, due to the application by its creators of better anti-detection features, more difficult to detect (Dargahi et al., 2019). Second, ransomware is evolving so fast that it often cannot be identified reliably using current detection techniques. Both factors ensure that malware developers can easily bypass detection and thus increase the chance of infection and damage.
A reliable method for analysing ransomware is static analysis, in which a file is converted into machine code and then thoroughly analysed by a human for known patterns. This requires specialist knowledge from an analyst and is considered to be very time-consuming (Chen, 2018; Nataraj et al., 2011). Machine learning can support analysts in classifying a file as benign or malware (Al-rimy et al., 2018, p.; Johns, 2017).

Nevertheless, researchers state that there are limitations to the use of traditional ML algorithms for this classification application, of which feature engineering is the most important.

Machine learning algorithms (MLAs) rely heavily on **feature engineering**, feature selection and feature representation techniques that require an extensive domain-level knowledge (Vinayakumar et al., 2019). We derived our definition of feature engineering from the works of Kubat (2017) and Sharma et al. (2019):
"Feature engineering is the process of looking at relevant features in a dataset that can be used by an ML algorithm"

We define traditional machine learning algorithms based on linear regression like support vector machines (SVMs) and K-nearest neighbour (K-NN) algorithems as shallow learning. In general, there are two types of application of ML, and the literature indicates that these can assist analysts in identifying malware (Vu et al., 2019):

1. *Feature-based on machine code and/or behaviour* (Kolosnjaji et al., 2017);
2. *Imaged-based* without *feature engineering* (Han et al., 2013);

In this study, the second method, introduced by Han et al. (2013), is further investigated. It is chosen, firstly, because it can be used to detect malware reliably and then identify it without having to convert the software to machine code (disassembly) or run it. Not having to interact with the binaries possess advantages because it (1) limits the chance of infecting the system used for analysis and (2) reduces the overall complexity of the analysis process. Secondly, it has proven to be robust enough against the aforementioned anti-analysis methods used by ransomware developers (Al-rimy et al., 2018; Vu et al., 2019). Examples of these anti-analysis methods are obfuscation, anti-disassembly, anti-debugging and anti-virtual machine, all of which prevent analysis as well as traditional feature-based identification of malware (Sikorski and Honig, 2012).

2.1.3 Summary

There are several applications deploying ML-based malware detection techniques that may be useful in ransomware detection. Of these, image-based detection appears to be the most promising, as it may offer a solution to the problem described in the introduction of this study.

2.2 AI and Deep Learning

Machine learning, introduced in the previous section, is a sub-domain of artificial intelligence (AI). Artificial intelligence is a research field that has existed since the 1950s, and its most recent evolution is deep learning (DL) (Müller and Bostrom, 2016).

2.2.1 Deep Learning

Deep learning is sometimes referred to in (popular) science as AI. Artificial intelligence is applied in various domains, including information security (Sewak et al., 2018). The following definition of DL is used in this study:

"A class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification." (Deng and Yu, 2013) *(p.199)*

Deep learning makes use of deep neural networks. This form of neural networks (NN) is based on the functioning of the neural networks of the human brain (Krohn et al., 2020). With DL, it is possible to develop algorithms that can be applied to solve (multi-) classification problems, such as distinguishing between good and malware and distinguishing different malware families without having to manually apply feature engineering in advance, which, as previously mentioned, is necessary for traditional ML algorithms (Hardy et al., 2016; Krohn et al., 2020).

Several existing studies classify malware based on an image. These methods are derived from computer vision (CV), which "looks" at an image of a binary (Vu et al., 2019; Yan et al., 2018) and the textures and patterns it contains. An example of this process is shown in Figure 3, below. Nataraja et al. (2011) use this same method, except in a traditional MLA, they apply an SVM algorithm to a grayscale-converted malware binaries dataset.

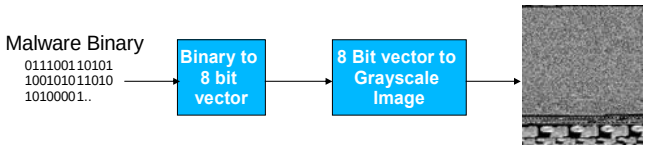


Figure 3 High level process of conversion of a binary file to image (Hassan, 2019)

2.2.2 Summary

Artificial intelligence in the form of deep learning is most often used to analyse malware. This application could also be useful to achieve the aim and objectives of the present study, given the hypothesis that ransomware families have similar properties in the form of patterns that can be recognized by an algorithm in a pixel representation of a binary file. This hypothesis has in any case been frequently confirmed by researchers for machine code (Sharma et al., 2019).

2.3 Convolutional Neural Networks

There are DL architectures that can be used for various applications (Hosseini et al., 2020). A convolutional neural network is an NN architecture that is deployed at DL in various domains, the best known of which is photo or image classification (Cawsey, 1998; Krizhevsky et al., 2012). A CNN is therefore frequently used in the aforementioned CV research domain (Baltrušaitis et al., 2018). The CNN architecture is based on the functioning of the human visual cortex (Krohn et al., 2020).

Deep learning algorithms with a CNN architecture are mentioned in several studies (Hardy et al., 2016; Vu et al., 2019; Yan et al., 2018) as a reliable application of ML to distinguish malware from benign software using feature engineering techniques such as image gists and Scale Invariant Feature Transform (SIFT) (Xie et al., 2017).
Because these types of NN are primarily intended to recognize patterns in data using the aforementioned techniques, the NN "learns" by itself the features of the input data using a technique called back propagation (Géron, 2019; Krohn et al., 2020).

In 2017, research was conducted by the FireEye company into CNN architectures that are used in a detection system. This method was found to be suitable for classifying malware packaged in Microsoft Windows Portable Executable (PE) files. In the FireEye study (Coull and Gardner, 2019), these files, before they were presented to the detector, were converted to grayscale representations as described in the previous section.

The method used was a form of supervised learning in which a person had pre-labelled what in this case consisted of malware and benign software (Kubat, 2017). FireEye's research looked at which features were identified by the ML model as interesting and which a malware analyst would find relevant in traditional static analysis. The scores achieved by manual classification and by the algorithm were compared. The accuracy of this model was **98%,** and a dataset of 3 million benign and 3 million malware samples was used.

2.3.1 Architecture

As described earlier, NNs have an architecture. A CNN is a network architecture with several layers between the input and the output layer; these are called hidden layers. An architectural drawing is shown in Figure 5, below. In general, the hidden layers consists of convolution, pooling and fully connected layers.

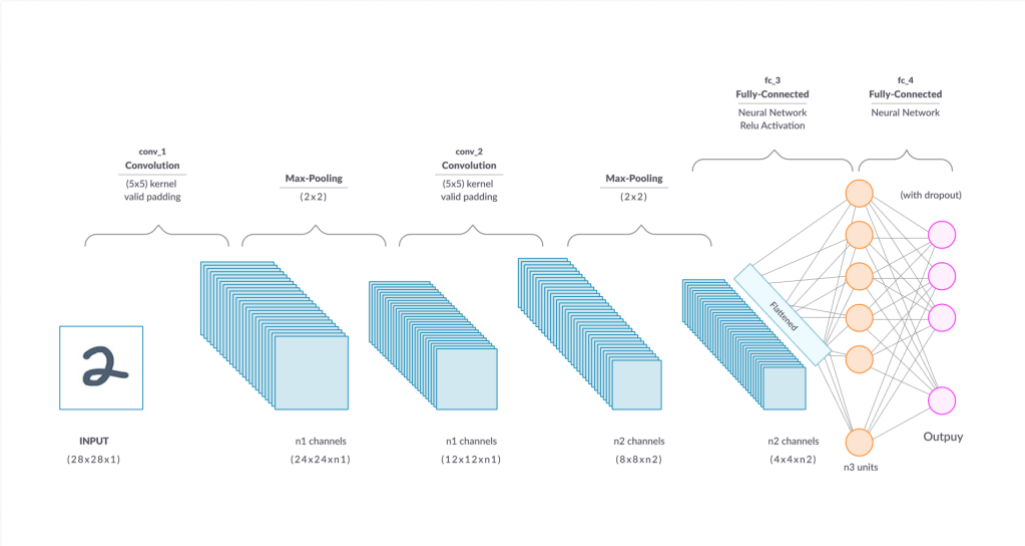


Figure 4 Typical Convolution Neural Network (Keras,2020)

These hidden layers are responsible for feature extraction, in which the features together form a part or a whole pattern that can be recognized by the algorithm in order to classify a sample as benign or malware (Géron, 2019; Krohn et al., 2020).

Deeper neural networks, in general, prove to have better accuracy then shallow networks, as Simonyan and Zisserman (Simonyan and Zisserman, 2015) and Bohida et al. (Bhodia et al., 2019) have demonstrated. Bohida et al. compared shallow K-NN models with deep ResNet models and proved deep neural networks (DNN) performed better due to the fact there are more features to extract resulting in more parameters to train. The downside of DNN's is the high need of training samples and CPU/GPU resources to train them effectively.

2.3.2 Resource intensive

Deep learning is resource intensive, which means that sufficient (1) training data and (2) hardware in the form of computing power (CPU / GPU) and memory (RAM) must be available to train a model sufficiently (Claufield, 2009; Rawat and Wang, 2017; Vu et al., 2019).
As described above, FireEye used 3 million malware and benign samples in their study to train their model; however, marshalling this amount of training data can be a challenge for independent researchers.

2.3.3 Summary

Deep CNN architectures (DCNN) could contribute to the solution to the problem of this study. Solutions are provided to the limitations in its usability, including the following:

- Deeper networks tend to have better accuracy;
- Large amounts of data are needed to train a CNN in general;
- Training a CNNs takes significant time, and without the right hardware it takes hours or even days.

2.4 Deep Transfer Learning

2.4.1 *Transductive Transfer Learning*

Pan and Yan (Pan and Yang, 2010) describe the different methods of deep transfer learning (DTL). The transductive transfer learning (TTL) method may offer a solution to the aforementioned limitations. There are similarities between the source and target tasks, but the corresponding domains are different. The source domain has a vast amount of labelled data, while the target domain, in this case ransomware, still has relatively little.

The transfer of parameters of the neurons is in this study considered TTL. These parameters are the biases and weights that can be used in ransomware classification to (1) shorten training time and (2) increase the reliability of an algorithm (Tan et al., 2018).

2.4.2 Pre-trained CNN's

The application of TTL is therefore the use of a pre-trained CNN, also referred to as a model, on another application domain. Several pre-trained CNN models are publicly available. These are trained on images from the ImageNet dataset, which contains more than 14 million labelled natural images in 1000 categories (ImageNet Project, n.d.).

There are CNNs implemented in publicly available deep learning software libraries such as TensorFlow (https://www.tensorflow.org) and Keras (https://keras.io).

2.4.3 Models applied in other studies

Table 1 shows the different deep models and their accuracy as described in (scientific) literature and the results of applying transfer learning on a malware dataset.

We can conclude that (1) the models used are very deep, (2) the weights and biases of the aforementioned ImageNet dataset were used and (3) the binaries in the dataset were converted into **grayscale** images. Furthermore, all but one study used the same malware datasets.

Table 1 Models found in literature

Model	Grayscale	Used in Study	Accuracy (%)	Dataset
Xception	Y	(Lo et al., 2019)	99.03	Malimg (9339 samples of malwares belonging to 25 malware types) & Microsoft (10868 samples of malware belonging to 9 malware types)
VGG16	Y	(Kalash et al., 2018)	98.52	Malimg (9339 samples of malwares belonging to 25 malware types) & Microsoft (10868 samples of malware belonging to 9 malware types)
ResNet50	Y	(Khan et al., 2017)	87.98	Microsoft (10868 samples of malware belonging to 9 malware types)
InceptionV3	Y	(Chen et al., 2019)	90.0	Other (10,849 samples of malware and 11,153 benign samples)
Ensemble of InceptionV1	Y	(Chen, 2018)	99.07	Other (584,606 samples of malware and 197,604 benign samples)

2.4.4 Transfer scenarios

There are various scenarios, described by Tan et al. (Tan et al., 2018), in which to use the pre-trained models, including (1) using the features from the model, (2) fine-tuning the model if the application domain is close to the original domain or (3) using a pre-trained model.

2.4.5 Summary

Deploying DTL by implementing a pre-trained model can significantly reduce the training data needed, reduce the overall training time and increase the reliability of the detector. The feature extraction scenario and fine-tuning could be a suitable approach because it is seen by several researchers as a solution for reducing the time required to train a model. Pre-trained models are non-existent in this line of research.

Chen (2018) proposes an approach for applying DTL in the classification of malware binary into two classes (benign or malware). This research is the main inspiration behind our approach. Chen offers the most detailed description of the approach that we found in the literature.

3 Proposed approach

We adapted the approach of Chen (2018), as shown in Figure 5, to **multi-classification** on our ransomware dataset. We use Figure 5 as a **conceptual model** throughout this chapter to clarify the research in conjunction with our approach.

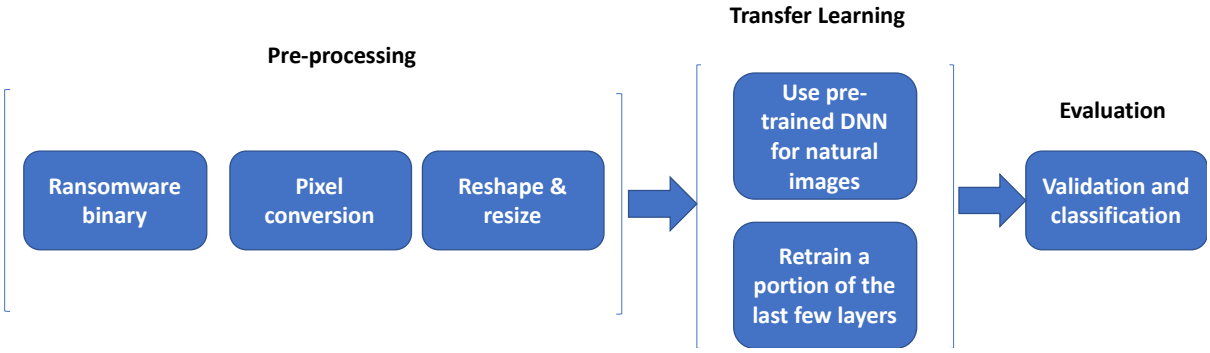


Figure 5 Proposed approach

To add to the validity of the three main phases of this approach, it is believed to be the current state of the art and best practice in (1) data science and (2) in ML projects in general.

3.1 Pre-processing

During pre-processing, the first phase shown in Figure 5, we convert a ransomware binary to an m x m, three channel matrix (RGB). The process and the result are shown in Figure 6. We apply a mean RGB subtraction to normalize the input data for the CNN. For that purpose, we need to calculate the mean value across one whole sample and subtract it from each number in it. In the Keras API we use the *sample-wise center* option for this purpose.

Our method differs from the ones found in the literature because we use **RGB** images instead of the grayscale images used by Nataraja et al (2011). Second, we deviate from his method by not converting binaries in opcode before converting them to images. In this way we avoid (1) the need to identify the packager used and (2) unpacking the binaries, which is an error-prone and time-consuming process, as stated by Gibert et al. (2020).

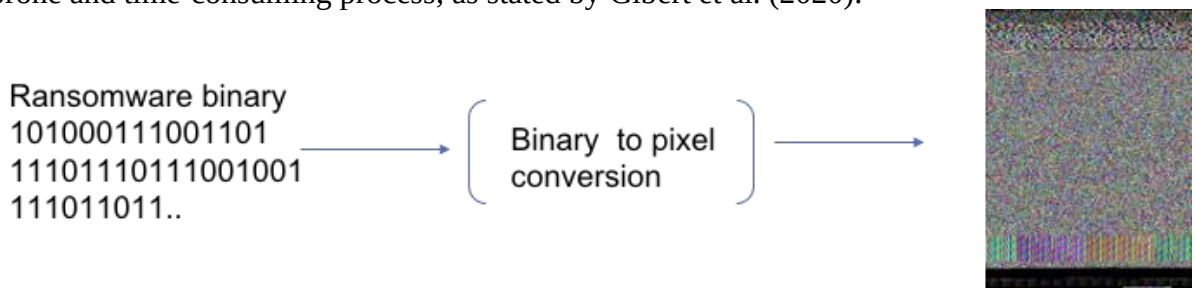


Figure 6 Binary to pixel conversion

We reshape and resize the mostly rectangular images (m x n) of pixel representation of a binary to the default-size square shape (m x m) of a CNN input tensor, following the CNN implementation guides provided by Keras.

3.2 Transfer learning

In the transfer learning phase of the process illustrated in Figure 5, we use a pre-trained CNN on the ImageNet dataset of natural images (ImageNet, n.d).

As stated in the literature review, it is well known that deep neural network topologies lead to better accuracy. However, many of the models have never been used to classify malware or ransomware in the published literature.

We add layers for additional feature extraction and replace the original classification layer of the CNN with a new dense layer of 8 classes with a *Softmax* activation function. This layer is used to classify a sample.

In the transfer learning phase shown in Figure 5, we evaluate the different pre-trained CNNs on the dataset and train the last few layers we added to the CNN. This approach is called *fine-tuning* (Keras, 2020) a model so it can be applied on the dataset and problem or use case of this study.

3.3 Evaluation

We adopt a naïve approach in selecting the best performing CNN implementation by iterating over the different pre-trained and *fine-tuned* models. Finally, in the evaluation phase shown in Figure 5, we evaluate the models' performance, searching for the best results to meet the stated requirements.

3.4 Requirements

We derived four requirements for our solution (the artifact) from literature and best practices in the security practitioner's community. The model used for the detector must achieve at least **90%** accuracy, a low false positive rate (FPR) and the lowest possible false negative rate (FNR). This is achieved in similar studies (Chen, 2018; Kolosnjaji et al., 2017). The detector is considered reliable and useful if it

1. can classify unknown binaries as ransomware;
2. can classify ransomware in different families;
3. can be trained using a limited amount of resources (e.g. number of samples).

4 Research methodology

In this chapter the research method is explained, beginning with design science research (DSR), consisting of the literature review and the description of the empirical part of the research.

4.1 Design science research

In this study, the DSR method is applied to develop a practical solution to the problem described in Chapter 1 and is scientifically validated and evaluated. This determines whether the solution can also be used in the context of the research (Wieringa, 2014). This method of research fits in with the problem defined in the proposed study because it tries to develop an artifact that contributes to or is a solution to a problem that occurs in practice: in this case, detecting and identifying ransomware with limited use of resources.

4.2 Literature review

A literature review was carried out in the form of a systematic literature study to form a theoretical framework and thus be able to answer the theoretical sub-questions. The insights obtained through this process are necessary for drawing up the requirements and ultimately the design of the artifact: a prototype of a ransomware detector and classifier.

The literature review was conducted using sources available through the library of The Hague University of Applied Sciences. A search was made for various keywords from the problem definition and combinations of those keywords, and a pre-selection was made within the literature found. For example, literature was chosen that is no more than five years old because the detection of ransomware and AI is a field that is developing very quickly. In addition, however, a number of fundamental research papers were consulted that are more than five years old.

4.3 Experiment

In the empirical part of the research, an experiment was carried out in which the artifact (detector) was tested, evaluated and validated for usability in a practical situation.

A lab environment was configured to (1) pre-process benign and ransomware samples and divide them into the necessary datasets to train, validate and test the model; (2) train the model for a longer period of time (e.g. a number of hours); and (3) perform the experiment (validating and testing the detector). This was done in several iterations in order to improve the detector and thereby increase its reliability, accuracy and usability.

4.3.1 Detector & classifier

In this study we define a detection system as an output of a single value $y = f(\mathrm{x})$, in the range 0 to 1, which indicates the maliciousness of the executable.
Furthermore, a classification system outputs the probability of a given executable to each output class or family, $y \in \mathbb{R}^n$, where *n* indicates the number of different ransomware families (Gibert et al., 2020).

The detector is developed in the programming language Python (https://www.python.org/) and uses the TensorFlow machine learning framework (https://www.tensorflow.org/) combined with Keras as TensorFlow API also written in Python (https://keras.io/) and further supported by a number of different supporting software libraries. The models implemented in the Keras API will be evaluated during the experiment.

This software stack was selected (1) because it was found in many sources in the literature search and (2) because the researcher has considerable experience with the programming language mentioned. Together, these form an open source framework that is used for the development of the detector and classifier. Using this opensource software stack contributes to the transparency of this research and the usability of the solution.

4.3.2 Constraints

Training a CNN is generally faster if systems are equipped with multiple graphics cards (GPUs) or Tensor Processing Units (TPUs); however, these systems are very expensive and not available for this research. Therefore, the speed of training and classification is not a decisive factor in assessing the results of the study. The most important limitation is in obtaining sufficient samples to compile an adequate ransomware dataset. Extensive malware datasets containing ransomware are generally not shared by anti-malware vendors with independent researchers because they form part of the business model used by these organisations; therefore, researchers must collect samples themselves. This method is used in several existing studies (Al-rimy et al., 2018).

4.4 Data gathering methods

For this research a dataset with malware was made available to us by VirusTotal (VT). VT is a community-driven security platform and is recognised in academia and by practitioners as an authority on malware analysis. The selected models, as described in the approach, were trained and tested on samples from this dataset.

We conducted a systematic analysis with the use of Exploratory Data Analysis (EDA) (Guthrie, 2020) techniques on the received samples. With EDA we aim to (1) gain insights into the distribution of the families versus samples size and (2) determine the data quality (e.g. correct labelling) and integrity of the files supplied.

To make the results of the research as representative as possible, samples of recent ransomware and a number of well-known ransomware families as described by Hassan (Hassan, 2019) were part of the selection. We selected the ransomware based on the family name. This name was human-readable and set by Microsoft as a label, which is available in our dataset. This is considered as our ground truth.

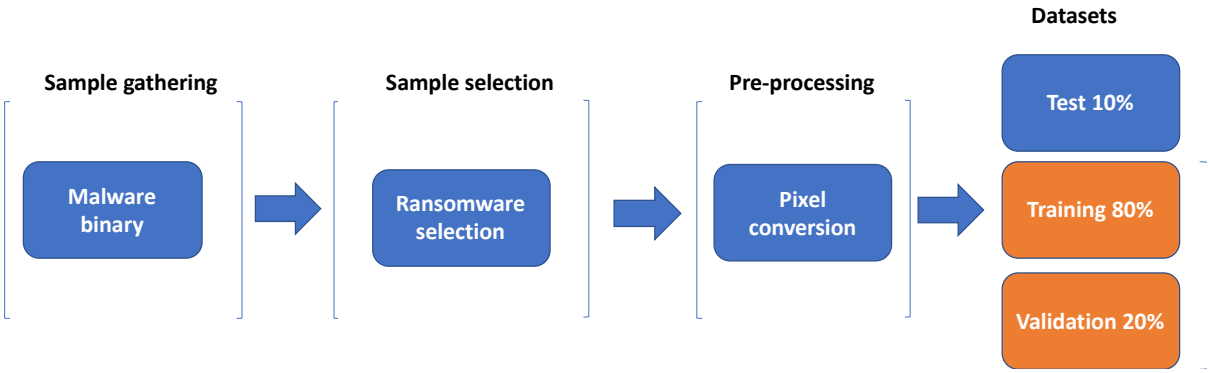


Figure 7 Data processing

The dataset for validation was used, as shown in Figure 7, to measure *validation accuracy*, which indicates the extent to which the model generalizes, meaning it does not under- or overfit during training.

The scripts and configuration files of the lab environment are stored in a GitHub repository (https://github.com/azeus404) which is publicly accessible so that the steps used during the data pre-processing and experimental part of the research can be reproduced. The dataset has been archived and can be made available for peer-review. Given its size (around **320 GB**) it cannot be uploaded to GitHub. More importantly, the legal limitations set by VT prevent us from further distributing this dataset. Due to these restrictions, it cannot be made available in the aforementioned repository.

4.4.1 Malware dataset

This dataset contained **347,307** malware samples with labels from multiple AV suppliers recorded in the JSON file accompanying each sample. These samples were collected between 2017 and 2020. From this dataset we selected the samples based the malware families as they were named by Microsoft and used this label as our ground truth. This resulted in a total of 282,650 unique samples consisting out of 10 types of malware. **Appendix 1: Dataset,** provides an in-depth explanation of contents of the aforementioned JSON file and an example. The distribution of malware types and the number of samples present in the VT dataset are shown in Figure 8.

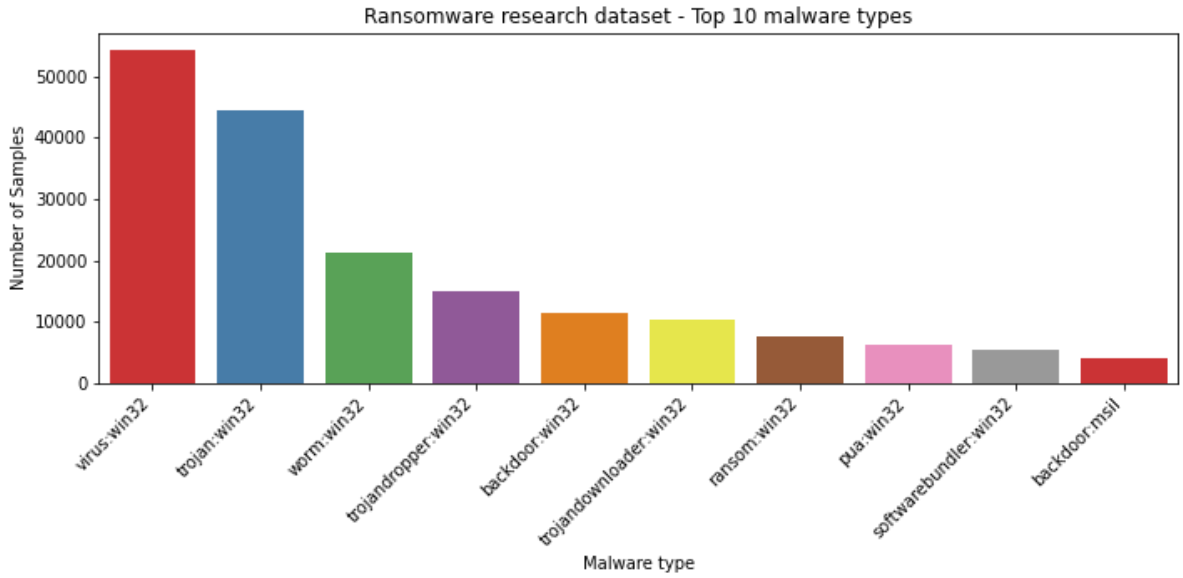


Figure 8 Malware in ransomware research dataset

4.4.2 Benign dataset

We added benign samples to the ransomware dataset, as in the study by FireEye (Coull and Gardner, 2019), so the classifier could identify ransomware or benign software. These samples were collected from executables from a clean-installed version of the Microsoft Windows 10 operating system. This benign dataset contains **900** unique samples and was added to the pre-processed VT dataset, in this study designated and referred to as the ransomware research dataset (RRD).

4.5 Selection of samples

We decided to use the Microsoft's classification label as ground truth, because here the names of the ransomware families in the dataset are easily identifiable. The dataset used by Nataraja et al. (2011) was also based on the Microsoft labels as ground truth. The MalImg dataset (Ronen et al., 2018), derivative of the Nataraja dataset, is also used in many of the

studies mentioned in the literature review. In total there are **53** unique ransomware families present in the VT dataset.

A selection of the ransomware families was taken from the dataset and ransomware, and families with fewer than 50 samples were excluded from the study, as they would contain insufficient data to make a training, validation and test set.

Table 2 shows the ransomware families (as detected and labelled by Microsoft) that were selected after processing the VT dataset. After some data cleansing steps, the dataset contained **7,042** samples ready to be pre-processed. The data cleansing activities consisted of removing duplicate samples and correcting spelling mistakes in the labels. We also added 900 benign samples as previously described.

Table 2 Ransomware families

Family name	Number of Samples
ransom:win32/tescrypt	3,412
ransom:win32/gandcrab	1,276
ransom:win32/crowti	618
ransom:win32/locky	600
ransom:win32/cerber	548
ransom:win32/genasom	243
ransom:win32/wannacrypt	138
ransom:win32/enestaller	98
ransom:win32/milicry	57
ransom:win32/haperlock	52
Total	7,042

4.6 Image distortion

Most of the CNNs trained on ImageNet dataset have a default input size, called an input tensor. The default sizes were retrieved from the documentation of the Keras API (Keras, 2020):

- 224 x 224;
- 299 x 299;
- 311 x 311.

This means that if the images in the training set are smaller or of a different shape, they will be distorted if they are resized and reshaped to the default input tensor size. Too much distortion has a negative impact on the accuracy of the model, as noted by Chen (2018). Therefore, smaller images were disregarded in the training of the models.

The distribution of image size over the selected ransomware is presented in Figure 9, which shows that the majority of images have a width of 384 pixels and a height of 342 pixels. Image size is regarded in the literature (Chen, 2018; Gibert et al., 2020) as an important hyperparameter for a model. We selected images with a minimum size of **112 x 112** to train our images to reach an optimum number of samples and minimize the effects of distortion. After this final selection, the total dataset (RDD) contained **6,310** (**n = 6,310**) samples.

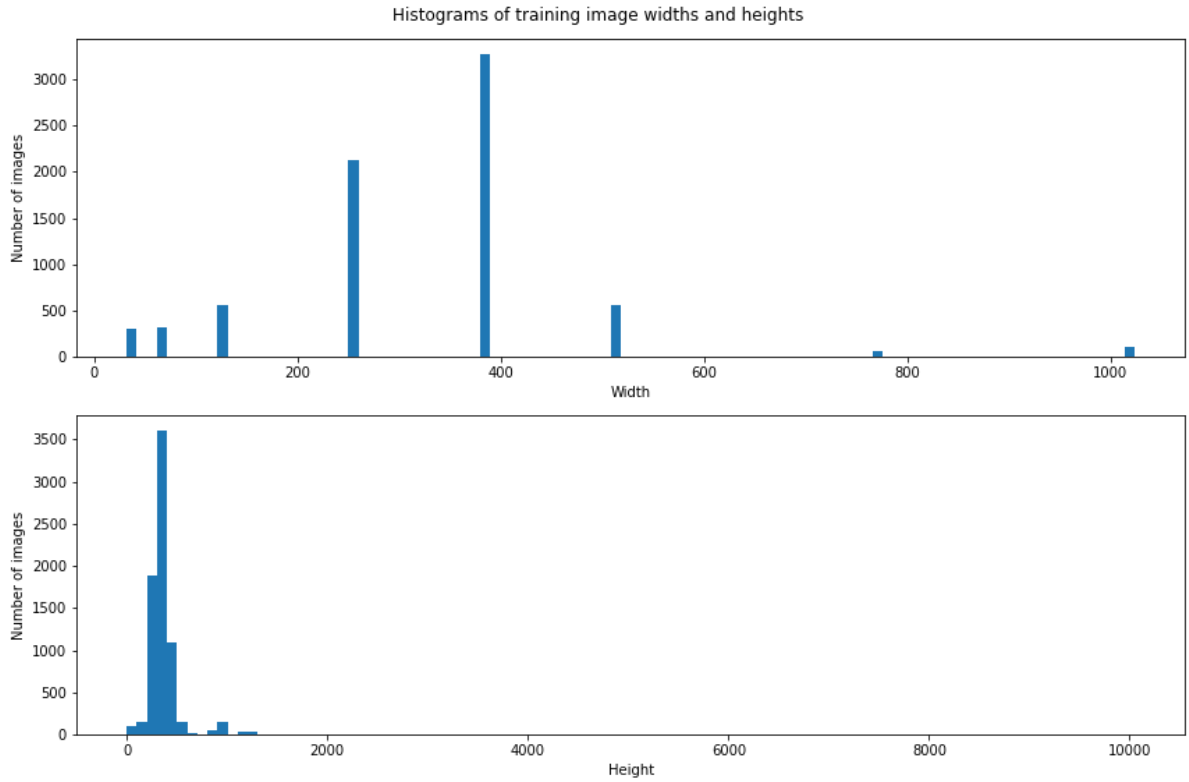


Figure 9 Distribution of image sizes by samples in the final dataset.

4.7 Imbalanced distribution of samples

Imbalanced datasets in general are a common phenomenon in all reviewed studies. We identified an imbalance of **36:1** in the number of samples per class. It was necessary to average out the differences between the majority (*Tescrypt* a.k.a *Teslacrypt*) and the minority class (*Wannacrypt* a.k.a *Wannacry*) and take measures to counter this imbalance while training a model because imbalanced datasets often results in a bias to the majority class and make the model less useful (Gibert et al., 2020). The imbalance in our dataset is shown in Figure 10.

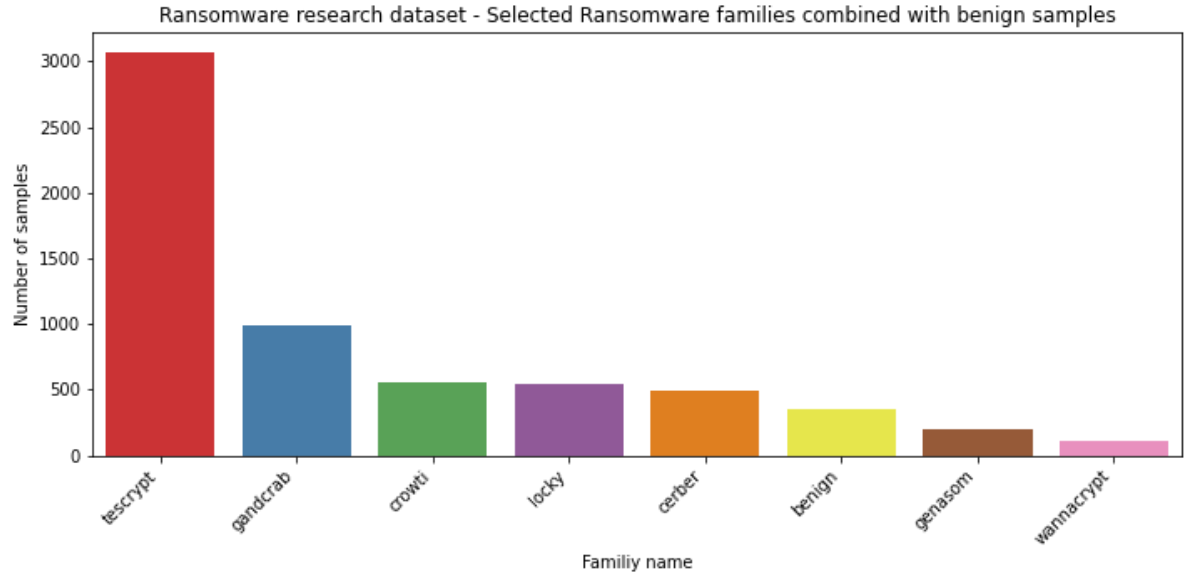


Figure 10 Distribution of families – imbalance dataset

The Python software libraries used in this study have their own implementations based on class weight functions. This function adds weights to classes, which causes the model to "pay more attention" to examples from an under-represented class.
We used one of the Scikit-learn (Sklearn) built-in methods to account for the imbalance and implemented it in code. Other methods, such as under- or oversampling, image augmentation and SMOTE (Yue, 2017), were not viable solutions for our use case because we use images of byte representations of binaries and not natural images (of cars or dogs, e.g.), which can be flipped horizontally to count as a new training sample.

4.8 Training, validation and testing set

We partitioned the RRD into a training set (90%) and a testing set (10%). This split ratio is common in ML-related studies and practical implementations when using a limited number of samples (Géron, 2019). We use a method of stratified sample selection to retrieve a 10% selection of samples from processed dataset; this data was set aside to be used for final evaluating (testing) the overall performance of the model. The remaining 90% of data was randomly split again in an 80% and 20% partition during the initialisation of the model and was used for training and validation of the model. The final partitioning and the number of samples in the aforementioned datasets are shown in Table 3.

Table 3 Training, validation & testing split

Dataset	**Number of samples (images)**
Training	4,547
Validation	1,132
Testing	631
Total	**6,310**

4.9 Model selection

In this study we implemented different models found in the literature and added the models found on the Keras webpage. As stated in the approach, we used a naïve method by iterating over viable solutions and checking whether they met the requirements. The best solutions, based on their performance, were selected. This method and approach are part of the design cycle common in DSR, the overall research method used in our study (Wieringa, 2014).

4.10 Model Evaluation

In this study we use different methods to evaluate the models' performance in classification. These methods were derived from those we found in the literature to aid reviewers in comparing and validating our research (Wieringa, 2014). We use a confusion matrix, recall, precision, F_1 score and accuracy. The equations and a short description are given below.

Confusion matrix: A method used to evaluate the model's performance. In the matrix classes are scored based on the instances of correct classification for a given class (Géron, 2019).

Recall: The ratio of positive correctly identified samples by the classifier to what the actual label or ground truth was (Géron, 2019). A perfect recall or sensitivity score equals 1.0 and implies no false negatives or FNR equal to 0.

$$Recall = \frac{TP}{TP + FN}$$

Precision: The ratio of correctly predicted positive observations to the total predicted positive observations (Géron, 2019). A perfect precision score equals 1.0 and implies no false positives or FPR is equal to 0.

$$Precision = \frac{TP}{TP + FP}$$

F_1 score: This score is a combination of the aforementioned metrics *recall* and *precision* and is used to compare classifiers or models (Géron, 2019).

$$F1 = \frac{TP}{TP + \frac{FN + FP}{2}}$$

Accuracy: Accuracy is the number of correct predictions out of the total examples (Géron, 2019).

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

TP = True Positive
FP = False Positive
FN = False Negative
TN = True Negative

FNR = False Negative Rate
FPR = False Positive Rate

5 Experimental environment & fine-tuning

This chapter provides a brief description of the experimental environment and the fine-tuning applied on the CNNs.

5.1 Experimental environment

We implemented the proposed approach in an experiment. The experiment was conducted in an isolated lab environment with the dataset made available by VT. The specifications of the hardware and software used and the configuration of the lab environment were recorded, and the datasets were saved so that the experiment could be repeated with the same parameters and the results could be peer-reviewed; this contributes to the validity and reproducibility of the research. The specifications of the computer system used are described in **Appendix 3: System specifications**. The pre-trained weights, on the ImageNet dataset, were downloaded during the initialisation of the model. The scripts and Jupyter notebooks used to analyse the data are available on GitHub https://github.com/azeus404/thesis .

5.2 Fine-tuning

Fine-tuning the model is the process of modifying the classification part of the selected CCNs for our use case. Figure 11 shows an overview of a typical CNN divided into input, feature extractor and classifier parts.

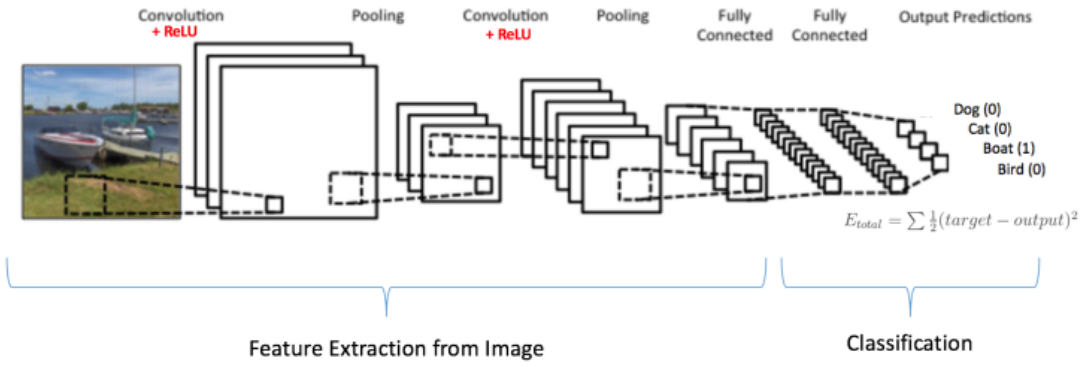


Figure 11 Example CNN feature extractor and classifier (Google,2020).

As discussed in Chapter 3 and illustrated in Figure 5, we modified the **model hyperparameters** by tweaking the classification part (Figure 11) of the model and adding a GlobalAveragePooling2D pooling layer for reducing the output (dimensions) from the pre-trained CNN.

For the classifier part of Figure 11, we

- added a fully connected or dense layer with 1,024 neurons with a *ReLu* activation function additional feature extraction. This neuron size is the output size of the pooling layer of the feature extraction part of our model.
- added a fully connected or dense output layer of 8 neurons with a *Softmax* activation function for the prediction of the classes.

A CNN has hyperparameters such as an optimisation algorithm and a loss function. We applied an Adam optimisation function with a relatively high learning rate and a categorical cross entropy loss function which computes the cross-entropy loss between the labels and predictions, suited for a multi-class problem.

6 Results of the study and analyses

6.1 Introduction

In this chapter we present the results of our study, beginning with a description of the experimental results and proceeding to an interpretation of the results.

6.2 Experimental results

The results of the experiment are described, and the detector is evaluated by looking at its performance using a set of evaluation metrics consisting of a confusion matrix (CM) and derivatives such as recall, precision and accuracy, as presented in Chapter 4.

The extent to which the detector is able to correctly classify samples based on these evaluation methods is described and compared with the techniques that have been applied in the existing literature (Chen, 2018; Vu et al., 2019).

We compared different CNN architectures as described in the research methodology and approach models presented in Table 4. In the table, the total parameters, the number of trainable parameters and the input shape or tensor are shown for a given model.

During the experiment the models were trained for **50** epochs, a batch size of **64** samples per epoch and a high learning rate of $1 * 10^{-3}$ (**1e-3**), the default value of the Adam optimizer, for stochastic gradient descent; this algorithm is used for back propagation (Kingma and Ba, 2017). Batch size and learning rate are **hyperparameters** (Géron, 2019) and can be further tuned during the final implementation of the detector. In the literature, 32 is used as a default batch size, but the decision depends mainly on the amount of available GPU RAM and the size of the training dataset (Géron, 2019; Masters and Luschi, 2018). We decided to use 64 because (1) we had sufficient GPU RAM available and (2) it gave the best results for our use case as it reduced the training time and led to a better-fitting model.

We applied an early stopping call-back function – if the validation accuracy did not improve for **5** epochs the training halted and the best model weights were saved – and an additional function to stop the training if a ***NaN*** value of loss was reached.

Table 4 Implemented models and parameters

#	Model	Total params	Number of Trainable params	Input shape (ImageNet)
1	ResNet50	25,694,088	25,640,968	224 x 224 x 3
2	InceptionV3	23,909,160	2,106,376	299 x 299 x 3
3	ResNet50V2	25,671,176	2,106,376	224 x 224 x 3
4	ResNet101V2	44,732,936	2,106,376	224 x 224 x 3
5	ResNet152V2	60,438,024	2,106,376	224 x 224 x 3
6	VGG19	20,557,896	533,512	224 x 224 x 3
7	Xception	22,967,856	2,106,376	299 x 299 x 3
8	VGG16	14,981,448	266,760	224 x 224 x 3
9	DenseNet121	8,095,304	1,057,800	224 x 224 x 3
10	DenseNet169	14,356,040	1,713,160	224 x 224 x 4
11	MobileNet	4,286,664	1,057,800	224 x 224 x 3

12	MobileNetV2	3,577,928	1,649,928	224 x 224 x 3
13[1]	NASNetMobile	5,326,716	1,124,648	224 x 224 x 3
14	NASNetLarge	89,184,122	4,267,304	331 x 331 x 3
15	InceptionResNetV2	55,873,736	1,582,088	299 x 299 x 3
16	EfficientNetB7	66,728,351	2,630,664	224 x 224 x 3

As shown in the input shape column, the input shapes are fixed because we used ImageNet weights. After the experiment we collected the data; the models' scores are shown, ranked by accuracy, in Table 5.

Table 5 Model performance

Model	F1	Recall	Precision	Loss	Accuracy
MobileNetV2	0.8479	0.8482	0.8498	0.6495	0.916
MobileNet	0.8503	0.8603	0.8429	0.3466	0.9049
ResNet50	0.8034	0.8239	0.8009	0.4271	0.8748
NASNetLarge	0.8134	0.8204	0.8163	0.7944	0.8605
InceptionV3	0.782	0.7749	0.8018	0.8289	0.8479
InceptionResNetV2	0.7615	0.7855	0.7468	0.4958	0.8463
VGG16	0.5764	0.6709	0.5414	1,05E+32	0.6149
ResNet152V2	0.5147	0.5943	0.5372	41.622	0.5563
ResNet50V2	0.2457	0.2865	0.2911	85.631.211	0.3994
ResNet101V2	0.26	0.2874	0.2864	4363799.5	0.3217
Xception	0.0135	0.125	0.0071	NaN	0.0571
VGG19	0.0135	0.125	0.0071	NaN	0.0571
EfficientNetB7	0.0135	0.125	0.0071	NaN	0.0571
DenseNet121	0.0135	0.125	0.0071	NaN	0.0571
DenseNet169	0.0135	0.125	0.0071	NaN	0.0571

The models with the required accuracy of 90% plus established in Chapter 3, section 3.4 are listed below in Table 6.

Table 6 Comparing model results on test dataset

Model	F1	Recall	Precision	Loss	Accuracy
MobileNetV2	0.8479	0.8482	0.8498	0.6495	0.916
MobileNet	0.8503	0.8603	0.8429	0.3466	0.9049

Due to the stochastic nature of training CNNs, it is possible to reach the same results on either of the models. This is a common problem when training models in general and on GPUs (Géron, 2018). The differences in performance are in our opinion too slight to justify favouring one over the other.

[1] Despite our efforts we failed to successfully implement this model.

In our study false positives are considered as important as false negatives for malware analysts because being unable to identify a sample as ransomware or as belonging to a known family does not help the malware analysts, as described in the aim of our study.

The *MobileNet* and *MobileV2* models reach around **90%** accuracy on the test data in 21 epochs on the validation data. The experimental results are shown in Table 6.
As shown Figure 12 the macro average recall of 0.86 and 0.85, macro average precision score of 0.84 and 0.85 and similar F_1 scores on **631** samples in the test set.

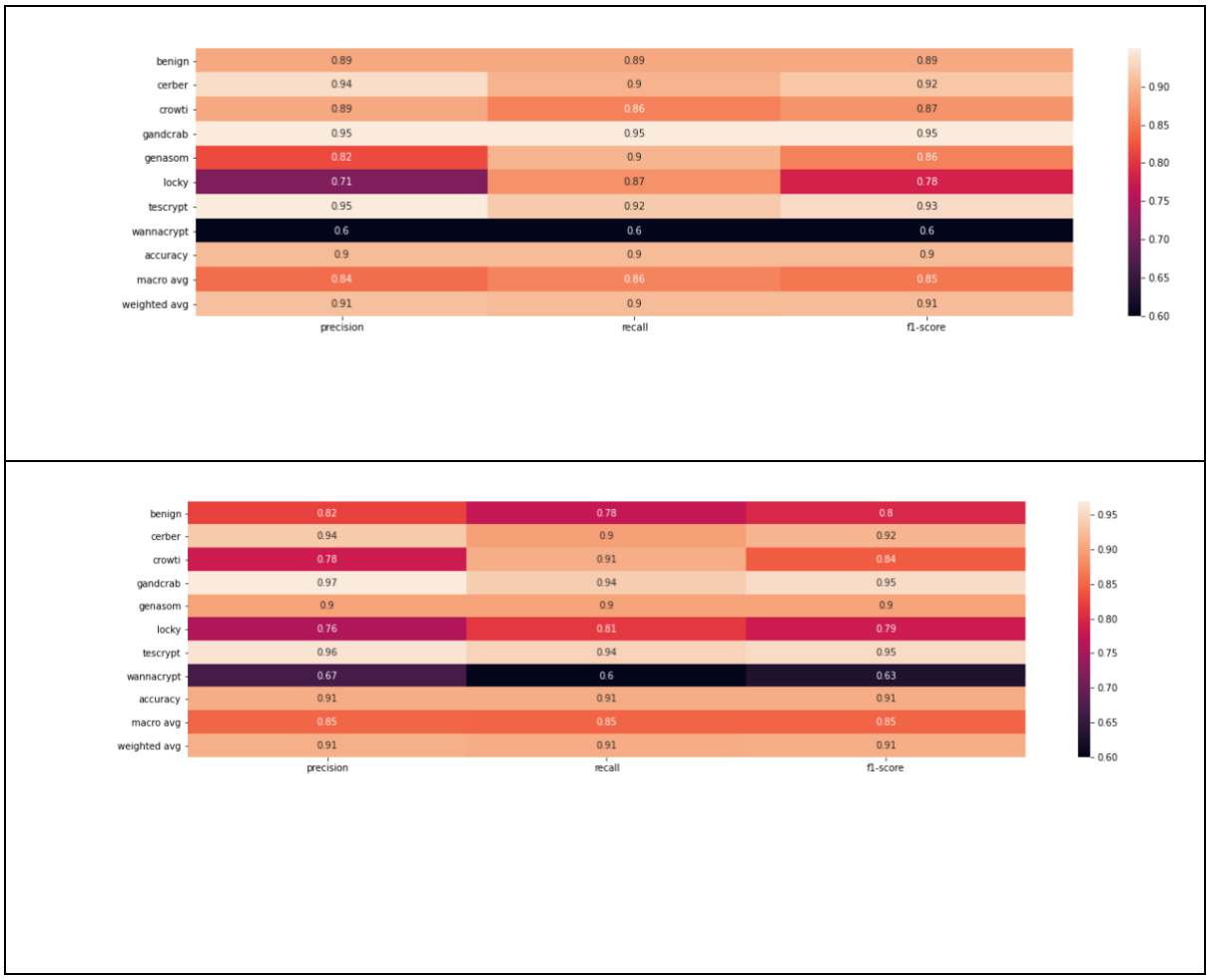


Figure 12 Classification reports (top MobileNet and bottom MobileNetV2)

The *MobileNet* and *MobileNetV2* models score around **90%** on the **631** stratified test samples over the **8** classes, the benign samples and the ransomware families. The confusion matrices of the aforementioned models are provided in **Appendix 2: Confusion matrices**

6.3 Interpretation and discussion

We began the experiment with the models with the deepest layers as suggested by the literature review. The experimental results show that it is possible to use CNN and apply transfer learning models trained on natural images from the *ImageNet* dataset on a new domain such as ransomware.

We initially tried grayscale images but ended up using RGB because training on grayscale images did not reach the desired **90%** accuracy.[2]

The deeper models, 3, 4, 5 and 16, are the models with significantly more parameters, as shown in Table 4. Surprisingly, they failed to fit the training data and had high losses or resulted in NaN errors. We consider model 14 (NASNetLarge) to be an outlier; this was the deepest network we evaluated, and it had a fair accuracy of **86%.** This result counters the insights found in the literature that deeper networks tend to have better accuracy; in our case, the less deep networks 11 (MobileNet) and 12 (MobileNetV2) performed better than the deeper ones on our dataset.

As shown in Figure 12, both *MobileNet* models score as required on the classification task. As expected, *Teslacrypt*, as the majority class, was identified correctly most of the time and *Wannacrypt*, as the minority, not at all, despite training the model with class weights to counter imbalance. What is more, some classes were confused or misclassified by the detector, but this could give malware analysts an indication of or clue to their origin.

Due to the stochastic nature of training CNNs, it would be possible to reach the same results on either of the models if the experiment were to be repeated. The differences in performance are, in our opinion, too minimal to justify favouring one model over the other.

[2] The results of the experiment on grayscale images are also available on https://github.com/azeus404

7 Conclusions and future research

7.1 Introduction

Our research adds to the body of knowledge on ransomware by adding a novel approach that applies a form of AI – deep learning and transfer learning, as we call it – to a ransomware classification problem referred to us as a use case. As we stated in the introduction to this study, we aim to aid malware analysts in identification of new ransomware families after detection.

7.2 Review of research questions

In this section we present the answers to our research questions, titled RQ 1 through 3, and the main research question.

7.2.1 Review of the research questions

RQ 1

The current methods for detecting ransomware are based on dynamic and static analysis. Static analysis relies on the heuristics, signatures and specialized domain knowledge of the analyst, who focuses on the behaviour of the ransomware interacting with a simulated victim system. In the literature review we found indications that image-based ransomware classification is still relatively new in this field of study. There were no publicly available research papers found on this subject. Current identification methods are aimed at the malware problem as a type classification without focusing on individual subclasses like ransomware families. State of the art ML or shallow learning approaches tend to need a significant amount of feature engineering and domain knowledge but can, with a limited number of features, be very effective in identifying malware and may be able to classify ransomware into families. To eliminate the need for extensive feature engineering, we transformed the ransomware classification problem into an image classification problem.

RQ 2

To resolve image classification problems, DNNs and especially CNNs have proven to be very capable in the field of image recognition and computer vision. However, they need a vast number of training samples. Deep learning and deep transfer learning involve the use of neural networks and transfer learning based on training the DNN in general on a dataset and using what is learned on another domain. However, although malware and ransomware are related, there is no trained model available. Therefore, ransomware image classification challenges have to rely on models trained on natural images. Deep learning, as opposed to traditional shallow learning methods, seems to be a viable solution and aids us in reaching our objectives.

RQ3 A and B

We successfully implemented a pre-trained CNN and changed the classifier part to our ransomware classes. In the empirical part of our study, we evaluated the performance of the model and reached satisfactory results. During the implementation of the classifier as part of a detector, there is a need for further fine-tuning of the hyperparameters. Additional fine-tuning can boost the accuracy by between 5% and 10% (Géron, 2018).

7.2.2 Review of the main research question

The DL models used in our research, *MobileNet* and *MobileNetV2,* were pretrained on an ImageNet dataset, and transfer learning was applied with the weights, resulting in a **90%** accuracy on our ransomware dataset and proving that a CNN can be used for detecting and identifying ransomware families based on images and that using DTL, and TTL in particular, to aid the speed of training and accuracy is possible.

We discovered that grayscale images, used in multiple prior studies, did not prove to reach a **90%** plus rate of accuracy on our dataset and in our use case, identifying and predicting families of ransomware. However, the results of our study demonstrate that we can reach the same or better results using 3-channel (RGB) images. This discovery adds a novel insight to the body of knowledge in this field.

There is a limited number of ransomware samples available and a lot of single instances, hence the need for application of deep transfer learning. We believe that our method is capable of detecting unseen ransomware based on the trained features of our model and is able to give at least a hint of possible family relations as stated in Chapter 6, section 3.

7.3 Future research

Image size is a key hyperparameter that needs to be taken into account during training and validation of the model, since malware developers can manipulate samples with additional random bytes, changing the file size after conversion of the image size. This results in a sample that cannot be detected using this image-based detection approach. Therefore, future research must be aimed at improving the robustness of the classifier in relation to variance in image size.

8 Discussion and reflections

In this chapter we discuss and reflect on the literature, methodology and outcomes of the research.

8.1 Literature

There was no literature available regarding image-based classification of ransomware samples. However, the classification of ransomware can be regarded as a sub-task of the classification of malware. We reviewed state-of-the-art articles in the field of machine learning and deep learning in security and other fields of study like medicine and computer vision in general. The literature and our review form a sturdy base for the empirical research in this study.

8.2 Methodology

We could achieve better results if more data were available for training a model; however, **2.5%** of the malware samples collected by VT were detected by Microsoft as ransomware. Therefore, this may not be feasible.

When adversaries use smaller or larger binaries, which result in our method in larger images, this has a negative effect on the detection rate, as noted by Chen (2018).
Therefore, as demonstrated by Chen (2018) and Gibert et al. (2020), image size is a hyperparameter that needs to be fine-tuned, and there is a constant need to re-train this model.

We did not add other forms of malware to our dataset because the effects of ransomware are clear. As a result, malware analysts know already whether a sample contains ransomware but need to identify the family.

8.3 Outcome

We consider this study to be an addition the existing body of knowledge regarding malware detection and identification in general. Analysts can be aided in their tedious work by our approach, so this study has practical value. In addition, it can serve as an inspiration for future research as stated in the Conclusions chapter of this study.

Glossary

AI	Artificial Intelligence
ANN	Artificial Neural Networks
API	Application Programming Interface
AV	Anti-virus
CM	Confusion Matrix
CNN	Convolutional Neural Network
COTS	Commercial Of The Shelve
CPU	Central Processing Unit
CV	Computer Vision
DNN	Deep Neural Network
FPR	False Positive Rate
GPU	Graphical Processing Unit
KNN	k-Nearest Neighbours algorithm
ML	Machine Learning
MLA	Machine Learning Algorithms
NN	Neural Networks
PCA	Principal Component Analyses
RGB	Red Green Blue
SIFT	Scale Invariant Feature Transform
SMOTE	Synthetic Minority Over-Sampling Technique
SVM	Support Vector Machine
TPR	True Positive Rate
VT	VirusTotal

References

Al-rimy, B.A.S., Maarof, M.A., Shaid, S.Z.M., 2018. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. Comput. Secur. 74, 144–166. https://doi.org/10.1016/j.cose.2018.01.001

Amro, S.A., Alkhalifah, A., 2015. A Comparative Study of Virus Detection Techniques 9, 8.

Baltrušaitis, T., Ahuja, C., Morency, L.-P., 2018. Multimodal machine learning: A survey and taxonomy. IEEE Trans. Pattern Anal. Mach. Intell. 41, 423–443.

Bhodia, N., Prajapati, P., Di Troia, F., Stamp, M., 2019. Transfer Learning for Image-Based Malware Classification. ArXiv190311551 Cs Stat.

Cawsey, A., 1998. The essence of artificial intelligence, Essence of computing series. Prentice Hall, Harlow, England ; New York.

Chen, C.-M., Wang, S.-H., Wen, D.-W., Lai, G.-H., Sun, M.-K., 2019. Applying Convolutional Neural Network for Malware Detection, in: 2019 IEEE 10th International Conference on Awareness Science and Technology (ICAST). IEEE, pp. 1–5.

Chen, L., 2018. Deep Transfer Learning for Static Malware Classification. ArXiv181207606 Cs Stat.

Claufield, B., 2009. What's the Difference Between a CPU and a GPU? NVIDIA. URL https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/

Coull, S.E., Gardner, C., 2019. Activation Analysis of a Byte-Based Deep Neural Network for Malware Classification. ArXiv190304717 Cs Stat.

Dargahi, T., Dehghantanha, A., Bahrami, P.N., Conti, M., Bianchi, G., Benedetto, L., 2019. A Cyber-Kill-Chain based taxonomy of crypto-ransomware features. J. Comput. Virol. Hacking Tech. 15, 277–305. https://doi.org/10.1007/s11416-019-00338-7

Deng, L., Yu, D., 2013. Deep Learning: Methods and Applications. Deep Learn. 7, 197.

Ganapathi, P., Shanmugapriya, D. (Eds.), 2020. Handbook of Research on Machine and Deep Learning Applications for Cyber Security:, Advances in Information Security, Privacy, and Ethics. IGI Global. https://doi.org/10.4018/978-1-5225-9611-0

Géron, A., 2019. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems. O'Reilly Media, Inc, Sebastopol, CA.

Ghafur, S., Kristensen, S., Honeyford, K., Martin, G., Darzi, A., Aylin, P., 2019. A retrospective impact analysis of the WannaCry cyberattack on the NHS. NPJ Digit. Med. 2, 1–7.

Gibert, D., Mateu, C., Planes, J., 2020. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. J. Netw. Comput. Appl. 153, 102526. https://doi.org/10.1016/j.jnca.2019.102526

Guthrie, W.F., 2020. NIST/SEMATECH e-Handbook of Statistical Methods (NIST Handbook 151). https://doi.org/10.18434/M32189

Han, K., Lim, J.H., Im, E.G., 2013. Malware analysis method using visualization of binary files, in: Proceedings of the 2013 Research in Adaptive and Convergent Systems. pp. 317–321.

Hardy, W., Chen, L., Hou, S., Ye, Y., Li, X., 2016. DL4MD: A deep learning framework for intelligent malware detection, in: Proceedings of the International Conference on Data Mining (DMIN). The Steering Committee of The World Congress in Computer Science, Computer …, p. 61.

Hassan, N.A., 2019. Ransomware Families, in: Ransomware Revealed. Springer, pp. 47–68.

Hosseini, M.-P., Lu, S., Kamaraj, K., Slowikowski, A., Venkatesh, H.C., 2020. Deep Learning Architectures, in: Pedrycz, W., Chen, S.-M. (Eds.), Deep Learning:

Concepts and Architectures, Studies in Computational Intelligence. Springer International Publishing, Cham, pp. 1–24. https://doi.org/10.1007/978-3-030-31756-0_1
ImageNet Project, n.d. ImageNet [WWW Document]. URL http://www.image-net.org/ (accessed 3.20.20).
Johns, J., 2017. Representation Learning for Malware Classification 23.
Kalash, M., Rochan, M., Mohammed, N., Bruce, N.D.B., Wang, Y., Iqbal, F., 2018. Malware Classification with Deep Convolutional Neural Networks, in: 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). Presented at the 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), IEEE, Paris, pp. 1–5. https://doi.org/10.1109/NTMS.2018.8328749
Kaspersky, 2020. IT threat evolution Q3 2019. Statistics. URL https://securelist.com/it-threat-evolution-q3-2019-statistics/95269/ (accessed 2.3.20).
Kaspersky, n.d. Machine Learning in Cybersecurity. URL https://www.kaspersky.com/enterprise-security/wiki-section/products/machine-learning-in-cybersecurity (accessed 2.2.20).
keras, 2020. Keras [WWW Document]. URL https://keras.io
Khan, R.U., Zhang, X., Kumar, R., Tariq, H.A., 2017. Analysis of resnet model for malicious code detection, in: 2017 14th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP). Presented at the 2017 14th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), IEEE, Chengdu, pp. 239–242. https://doi.org/10.1109/ICCWAMTIP.2017.8301487
Kingma, D.P., Ba, J., 2017. Adam: A Method for Stochastic Optimization. ArXiv14126980 Cs.
Kolosnjaji, B., Eraisha, G., Webster, G., Zarras, A., Eckert, C., 2017. Empowering convolutional networks for malware classification and analysis, in: 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, pp. 3838–3845.
Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems. pp. 1097–1105.
Krohn, J., Beyleveld, G., Bassens, A., 2020. Deep learning illustrated: a visual, interactive guide to artificial intelligence.
Kubat, M., 2017. An introduction to machine learning. Springer Science+Business Media, New York, NY.
Lo, W.W., Yang, X., Wang, Y., 2019. An Xception Convolutional Neural Network for Malware Classification with Transfer Learning, in: 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS). Presented at the 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS), IEEE, CANARY ISLANDS, Spain, pp. 1–5. https://doi.org/10.1109/NTMS.2019.8763852
Loman, M., 2019. How Ransomware Attacks.
Masters, D., Luschi, C., 2018. Revisiting Small Batch Training for Deep Neural Networks. ArXiv180407612 Cs Stat.
McAfee, 2019. McAfee Labs Threats Report August 2019.
MITRE, 2020. Data Encrypted for Impact [WWW Document]. URL https://attack.mitre.org/techniques/T1486/
Müller, V.C., Bostrom, N., 2016. Future progress in artificial intelligence: A survey of expert opinion, in: Fundamental Issues of Artificial Intelligence. Springer, pp. 555–572.

Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B., 2011. Malware images: visualization and automatic classification, in: Proceedings of the 8th International Symposium on Visualization for Cyber Security. pp. 1–7.
NCSC, 2020. Ransomware: wat kunt u doen? [WWW Document]. URL https://www.ncsc.nl/actueel/nieuws/2019/september/5/ransomware-wat-kunt-u-doen (accessed 2.2.20).
Pan, S.J., Yang, Q., 2010. A Survey on Transfer Learning. IEEE Trans. Knowl. Data Eng. 22, 1345–1359. https://doi.org/10.1109/TKDE.2009.191
Rawat, W., Wang, Z., 2017. Deep convolutional neural networks for image classification: A comprehensive review. Neural Comput. 29, 2352–2449.
Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M., 2018. Microsoft Malware Classification Challenge. ArXiv180210135 Cs.
Sewak, M., Sahay, S.K., Rathore, H., 2018. Comparison of deep learning and the classical machine learning algorithm for the malware detection, in: 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). IEEE, pp. 293–296.
Sharma, S., Rama Krishna, C., Sahay, S.K., 2019. Detection of Advanced Malware by Machine Learning Techniques, in: Ray, K., Sharma, T.K., Rawat, S., Saini, R.K., Bandyopadhyay, A. (Eds.), Soft Computing: Theories and Applications, Advances in Intelligent Systems and Computing. Springer Singapore, Singapore, pp. 333–342. https://doi.org/10.1007/978-981-13-0589-4_31
Sikorski, M., Honig, A., 2012. Practical malware analysis: the hands-on guide to dissecting malicious software. No Starch Press, San Francisco.
Simonyan, K., Zisserman, A., 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. ArXiv14091556 Cs.
Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C., 2018. A Survey on Deep Transfer Learning. ArXiv180801974 Cs Stat.
Trend Micro, 2020. THE SPRAWLING REACH OF COMPLEX THREATS 2019 ANNUAL SECURITY ROUNDUP [WWW Document]. URL https://www.trendmicro.com/vinfo/us/security/research-and-analysis/threat-reports/roundup/the-sprawling-reach-of-complex-threats (accessed 2.2.20).
Vinayakumar, R., Alazab, M., Soman, K., Poornachandran, P., Venkatraman, S., 2019. Robust intelligent malware detection using deep learning. IEEE Access 7, 46717–46738.
Vu, D.-L., Nguyen, T.-K., Nguyen, T.V., Nguyen, T.N., Massacci, F., Phung, P.H., 2019. A convolutional transformation network for malware classification. ArXiv Prepr. ArXiv190907227.
Wieringa, R.J., 2014. Design Science Methodology for Information Systems and Software Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-43839-8
Xie, B., Qin, J., Xiang, X., Li, H., Pan, L., 2017. An Image Retrieval Algorithm Based on GIST and SIFT Features 8.
Yue, S., 2017. Imbalanced malware images classification: a CNN based approach. ArXiv Prepr. ArXiv170808042.

Appendix 1: Dataset

VirusTotal provided us with a dataset which contained **347,307** Microsoft Windows binaries and JSON files. As described in Chapter 4 paragraph 4.1, we selected samples based on the label as provided by Microsoft and we used it as ground truth during training and testing of our models. In other words, the label is the **name** of the ransomware family as it was given by a Microsoft antivirus scanner.

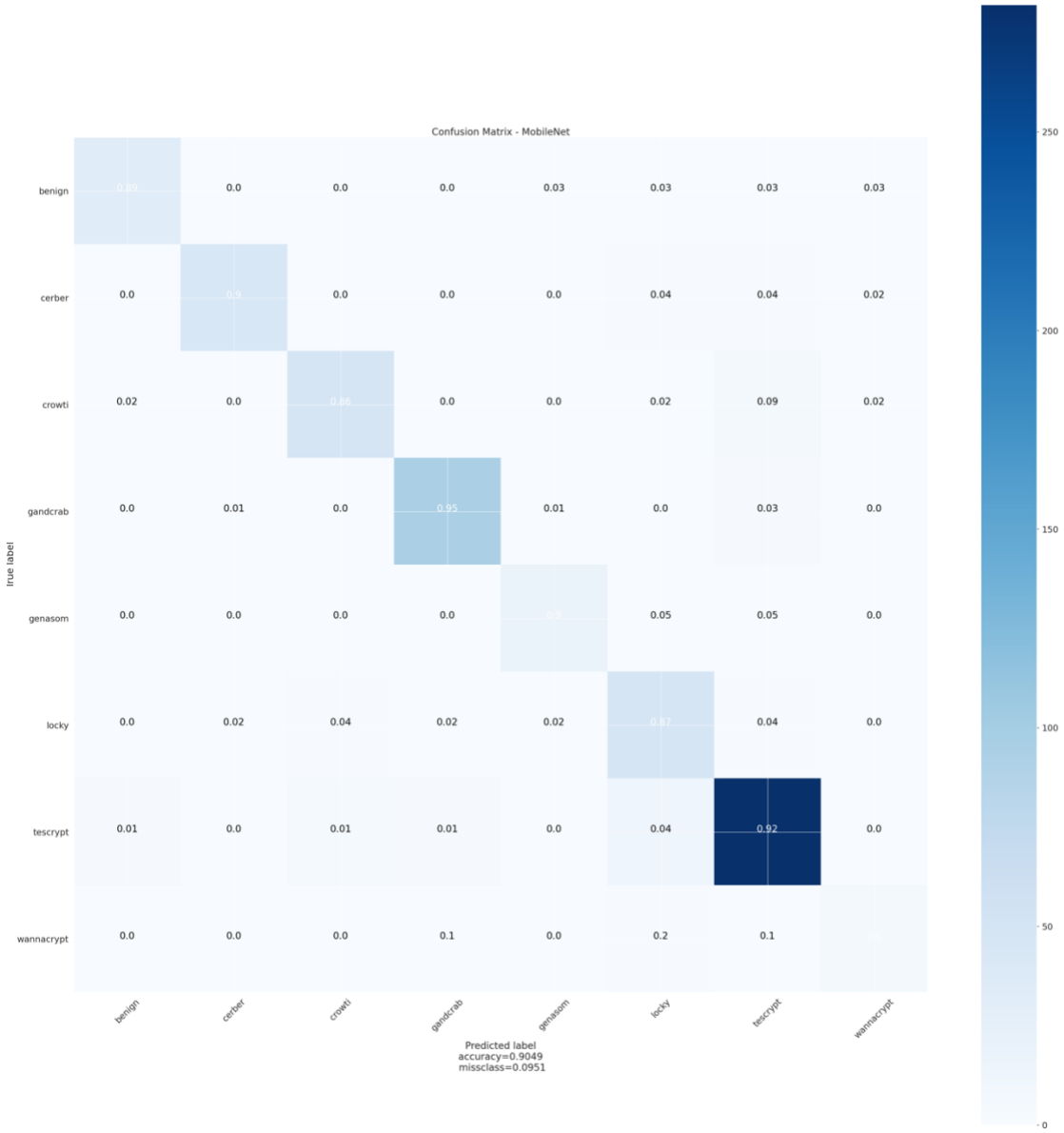
An example of a binary and its corresponding JSON file is listed below:

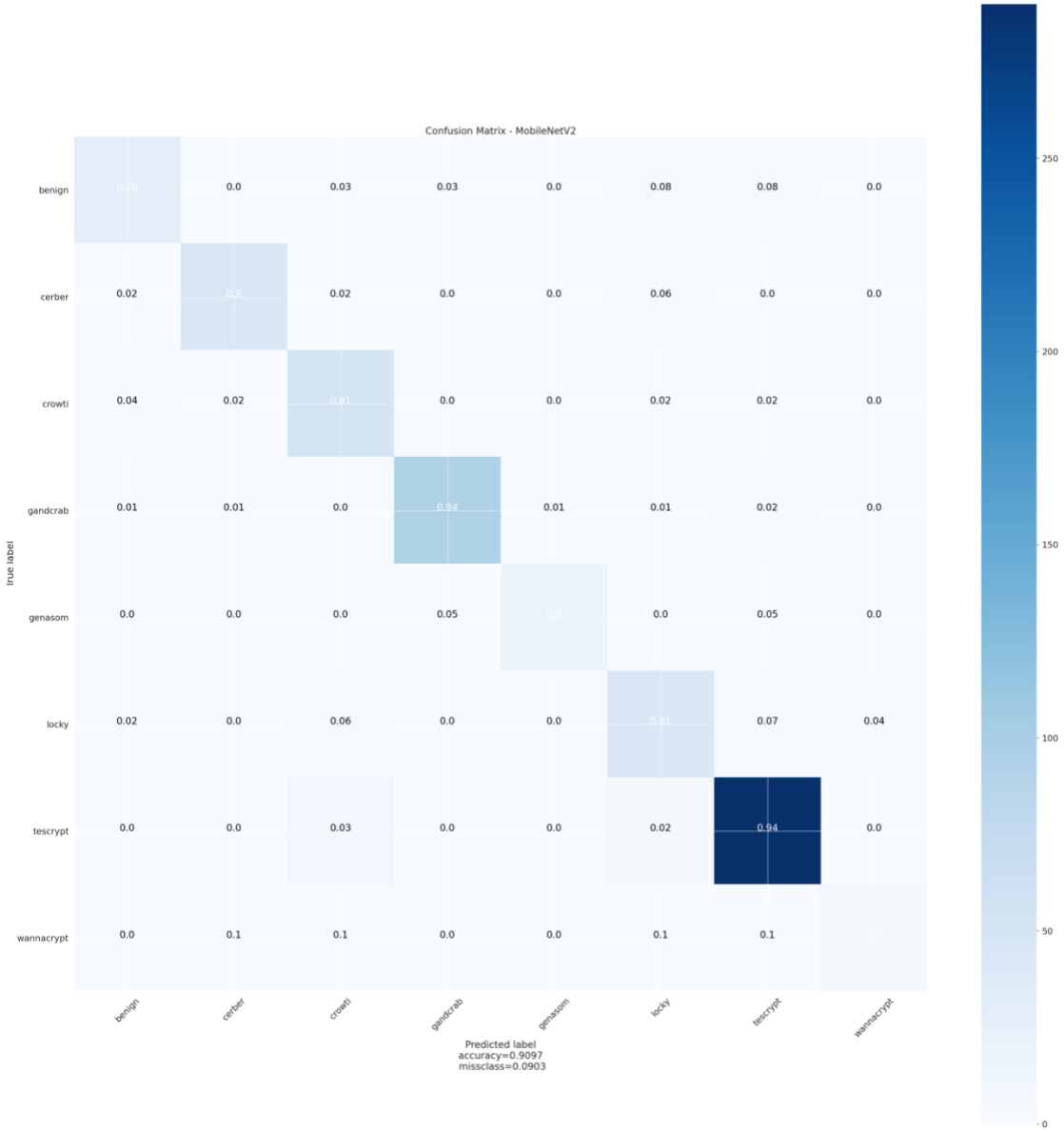
- 0ad4fd25a9611201d3125cdf129bb3ca3738cc49bf3e3bfd53f6a37dc6aa6768
- 0ad4fd25a9611201d3125cdf129bb3ca3738cc49bf3e3bfd53f6a37dc6aa6768.json

The JSON file contains the results of the analysis done by 70 antivirus scanners from different AV companies on the binary and includes the relevant meta data of the file.
A formatted output of the JSON file can be found on
https://www.virustotal.com/gui/file/0ad4fd25a9611201d3125cdf129bb3ca3738cc49bf3e3bfd53f6a37dc6aa6768/detection
The sample was detected as malicious and identified by Microsoft as a sample from the Cerber ransomware family. With the name (label): **Ransom:Win32/Cerber.J.**

Appendix 2: Confusion matrices

As shown in the images below the Confusion Matrices of the MobileNetV1 and V2 models.




Confusion Matrix - MobileNetV2
benign
cerber
crowti
gandcrab
genasom
locky
tescrypt
wannacrypt
True label
0.78 0.0 0.03 0.03 0.0 0.08 0.08 0.0
0.02 0.9 0.02 0.0 0.0 0.06 0.0 0.0
0.04 0.02 0.91 0.0 0.0 0.02 0.02 0.0
0.01 0.01 0.0 0.94 0.01 0.01 0.02 0.0
0.0 0.0 0.0 0.05 0.9 0.0 0.05 0.0
0.02 0.0 0.06 0.0 0.0 0.81 0.07 0.04
0.0 0.0 0.03 0.0 0.0 0.02 0.94 0.0
0.0 0.1 0.1 0.0 0.0 0.1 0.1 0.6
benign
cerber
crowti
gandcrab
genasom
locky
tescrypt
wannacrypt
Predicted label
accuracy=0.9097
missclass=0.0903
250
200
150
100
50
0

Appendix 3: System specifications & repository

For the experiment, a system with the following specifications was used:

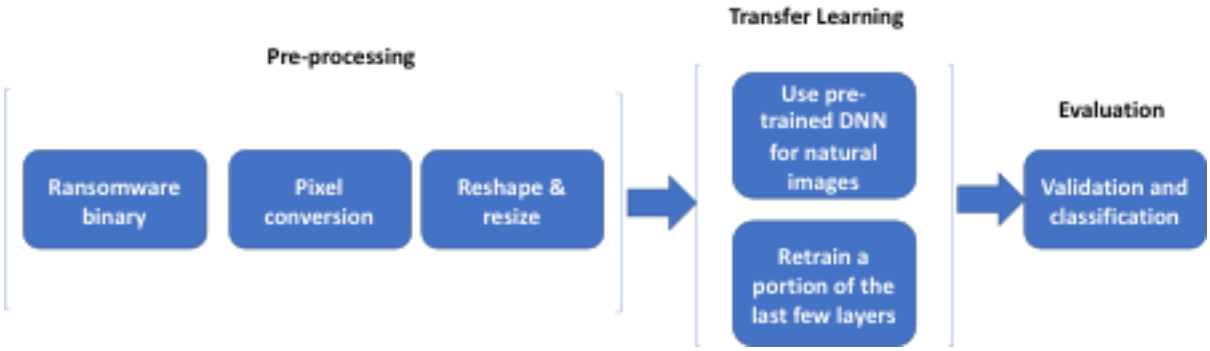
- Intel Core i7 8559U Processor (2.7 GHz)
- 32GB RAM
- GeForce RTX 2070 8GB

This system used a GPU with 2,304 CUDA cores and 8GB of memory in training the DL algorithm.

The Ubuntu 18.0.5 LTS Linux OS was used. Linux is suitable for our needs because (1) malware is used, and we do not want to infect the underlying operating system while processing the samples, and (2) the tools used are optimised for this operating system.

The software libraries used and their versions are included the requirements.txt file accessible in a public GitHub repository: https://github.com/azeus404/thesis

The different scripts used in this study are divided into categories as described in the approach (Chapter 3) and shown in the image below. Please consult the README.MD file in the aforementioned repository.



The scripts used for the pre-processing and the model evaluation analysis are also available as Jupyter Notebooks and were used in Chapter 4 of the study. Finally, a prototype of the implemented model, in a detector is included in the aforementioned GitHub repository as described in Chapter 4.