Radboud University Nijmegen

Faculty of Science

Where does this malware come from?

Multi-class malware classification using static analysis and deep learning with the Malconv machine learning model

Thesis BSc Computing Science

Author:
Wietze Mulder

Supervisors:
Alexandru C. Serban
Dr. Ir. Erik Poll

January 2020

Abstract

This thesis is about classifying state-sponsored malware using static analysis and a convolutional neural network. The neural network we used is based on the one described in [25] called Malconv. Small modifications are made to get better results and to apply it to our multi-class classification problem. We have trained a neural network on nation state malware so we can predict which country a piece of malware belongs to. Previous research has proven that machine learning is an interesting approach at classifying malware. Malconv is a convolutional neural network used for malware classification by training the network with the binary as a whole. We try to classify the malware by country or Advanced Persistent Threat group, APT-group for short. APT-groups are hacker groups with malicious intent to, for example, attack government agencies. In this research the APT-group are financially backed by nation states.

The data set we use was previously published by a Radboud student named Coen Boot from the Digital Security group. Next to publishing this data set we also showed a method for classifying the data set. With static analysis we only look at the binary content of the malware, without running the malware. With dynamic analysis analyse what the binary does when it is executed. We use static analysis, however the approach of Boot was based on dynamic analysis with machine learning. Dynamic analysis requires running the samples in a sandbox environment. However most sandbox tools are commercial products. Therefore static analysis is more suitable for open-source distribution and often faster than dynamic analysis.

We have found out that using Malconv, We reached an accuracy of 89.3% when classifying the data set by 5 different countries. We ran multiple experiments to improve accuracy and to identify the mis-classification. We have stated that the some characteristics are shared across countries and thus may share their malware.

Moreover, it is interesting to say we achieved satisfactory results with our method, we have reached the same accuracy as the Malconv research[25]. Our results differ from Boot's machine learning with a dynamic analysis approach on the same data set. Boot got a better accuracy for this data set, namely 95% compared to our 89%. Although we did not surpass the high-class performance required to compete with other models in this field, we can say that this methodology is suitable for further academic research. Future work could extend the data base or learn more from the classifications with CAM mappings

Contents

List of Figures

List of Tables

1. Introduction

Malware is a steadily growing problem with new samples occurring every day[3]. This is a problem because most systems, even those that are critical for our infrastructure are connected to the internet, thus available for attack for everyone. Either to gain private information or to derange systems, there is an incentive to attack such systems for cyber-criminals, Specialists have seen that cyber-criminals become more sophisticated in their skill of malware development[3]. Malware is used to infect computers for malicious purposes, acting against the interests of the computer user. This threat gives criminals the opportunity to gather private information or to derange systems. Cyber-threats have extended from individual or small organized groups to professional intelligence officers leading state-sponsored cyber-missions. The complexity of the malware has also grown, with its ever-growing techniques to evade anti-virus software and technical analysis. State-sponsored Advanced Persistent Thread (APT) groups make use of malware to exploit systems, spy on their targets or disorganize systems. With the increasing competence of attackers and their incentive to hide their traces makes the authorship attribution increasingly harder. The complex and laborious task asks for automation (or at least partly) of the analysis.

In order to fight against cyber-treats, it is an important part to know who the attacker is. In state-sponsored malware this is important. In order to accuse internationally an attacker, we need evidence for our accusation to open a trial or other diplomatic mechanisms. Thus, we wish to analyse malware samples to find the actor behind the malware and have a certain proof this actor is indeed the author. This is called authorship attribution. Already a lot of research has been done on authorship attribution of malware[2] and harmless software[7]. But these techniques are not always easy to implement. Many developed techniques require the source code of the malware. However, for most of the malware the source code is not available. Furthermore, authors put a lot of effort into obfuscating their code and hiding their traces to prevent attribution.

One single malware sample is most of the time part of a larger malware family, developed by the same threat group. This collection contains important knowledge to develop a broader picture of a certain APT group. Since malware samples from the same malware families share a lot of properties and are most of the time created by the same author, we can detect similar "offspring" in the same manner. An efficient way to perform family classification can be helpful for authorship attribution.

Both authorship attribution and family classification are classification problems. In order to automate a part of the analysis, we can train a system to determine the author of a piece of malware, by means of classification. Classifying malware samples with machine learning already had some history[19]. The field really started about a decade ago[27], especially there is quite a lot of research about android malware[28, 21]. In this research we want to focus on neural networks. Neural networks have excelled on image[20], signal[10] and text[18] problems. By learning from this success we may simplify our problem in classifying binaries.

In this research, we have chosen to use the framework called Malconv. Malconv is a convolutional neural network that use the whole binary to train its model on. This comes with many challenges and more information is given in Chapter 3.

Building a malware classification system requires a data set. A recent paper[4] published a free-accessible data set with 3500 state-sponsored malware samples. This data set can now be used for future research like this research. We will use this data set for our research

Our goal of this bachelor thesis is to verify if it is possible to perform state-sponsored malware authorship attribution using the malconv supervised learning model?

1.1 Outline

The structure of the thesis will be as the follows: First some background is presented about malware and malware analysis in Chapter 2. After that we cover the related work about other methodologies used in automated malware classification, especially on supervised learning, in Chapter 3. The data set we use for this thesis is interesting and we recommend to take a glance at Chapter 4. We also explain how to deal with an imbalanced data set and this chapter is a good read on performance metrics and which is also useful outside this research. Finally we arrive at the most important part of this thesis in Chapter 5, the methodology and results. If you cannot wait to research further on this thesis, take a look at Chapter 6 where you can find the future work. At last we conclude and discuss our research in Chapter 7. Supplemental information about the scripts and hardware used in this research can be found in Appendix A.

2. What is Malware?

In this chapter we explain what malware is. We will also highlight some special traits malware has compared to benign software. We also dive into malware analysis in section 2.2, and the two main different approaches of analysis in section 2.3 and section 2.4. If you are already very familiar with malware and its analysis, you can skip to chapter 3 about the related work and recent trends in this area of research.

2.1 Characteristics of a malware binary

The name *Malware* is derived from two words, malicious and software. The best description would be software designed to intentionally execute malicious practices. All computer programs have been designed with a certain purpose in mind, a task they should execute. The design goal of a program has great influence for the end-result of the binary. This also holds with malware, only their goal is of malicious intent. The malicious intents may have different goals. In most cases, its goals is one of the following: To disturb computer systems, gather private information or to gain access to private computer systems.

Malware compiles to binaries just like ordinary programs, but there is a difference. Malware does not want to show their purpose in order to evade detection from anti-virus, malware-detection and most of the times the victim. It could therefore include information in the binary to trick the victim. In the next sections we will explain the various ways a malware sample may use to hide its intent.

2.1.1 Obfuscation methods

Anti-virus software has multiple known strategies to detect malicious intent. However, malware evolves and has found multiple techniques to hide from malware scanners and anti-virus software. In the paragraph different scanning methods and their hiding techniques will be discussed.

Hash signatures First of all, the most simple approach, is to check the signature hash of a sample and look up in a database with signatures of known malware sample. This approach can be easily mitigated by adding randomized useless data in your binary[36].

System calls Second of all, a more sophisticated approach is to detect malware from the order of system calls a sample makes to detect malicious intent[36]. This can be bypassed by adding noise in terms of nonsense system calls.

Heuristics Third of all, Anti-virus scanners perform heuristics to identify suspicious characteristics. Even this can be mitigated by for example packing the binary or encrypting the suspicious parts of your code.

As we have just learned, there are a lot of tactics to prevent being discovered. And this list is only to name a few examples to show the difference between malware and benign software. In the next subsection we will explain the techniques and the challenges of detecting malware.

2.1.2 Detecting these traits

Hiding tactics can be implemented in multiple ways, and the style is depending on the programmer or group. This means that the same implementation of traits in malware samples can be linked together. Coding style is on a great level an identifier to the autho[7]. Even more simple traits, like comments or variable names in code may reveal the language of the programmer. For example, Chinese comments may reveal the origin of the author, but it could be the case that they were put there with intent by a smart non-Chinese author to trick the analysts.

Malware development is similar to developing normal software, it requires a lot of complexity for a good end-result. One does not start from scratch when starting a new project, we most of the times use code from previous projects to use in the new project. Most of the times malware is build from older code from previous malware. This means that characteristics resonate in previous work of a hacker group or country. This is really useful for analysing malware families, since they heavily share the same traits.

Since malware design starts to reach a great level of sophistication, undetected solutions have a great value. These solutions can be sold on a black market and may be used by other parties. This means that certain characteristics of a malware family overlap with another family, since code is shared or traded between parties.

2.2 Malware analysis

Analysis of malware can be divided into two categories. The first is dynamic analysis and the latter is static analysis. Both are great methods to analyse binaries and in the next two paragraphs we will discuss about the two methods.

In the next sections we will give an overview of the methods to analyse a piece of malware, we will cover dynamic analysis and static analysis and different related work that used machine learning to classify or attribute authors from binaries.

2.3 Dynamic analysis

Dynamic analysis means running the executable and analysis its behaviour inside a sandboxed environment, for example an emulator. We can observe the behaviour at run-time and observe the traces a sample produces like altered files or network traffic. Most of the times, this gives a clearer picture about the malware and it's identity. But there are downsides using dynamic analysis. Running each binary you want to classify in a dynamic fashion is very resource intensive, because each time there needs to be a fresh emulator ready to analyse. And above that, you also have to run the binary, and without knowing what it will do this may impose risks, since the malware might contain sandbox escalation techniques (although very rare). Besides, the malware may have obfuscation methods to detect the sandboxed environment and not show their behaviour[26]. Dynamic analysis also implies you can run the binary, however this is not always the case.

Previous work has made multiple attempts to classify malware with a dynamic approach. For example, by analysing the system call sequences[15]. System calls are needed for any meaningful action like, opening a file, running a thread, writing to the registry, or opening a network connection. this research uses the system calls a piece of malware produces while running to identify the author. The research achieves a good accuracy of 89.4%, and most of the error was because one family in their data set is entirely mis-classified for another family. It classifies the rest of the families with great accuracy.

Another way of analysis is by looking a reports made by a virus scanner. In this paper[4] the researchers uses the scan report of the Cuckoo virus scanner to identify the treat-group of a malware sample. The result of these reports was the input for a neural network that classified the samples. The model was trained and tested on state-sponsored malware.
However this comes with the negative aspects of dynamic analysis. Also the software used in this research was proprietary and this means we are not able to easily replicate the result.
We use the data set of nation-sponsored malware samples published in this research. However, our research takes a static approach instead of their dynamic approach. We also aim for the same goal, namely to try to classify the data set by Country and by APT group.

Because of the hassle of setting up dynamic analysis and the use of proprietary software that comes with dynamic analysis we have chosen to use static analysis for this research. What static analysis is and all the good (and bad) parts about it are explain in the next section.

2.4 Static analysis

Static analysis is the collection of methods analysing a sample without executing it. One can already retrieve quite some information just by looking. The header of a binary contains quite some useful information. We can see for what OS it is written and what programming language or compiler is used. Even before further analysis we can retrieve quite some useful information by just only looking at the readable characters in a binary[36]. Windows system calls for example are readable as a human in the binary and in some cases variable names are left in so we can figure out their purpose. Depending on the compiler and compiler flags, we can even find comments in the compiled code. Variable names and comment give away quite some information about the user. We can now identify the language and understand the goal of the program better and its way of working. It is also possible to disassemble or decompile the code to gain knowledge about its source code. With this method we can substract the program flow and find all system calls used in the code. However, static analysis is susceptible to obfuscation of the binary. A piece of software called a "packer" can compress and encrypt a binary to conceal its content. One may think that only malware packs its binaries, but this is not the case[11]. Normal benign binaries also use packers for their executables. Also an important aspect of static analysis is that is it fast and it is most of the times faster than a dynamic approach. With the leading reason being the hassle to boot up and emulator and waiting for the malware to reveal its behaviour.

There are more challenges when performing static analysis, especially when automating this process by machine learning or software. Binaries contain multiple modalities of information. this means that the malware contains human-readable text, binary code or even arbitrary objects such as images. Due to the characteristics of a binary, compiled functions in the program can be at every location in the binary. The location of critically important functions can extremely differ per binary. Because of function calls or jump commands, resulting in that the flow of a program is not easily visible in the binary. Another problem is that applications, build tools, and libraries developers use will be updated and alternatives may be used in the future. This means that newer malware may have the exact same behaviour, but due to updates in these tools, the resulting binary may significantly differ from its outdated sibling. Also malware is written by a person and is often, as mentioned earlier, adjusted to avoid detection. And when it comes to feeding the binary into a network, another problem arises. If we treat each binary as a unit in a sequence, and since binaries can be of an arbitrary size, how do we deal with a sequence classification problem of two million bytes?

3. Related Work

Quite some interesting research already has been done in this field. But there is still a lot to be researched. In this chapter we show what already has been done and we show work related to our research. You can say that our research overlaps two fields of research, the first being malware analysis and the other being machine learning on software. We will highlight both fields

Previous research[33] showed promising results on classifiying binaries using the PE-header of binaries. The research of Shafiq et al.[33] used a lot of domain knowledge to achieve its more than 99% detection rate of their framework called PE-miner. But we are interested in a framework without the use of domain knowledge. Raff et al.[24] have made an really good attempt at classifying whether malware is benign or malicious. By using only the PE-header of the malware binaries, the researchers used using deep learning methods and without the use of domain knowledge of malware. They have tested of different networks, Fully connected neural network and a recurrent neural network, which were trained from only the raw bytes of the PE header. The results were very positive and they achieved an AUC of 97.7%. The most important message of these results current is that neural networks are able to learn from raw byte data, without domain knowledge.

Huang and Stokes[13] have created a multi-task neural network to classify it whether it is benign or malicious and to classify it in 100 malware families. They have designed a shallow neural network that was trained on dynamically obtained features, namely a sequence of application programming interface (API) call events plus their parameters and a sequence of null-terminated objects recovered from system memory during emulation. With their huge database of 4.5 million samples, they achieved an excellent performance in binary classification and the family classification. An interesting note is that they mention that dropout significantly reduces the error rate for both shallow and deep neural architectures.

In the research of Saxe and Berlin[30] the researchers have designed a shallow neural network on 4 features retrieved from static analysis all four in a 2 dimensional array. The features were retrieved from the entropy of the file, the PE header and the readable characters in the code.

N-gram is a contiguous sequence of n items from a given sample. This n-gram model is widely used in text mining[8, 6]. Using binary n-gram features for malware classification seems to be a previous preference[31, 22, 17]. These papers have achieved great performance using this type of feature extraction. In Kolter and Maloof[17], they show that the use of established methods of text classification to detect and classify malicious executables using byte n-gram feature selection and achieve great results. Not only bytes in binaries, but also the disassembled opcodes may be used for classification. Opcodes are the building block of a binary executable. opcode stands for "Operation Code" and it is a single instruction for the CPU. In Assembly language all individual opcode have a name, like MOV, ADD, or JMP. By learning opcode sequences Moskovitch et al.[22] were able the achieve great performances using opcode n-gram feature selection.

But there seem to be downsides using n-gram. Raff et al.[26] have investigated in byte n-gram features and have found some flaws using this method. First, they discovered a flaw in how previous data sets were created that lead to an over-estimation of classification accuracy. Second, they discovered that most of the information contained in n-grams arise from string features that could be obtained in simpler ways. Finally, they demonstrated that n-gram features promote over-fitting, even with linear models and extreme regularization. Although n-gram is a proven method to gather features, we have chosen not to use N-grams in our approach.

In the research of Milosevic et al.[21], the researchers developed two static approaches to classify Android malware with machine learning. The first one was a permission-based approach and the other one is based on source code analysis utilizing a bag-of-words representation model.

Malware is also just software written by real adversaries. We want classify malware by the origin of the malware, since these malware is likely written by the same author, we may use the knowledge of authorship attribution on malware to gain understanding of its origin. Alrabaee et al.[2] created a compiler-agnostic method for identifying the authors of program binaries. They designed a system to recognize author coding habits by extracting author's choices from binary code. It filters out all compiler related functions and labels the library related functions using their previous work, Binshape[34] and FOSSIL[1]. Then they convert the user-related functions into canonical form that is robust against heavy obfuscation. Then they collect author's choices made during coding. They trained their model on a large collection of source code and the corresponding assembly instructions. They applied their system *BinAuthor* on real-world malware and achieved great results.

All the previous methods are great, but we are looking for something that is: open-source, free to use and has the possibility for adjustment to act as a multi-class classifier. Luckily, we have found a fitting model that performs great, namely Malconv[25]. A neural network that eats the binary as a whole.

3.1 Malconv

The goal of Malconv[25] is to decide if a binary is malicious or benign. The design goal was to minimize domain knowledge in the model. When deep learning raw binaries there is a problem. Namely, these is a locality problem, the important coding functions can be anywhere in the binaries, since when the critical code in the source code changes location, it also changes location in the binary. They managed to solve this problem by using a convolution neural layer. After a lot of design attempts, the researchers figured out the best solution was a simple design. An embedding layer, a 1D convolution, temporal MAX-Pooling layer, a fully connected layer and a Softmax function at the end. The researcher also optimized the model to handle huge data set. This is not important for this research however but worth mentioning. In this thesis we use this model for our classification, because it has achieved a great performance on classifying and it is easy to implement because of the design where no pre-processing is needed. We are curious to find out if the model is also capable of distinguishing multiple countries of APT-groups

4. Dataset for nation-state malware and performance metrics

The data set used in this research is from the research of Boot [4] and is publicly available. The data set contains of 3594 real world samples found on various malware databases, labeled by Advanced Persistent Threat (APT) group and country. The samples are compressed and encrypted with a password and stored in the corresponding APT group folder. All sampled have been downloaded from VirusTotal, a malware database and analysis platform where anyone can upload samples to. Along with all the samples there is a csv-file with all labels and hashes of the malware. No real pre-processing is needed. The whole data-set is available at a Github repository. More information on retrieving the data set and using it can be found in Appendix A

Note these malware samples are not syntactically created, they are real malware samples developed by nation funded developers. In order to focus on the academic computer science side of this data set, we have anonymized the data set for this research. But since interesting discussion rises from the geographic origin of the malware we have the names of the countries presented in table 4.1.

Table 4.1: Data set per country

Country	**Anonymized**	**# of samples**
China	Country 0	1338
Russia	Country 1	627
North- Korea	Country 2	273
USA	Country 3	395
Pakistan	Country 4	961
Total		3594

4.1 Training and Test Sets

This data set is split-up in a training set and a prediction set with a ratio of 9:1. We do this to have an accurate accuracy score on our data set. Since this data set is relatively small, we use 10-fold cross validation to get accurate performance measures.

4.2 How do we deal with unbalanced data sets?

Is we can observe in Chapter 4, our data set is unbalanced, almost half of our data set is Country 0 and this brings extra challenges. The medical research almost always deals with an unbalanced data set of its measures on their patients[5]. Luckily most of the patients are healthy, and a minority is ill. However, identifying all the ill is way more important than identifying all the healthy patients. A lot of research has been done on evaluating the quality of a model that deal with an unbalanced data set.

We introduce class weights to Keras to inform the data set is imbalanced. This will treat less occurring classes as important as the over-represented one. This will give us a weighted loss function, which yields a more honest performance.

We can also improve the model by balancing the data set. This has two possibilities, the first is syntactically creating samples for the the under-represented classes and the second one is removing over-represented classes until we have a balanced data set. Unfortunately, we cannot syntactically create samples, since binaries have a vary complex structure. Luckily, we can always remove over-represented classes. In our research we use a random under-sampler. We cannot use a smart over-sampling method,because of the complex structure of the binary.

The most straightforward method of calculating the performance of a model is to calculate the accuracy that is: $\frac{\#correctpredictions}{\#samples}$. But this metrics is not great for unbalanced data sets, since the over-represented classes will influence the score a lot more than underrepresented classes. But these classes are most of the times as important if not more important as the over-represented classes. Luckily, the literature has developed other metrics to calculate the performance of of a model. Since we also deal with balanced data set, we also use an ROC curve, more information about it later on in this chapter. Notice we decided not to choose Cohen's Kappa for our research. Recent research[9] shows us we should avoid using Cohen's Kappa in classification and favor Matthews correlation Coefficient over Cohen's Kappa. The next section we will discuss about different metrics and explain why we did use them in our research.

4.3 Matthew Correlation Coefficient (or MCC)

MCC takes into account all four values in the confusion matrix, and a high value (close to 1) means that both classes are predicted well, even if one class is disproportionately under- (or over-) represented[35]. In a multi-class calculation, the four values are the sum of all the combinations of two-class confusion matrices. More details about the calculation can be found here[14]. The MCC is in essence a correlation coefficient between the observed and predicted binary classifications; it returns a value between -1 and $+1$. A coefficient of $+1$ represents a perfect prediction, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation.[37] In this paper we have chosen for the Matthew Correlation Coefficient, because MCC works really well when using it for classification of imbalanced data[5]. It also works well for a balanced data set.

4.4 Confusion Matrix

A confusion matrix is a matrix of the predictions per class. Each row in the matrix represents the instances in an actual class and each column represents the instances in the predicted class. instead of the number of instances we can also normalize this by displaying the percentage of the number of instances of the actual class. We can now see what classes the classifier confuses as another class. The most useful information is that we are able to observe how classes are wrongly predicted. It is interesting to know what countries or APT groups are confused with other classes. This is the reason we use confusion matrices for our research.

4.5 Precision Recall Curve

Before we can understand what a precision recall curve is, we need to understand what precision and recall are. Precision expresses the proportion of the malware samples our model says that were classified to, for example, Country 0 actually were correctly from Country 0. Recall is the ability of a model to find all the malware of a certain country within a data set Take colour prediction between blue and red for example and a classifier that does not perform really well. Lets take a data set of 12 samples with red or blue colors. Of the 8 identified as blue, 5 actually are blue (true positives), while the rest are red (false positives). The precision would be 5/8. The recall would be 5/12. A precision Recall Curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds.The strong point of these curves is that they treat a classes in an unbalanced data set the same, since precision and recall only look at the relevant data points per class. In this paper we also have chosen for the Precision Recall Curve. The practical argument to use a precision recall curve is that we can now clearly see which countries or APTs under-perform or over-perform compared to the others.

4.6 ROC curve and AUC

An Receiver Operating Characteristic (ROC) curve is a plot of the True Positive Rate (TPR) on the y-axis and the False Positive Rate (FPR) on the x-axis. This means that the top left corner of the plot is the “ideal” point - a false positive rate of zero, and a true positive rate of one. This is not very realistic, but it does mean that to maximize the area under the curve (AUC) is usually better. The “steepness” of ROC curves is also important, since it is ideal to maximize the true positive rate while minimizing the false positive rate. It ultimately helps us to understand how well-separated our data classes are.[32] ROC curves are a great choice for comparing models[12]. However, ROC curves are over-optimistic when used on a imbalanced data set. Precision Recall curves show a more correct score in this case[29] ROC curves and the AUC can be of great value when the data set is balanced. We use the ROC curve when we train on a balanced data set.

5. Methodology and results

In this chapter we describe and use the different methods we shall use to analyse the data set that is described in Chapter 4. We will run the experiments and discuss the results for every experiments.

Dynamic and static analysis is just a first stage of the malware analysis, in most research it is used to produce the features that are used for the machine learning. Our process slightly differs from the standard method of automated analysis, since we just feed the whole binary into the Malconv model. More explanation about Malconv can be found in Chapter 3. The Figure 5.1 describes the process used in this research.

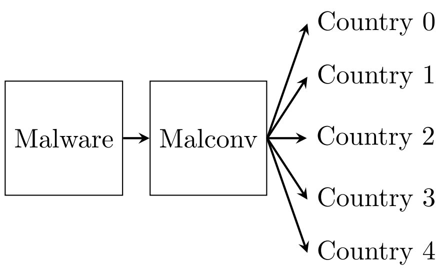


Figure 5.1: Our process to classify malware samples

The model presented in Figure 5.1 presents our flow. Malconv reads the malware samples in a sequential manner and learns to classify the samples per country or, in some experiments later discussed in this chapter, per APT-group.

The model we use in this research is inspired from the Malconv[25] model. At first the model was designed for binary classification on an mostly balanced data set. There are some parameters we have to change in order to set the model towards an unbalanced multi-class classification problem. At first, we can adjust the loss function. A loss function is used to measure the performance of a model and adjust the weights in a network when training. The original model used binary cross entropy which is well suited for binary classification, however using multi-class cross entropy is needed for our multi-class research.

First, some practical information about the experiments. Every experiments is validated using 10-fold cross validation. We have trained with 90% of the data set and the remain 10% was used as the validation test Almost all of the experiments converged quickly. This is the same as the Malconv research mentioned.

The Malconv neural network is designed to decide if a sample is malware or not.

So without configuration, this did not work for our classification for multiple classes. To classify the samples by country, the malconv model needs a slight adjustment. The last layer, previously a single node, is changed to a dense layer with 5 node and softmax activation.

5.1 Experiments with a balanced data set

As we can observe in Table 4.1 the data set in unbalanced. Country 0 has almost half of the samples while the Country 2 only a fraction, 8% to be exact. This means that the data-set is unbalanced, what could mean that by training on more samples from Country 0 than other countries, the model favors Country 0 when predicting unknown samples. These problems can be solved by balancing the data-set. To mitigate this problem we can balance out the training set by means of over-sampling or under-sampling. notice that, over sampling is not possible here because we cannot syntactically create working binaries from our data set. Under-sampling is possible however. We have randomly under-sampled our data set. With this under-sampled data set we have trained our model and and the results can found in Figure 5.2 and Figure 5.3. As already mentioned in Chapter 4, we used an ROC-curve here because this metric is of best use when analysing a balanced data set.

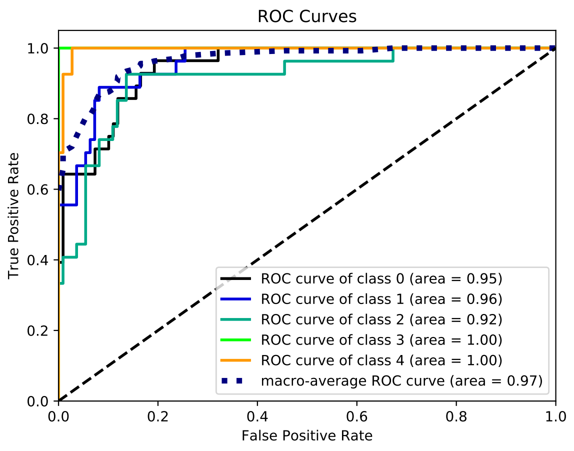
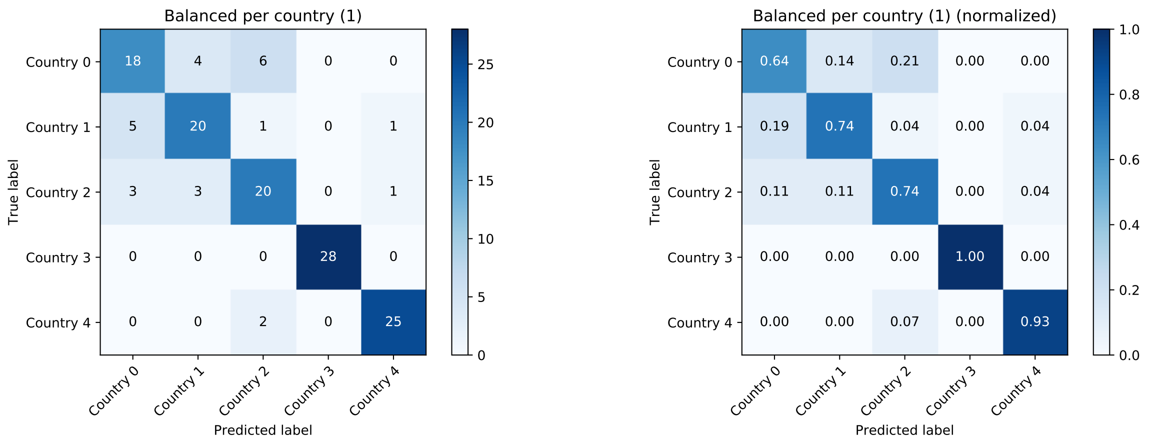


Figure 5.2: ROC-curve for under-sampled data set per country. In the legend, country is replaced with class.

As we can see in the results displayed in figure 5.2, we can see Country 0, Country 1 and Country 2 perform the same. we can see Country 3 has perfect accuracy. By looking at the AUC, we might say Country 4 also performs with perfect accuracy, but notice the small bend at the left-top corner. This means that there is still some mistakes made with classifying Country 4, only really few since the AUC is rounded off to 1.00



(a) Confusion matrix (b) Normalized confusion matrix

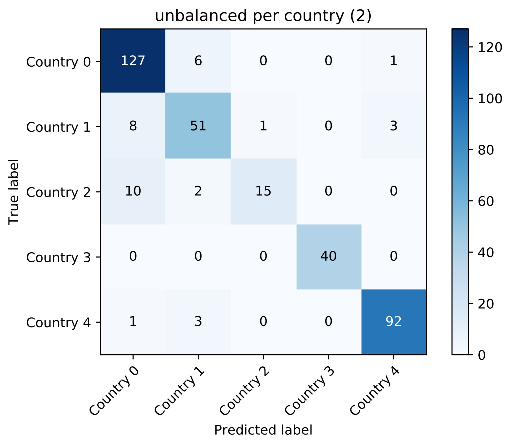
Figure 5.3: Confusion matrices under-sampled per country. left:confusion matrix, right: normalized confusion matrix

Since we concluded that the results are not perfect we wish to look where the error is located. In figure 5.3, we can find in these confusion matrices where the mis-classification is between the countries. We have chosen to show two confusion matrices, one with the number of predicted samples and one with normalized results, which means that it is a percentage of the actual label. More information on confusion matrices can be found in section 4.4. If we look at both the confusion matrices, we notice that our model has difficulty between three countries, namely Country 0, Country 1 and Country 2. These three countries are equally mis-classified with China.

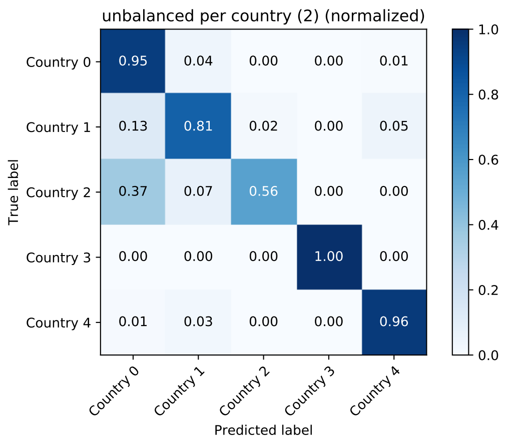
5.2 Using all data and have an imbalanced data set

The balanced experiment we balanced out the data set with the use of under-sampling, removing samples until the data set is balanced. However, we lose a lot of useful training data when we under-sample our data set in order to make it balanced. It can be useful to use all the samples. So in this experiment we do not balance our data set to see if adding more data will improve the performance of our model. With a training set of 3234 samples and a validation set of 360 samples, we reached an accuracy of 89.3% using 10-fold cross validation.

Here we have two confusion matrices, one with absolute results (Figure 5.4) and one with normalized results (Figure 5.4).



(a) Confusion matrix



(b) Normalized confusion matrix

Figure 5.4: Confusion matrices imbalanced per country. left:confusion matrix, right: normalized confusion matrix

We can see the results of our model on the prediction data set in Figure 5.4. Compared to the results in the balanced experiment, we can see that there is a change in mis-classification. Whereas samples from Country 0, wrongly predicted as Country 2 cover 21% of the samples from Country 0 in the balanced experiment, this mis-classification is reduced to 0%. But the mis-classifcation "the other way around" is increased. in the balanced dataset malware samples from Country 2 wrongly predicted as Country 0 cover 11% of the samples from Country 2. In this imbalanced experiment this mid-classification is increased to 37%. It becomes clear that this experiment performs better on Country 0. Now more of the mis-classification is from wrongly predicting countries as Country 0. The model had in particular trouble with predicting Country 1 and Country 2, But it is good in predicting the other countries namely Country 3, Country 4 and Country 0. Notice that Country 3 remains their perfect score.

These mis-classifications and correct classifications may have a geo-political origin. Different countries have different design flows. The model will have trouble picking up the origin country because they share many of the same features if these design flows or frameworks are shared between countries. It could be the case that the malware in Country 2 and Country 1 share many traits with malware from Country 0 because they are bought or stolen from this country. This assumption also explains why the Country 3 and Country 4 are correctly identified, because the countries have different frameworks and probably unique identifying ways of designing the binaries.

In order to gain understanding how countries perform, we use a Precision Recall curve (PR-curve). As we already mentioned Chapter 4, PR-curves are great for measuring imbalanced data set. Because of this we use it for this experiment. We do not use a ROC-curve this experiment, because an ROC-curve does not work as well on imbalanced data set as the PR curve. The PR-curve can be found in Figure 5.5.

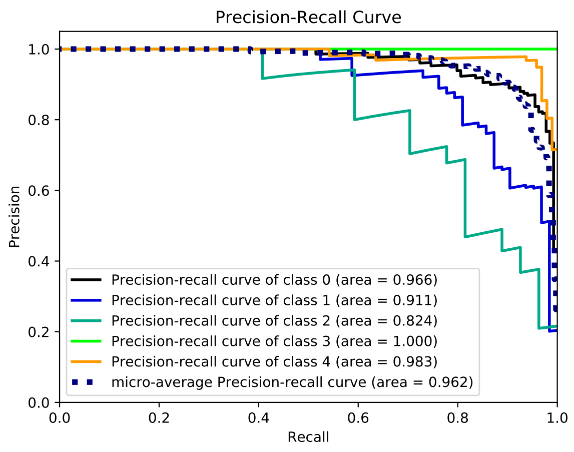
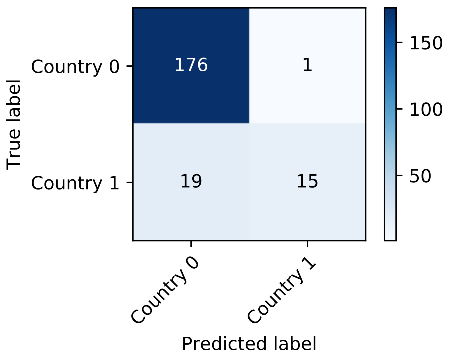


Figure 5.5: Precision recall curve per country. In the legend, country is replaced with class.

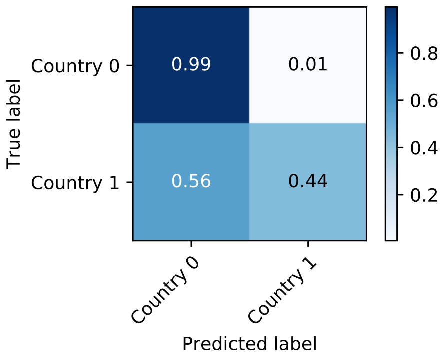
As can be seen in Figure 5.5, we notice that two curves are below the rest of the curves, namely to curve of Country 2 and Country 1. This means that these two countries are now performing worse than Country 0, compared to the balanced experiment. They have both the least amount of area covered in the plot and thus are the two countries that under-perform compared to the other countries. Country 0 seems to be improved however.

5.3 Scoping in on Country 0 vs. Country 2

Since the model has trouble distinguishing between Country 0 and Country 2 we have our neural network train only on these countries, in order to understand the biggest mis-classification of the model.



(a) Confusion matrix



(b) Normalized confusion matrix

Figure 5.6: Confusion matrices for Country 0 vs. Country 2. left: confusion matrix, right: normalized confusion matrix

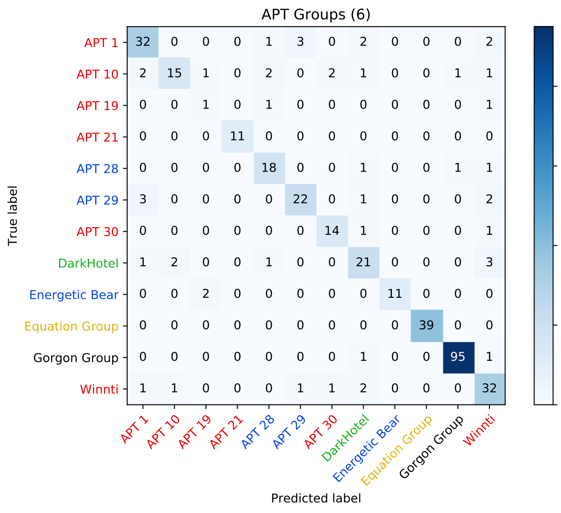
As seen in Figure 5.6, we can observe we do not gain any accuracy by scoping in on those two countries. We may observe from this that the code or framework used in the malware is very similar between the two countries. This may confirm our our geo-political hypothesis that malware between the countries is shared.

5.4 Per APT-group

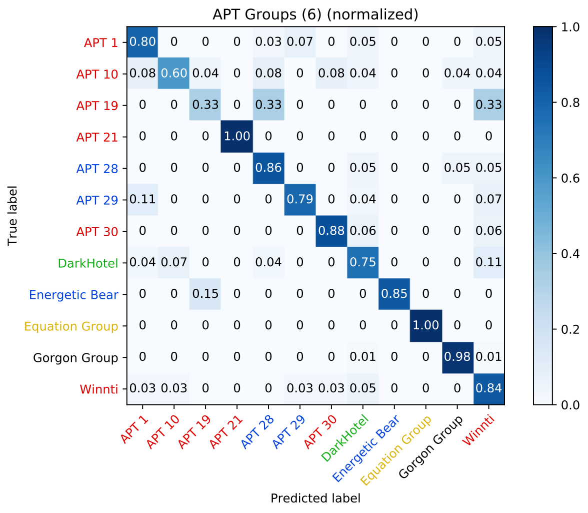
The samples can also be classified on APT-group (Advanced Persistent Threat group). Advanced Persistent Thread groups are multiple attacks over time identified from a single instance. a APT group may be a criminal hacker group or a long-term plan from a nation state. In our research all APT-groups are sponsored These labels are available in the data set and the same methodology is used as for the country classification. In table 5.1 we can see all APT group with their corresponding country and sample count. Note that the Countries are colored and these colors will also be used in the confusion matrices of Figure 5.7. We have decided not to add the PR-curve these because with 11 colored lines, the figure becomes very unclear and confusing to read.

Country	APT Group	Sample count
Country 0	APT 1	405
Country 0	APT 10	244
Country 0	APT 19	32
Country 0	APT 21	106
Country 0	APT 30	164
Country 0	Winnti	387
Country 1	APT 28	214
Country 1	APT 29	281
Country 1	Energetic Bear	132
Country 2	DarkHotel	273
Country 3	Equation Group	395
Country 4	Gorgon Group	961

Table 5.1: Data set per APT-group



(a) Confusion matrix



(b) Normalized confusion matrix

Figure 5.7: Confusion matrices under-sampled per APT-group left: confusion matrix, right: normalized confusion matrix

If we look at Figure 5.7, we can see that the mis-classification is somewhat scattered over all different APT-groups. Although not visible in these figures, the mis-classification varied quite a lot between experiments. except the APT group related to Country 3 and Country 4. The reason for this could be that some of the APT groups do not have much malware samples available in this data set. And thus contain too few samples too train on and learn this identifying characteristics and even less to predict on, increasing the variance in measured performance. Nevertheless, these scattered mis-classification also show that the model does not more have trouble distinguishing between APT groups from the same country. This works hand in hand with the results Boot had when predicting APT groups it was not trained on on a model that was trained on different APT groups from the same country. There seems to be enough difference between these groups to differentiate them, although the origin of the country is the same.

5.5 Final comparison

The Matthew correlation coefficient, MCC for short, is in essence a correlation coefficient between the observed and predicted binary classifications; it returns a value between -1 and $+1$. A coefficient of $+1$ represents a perfect prediction, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation. As already mentioned in Chapter 4, We use this metric to gain knowledge about the classifier's performance.

Table 5.2: Accuracy and MCC table

data set	**Accuracy**	**MCC**
Per country under-sampled	80.7% (σ: 5.2%)	77.8% (σ: 6.7%)
Per country imbalanced	89.3% (σ: 2.2%)	85.9% (σ: 3.5%)
Country 0 vs. Country 2	86.8% (σ: 3.6%)	35.8% (σ: 5.2%)
Per APT-group	85.2% (σ: 3.9%)	83.0% (σ: 2.0%)

In order to get a good representation of the performance, all the results were retrieved by 10-fold cross validation. The variance (σ) is also presented in the accuracy table in Table 5.2. A we can see in the accuracy tables in Table 5.2, we increase our performance when adding more sample to our data set. We gain performance when we look at the Matthew's Correlation Coefficient. Interesting to notice is that the APT group experiment has quite some good performance compared to the imbalanced country experiment. This is very interesting because we now know Malconv is also to differentiate not only to two classes but also to 12 classes. It is really interesting to see we had an overlap between Country 0 and Country 2 and Country 1. This could mean the framework or code of the malware is shared or at least share a lot of similarities, but the behaviour per country is significantly different. The most important part of this finding is that this wasn't the case with Boot's dynamic analysis on the same data set. What also is interesting to notice is that every one of the experiments we ran, It has a perfect accuracy for Country 3. Every time it had 100% precision and recall on Country 3 when the model struggled with the other countries.

6. Future work

Adversarial Malware binaries Neural networks are great for classification purposes but they have a flaw. Because they do not see at a picture a normal human does, certain trivial details may be critical for the classification choice of the network. an adversarial attack tries to exploit this issue and tries to find an slightly altered input not visible for the human eye that is wrongly classified. This method of attack is also possible for binaries, but it is a challenge because the code may break when slightly changing bytes. However, there are some places where garbage data can be placed. Some parts in the PE header may be altered and unused sections at the and are ignored. [16] has found an adversarial attack on malware binaries by appending superfluous data at the end of the binary. This will not alter the behaviour of the binary, but may alter the classification of the malware binary. With their method they achieve an evasion rate of 60% on the *Malconv* model, also used in this research. This is alarming and is interesting for further research.

Extending the data set Malconv managed to increase the performance to even 94% by training with and additional 2 million samples. Although this amount is unrealistic, we think the data set can greatly improved when more samples are added. As we read in the master thesis of Boot[4] we see that some binaries are still missing. Maybe in the future these are published and retrieving these samples will improve the data set. It could be interesting to expand on these malware and finding out it is useful for expansion on this data set.

Learning more from the data set with CAM mappings Although our model learns from training, we do not gain any knowledge about the malware. Since our method is really fast in classifying, it could suffice to draw a quick estimation in the probability of the malware samples belonging to a certain task. Future work could look into producing a class activation map (CAM)[38] for each class in order understand where the "important" regions are in the malware. This was also used in the Malconv research itself where they used in to locate important sections in the binary.

7. Conclusion and discussion

In this work, we evaluated the use of static analysis on state-sponsored malware. The data set we used contain binary samples per country and per APT-group. the data set was from a publicly available repository. We used the Malconv[25] model for our research. We have adjusted this model to have it classify by country or by APT-group. We reached an accuracy of 89.3% and have stated that the some characteristics are shared across some countries that may share their malware.

In the research of Malconv itself[25], Raff et al. reached an accuracy of 88.1%[1]. We can say we reached an expected performance with the use of Malconv. We could say we even did better than expected, since the baseline accuracy of a multi-class classification problem is lower than a binary classification problem. this means that it is harder to guess a multi-class classification problem than a binary counterpart.

An interesting finding is that this our results differ from Boot's machine learning with a dynamic analysis approach on the same data set. Although the methods of classification differ quite a lot, we conclude that the approach of Boot got a better accuracy for this data set, namely 95% compared to our 89%. As mentioned in Chapter 2, dynamic analysis is able to gather more useful features to identify a certain piece of malware. For this reason dynamic analysis will be able to reach an higher accuracy than static analysis. We also see in other research[15] dynamic analysis is able to perform better. This could be the reason that Boot's classification performed better than ours. Although our method is faster then the dynamic approach of Coen, the time is well spent.

This biggest downside with this approach is that we are not able to find out what identifying characteristics are for every country or APT group. Although our model learns from training, we do not gain any knowledge about the malware. Since our method is really fast in classifying, it could suffice to draw a quick estimation in the probability of the malware samples belonging to a certain task. Future work could look into producing a class activation map (CAM)[38] for each class in order understand where the "important" regions are in the malware. This was also used in the Malconv research itself where they used in to locate important sections in the binary.

This data set is of great quality, it is of decent portable size but not to small, perfect for future bachelor theses. So we really recommend to see future research on this data set. We thank Boot for publishing this data set.

It is worth noting that computing power was not a big issue. With 10-fold

[1]Malconv managed to increase the performance to even 94% by training with 2 million samples. This is of course very impressive, but since we also do not train with 2 million samples, this performance is ignored.

cross-validation I started to reach some boundaries on my laptop (see hardware specs in Appendix A). Training my model took about 15 minutes, enough time to pick up another task in the meantime or grab a coffee. It is worth noting that with a modern laptop and possibly with GPU enabled training, the waiting times will not be a problem at all. Hardware specs can be found in Appendix A.

7.1 Personal discussion

In retrospect, the course Data mining was really useful. It helped me a lot in the skills of using libraries like `numpy`, `pandas`, `matplotlib` and `sklearn`. The course was also useful because it learned me the methodology of a machine learning experiment. Since I am following the Cyber-security track, I am glad that a took this course in my free elective.

Bibliography

[1] Saed Alrabaee, Paria Shirani, Lingyu Wang, and Mourad Debbabi. Fossil: a resilient and efficient system for identifying foss functions in malware binaries. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):8, 2018.

[2] Saed Alrabaee, Paria Shirani, Lingyu Wang, Mourad Debbabi, and Aiman Hanna. On leveraging coding habits for effective binary authorship attribution. In *European Symposium on Research in Computer Security*, pages 26–47. Springer, 2018.

[3] Christiaan Beek, Taylor Dunton, John Fokker, Steve Grobman, Tim Hux, Tim Polzer, Marc Rivero Lopez, Thomas Roccia, Jessica Saavedra-Morales, Raj Samani, and Ryan Sherstobitof. Mcafee labs threats report august 2019. Technical report, McAfee, 2019.

[4] Coen Boot. Applying supervised learning on malware authorship attribution. Master's thesis, Radboud University Nijmegen, 2019.

[5] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PloS one*, 12(6):e0177678, 2017.

[6] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[7] Aylin Caliskan, Fabian Yamaguchi, Edwin Dauber, Richard Harang, Konrad Rieck, Rachel Greenstadt, and Arvind Narayanan. When coding style survives compilation: De-anonymizing programmers from executable binaries. *arXiv preprint arXiv:1512.08546*, 2015.

[8] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, volume 161175. Citeseer, 1994.

[9] Rosario Delgado and Xavier-Andoni Tibau. Why cohen's kappa should be avoided as performance measure in classification. *PloS one*, 14(9), 2019.

[10] A Fehske, J Gaeddert, and Jeffrey H Reed. A new approach to signal classification using spectral correlation and neural networks. In *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005.*, pages 144–150. IEEE, 2005.

[11] Fanglu Guo, Peter Ferrie, and Tzi-Cker Chiueh. A study of the packer problem and its solutions. In *International Workshop on Recent Advances in Intrusion Detection*, pages 98–115. Springer, 2008.

[12] Karimollah Hajian-Tilaki. Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation. *Caspian journal of internal medicine*, 4(2):627, 2013.

[13] Wenyi Huang and Jack W Stokes. Mtnet: a multi-task neural network for dynamic malware classification. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 399–418. Springer, 2016. Great read, explains every design decision.

[14] Giuseppe Jurman, Samantha Riccadonna, and Cesare Furlanello. A comparison of mcc and cen error measures in multi-class prediction. *PloS one*, 7(8):e41882, 2012.

[15] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. Deep learning for classification of malware system call sequences. In *Australasian Joint Conference on Artificial Intelligence*, pages 137–149. Springer, 2016.

[16] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 533–537. IEEE, 2018.

[17] J Zico Kolter and Marcus A Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7 (Dec):2721–2744, 2006.

[18] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[19] Terran Lane. Machine learning techniques for the domain of anomaly detection for computer security. *Purdue University, Jul*, 16, 1998.

[20] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571. IEEE, 2016.

[21] Nikola Milosevic, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Machine learning aided android malware classification. *Computers & Electrical Engineering*, 61:266–274, 2017.

[22] Robert Moskovitch, Clint Feher, Nir Tzachar, Eugene Berger, Marina Gitelman, Shlomi Dolev, and Yuval Elovici. Unknown malcode detection using opcode representation. In *European conference on intelligence and security informatics*, pages 204–215. Springer, 2008.

[23] Edward Raff, Richard Zak, Russell Cox, Jared Sylvester, Paul Yacci, Rebecca Ward, Anna Tracy, Mark McLean, and Charles Nicholas. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques*, 14(1):1–20, 2016.

[24] Edward Raff, Jared Sylvester, and Charles Nicholas. Learning the pe header, malware detection with minimal domain knowledge. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 121–132. ACM, 2017.

[25] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[26] Thomas Raffetseder, Christopher Kruegel, and Engin Kirda. Detecting system emulators. In *International Conference on Information Security*, pages 1–18. Springer, 2007.

[27] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[28] Justin Sahs and Latifur Khan. A machine learning approach to android malware detection. In *2012 European Intelligence and Security Informatics Conference*, pages 141–147. IEEE, 2012.

[29] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[30] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20. IEEE, 2015.

[31] Matthew G Schultz, Eleazar Eskin, F Zadok, and Salvatore J Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, pages 38–49. IEEE, 2000.

[32] scikit learn. Receiver operating characteristic (roc), 2019. URL `https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html`.

[33] M Zubair Shafiq, S Momina Tabish, Fauzan Mirza, and Muddassar Farooq. Pe-miner: Mining structural information to detect malicious executables in realtime. In *International Workshop on Recent Advances in Intrusion Detection*, pages 121–141. Springer, 2009.

[34] Paria Shirani, Lingyu Wang, and Mourad Debbabi. Binshape: Scalable and robust binary library function identification using function shape. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 301–324. Springer, 2017.

[35] Boaz Shmueli. Matthews correlation coefficient is the best classification metric you've never heard of, 2019. Towards Data Science.

[36] Michael Sikorski and Andrew Honig. *Practical malware analysis: the hands-on guide to dissecting malicious software.* no starch press, 2012.

[37] Wikipedia. Matthews correlation coefficient, 2019.

[38] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

A. Appendix

A.1 Scripts used in this research

We have the scripts we wrote in this thesis made publicly available. The installation procedure and more instructions are described on the Github Repository here:
`https://github.com/keukentrap/thesis`

A.2 Data set used in this research

You can find the data set used in this research here:
`https://github.com/cyber-research/APTMalware`

Pre-processing the data set is quite simple. We need to download the repository and extract all binaries. After that, we need to strip superfluous information in the csv-file and the data set is ready to use. We have provided a script automating this process.

The Github repository of the data set consists of many encrypted zip-files. Boot already made an effort to extract these and the scripts are available here:
`https://github.com/cyber-research/APTAttribution`

A.3 Parameter optimization

In this research, we tested many learning rate schedulers and learning rate optimizers for our research, a lot of them resulted in over-fitting, adam and amsgrad for example. We ended on using SGD with a learning rate of 0.01. We also experimented with different learning rate schedulers. We have covered: Early stopping, step decay, ReduceLRonPlateau We have also settled for Early stopping, to speed up the training time and the learning rate scheduler called "ReduceLROnPlateau". This will reduce the learning rate when the validation loss does not improve for 3 epochs. Also we tested a few loss functions, like binary cross entropy and kullback leibner divergence. We resulted in using categorical cross entropy. More information about these parameters can be found in the repository.

A.4 PlaidML

Tensorflow has a possibility to run on a GPU. This makes training and predict a lot faster. However, by default this only runs on a dedicated Nvidia graphics card. Most laptop and PC's do not have this available. PlaidML could solve this issue. PlaidML is a so-called tensor compiler that "brings Deep Learning to

Every Device".[1] PlaidML could make your graphics card in your device available for tensorflow, which will significantly decrease training times. Unfortunately, It did not work yet on my (old) laptop, but could be really useful on other machines.

A.5 Hardware used in this research

The hardware used in the start of this research was a business laptop from 2012. It was do-able but tedious when running a lot of experiments. We switched to a GPU-accelerated VPS from Microsoft Azure that is in the first $100 free for students. This worked as a charm and really sped up the training time.

Table A.1: Hardware specifications

Component	**Laptop**
Model	Dell Latitude E6430
CPU	Intel(R) Core(TM) i5-3320M
GPU	Intel® HD Graphics 4000 (not used in this research)
RAM	8GB

[1]https://www.intel.ai/plaidML/