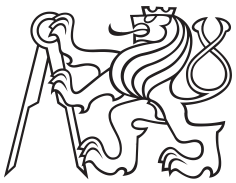
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of control engineering

Hierarchical models of network traffic

Vojtěch Kozel

Supervisor: doc. Ing. Tomáš Pevný, Ph.D.
Field of study: Cybernetics and Robotics
May 2021

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kozel Vojtěch** Personal ID number: **481891**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Hierarchical models of network traffic

Bachelor's thesis title in Czech:

Hierarchické modely síťové komunikace

Guidelines:

1.Study prior art on automatic analysis of network traffic.
2.Capture network traffic of malware from public sources.
3.Learn the hierarchical multiple instance learning framework.
4.Analyse captured malware / cleanware using HMill.
5.Using Mill, identify the artefacts corresponding to different malware strain.

Bibliography / sources:

[1] Mandlík Šimon: Modelling Entity Interactions in Complex Heterogeneous Networks (Master's thesis), Prague, 2020
[2] Tomáš Pevný, Marek Dědič: Nested Multiple Instance Learning in Modelling of HTTP network traffic, Prague, 2020
[3] Tomáš Pevný, Petr Somol: Using Neural Network Formalism to Solve Multiple-Instance Problems, Prague, 2017
[4] A. Tibo, M. Jaeger, P. Frasconi: Learning and Interpreting Multi-Multi-Instance Learning Networks, October 6, 2020
[5] Gueltoum Bendiab et al.: IoT Malware Network Traffic Classification using Visual Representation and Deep Learning, Ghent, 2020

Name and workplace of bachelor's thesis supervisor:

doc. Ing. Tomáš Pevný, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **28.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until:
by the end of summer semester 2021/2022

doc. Ing. Tomáš Pevný, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. | Date of assignment receipt

Student's signature

Acknowledgements

I would like to thank my supervisor doc. Ing. Tomáš Pevný, Ph.D. for his patience, guidance and help. But most of all, I thank him for the knowledge and experience he gave me.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 20. May 2021

Abstract

The spread of malware is constantly growing, and along with the transformation of the world into digital form, this problem is an increasingly essential and discussed topic. There are various ways to detect it: analyzing a suspicious file, analyzing processes and activities inside the computer, or analyzing network communication. This work aims to compare completely different approaches to the classification of network communication of malware. The research is about the three approaches: the use of computer vision methods, examining network communication as a time series, and focusing on the hierarchical structure of communication. The hierarchical approach in this research gives the best results, as it allows to build a computational graph reflecting the structure of the problem.

Keywords: cybersecurity, computer vision, ResNet, LSTM, multiple-instance learning, network traffic

Supervisor:
doc. Ing. Tomáš Pevný, Ph.D.
Artificial Intelligence Center, FEE

Abstrakt

Šíření malwaru neustále roste a spolu s transformací světa do digitální podoby je tento problém stále důležitějším a diskutovaným tématem. Existují různé způsoby, jak jej detekovat: analýza podezřelého souboru, analyzování procesů a aktivit uvnitř počítače nebo analyzování síťové komunikace. Tato práce si klade za cíl porovnat zcela odlišné přístupy ke klasifikaci síťové komunikace malwaru. Jedná se o tyto tři přístupy: využití metod z oblasti počítačového vidění, zkoumání síťové komunikace v podobě časové řady a zaměření se na hierarchickou strukturu komunikace. Hierarchický přístup v tomto výzkumu podává nejlepší výsledky, jelikož umožňuje vybudovat výpočetní graf reflektující strukturu problému.

Klíčová slova: kybernetická bezpečnost, počítačové vidění, ResNet, LSTM, multi instanční učení, síťová komunikace

Překlad názvu: Hierarchické modely síťové komunikace

Contents

Appendices

Figures

Chapter 1

Introduction

1.1 Motivation

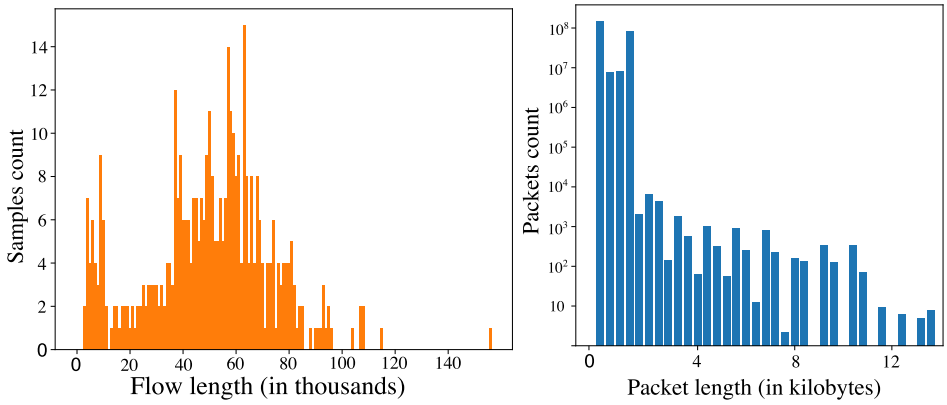
Malware is software designed to damage a target system, steal information, blackmail or in other ways harm users. With the amount of new perpetually generated malwares, it would be difficult to use manual methods for detection. An automatic analysis provide the only one possible solution, which is useful for mass usage. Automatic detecting of such software is a task solvable in different ways and approaches depending on the type and structure of available data. Possible ways are finding specific and pre-known signatures such as URLs, IP addresses, file paths or comparing fingerprints of suspicious files to known hashes in databases. In analyzing computer behavior, one of the detection options is at the level of internal-computer processes (for example, registry entries, DLL's usage, user interface accesses, and peripheral devices). The second option is the network behaviour analysis - examining the external communication of the computer - the movement of data over the network. This thesis deals with the topic of detection of infected computers based on the network communication behaviour of malware.

This thesis aims to explore the existence of a common concept of malware communication. The prerequisite for the above idea is the existence of a specific infrastructure of cybercrime [1]. Cybercrime has a hierarchical social structure with a small group of highly skilled actors at the top. The highly skilled group products the most of malwares in order to profit from its sale to wider communities of less qualified actors. After that, the malwares are adjusted to the final form and distributed to the targets. A narrow

group or groups of major producers only develop "semi-finished products" of malwares, tools for attacks on vulnerabilities in computer systems. However, especially these developers create the concept of communication infrastructure of malwares with command and control (C&C) servers. Given the above, it can be assumed that malwares collected simultaneously could have a similar network communication concept.

1.2 Problem statement

The network communication records represent a time-ordered flows of blocks of information transmitted in a computer network. These blocks of information, packets, contain information (can be hierarchically structured) about the recipient, the sender and the type of the block itself. Processing such data using machine learning methods then represents a more demanding task due to a more complex input data structure. The complexity of the input data of the solved problem lies in the following.



(a) : Size of flow. **(b) :** Size of packet.

Figure 1.1: Packets histograms

- At first, each computer, depending on the running processes, emits different amount of packets and communicate with different count of servers. According to the histogram of flow lengths 1.1a, it can be seen that the distribution of the packets count in flows is very uneven in the examined dataset (appendix B).
- At second, over the computer network are sent packets of defferent types (protocols) and each packet can be of different length. According to the attached histogram of packet lengths 1.1b in the used dataset, it is evident that packets of up to approximately two thousand bytes have

the most significant representation. The proportion of packets larger than five thousand bytes is declining considerably.

- At third, it is not clear how to structure the data: whether to sort them as a time series, or whether to group them according to the communicating servers.

Network communication takes place via various protocols according to the ISO/OSI model. Different protocols have different forms, designations and tasks. The Network layer (responsible for packet forwarding) includes, for example, the Internet Protocol for transporting data on packet-switched networks. The Transport layer performs communication services for applications over protocols such as UDP and TCP. Packets may also contain additional information from the Application layer that allows applications to access the communication system. This information includes, for example, a DNS record or TLS/SSL cryptographic protocols. Communication can take place via various packets containing various types of information. Within one communication flow, there can be packets with completely different purpose and headers (W. Richard Stevens in [2]). The data are thus very heterogeneous. The comparison of TCP and UDP packet headers is in 1.1 and 1.2.

Bit	0 7	8 15	16 23	24 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160 ...	Options and Padding			

Table 1.1: *TCP header*

Bit	0 7	8 15	16 23	24 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

Table 1.2: *UDP header*

As it is seen, the TCP protocol contains much more information in its header than UDP. It is not clear which of this information is relevant to the classification of the communication and if primary data (provided by the UDP header) are sufficient.

The goal of this thesis is to compare different machine learning approaches and data representation in solving a given problem. In this thesis, three approaches to solve a given problem are presented.

The first approach is the application of computer vision. The method converts packets flows into images and performs their classification. The advantage of this approach is the possibility of using various methods commonly used in machine perception. In the field of machine learning, there is currently a massive increase in surveillance such as face recognition or monitoring the movement of people. Furthermore, in recent years there is the development in mobile robotics. That causes the extensive development of methods for processing information from sensors sensing the robot's workspace and recognition of objects in its vicinity. From the above mentioned reasons, computer vision methods are well developed and popular in machine learning.

Another approach is to treat flow as a sequence, sequences are probably the second most common topic in machine learning presently. They are used, for example, in solutions of automatic translation, prediction and autocorrection of words on a smartphone keyboard, processing of DNA sequences or predicting market price developments. Depending on the needs, the prediction, generation or classification are performed. The last of these is the case of this research. There were used Long Short Term Memory recurrent neural networks. It examines the classification as a time series of packets. The advantage of this method is that the packet order information is maintained. It is not clear, whether time dependence is important for the identification of infected computers. If the packets' orders were not significant, there would be a great saving of computational time during neural network learning.

The third approach uses hierarchical multiple-instance learning. Thus, it allows easily to propagate the data structure to the model. The neural network architecture then reflects the form of network communication. Thanks to this, it is possible to cope with heterogeneous, incomplete data, such as packets. As mentioned above, the complexity of the task lies in the fact that the problem has sequence effects, and at the same time, each item has its hierarchical description. The ambiguity of the hierarchical approach is in the problem of data structuring. Since the goal of the hierarchical approach is to model interactions in a computer network, the following approach is offered. The sample is modeled as a set of servers (identified by IP addresses) that communicate with the monitored computer. Each of these servers has as its hierarchically structured features packets that represent the communication of the monitored computer with the server in question (introduced by Pevny and Dedic in [3]).

The processing of large, especially heterogeneous, data places great demands on hardware. The identification of the significian properties of the data for classification can allow modification the network architecture, mainly hierarchical models, to better focus on proper information. For hierarchical multi-instance learning models, removing less significant instances will reduce

the number of model parameters and ultimately saves computational time during network learning. Therefore, this thesis aims to explain the decisions of neural networks and identify a subset of samples or instances which are considered crucial to the correct classification; and identify artefacts that could characterize some malware strains.

If malware architects want to stay underdetected all time, they must constantly improve, change and mask their products. Because of this, malwares mutates and perpetually changes its characteristics. Due to this fact, more malwares of the same strain, purpose and the infection target, produced with a longer time interval, may have completely different attributes. This fact creates extensive non-stationarity in the data from a long-term perspective on the problem. For this reason, the instances that the model considers essential for classification also change, and their values may be affected by the particular dataset used.

This thesis is organized as follows. The **Part I** includes two chapters (*Prior art* and *Proposed approaches*) in which are compared approaches of classification. The *Prior art* chapter deals with the approaches from the computer vision and sequences classification. The *Proposed approaches* introduces new methods for approaches from the prior art. Next this chapter describes the concept of multiple-instance learning and introduces the new method of classification using hieararchical multiple-instance learning. The **Part II** includes three chapters (*Results*, *Interpretations* and *Conclusion*), which summarize the results and interpretations of the individual approaches.

Part I

Compared approaches

Chapter 2

Prior art

At the state of the art, malware communication is often classified using computer vision methods or sequence processing methods. These methods have in common that they convert packets flows of different lengths into samples of constant dimensions. From the heterogeneous input data, homogeneous samples must first be obtained by initial preprocessing.

2.1 Visual representation classification

The first approach to solve the introduced problem verifies the solution presented by Bendiab et al. in [4] and [5] proposing a novel IoT malware traffic analysis. The method consists of converting a complex problem into an easier-to-solve problem in the field of computer vision or machine perception. Hence, the first part of the research lies in malware network communication classification through a visual representation (transformation into images) of packets captured files. The method's goal is to convert a problem of hierarchically formatted (time dependent) data classification into a problem focusing on classification of images in computer vision. The incoming pcap (packet capture) files are converted into images by the Binvis tool (more detail in appendix A.1) [6]. Binvis treats the network capture as a sequence of bytes and convert this one-dimensional sequence to two-dimensional image using space filling (hilbert) curves. Simultaneously with the mapping, bytes that are close in packets are projected onto pixels close in the image. The possible disadvantage of the method is the fact that the hierarchisation is completely neglected here. If its unique communication structure typically

characterises the malware, this fact is unlikely to be sufficiently highlighted in the visual representation.

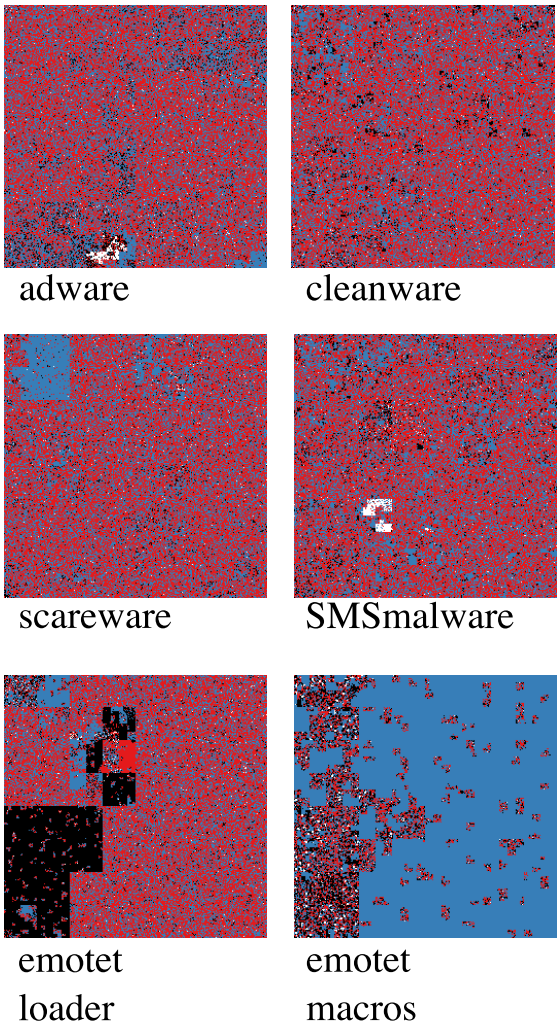


Figure 2.1: Examples of Binvis images of malwares.

The figure 2.1 shows the examples of the encoded communication of various malwares. Malware traffic images include a predominance of black pixels (null bytes) or blue (ASCII readable) in some images' parts. Compared to this, cleanware traffic images do not contain any clusters of monochromatic pixels or any characteristic patterns. For example, Emotet (specifically malware that generates a macros-using document) is similar to Scareware malware in blue patterns. That is caused by a larger volume of downloaded human-readable text data. This preprocessing enables to use plethora of methods from the field of computer vision. The figure 2.1 shows, that these 2D images of network traffic of different malwares can be easily recognizable by naked eye. In machine learning is generally the most common application

a computer vision, so this method has a tremendous advantage in accessing many different libraries and architectures designed to process visual data. Bendiab et al. states that the best accuracy is achieved by residual neural networks. Bendiab et al. state that although ResNet50 accuracy is above 92% on binary classification, there was a problem with the convergence of the training data during training.

2.2 Classification of sequences

The second approach of this thesis considers network connection as a time flow of non-hierarchical data. This research's primary goal is to verify if the time dependencies in sequences of network communication plays a role in the classification of infection. The secondary goal is to identify which part of the packet (header or data body) is considered more important by the neural network model.

A packet is a block of information written in bytes. Their flow can thus be formally expressed as an ordered tuple of vectors whose items correspond to bytes. The solution methods work with these ordered tuples of vectors. The initial problem that had to be solved lies in the number of packets' inhomogeneity and flows dimensions. Data inhomogeneity makes it difficult to use convolutional neural networks (preprocessing such as interpolation would be needed). Furthermore, too long a packet flow length would place a significant burden on computing power when training recurrent neural networks. The elimination of that problems lies in setting the threshold hyperparameters for the input flows and packets. The first n packets of the flow are considered as input data, and the rest will be truncated. Furthermore, at the same time, setting a uniform fixed length for each packet (if the packet is shorter than the set limit is zero-padded). The two steps mentioned above give samples of fixed dimensions to which convolutional and Long Short Term Memory recurrent neural networks can be applied without the need for further preprocessing. At the same time, it can affect the computational demands during training.

Bernaille et al. in [7] performed network traffic classification with only the first five packets of the flow. Lin et al. in [8] states that traffic classification could be based on only headers of packets - packet payload may be completely ignored. Gonzalo Marín, Pedro Casas Germ and Germán Capdehourat in the [9] presented the method of solving detection using only convolution.

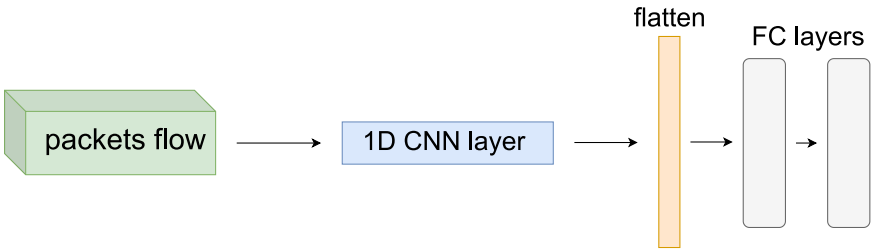


Figure 2.2: Marín et al. network.

Marín et al. state that model with more convolution layers started to overfit quickly. After the gradual reduction of parameters through removing model's layers, the final network 2.2 consists of a 1D convolution layer (processing individual packets) followed by two fully connected layers. The model is simple and needs a large dataset for its proper generalisation (Marín et al. used a dataset of 67,000 samples). Requiring a large dataset is a significant problem with this method.

Thapa and Duraipandian in the [10] presented the approach implementing Long Short Term Memory recurrent neural networks. The feed-forward neural networks can only very poorly detect the interdependencies between elements. The long-term dependencies in series are thus lost. The Long Short-Term Memory (LSTM) used instead of feed-forward neural network could solve that problem (described in more detail in appendix A.6). Thapa and Duraipandian proposed architecture with LSTM nodes and fully-connected layer.

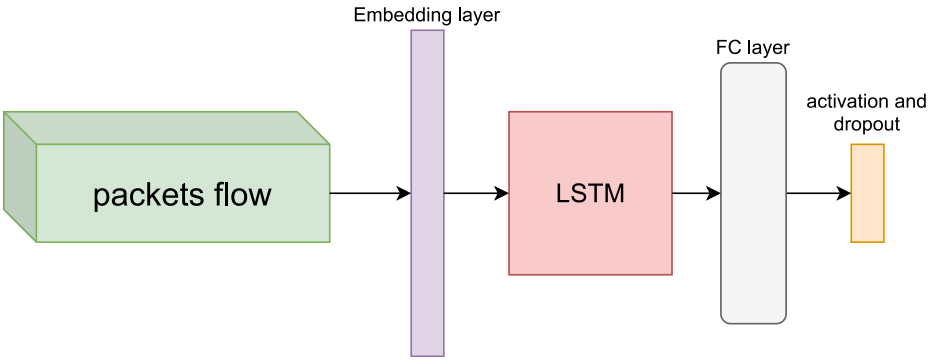


Figure 2.3: Thapa and Duraipandian network.

In 2.3 the fixed sizes packets flow enters an embedding layer; then is placed LSTM node (processing packets as features of timestamps) and a fully-connected layer.

Chapter 3

Proposed approaches

3.1 Proposed approaches in visual representation

Due to the poor convergence of models with a high number of parameters presented by Bendiab et al. this work proposes another classifiers. However, the data for the proposed methods are preprocessed in the same way (conversion of packet flows into images) as described in the prior art chapter. The aim of the newly proposed methods is to perform the classification with a model that has fewer parameters and thus avoid overfitting.

3.1.1 k-NN

The first approach was to perform the k-NN classifier, which is considered the simplest classification method in machine learning and data mining (Asim and Zakria [11]). The fundamental advantage of k-NN over other classification methods and especially over neural networks lies in the fact that it is a lazy learning because there is no need to build a model. The main problem is how to define the metric between samples. The problem in choosing the right metric for the k-NN classifier in machine perception is that for some metrics (such as L2) only a tiny difference in the image's pixels changes the sample's distance from the origo. Suppose a visually apparent anomaly characterises a malware class in the image. However, the event occurs in a different (temporal) part of the communication than in the training data. In

that case, the anomaly is also encoded in another part of the image. In this situation, the nearest neighbour classifier may fail because other validation data samples than those that are part of the training set become the proper classification information's bearer. A possible solution to this problem could be to use maximum cross-correlation as a metric.

3.1.2 Neural networks

The second approach of classification was based on the usage of convolutional residual neural networks. Convolutional neural networks are among the most common ways to classify problems with images as inputs and recognise specific patterns (such as face recognition) while preserving information about their positions (described in more detail in appendix A.3.1). The choice of residual neural networks A.3.2, introduced by He et al. in [12], was due to they may solve the problem of vanishing gradients. In this approach were compared two ResNet architectures that have significantly fewer parameters than the proposed ResNets by Bendiab et al. The first network, ResNet18, is an architecture with 11,188,941 parameters (11,180,999 trainable). The second network, Resnet_s ("s" means "smaller"), is an architecture designed because of the need to have a residual network with fewer parameters (467,661 parameters, 466,119 trainable) to prevent overfitting.

Experimental settings. The networks are built in Keras-TensorFlow with usage of Classification models Zoo library [13]. Both networks has an input image of shape (256, 256) and an output vector of four classes. Networks' training was performed with Adam optimizer, which is one of the most common optimizer algorithms used to update network weights parameters based on a training dataset. The algorithm combines Adaptive Gradient Algorithms (AdaGrad) and Root Mean Square Propagation (RMSProp). As a loss function was used the Cross Entropy. It is a good and common used loss function for classification problems, because it minimizes the distance between two probability distributions - predicted and actual.

3.2 Proposed approach in classification of sequences

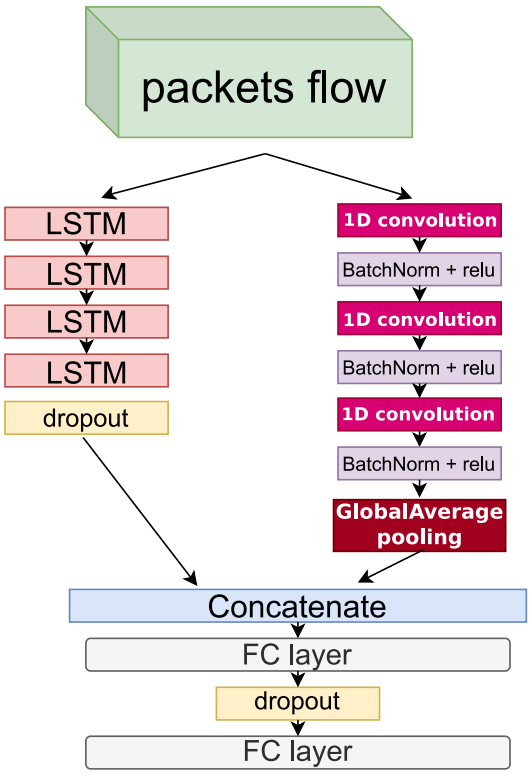


Figure 3.1: Proposed network with LSTM.

In order to improve the accuracy of the classification even when training on a small dataset, there was changed the concept of neural network architecture. The idea was based on the architectures of the previous researches, but the network was split into two streams, which handle the same input (*CNN-LSTM combined architecture*). The streams (CNN and LSTMs) are connected to the last decision-making layer. Given that two completely different methods are processing the input information, the model has gained greater robustness. The combined model returned better accuracy results than separate usage of CNN and LSTMs models.

The first stream of the network contains a sequence of stacked recurrent LSTM blocks and handles time dependencies between packets. The second part of the network implements a sequence of 1D convolutions terminated by global average pooling, thereby processing the packets' contents. The streams are concatenated and followed by a fully-connected layer with softmax.

Packet inhomogeneity and extent of their flow make the approach a significant load in hardware memory requirements during training of the network. For this reason, it was forced to select the sample only the first n packets from each stream. At the same time it was decided to choose a threshold of k bytes for each packet.

Experimental settings. The training was performed with Adam optimizer with the preset learning rate 10^{-4} and Cross Entropy loss. According to the histograms of packets and flows in the introduction problem statement was choiced to set first hundred of packets as input data, which means that the method only examines the effect of the type of infection on the infected computer's initial communication. Since, according to the packets histogram, the largest representation is up to two thousand bytes in packet length, and from five thousand bytes the packet frequency decreases significantly, the bytes threshold was set to 4096.

3.3 Multiple instance learning and hierarchical concept of network communication

Given that the presented problem can be formulated by a hierarchical structure, by generalizing multi-instance learning into hierarchical multiple-instance learning, it is then possible to build neural networks that accurately reflect the data structure of the problem. The leaves of such a graph then correspond to the instances in hierarchical multiple-instance learning.

3.3.1 MIL overview

Solving problems that deal with real-world data is very difficult to describe by fixed size numerical vectors or tensors. Problems can lie in the incompleteness of available data or inhomogeneity (heterogeneous data can be represented by vectors of different lengths). Most traditional approaches in machine learning (such as convolutional neural networks) can not be easily used or do not make sense in solving such a problem. That problem can be partially or sometimes completely solved with the help of multiple-instance learning. Multiple-instance learning (MIL) is a type of supervised learning. The first concept of MIL, Learning with many irrelevant features, was introduced by Dietterich et al. in 1991 [14]. Dietterich et al. also introduced MIL in

[15]. In the standard typical machine learning technics are input samples represented by tensors or vectors of fixed dimensions; however, as opposed to that, Multiple-instance learning samples are sets of tensors and vectors. In the MIL terminology, these sets are called bags, and contained vectors are called instances. There exist labels for each instance, but these instance-level labels are not known, even during the training. The known labels (ground truth information) are available only on the higher level of samples (bags). Let $\mathbf{b}$ be a bag from a bag space $\mathcal{B}$; let be y its label from a finite set $\mathcal{C}$ and x_i instances in the bag from come from instance space $\mathcal{X}$, then

$$\mathbf{b} = \{x_i \in \mathcal{X} | i \in \{1, ..., |\mathbf{b}|\}\}. \tag{3.1}$$

Based on the above terminology, in multi-instance learning, the model is defined as mapping $f : \mathcal{B}(\mathcal{X}) \to \mathcal{C}$. There are three approaches of bag classification: *instance-space paradigm*, *bag-space paradigm* and *embedded-space paradigm* (Pevny and Somol in [16], Tibo et al. in [17]).

Instance-space paradigm

In the *instance-space paradigm* is the classification function trained on the level of raw instances in the meaning $f : \mathcal{X} \to \mathcal{C}$ (Carbonneau et al. in [18]). An aggregation function gives the result of classification:

$$f(\mathbf{b}) = g(\{f_I(\vec{x})\}_{\vec{x} \in \mathbf{b}}), \tag{3.2}$$

where f_I is a instance-level classifier. In the standard MIL assumption is the aggregation function defined as a max function. This choice implies for a binary classification that a positive bag contains at least one positive labelled instance. The model is then designated as

$$f(\mathbf{b}) = \max_{\vec{x} \in \mathbf{b}} f_I(\vec{x}). \tag{3.3}$$

Frank and Xu [19] introduced a mean aggregation function that averages the sum of probabilities of all classes determined by the instance-level classifier by the number of instances in the bag.

$$g(\{f_I(\vec{x})\}_{\vec{x} \in \mathbf{b}}) = \frac{1}{|\mathbf{b}|} \sum_{\vec{x} \in \mathbf{b}} f_I(\vec{x}), \tag{3.4}$$

where $|\cdot|$ denotes the cardinality of a set. This way, the average class belonging to the bag is obtained.

Generalization leads to problem-solving when a bag class is identified by mutual interactions of certain instances or accumulating several instances (Foulds and Frank [20]).

Bag-space paradigm

The bag-space paradigm is defining principle which assumes an existence of a function measuring the similarity of samples. Based on their similarity, the classifier makes the decision. This corresponds to mapping from bag space to labels space $f : \mathcal{B} \rightarrow \mathcal{C}$. For machine learning methods based on the existence and meaningfulness of a normalization function (k-NN classifier or SVM) to be used for bag classification, the distance between two elements (bags) must be defined in the given space as a distance function $\text{dst} : \mathcal{B} \times \mathcal{B} \rightarrow \mathbb{R}_0^+$. Unlike the instance space, bag space is not very often expressed in Euclidean space. Assuming that instance-space has metrics in place, the following relationships can be used, for example, to calculate bag spacing.

Let $\mathbf{b}_i$ be a bag of instances x_{ij}. *Earth Mover's Distance (EMD)* is defined as

$$\text{dst}(\mathbf{b}_1, \mathbf{b}_2) = \frac{\sum_{\vec{x_1} \in \mathbf{b}_1} \sum_{\vec{x_2} \in \mathbf{b}_2} w_{x_1,x_2} \, \|x_1 - x_2\|}{\sum_{\vec{x_1} \in \mathbf{b}_1} \sum_{\vec{x_2} \in \mathbf{b}_2} w_{x_1,x_2}}, \tag{3.5}$$

where the weights w_{x_1,x_2} are gained through an optimization process that minimizes the introduced function (for example, using the simplex method). The minimal *Hausdorff distance* is defined as a distance between the two nearest instances of the two bags.

$$\text{dst}(\mathbf{b}_1, \mathbf{b}_2) = \min_{\vec{x_1} \in \mathbf{b}_1, \vec{x_2} \in \mathbf{b}_2} \|\vec{x_1} - \vec{x_2}\| \tag{3.6}$$

Thanks to the metrics introduced in this way, the *k-NN classifier* or a kernel-based classifier such as *support-vector machines* at the bag level can be used (J. Amores in [21]).

Embedded-space paradigm

Unlike the Bag-space paradigm, which defines the distance between bags according to the instances' distances, the embedded-space paradigm performs the explicit mapping from bag-space to the feature space in a way $\mu : \mathcal{B} \rightarrow \mathbb{R}^n$. The feature vector carries information that is essential for the characterization of the bag. Space to which the vector belongs is constructed as a Cartesian product of partial mappings of the bag $\mathbf{b} \in \mathcal{B}$.

$$\mu(\mathbf{b}) = (\mu_1(\mathbf{b}), ..., \mu_m(\mathbf{b})) \tag{3.7}$$

The selection of the information depends on the used mapping function. Lin Dong [22] proposed the Simple MI method (also proposed by Bunescu and Mooney [23]), that maps each bag as an average of its instances.

$$\mu(\mathbf{b}) = \frac{1}{|\mathbf{b}|} \sum_{\vec{x} \in \mathbf{b}} \vec{x} \tag{3.8}$$

Gärtner et al. in [24] propose a max-min vector strategy

$$\mu(\mathbf{b}) = (\mu_{1,1}(\mathbf{b}), ..., \mu_{1,m}(\mathbf{b}), \mu_{2,1}(\mathbf{b}), ..., \mu_{2,m}(\mathbf{b})), \tag{3.9}$$

where

$$\mu_{1,i}(\mathbf{b}) = \min_{\vec{x} \in \mathbf{b}} x_i \tag{3.10}$$

and

$$\mu_{2,i}(\mathbf{b}) = \max_{\vec{x} \in \mathbf{b}} x_i. \tag{3.11}$$

The advantage of the above embeddings lies in their very low computational complexity, but they do not always prove to be ideal for distinguishing bags with differently structured instances. Vocabulary-based methods provide solutions. They consist of determining predefined patterns of typical bags and their structured instances. Instances are then compared to how well they match their patterns. The degree of similarity can be calculated either as the distance of instances from their patterns. That is the Distance-based method. The partial mapping function is defined as

$$\mu_i(\mathbf{b}) = \min_{\vec{x} \in \mathbf{b}} \left\| \vec{x} - \Theta_i \right\|, \tag{3.12}$$

where the Θ_i is the corresponding pattern for instance $\vec{x}$. Another way to grasp the solution to the problem is histogram. When assembling it, the distances of the instances from the corresponding patterns are not measured, but the degree of similarity is measured based on some likelihood function. The histogram is in a form $\vec{v} = (v_1, ..., v_n)$, where v_i corresponds to the partial mapping

$$v_i = \frac{1}{z} \sum_{\vec{x} \in b} l(\vec{x}, \Theta_i), \tag{3.13}$$

where l is a likelihood function normalized by z constant (J. Amores in [21]).

Adapting neural networks to MIL

There were introduced multiple-instance learning approaches. Along with this, the following have been introduced: a function for classifying individual

instances $f_I(\vec{x}, \Theta_I)$ (with parameters Θ_I and input instance vector $\vec{x}$), an aggregation function g, which is a necessary part of a partial mapping in the embedded-space paradigm, and a bag-level classifier $f_B(\bar{x}, \Theta_B)$ with parameters Θ_B. Pevny and Somol introduced the MIL model in [16], in which individual instances are first mapped at the lowest level. The obtained intermediate results pass through the element-wise aggregation function. As the last step, the information will be processed using the network as a bag-level classifier. The formal expression of the process described above is:

$$\begin{aligned} \tilde{x}_i &= f_I(\vec{x}_i, \Theta_I) \\ \bar{x} &= g(\{\tilde{x}_i\}_{i=1}^{|b|}, \Theta_g) \\ y &= f_B(\bar{x}, \Theta_B) \end{aligned} \tag{3.14}$$

The main advantage of the method presented by Pevny and Somol is the optimization of the classifier at the same time as the optimization of embedding. Their method performs the calculation recursively. It assigns instances to the computational tree and then sequentially from leaves (instances) performs calculations toward the root node.

3.3.2 HMill overview

As mentioned in the introduction, network communication is a set of hierarchical data; hence was used the HMill framework (Hierarchical multiple-instance learning library; described by Simon Mandlik in 2020 [25]; created by Mandlik, Pevny and Racinsky [26], [27]) to model them and generate a neural network. HMill was created by generalizing the multiple-instance learning described above. It accurately takes into account the hierarchical structure of the problem, using the MIL paradigms. The input data can be organized into the hierarchical structure, which is reflected by the model. The structure consists of nodes, which together form a tree-type graph. The tree leaves process input instances (low-level raw information). The middle part of the model is is responsible for processing abstract intermediate results and the root of the tree model corresponds to the model output. The evaluation takes place gradually from leaves to roots - parents are waiting for the results processed by their children. Thus, it is a tree-based computational graph, where each of the partial functions is differentiable from its inputs (Pevny and Dedic [3]).

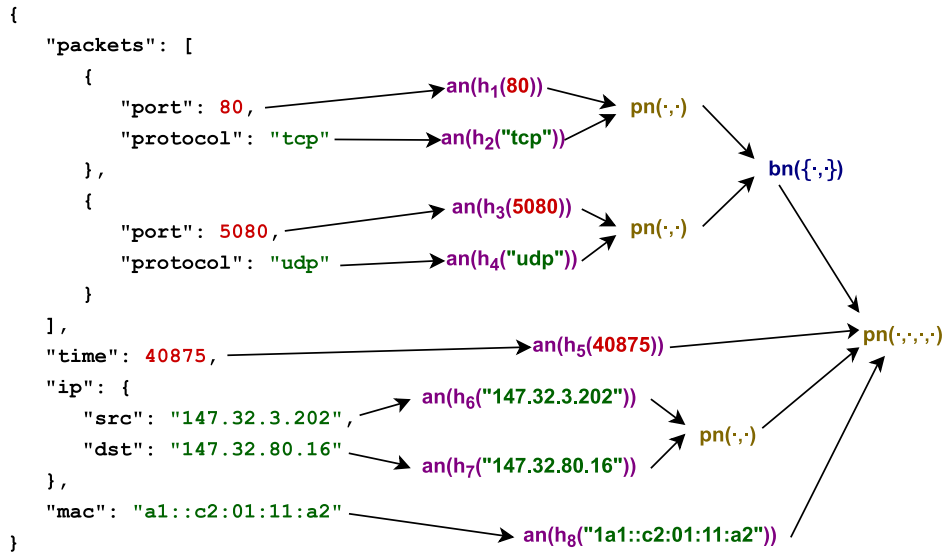


Figure 3.2: A transformation of JSON into a HMill sample.

The figure 3.2 shows an example of the transformation of hierarchically structured data from JSON to HMill sample. All instances (leaves of the computational graph) are mapped into array nodes an() with mapping h_i (*n-gram histograms*, *one-hot encoding*, *identity mapping*). Array nodes are stored into product nodes pn(). Product nodes enable as inputs nodes of different types. Bag nodes bn() enable as inputs only nodes of the same type. Individual mappings between nodes may use different layers (or neural networks) and different aggregation functions.

Array Node / Model

All low-level input information is stored in Array nodes. Input data can be very variable - it can be boolean variables, text strings, numbers, arrays or any other categorical variables. Various procedures are used to encode inputs into a mathematically graspable vector form. Boolean is converted as a binary value using one-hot encoding. Text strings are encoded using *n-gram histograms* [1]. Numerical vectors are themselves in Euclidean space elements, and thus identity mapping is entirely sufficient for their conversion. Categorical variables are processed using one-hot encoding. The above methods transform the input instances into a numerically graspable form and store them in an Array Node. The process of mapping to Euclidean space is defined as the Array Model.

[1] An n-gram is defined as a sequence of n consecutive arbitrary items from a given series.

Bag Node / Model

Bag Node is an analogous concept of storing information to the concept of a bag from multiple-instance learning. A Bag Node can contains various items of exactly the same type. The count of items can be arbitrary (it can also be an empty set). If all elements come from the instance space (these are the tree model leaves), they are stored in array nodes. Elements that are themselves trees are stored in bag nodes. The Bag Model bm(f_I,g,f_B) is a composition of Bag Node elements models (processed by f_I), an aggregation (element-wise) function g and bag mapping f_B (transformation into the target space). The Bag Model applies the same mapping to all its children.

Product Node / Model

Product Node (it is the Cartesian product) joins and combines heterogeneous data from various sources - whether other Product Nodes, Array Nodes or Bag Nodes. Product Node accumulates data (hierarchical trees) with different structure, meaning and type. Product Model pm(f_1, ..., f_n, f) analogously to Product Node contains submodels of various types. Unlike the Bag Model, it can apply a unique mapping function f_i to each submodel. The results of these mappings are concatenated and transformed by the f function into the target space.

3.3.3 Proposed hierarchical models

As mentioned above, HMill is a purely hierarchical approach to solving the given problem. The disadvantage of this method is the loss of information about the time sequence of packets. The pattern of network communication in this method can be interpreted as a continuous graph without loops - a tree whose root represents the monitored system or sandbox. Formally written: let $G(\mathbf{V}, \mathbf{E})$ be a tree graph, $\mathbf{V}$ be a set of all vertices and $\mathbf{E}$ set of edges. Let R be the root of the tree and $\mathbf{W}$ be the set of all its neighbours ($\mathbf{W} = \mathbf{V} \setminus R$). All vertices W_i of the graph $G(\mathbf{V}, \mathbf{E})$ represent the systems communicating with the monitored sandbox. The degree of the vertex that is the root R of such a graph is equal to the cardinality of the set of all communicating IP addresses (figure 3.3).

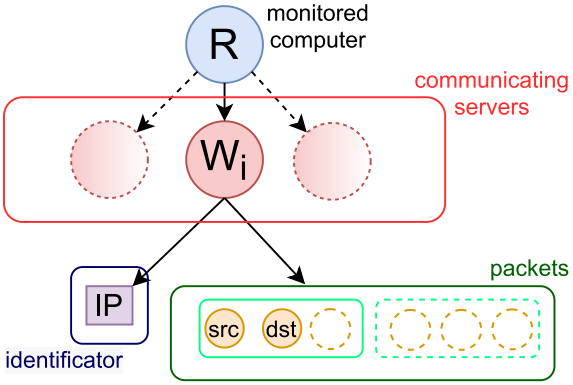


Figure 3.3: Communication representation.

All vertices from the set V are uniquely identifiable by their IP address. Each W_i vertex is a subroot of the tree subgraph, which carries information about the communication between W_i and the main root R (packets content). Depending on the available data and considering their hierarchical structure, different models can be designed.

JoyHMill

In the first approach, there is the network traffic represented in the same way as is provided by JOY tool [28]. The data contain information on the total volume of bytes transferred between the monitored computer and the communicating server, the total time of their mutual communication and the number of packets. Packets are identified, by the unique number, in the flow and are simultaneously written in the two variants of structures. The first variant divides them into two groups according to the direction of their flow. The second variant collects an array of all packets. There are assigned the properties of the packets: data part bytes, direction, and the time within the stream. The main difference between this approach and the following is that Joy is numbering packets (in the following approaches, packets are only in the form of an unordered set).

L-HMill

The second approach (Larger schema HMill) structured the data differently from the first one - it assigned only a packet array to each communicating node W_i. The 3.4 schema shows a modeled tree. The W_i vertex has got as its children vertices the pakets sent between monitored root computer R and W_i. Each packet, as its children, contains a set of properties that describe itself (DNS record, UDP, TCP, IP). These properties have as children input instances.

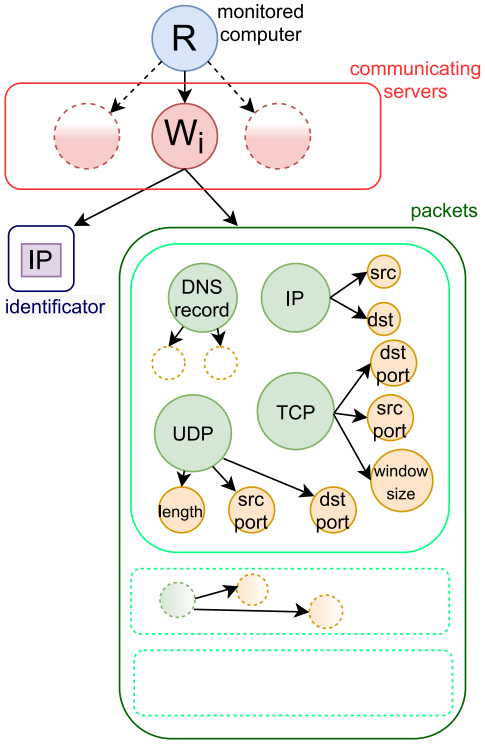


Figure 3.4: L-HMill schema.

In addition to packet length information and communicating ports, the tree also contains the information provided by DNS servers in the form of a *DNS record*. DNS records determine which services run on a given Internet domain and the appropriate type activates the service and sets its parameters. It can be used (among other) for the following purposes.

- Translating the domain to the specific IP address.
- Specification which certification authority (CA) can issue the SSL certificate about domain. It ensures a response from the authoritative DNS

server and not from another server whose response could be fraudulently pushed to the computer.

- An indication of which domain server manages DNS records.
- Specify information about available services on the domain.
- Determining to which mail server is the domain routed.

S-HMill

The third approach (Smaller schema HMill) structured the data in the same way as the *L-HMill* approach. The only difference is that this method does not consider a *DNS record* and therefore it is a model with significantly fewer parameters, which can save computing time.

Experimental settings. In all models there were used as instance-level classifiers dense layers to process information from tree leaves inside the array models. Also, bag models and product models have dense layers set as classifiers with twenty neurons per layer and ReLU activation functions (benefits of ReLU are sparsity and a reduced likelihood of vanishing gradient). Furthermore, in order to increase the accuracy of the classification it was experimentally tried to use a residual network and a neural network containing a dropout instead of a single dense layer as product and bag models classifiers. But this did not affect the accuracy and only slowed down convergence. Mean-max (concatenation of mean and max) was chosen as the aggregation function. The choice of mean-max is such as it is not clear whether one of the most important instances of the bag is more important for classification, or whether it is more appropriate to identify the global trend of the bag. Cross Entropy (well used in classification problems) was chosen as the loss function and Adam as the optimizer. The training was performed on 400 epochs, because the convergence was very slow.

Part II

Results and conclusion

Chapter 4

Experimental results and conclusions

4.1 Results

The training dataset consisted of 342 malware samples (the validation consisted of 84 samples), and samples were split into four classes. The malware dataset is described in more detail in the appendix B.

4.1.1 Visual representation

When verifying the use of computer vision, the first approach was a k-NN classifier with metrics *p-norm* and *maximum cross-correlation*. The following graph 4.1 shows that the best accuracy for validation is achieved by the k-NN classifier for k = 5 using the euclidean distance - the best accuracy is 41.75% (for comparison, the random choice classifier has got an accuracy of 25%). The second approach was the application of residual neural networks. The ResNet50 and ResNet34 networks have been (during replication of Bendiab et al. approach) highly overfitted. Although networks with a smaller number of parameters also had a problem with overfitting, they already achieved better results. The ResNet18 has over validation dataset accuracy of 45.99%. As is visible in the figure 4.2, the ResNet18 has a problem with overfitting. The training accuracy is in thirty epochs, almost at 100%; however, the validation accuracy oscillates all along. Very similar behaviour can also be observed on the loss curves. During the fluctuation of the validation accuracy

curve, the method gets into the local minima. In the process, the best result has a minimum in the sixty-fifth epoch. ResNet_s converged significantly better than ResNet18 but had significant problems exceeding the validation accuracy of 45%. Because learning a neural network with an Adam optimizer is a stochastic method, different pieces of training can produce different results. After several repeated learning, the model converged to a point with parameters that ensured a validation accuracy of 55.3%. As shown in the figure 4.2, the training accuracy is forty-three epochs, almost 100%. Although the training loss curve has only changed in the order of hundredths since then, the validation loss is still slightly decreasing to a local low in the sixty-second epoch.

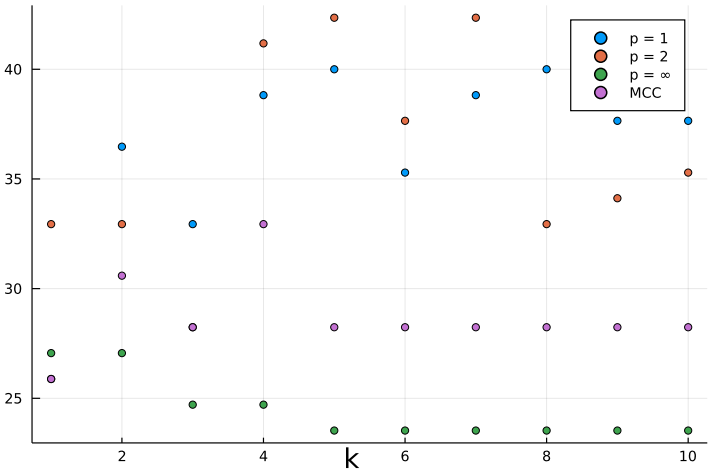


Figure 4.1: k-NN: accuracy.

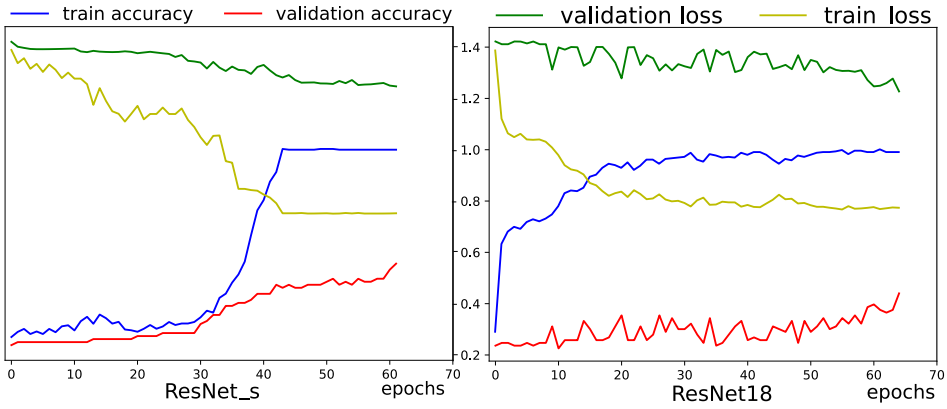


Figure 4.2: ResNets epochs.

4.1.2 Classification of sequences

In the first approach in classifying data sequences (replication the approach designed by Marín et al.), the model did not converge at all (nor on training data). The model could not distinguish the features of the instances from the noise; this failure could be attributed to the defect of dataset (dataset size or poorly collected data). Model processing data using LSTM and FC layer could not exceed 36% validation accuracy. Although it is more accurate than the random classifier, it is still significantly less than the accuracy of the k-NN classifier used in the visual representation approach. The problem is highly probably caused by the too small dataset.

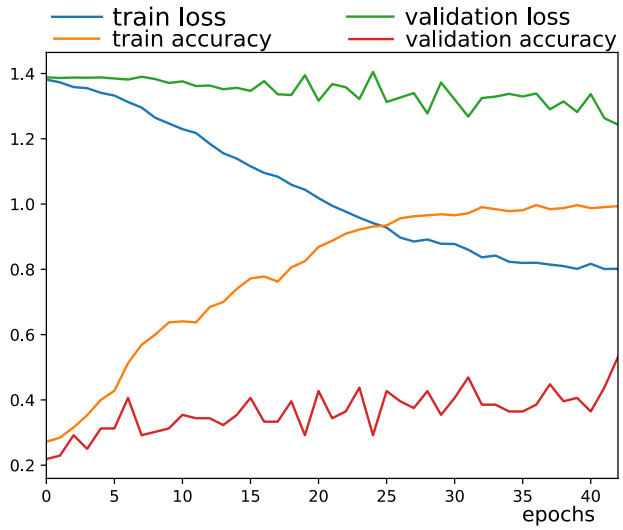


Figure 4.3: CNN-LSTM combined architecture epochs.

The newly designed, *CNN-LSTM combined architecture* achieved in classifying packet flows much better results. As can be seen from the chart 4.3, the training accuracy converged relatively stably, but the validation data did not exceed (even after repeated learning attempts) the accuracy of 53.25%. The biggest problem with the classification was with Ransomware, which confused the model with SMSMalware. From this, it can be concluded that the initial phase of communication with this two malware is similar. Simultaneously, according to the achieved results, it can be assumed that Scareware has a typical initial communication for its class.

4.1.3 HMill

The *JoyHMill* neural network (built according to the first approach based on data from CiscoJoy) did not converge even on the training dataset. The second and the third approaches to modelling hierarchical data (*L-HMill* and *S-HMill*) showed better results on both the training and validation datasets. Due to the slow learning of neural networks, training was stopped prematurely after 400 epochs. The larger model (*L-HMill*) reached an accuracy of 100% on the training data and 82.75% on the validation accuracy. The smaller of the models (*S-HMill*) achieved a training accuracy of 96.67% and a validation accuracy of 86.96%. Since the models differ only in the DNS part, it can be concluded that the DNS record slightly negatively affects the classification. Above all, however, it is a good knowledge that a model containing less input information achieves sufficient results (more than three times better accuracy than a random classifier) and thus computational time can be saved considerably.

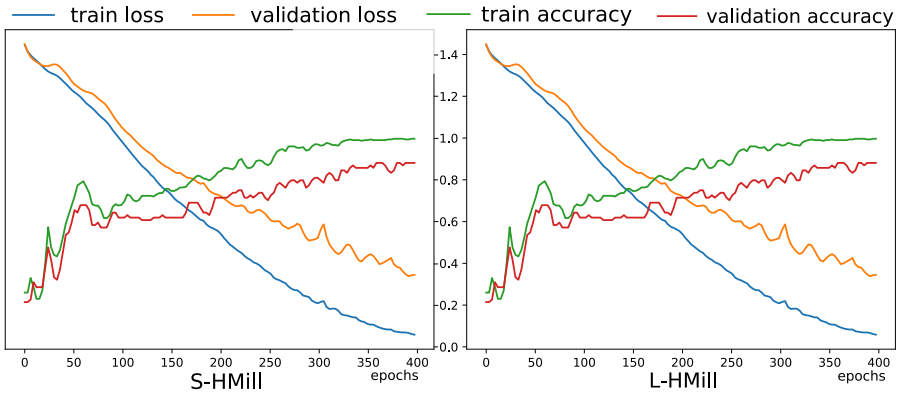


Figure 4.4: HMill epochs.

4.1.4 Summary

model	class accuracy				average accuracy
	Adware	Ransomware	Scareware	SMSMalware	
5-NN (L2)	0.25	0.77	0.24	0.41	0.42
ResNet18	0.55	0.50	0.43	0.36	0.46
ResNet_s	0.40	0.55	0.43	0.83	0.55
CNN-LSTM	**0.67**	0.19	0.77	0.5	0.53
S-HMill	**0.67**	0.89	**1.00**	**0.92**	**0.87**
L-HMill	0.56	**1.00**	0.87	0.88	0.83

Table 4.1: *HMill: accuracy on validation dataset.*

4.2 Interpretations

4.2.1 Interpretation of visual representation

The task is to classify images. The visual explanation algorithms, such as Grad-CAM or SHAP, will serve to create attention maps and gain knowledge which parts or regions of images are essential for each class. The SHAP method's implementation did not show any significant focus of neural networks on the features of entities. According to Grad-CAM (figure 4.5), ResNet18 focuses, in addition to corners, on practically the entire image. That could indicate that the model is too complex for the problem and dataset size. In contrast, heatmaps for the ResNet_s model already show some differences. However, the disadvantage, is that it is not easy to read from the heatmap which parts of the network communication are essential for the attribution. The transformation of the input data is performed to gain fixed-sized images so that some bytes are skipped. Therefore, performing decoding of images back to pcap (packet capture) files would not be possible. Thus it is impossible to decide whether the infection types differ in the header of the packet or its data part, nor if some IP addresses are typical for infection.

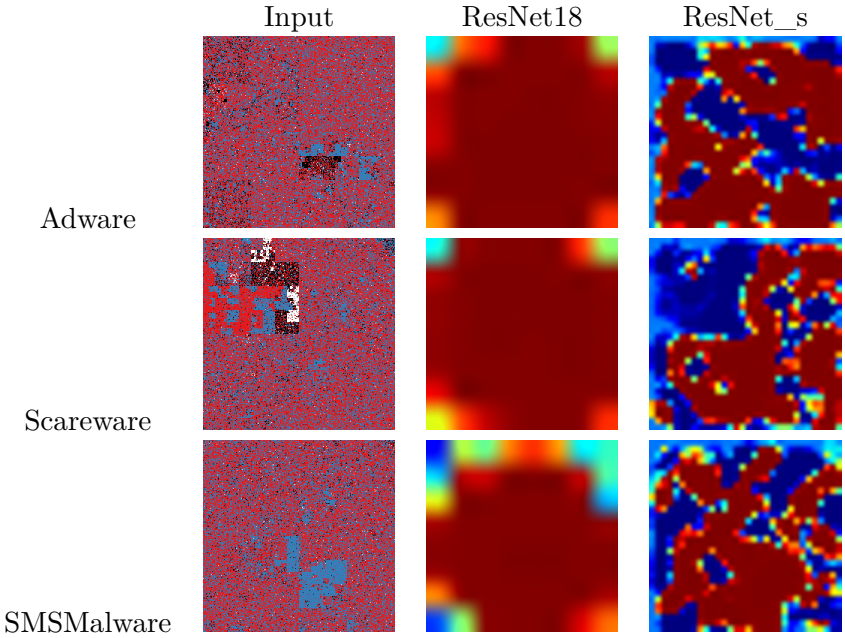
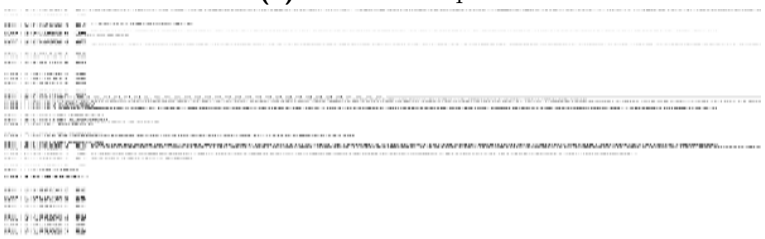


Figure 4.5: Grad-CAM heatmaps

4.2.2 Interpretation of sequences classification

The task is to classify inputs in byte format, so the samples could be screened as grayscale images (figure 4.6). The results of the Score-CAM algorithm show that the most important for the CNN-LSTM combined architecture are headers (visible in the left part of explanation images) of all packets, as well as complete packets with a large (above-average) data part. The headers of packets can contain enough information for classification. That is the crucial knowledge that can reduce the memory load of the input information. This information could be also used in data preprocessing to build hierarchical models.

(a) : Adware: input



(b) : Adware: heatmap.

Figure 4.6: Example of Score-CAM interpretation.

4.2.3 Interpretation of HMill results

The best accuracy on validation data was achieved by the S-HMill model. For this reason, it was examined which artefacts are targeted by that model. According to the results of the explainer based on Shapley values, the neural network focuses on IP addresses. The proper classification probably depends on the combination of particular IP-ranges or may-be even depends on a combination of particular IP addresses. This finding is consistent with the results of the CNN-LSTM network, which considered packet headers to be essential. The following figures 4.7 and 4.8 show heatmaps of the geographical affiliation of IP addresses according to the *whois* service.

The United States has the most frequent representation, both among all IP addresses in the dataset (63%) and among IP addresses focused by the model (55%). This is most likely due to the fact that approximately 35.9% of all IP addresses in the world are located in the United States [29]. Servers in the USA run a large number of ordinary (clean) services. Nevertheless, precisely, for this reason, it is effortless for an attacker to hide among them, and the effectiveness of state control decreases, of course. The fact that C&C malware servers are hidden under cloud hosting services proves that 48% of targeted addresses are related to servers belonging to Google, Microsoft and Amazon.

In addition, China, Russia, Canada and Japan were frequently represented countries, but orders of magnitude less than the USA. The differences in the frequency of country representation within the different malware classes were not significant.

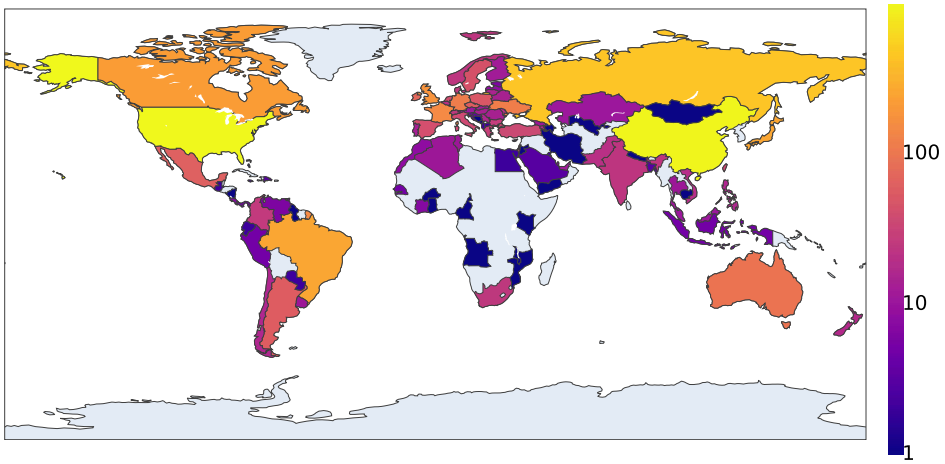


Figure 4.7: Malware dataset IPs heatmap.

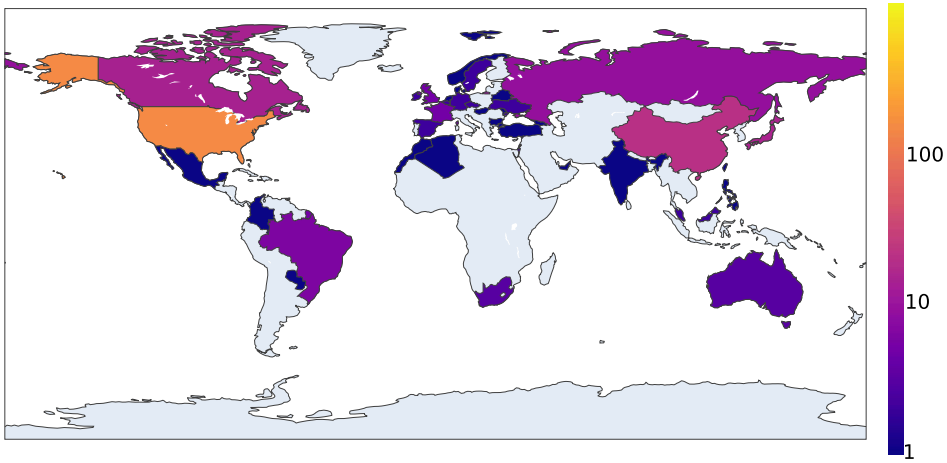


Figure 4.8: Malware dataset heatmap of IPs targeted by the S-HMill model.

Chapter 5

Conclusion

This thesis introduced one approach of malware network communication classification and compared it to two others. The visual representation approach proposed a non-hierarchical method to solve malware detection's hierarchical (time-dependent) problem. The method has less memory requirements for the hardware but at the cost of losing information while preprocessing. Classifying flows of packets as time sequences required such high demands on hardware that it was necessary to reduce the largeness of packets' flow significantly. Therefore, the time sequences method cannot be considered appropriate. The approach of hierarchical modelling neural networks that precisely reflects the input data structure yielded the best results. The significant advantages of this approach lie in that it simultaneously optimizes the classifier and the embedding. Building a computational graph enables to the solution precisely filter significant instances from the others. Given that HMill models targeted mainly IP addresses, it can be said that IP addresses are the main carrier of information for the classification of malware communication.

Extending HMill with the ability to process hierarchical structures as a time sequence would significantly increase the hardware requirements for neural network training. A much more interesting future direction for the research is applying HMill to a fragments of the Internet - modelling a general graph with loops (not a network with a star topology). The model could then not only be able to classify the infection on a separate computer but would also be able to detect suspicious structures within a part of the computer network.

Appendices

Appendix A

Background on tools

A.1 Binvis

Binvis is a toll for images representation of binary files. It samples the pcap (packet capture) files' content at regular intervals and translates each sampled byte into the output image's pixel. In the basic version, it compresses the content of packet capture files into the four classes. By ASCII value of a sample, the black colour corresponds to 0x00 (null), white to 0xFF (non-breaking spaces), the blue colour represents printable characters, and the extended ASCII bytes are assigned a red colour. The advanced version produces RGB images by clustering a 3D colour cube by Hilbert curve sets. There are three methods of arranging pixels (and sampling the original input file) in that tool. The first one is the *Zig-zag*, which lay the pixels row by row. This method has a low complexity (so it is quick); however, there is a problem with small scale features (the method tends to skip them). The second method, *Z-order*, partially avoids the previous problem. It is not the optimal solution, but the advantage (calculation speed) remains the same as in the first solution. The third offered way, the *Hilbert curve*, is as good as possible to get locality preservation at the cost of more complex calculations.

The following equation applies to the Hilbert curve that

$$s = 2^{p \cdot n}, \tag{A.1}$$

where s is a count of sampled points, p is an order of the curve and n is a dimension of the curve. In this research are used squared (two dimensional) images of size 256x256 pixels, so the needed order of curve (p) is 8.

A.2 k-NN classifier

A necessary condition for using k-NN is the existence of a norm function over a given data space. The class is assigned to the instance based on its distance to other neighbouring instances from the previous (training) dataset. In contrast with model-based learning algorithms, instead of model parameters, are all training data kept in memory. That is why it is crucial to have a balanced dataset of training samples for the classification's correct functionality.

The method with the k-NN classifier tested the application of the metrics below.

- The *Manhattan metric* is defined as: $\|\vec{x}\|_1 = \sum_{i=1}^{n} |x_i|$.
- The *standard euclidean metric* is defined as: $\|\vec{x}\|_2 = \sqrt{x_1^2 + ... + x_n^2} = \sqrt{\vec{x}^T \vec{x}}$.
- The uniform norm also called as *Chebyshev metric* is defined as: $\|\vec{x}\|_\infty = \lim_{p\to\infty} \|\vec{x}\|_p = \max\{|x_1|, ..., |x_n|\}$.
- The *maximum cross-correlation metric* is defined as: $\max((f \star g)[n]) = \max(\sum_{m=-\infty}^{\infty} f[m]g[m+n])$.

A.3 CNN, ResNet

A.3.1 Convolutional neural networks

Convolutional neural networks are primarily used in computer vision applications such as segmentation, captions recognition, classification, and image anomalies detection. It is a sequence of many layers which architecture depends on the purpose of usage. The main components of convolutional network architecture are bellow:

- **Convolutional layer** applies a kernel mask (convolutional matrix) on

an input matrix in a way (2D convolution):

$$g(x,y) = \text{bias}(x,y) + \sum_{dx=1}^{a}\sum_{dy=1}^{b} \mu(dx,dy)\cdot f(x+dx, y+dy), \qquad \text{(A.2)}$$

where $f(x,y)$ is an input feature map, $\mu(x,y)$ is a kernel mask of sizes $(a,\ b)$.

- **Activation function** is a threshold that adds a non-linearity into the network; it checks if the input is higher than a critical value.
- **Pooling layer** is an operation that slides a filter over the feature map channels and selects a valid value (by predefined rule - max / min / average) from the covered region. Pooling layer reduces the dimensions of the feature map; that implies reducing the count of parameters to learn, and the amount of computation. In contrast with convolution, pooling does not consider the exact position of the feature in the map. Thanks to, the model is more robust and resistant to minor input changes.

There are two main problems in learning during the backpropagation for convolutional neural networks without any skip-connections. There is a need to compute a partial derivative of the error function. Due to the chain rule, multiplying many (according to the count of layers) small numbers will become zero. In the case derivative of the loss function is a high number, multiplying many high numbers leads to infinity. The first problem is the vanishing of gradients, the second one the gradient explosion. If the gradient becomes too small and almost zero, it does not occur to update the early layers. It leads to a state when algorithm is only learning the last layers of the network - parameters (of the last few layers) are adapting to the training data.

A.3.2 Residual neural networks

The residual neural network is one of the architectures of convolutional networks. To solve the problems described above residual neural networks include skip-connections, which skip one or more layers in the network architecture; that provides alternative paths for a gradient in backpropagation, but there is a possible uncertainty with a convergence of learning. In ResNet are skip-connections performed by more possible approaches. The first one, identity mapping, is defined as

$$y = \Phi(\boldsymbol{x}, \boldsymbol{W_i}) + \boldsymbol{x}, \qquad \text{(A.3)}$$

where $\Phi(\boldsymbol{x}, \boldsymbol{W_i})$ represents layers to be learned. In case there are chained more convolutional bocks, there are inserted between them activation functions (in ResNet ReLU) as

$$\Phi(\boldsymbol{x}, \boldsymbol{W_i}) = \boldsymbol{W_{i_2}}\sigma(\boldsymbol{W_{i_1}}\boldsymbol{x}), \tag{A.4}$$

where σ is an activation function. The skip-connection itself is an element-wise addition of input without any additional changes, but it requires the same dimensions on the block's input and output. When dimensions increase, it can be solved by zeros padding. This principle adds neither an extra parameter nor computational complexity or time-consuming learning. The second approach performs a linear projection by the skip-connection.

$$y = \Phi(\boldsymbol{x}, \boldsymbol{W_i}) + \Theta(\boldsymbol{x}, \boldsymbol{W_s}), \tag{A.5}$$

where Θ is a convolution with parameters $\boldsymbol{W_s}$. This process can change dimensions but at the cost of increasing the number of parameters.

A.4 CAM, Grad-CAM, Score-CAM

The CAM method (Class Activation Mapping) modifies the original network by replacing fully-connected layers with a global-average-pooling layer and a single dense layer before softmax (Zhou et al. [30]). Product nodes of this modified network are output features maps. The class activation map is computed as a linear combination of each feature map (equation below express CAM heatmap for the class k).

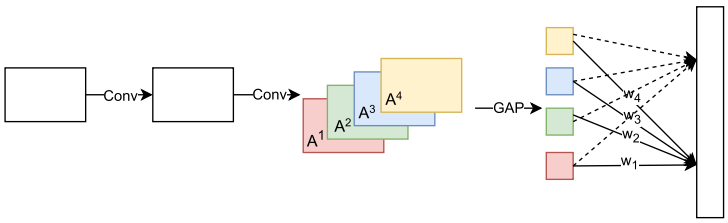


Figure A.1: Diagram of CAM-network.

$$\mathrm{CAM}^i = \sum_k w_i^k \mathbf{F}^k, \tag{A.6}$$

where w_i^k is a weight connecting the k-th feature map with the i-th class, and $\mathbf{F}^k$ is the global-average-pooled output defined by

$$\mathbf{F}^k = \frac{1}{\mathrm{Z}} \sum_m \sum_n \mathbf{A}_{ij}^k, \tag{A.7}$$

where $\mathbf{A}^k$ is the k-th feature map at location (m, n). The possible problem related to the CAM computation is that there might be a loss of spatial information.

The Grad-CAM method uses gradient information flowing into the last convolutional layer (between the last convolutional layer and softmax, there can be any network). In contrast with CAM, where features maps combination depends on weights of a single fully-connected layer, Grad-CAM computes weights (parameters of the linear combination of features maps) from gradient information (Selvaraju et al. [31]). The algorith has three steps (implementation from [32] by Isaac Castro).

Algorithm 1 Grad-CAM algorithm

Input: neural network model, input image or tensor), label.
Output: heatmap according to the label.

1. Let be y^i output for the i-th class. The gradient for the feature map $\mathbf{A}^k$ will be

$$\frac{\partial y^i}{\partial \mathbf{A}^k}. \tag{A.8}$$

2. The outputs of the global-average-polling will be

$$\alpha_k^i = \frac{1}{\mathrm{Z}} \sum_m \sum_n \frac{\partial y^i}{\partial \mathbf{A}^k}. \tag{A.9}$$

3. The heatmap $\mathbf{H}^i$ for the i-th class is defined as

$$\mathbf{H}^i = \mathrm{ReLU}(\sum_k \alpha_k^i \mathbf{A}^k). \tag{A.10}$$

The improvement of the previous method is Score-CAM. In this method, each activation map is used as a convolutional mask on the original input. The acquired instances forward-pass the network model and create score-based weights relative to classes. The Score-CAM result is a linear combination of activation maps and score-based weights (Wang et. al [33]). Implementation from [34].

A.5 SHAP

The SHapley Additive exPlanation (SHAP) method using Shapley values assumes fairly distributing the prediction contribution among each feature value of the instance (Strumbelj and Kononenko in [35]). The task is in how each feature affects the prediction of the model.

Let S be a feature subset $S \subseteq F$, where F is the set of all features. The $f_{S\cup\{i\}}$ is a model trained with the i-th feature and the model f_S is trained with the feature withheld. The x_s represents features values in the set S. The Shapley value weight over all possible differences for the i-th feature is defined by

$$\Phi_i = \sum_{S\subseteq F\setminus\{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!}[f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_s(x_s)]. \tag{A.11}$$

A.6 LSTM

The Long short-term memory (LSTM) is a type of recurrent neural network with the ability to train and remember long addictions of information sequences. LSTM neural network is built with many chained repeating cells, as is visible in the figure A.2 reprinted from [36].

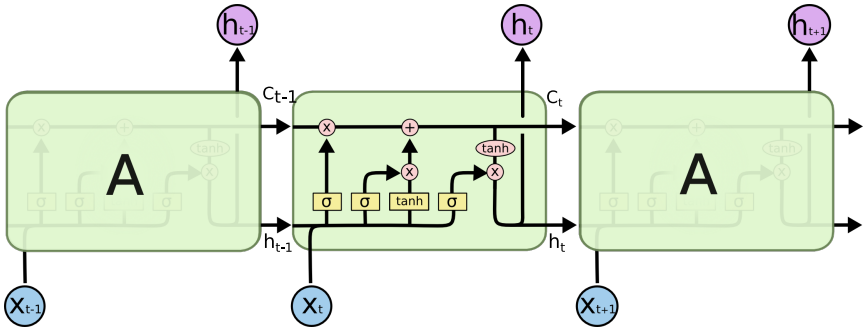


Figure A.2: LSTM chain.

The LSTM cell has three entry instances: previous internal cell state vector $\vec{C}_{t-1}$, previous output vector $\vec{h}_{t-1}$ and actual input vector $\vec{x}_t$. There are two resulting instances: current internal cell state vector $\vec{C}_t$ and current output vector $\vec{h}_t$.

The first sigmoid from the left in the cell diagram A.2 is called the *forget gate layer*, which decides how the previous internal state will be treated in the current cell. The second sigmoid from the left and the hyperbolic tangent block represent the *input gate layer* through which the internal state of the cell is updated. The right part of the cell with the last sigmoid acts as an output gate layer for calculating the output vector $\vec{h}_t$. The result depends on the modified cell internal state vector and on the previous inputs (Fathi M. Salem [37]).

Let be

$$\vec{z} = \boldsymbol{W}_i \begin{pmatrix} \vec{h}_{t-1} \\ \vec{x}_t \end{pmatrix} + \vec{b}_i, \tag{A.12}$$

where $\vec{b}_i$ is a bias and $\boldsymbol{W}_i$ is a weights matrix. Then outputs are defined as

$$\begin{aligned} \vec{c}_t &= \vec{c}_{t-1} \odot \sigma(\vec{z}) + \tanh(\vec{z}) \odot \sigma(\vec{z}), \\ \vec{h}_t &= \tanh(\vec{c}_t) \odot \sigma(\vec{z}), \end{aligned} \tag{A.13}$$

where $\odot$ is a *element-wise (Hadamard) product.*

Appendix B

Dataset

The dataset comes from the Canadian Institute for Cybersecurity from 2017 [38], [39]. There are captured malware samples infecting smartphones with the Android operating system. The dataset consists of many malware families divided into four classes; a total of 429 samples B.1.

Malware label	Total captured	Family
Adware	104	Dowgin, Ewind, Gooligan, Kemoge, koodous, Mobidash, Selfmite, Shuanet, Youmi
Ransomware	101	Charger, Jisut, Koler, LockerPin, Simplocker, Pletor, PornDroid, RansomBO, Svpeng, WannaLocker
Scareware	112	AndroidDefender, AndroidSpy.277, AV for Android, AVpass, FakeApp, FakeApp.AL, FakeAV, FakeJobOffer, FakeTaoBao, Penetho, VirusShield
SMSMalware	112	BeanBot, Biige, FakeInst, FakeMart, FakeNotify, Jifake, Mazarbox, Plankton, SMSsniffer, Zsone

Table B.1: *Dataset division.*

Arash Habibi Lashkari et al. performed capturing samples behaviour in three steps to overcome the inconspicuousness and strategic behaviour of more complex malware. The first one lasts 15 seconds after the smartphone's

first interaction with the malware (installing the application, downloading the file, opening the link). The second part takes 15 minutes to reboot the system. In the third part, the behaviour is captured after rebooting for 15 minutes.

The included classes:

- **Adware** is the software most often included in free ad-supported softwares. It manifests itself in the form of automatically pop-up windows and advertising accessories in the phone. In the case of a more complicated infection, it can establish a connection with the Command and Control servers, and in this way, more harmful malware can enter the infected phone.
- **Scareware** tries to gain the average user's trust quickly or scare him in shock about the threat to the device. It is typical in that it shows the user a fictitious warning and offers a solution in a paid service. Scareware bets its attack on human psychology, where panic behaviour leads to a short-lived mistake on which the malware's creator makes money.
- **Ransomware** blocks users or some files from users, requiring money to unlock them.
- **SMSMalware** uses a short message service containing a link to interact with the user. This type of malware is one of the initial infections when the infected device is allowed to communicate with the C&C server and downloading suspicious files to the device (for example one of the above types of malware).

Appendix C

Bibliography

[1] B. Collier, R. Clayton, A. Hutchings, and D. Thomas, "Cybercrime is (often) boring: maintaining the infrastructure of cybercrime economies." [Online]. Available: https://www.repository.cam.ac.uk/handle/1810/306682

[2] W. Stevens, *TCP/IP illustrated*. Reading, Mass: Addison-Wesley Pub. Co, 1994.

[3] T. Pevny and M. Dedic, "Nested multiple instance learning in modelling of http network traffic," 2020. [Online]. Available: https://arxiv.org/abs/2002.04059

[4] G. Bendiab, S. Shiaeles, A. Alruban, and N. Kolokotronis, "Iot malware network traffic classification using visual representation and deep learning," *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, Jun 2020. [Online]. Available: http://dx.doi.org/10.1109/NetSoft48620.2020.9165381

[5] R. Shire, S. Shiaeles, K. Bendiab, B. Ghita, and N. Kolokotronis, "Malware squid: A novel iot malware traffic analysis framework using convolutional neural network and binary visualisation," pp. 65–76, 2019.

[6] A. Cortesi, "binvis.io," *Visualizing Binaries With Space-Filling Curves: binvis.io*. [Online]. Available: https://binvis.io/#/

[7] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *Computer Communication Review*, vol. 36, pp. 23–26, 04 2006.

[8] Y.-D. Lin, C.-N. Lu, Y.-C. Lai, W.-H. Peng, and P.-C. Lin, "Application classification using packet size distribution and port association,"

Journal of Network and Computer Applications, vol. 32, no. 5, pp. 1023–1030, 2009, next Generation Content Networks. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804509000484

[9] G. Marín, P. Casas, and G. Capdehourat, "Deepmal – deep learning models for malware traffic detection and classification," 2020. [Online]. Available: https://arxiv.org/abs/2003.04079

[10] K. N. K. Thapa and N. Duraipandian, "Malicious traffic classification using long short-term memory (lstm) model," *Wireless Personal Communications*, 2021. [Online]. Available: https://doi.org/10.1007/s11277-021-08359-6

[11] M. Asim and M. Zakria, "Advanced knn: A mature machine learning series," 2020. [Online]. Available: https://arxiv.org/abs/2003.00415

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[13] P. Yakubovskiy, F. Camargo, and G. Anand, "Classification models zoo - keras (and tensorflow keras)," *GitHub repository*, 2019. [Online]. Available: https://github.com/qubvel/classification_models

[14] H. Almuallim and T. G. Dietterich, "Learning with many irrelevant features," in *Proc. of the 9th National Conf. on Artificial Intelligence*, vol. 2, 1991, pp. 547–552.

[15] T. Dietterich, R. Lathrop, and T. Lozano-Pérez, "Solving the multiple instance problem with axis-parallel rectangles," *Artificial Intelligence*, vol. 89, pp. 31–71, 03 2001.

[16] T. Pevny and P. Somol, "Using neural network formalism to solve multiple-instance problems," 2017. [Online]. Available: https://arxiv.org/abs/1609.07257

[17] A. Tibo, M. Jaeger, and P. Frasconi, "Learning and interpreting multi-multi-instance learning networks," 2020.

[18] M.-A. Carbonneau, V. Cheplygina, E. Granger, and G. Gagnon, "Multiple instance learning: A survey of problem characteristics and applications," *Pattern Recognition*, vol. 77, p. 329–353, May 2018. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2017.10.009

[19] E. Frank and X. Xu, "Applying propositional learning algorithms to multi-instance data," University of Waikato, Department of Computer Science, Working Paper, Jun. 2003. [Online]. Available: https://researchcommons.waikato.ac.nz/handle/10289/1006

[20] J. R. Foulds and E. Frank, "A review of multi-instance learning assumptions." *Knowledge Eng. Review*, vol. 25, no. 1, pp. 1–25, 2010. [Online]. Available: http://dblp.uni-trier.de/db/journals/ker/ker25.html#FouldsF10

[21] J. Amores, "Multiple instance classification: Review, taxonomy and comparative study," *Artificial Intelligence*, vol. 201, pp. 81–105, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370213000581

[22] L. Dong, "A Comparison of Multi-instance Learning Algorithms," Thesis, The University of Waikato, 2006. [Online]. Available: https://researchcommons.waikato.ac.nz/handle/10289/2453

[23] R. C. Bunescu and R. J. Mooney, "Multiple instance learning for sparse positive bags," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 105–112. [Online]. Available: https://doi.org/10.1145/1273496.1273510

[24] T. Gartner, P. Flach, A. Kowalczyk, and A. Smola, "Multi-instance kernels," *Proceedings of 19th International Conference on Machine Learning*, 11 2003.

[25] S. Mandlik, "Mapping the internet — modelling entity interactions in complex heterogeneous networks," *CVUT DSpace*, Jun 2020. [Online]. Available: https://dspace.cvut.cz/handle/10467/87851

[26] T. Pevny and M. Racinsky, "JsonGrinder.jl," https://github.com/CTUAvastLab/JsonGrinder.jl, 2019.

[27] T. Pevny and S. Mandlik, "Mill.jl framework: a flexible library for (hierarchical) multi-instance learning," https://github.com/CTUAvastLab/Mill.jl, 2018.

[28] cisco, "Joy tool," Nov. 2019. [Online]. Available: https://github.com/cisco/joy

[29] "Ip address by country 2021." [Online]. Available: https://worldpopulationreview.com/country-rankings/ip-address-by-country

[30] B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," *CoRR*, vol. abs/1512.04150, 2015. [Online]. Available: http://arxiv.org/abs/1512.04150

[31] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," *International Journal of Computer Vision*, vol. 128, no. 2, p. 336–359, Oct 2019. [Online]. Available: http://dx.doi.org/10.1007/s11263-019-01228-7

[32] I. Castro, "Gradcam-keras," *GitHub repository*, 2019. [Online]. Available: https://github.com/isaaccasm/GradCAM-keras

[33] H. Wang, M. Du, F. Yang, and Z. Zhang, "Score-cam: Improved visual explanations via score-weighted class activation mapping," *CoRR*, vol. abs/1910.01279, 2019. [Online]. Available: http://arxiv.org/abs/1910.01279

[34] tabayashi0117, "Score-cam," *GitHub repository*, 2020. [Online]. Available: https://github.com/tabayashi0117/Score-CAM

[35] E. Štrumbelj and I. Kononenko, "Explaining prediction models and individual predictions with feature contributions," *Knowledge and Information Systems*, vol. 41, no. 3, pp. 647–665, 2014. [Online]. Available: https://doi.org/10.1007/s10115-013-0679-x

[36] C. Olah, "Understanding lstm networks," *Understanding LSTM Networks – colah's blog*, Aug 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[37] F. M. Salem, "Slim lstms," 2018. [Online]. Available: https://arxiv.org/abs/1812.11391

[38] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in *2018 International Carnahan Conference on Security Technology (ICCST)*, 2018, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/8585560?denied

[39] "Android malware dataset (cic-andmal2017)." [Online]. Available: https://www.unb.ca/cic/datasets/andmal2017.html