ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

Deep Learning for the Analysis of Network Traffic Measurements

Tesis presentada a la Facultad de Ingeniería de la Universidad de la República por

Gonzalo Miguel Marín Freire

en cumplimiento parcial de los requerimientos para la obtención del título de Magíster en Ingeniería Eléctrica.

Montevideo
Jueves 28 de marzo de 2019

Deep Learning for the Analysis of Network Traffic Measurements, Gonzalo Miguel Marín Freire.

ISSN 1688-2806

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 70 páginas.
Compilada el domingo 16 de junio de 2019.
`http://iie.fing.edu.uy/`

I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted.

Alan Turing
Computing Machinery and Intelligence
Oct., 1950

Acknowledgments

Este trabajo no podría haber sido posible sin el apoyo de muchas personas. En primer lugar, quiero agradecer a Xime, quién está junto a mí acompañándome día a día a través de los distintos caminos que venimos transitando. Por su amor y compañerismo, sin ella esto hubiese sido mucho más difícil.

También quiero agradecer de forma especial a mis tutores. Al “Doc” –Germán– que fue quien me motivó desde el primer momento en continuar mis estudios hacia la Maestría en Ingeniería Eléctrica. A Pedro, que me recibió en Viena como si fuese parte de su familia. Por sus consejos, su motivación, su compañerismo, su amistad y por naturalmente convertirse en mis mentores. Estoy y estaré por siempre agradecido por ello. Este logro también es de ellos.

A Plan Ceibal, en particular a Gastón Arismendi, Enrique Lev, Ana Hernández y Fiorella Haim, quienes me apoyaron para que pueda construir este camino.

A todos los que de alguna manera u otra me apoyaran para que haya podido realizar la pasantía en Austria: a la SCAPA-IE, al Instituto de Ingeniería Eléctrica, en particular, Leo Steinfeld, María Misa y Fede La Rocca. A AIT, que me brindó un lugar donde poder trabajar gran parte de esta tesis.

A mis padres, a quienes debo todo. A Ana y Peti, que sin su inspiración de seguro no podría haber llegado hasta aquí. Por último y no menos importante, a mis amigos. Los de siempre y los nuevos que hice durante mi pasaje por Ceibal y AIT.

A Ximena, mis padres, Peti y Ana. Mis amigos–mi familia.

Summary

The application of machine learning models to the analysis of network traffic measurements has largely increased in recent years. In the networking domain, shallow models are usually applied, where a set of expert handcrafted features are needed to fix the data before training. There are two main problems associated with this approach: firstly, it requires expert domain knowledge to select the input features, and secondly, different sets of custom-made input features are generally needed according to the specific target (e.g., network security, anomaly detection, traffic classification). On the other hand, the power of machine learning models using deep architectures (i.e., deep learning) for networking has not been yet highly explored. These models have had huge success in various domains, notably in computer vision, natural language processing, machine translation, and more recently in gaming. The main goal of this work is to explore the power of deep learning models to enhance the analysis of network traffic measurements. To this end, the specific problem of detection and classification of network attacks is studied. As a major advantage with respect to the state-of-the-art in the field, the evaluation of different raw-traffic input representations, including packet and flow-level ones, is considered. Different deep learning architectures are explored, including convolutional neural networks and long short-term memory recurrent neural networks as core layers. In addition, three different datasets are crafted from publicly available network traffic captures and used for calibrating the considered input representations, as well as training and validating the proposed models. Different deep learning models are compared to a random forest model – commonly accepted as a highly accurate model for network traffic analysis, using the same raw input representations. In the malware detection task, a detection accuracy of 77.6% and 98.5% was achieved for packet and flow input representations respectively. For the malware classification task, an overall accuracy of 76.5% was achieved. In all evaluation tasks, the proposed deep learning models outperform the random forest ones. These initial results suggest that deep learning can be used to enhance malware detection without requiring expert domain knowledge to handcraft input features, opening the door to a broad set of potential applications for deep learning in networking.

Contents

Acknowledgments **iii**

Summary **vii**

1 Introduction **1**

2 Deep Learning **5**
- 2.1 Mathematical model of a single unit . . . 7
- 2.2 Fully-Connected . . . 8
- 2.3 Convolutional Neural Networks . . . 9
  - 2.3.1 Pooling . . . 9
- 2.4 Recurrent Neural Networks . . . 10
  - 2.4.1 Long Short-Term Memory . . . 10
- 2.5 Easing the training process . . . 11
  - 2.5.1 Batch Normalization . . . 11
  - 2.5.2 Dropout . . . 12

3 Related Work **13**
- 3.1 Shallow Machine Learning . . . 13
- 3.2 Deep Learning . . . 14

4 Input Representations and Datasets Construction **17**
- 4.1 Input Representations . . . 17
- 4.2 Building the Datasets . . . 18
  - 4.2.1 Malware Captures . . . 19
  - 4.2.2 Normal Captures . . . 21
  - 4.2.3 Design Criteria . . . 23

5 Deep Learning Architectures for Malware Detection **27**
- 5.1 Raw Packets deep learning architecture . . . 27
- 5.2 Raw Flows deep learning architecture . . . 29
- 5.3 Remarks on the number of parameters . . . 31
- 5.4 Hyperparameter optimization . . . 32
  - 5.4.1 Number of filters and filter size of CNNs layers . . . 33
  - 5.4.2 Learning rate . . . 33
  - 5.4.3 Optimizer . . . 34

5.4.4 Regularization 35
5.4.5 Checking the model capacity 35

6 Experiments and Results 37
6.1 Malware Detection: A First Approach Using Deep Learning 37
6.1.1 Deep learning vs. shallow models with raw inputs 38
6.1.2 Packet vs. flow representation performance 38
6.1.3 Domain knowledge vs. raw inputs 39
6.2 One Step Further: from Malware Detection to Malware Classification 40

7 Conclusions and Further Work 43

Bibliography 45

List of Tables 52

List of Figures 54

Chapter 1

Introduction

The popularity of Deep Learning has dramatically increased in the last few years, due to its outstanding performance in various domains, notably in image, audio, natural language processing, and more recently in gaming.

A major breakthrough in deep learning came in 2012 when a deep Convolutional Neural Network (CNN) won for the first time the largest contest in object recognition (ImageNet Large Scale Visual Recognition Challenge, ILSVRC). Krizhevsky et al. [1] presented a model based on a CNN trained using Graphics Processing Units (GPUs). The authors of the *AlexNet* model won the challenge by a wide margin, bringing down the state-of-the-art top-5 error rate from 26.1% to 15.3%. One of the most powerful characteristics of these models is their ability to learn feature representations from raw input or basic, non-processed data. For example, a CNN trained for image classification can learn to recognize edges and more complex structures along the sequence of neural layers, using as input only the image RGB pixel values (refer to [2] for a detailed example). One of the key reasons making deep learning models widely used today is the increasing computational power and the availability of larger datasets to perform the training, as well as several improvements in the techniques associated to train deep models [3]. Regarding this last point, it is notable the work of Hinton et al. [4] – providing a novel training algorithm for deep networks to achieve good solutions in real practical problems. A brief introduction to the basics of Deep Learning and its most-well used layers is presented in Chapter 2.

In the networking field, there have been many efforts over the past 20 years applying mainly conventional, shallow-like machine learning models to network measurements; there is a broad number of surveys [5–11] over-viewing the literature on the application of machine learning to diverse networking problems, including traffic prediction, traffic classification, traffic routing, congestion control, network resources management, network security, anomaly detection, QoS and QoE management. Nowadays, papers addressing more modern flavors of machine learning and their application in Networking are starting to be seen, e.g., with initial results on deep learning [12, 13], transfer learning [14], explainable AI for network security [15], deep reinforcement learning for network management [16–18], etc. A common trend that is found in the existing literature is that, for the most part of the papers doing machine learning for networking, there is a very strong dependence on the particular features used to represent the inputs, these being in general network packets aggregated at different granularities, such as flows, sessions, or even time-based aggregations. In this context, a feature vector of expert-handcrafted features is built to achieve the best results, being this the critical step on which the success of the model relies. More details about the related work on machine learning and deep learning applied to the analysis of network traffic measurements are presented in Chapter 3.

In this work, three specific limitations – among other challenges, are identified when addressing network traffic measurement problems using conventional, feature-dependent machine learning approaches. Firstly, there is a lack of standardized and well-accepted (labeled) datasets to train and test these models, and in particular of *raw*, full-packet capture datasets, which could be eventually used to build different input representations for the particular tasks. Different from other AI-related domains, where well established, publicly available datasets are available for testing, evaluation and benchmarking purposes (e.g., ImageNet in image processing), it is very difficult to find appropriate public datasets to assess machine learning for networking. While one of the main reasons for this lack clearly arises from the data's sensitive nature – including end-user privacy, other limitations come from the efforts required to build proper and representative datasets in networking. Given the scale of Internet-like networks, the massive volumes of data, and the multiplicity of operational conditions, building such a representative dataset is a daunting task. Secondly, there is a systematic lack of a consensual set of input features to tackle specific tasks, such as network security, anomaly detection or traffic classification; each paper in the literature defines its own set of input features for the considered application, hindering generalization and benchmarking of different approaches. Last but not least, networking data is very dynamic and consists of constantly occurring concept drifts – changes in the underlying statistical properties, causing static handcrafted features to fail over time.

To take steps in improving these limitations, this work explores the end-to-end application of deep learning models to complement conventional approaches for network measurements analysis, using different representations of the input data. As a representative networking application, the specific problem of malware traffic detection and classification through deep learning models is addressed, using raw, bytestream-based data as input. Two different raw input representations are considered: *Raw Packets* and *Raw Flows*. Also, three different datasets are crafted from publicly available network traffic captures, and used to construct the corresponding input representations and to train and validate the malware detection and classification models. The details of the input representations and the construction of the datasets are provided in Chapter 4, and the deep learning architectures used to tackle each specific problem are introduced in Chapter 5. In terms of malware detection performance, detection accuracies of 77.6% and 98.5% were achieved using the *Raw Packets* and *Raw Flows* input representations, respectively. As for the malware classification task, an overall accuracy of 76.5%

was achieved. In all scenarios, the proposed deep learning models outperform a powerful random forest model. The random forest model was chosen based on the generally outstanding detection performance shown by the model in the literature, using domain expert input features. The details of the different experiments and the corresponding discussion on the obtained results are presented in Chapter 6. Initial results of this work suggest that deep learning can be used to enhance malware detection without requiring expert domain knowledge to handcraft input features, opening the door to a broad set of potential applications for deep learning in networking. Concluding remarks and future work are presented in Chapter 7.

Chapter 2

Deep Learning

Goodfellow et al. [3] define *deep learning* [19,20] as *an approach to Artificial Intelligence (AI)*. Specifically, it is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler ones, and more abstract representations computed in terms of less abstract ones. Fig. 2.1 illustrates the relationship between different AI disciplines, and shows how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI.

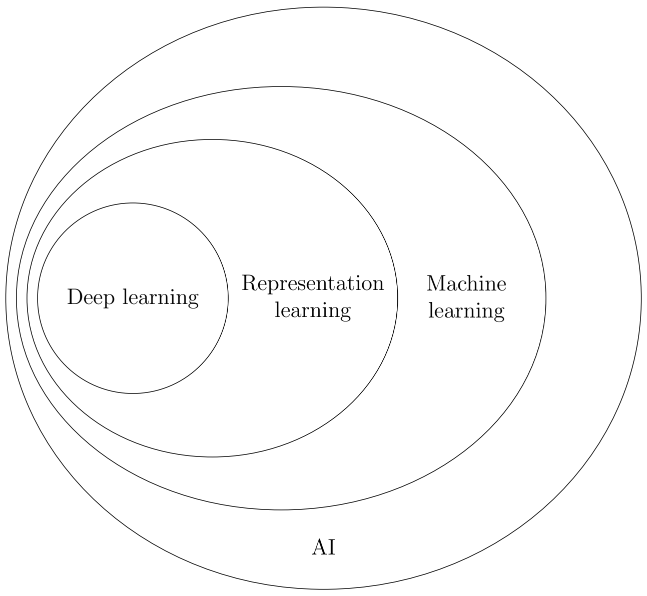


Figure 2.1: A Venn diagram showing the relationship between AI, machine learning, representation learning and deep learning [3].

With the major advances in computational processing capacity, notably through the massive production and adoption of Graphics Processing Units

(GPUs) and more recent Tensor Processing Units (TPUs)[1], and the surge of data availability over the past decade, deep learning has made it to the top of the AI agenda. Deep learning has dramatically improved the state-of-the-art in multiple domains, including speech recognition, visual object recognition, object detection and many others such as genomics. Different from conventional machine learning, deep learning is extremely data-driven and somehow agnostic to the specific type of data, requiring very big amounts of it to learn, in a completely black-box fashion.

One of the most powerful characteristics of deep learning models is their ability to extract high-level, abstract features from raw or basic, non-processed data. This is done by inferring representations that are expressed in terms of other, simpler representations. The key behind deep learning is the so-called representation learning paradigm [21], which offers a set of methods allowing a machine learning algorithm to automatically discover the best data representations or features from raw data inputs. Deep learning methods are basically representation learning methods with multiple levels of representation or abstraction, obtained by composing simple but non-linear consecutive transformation steps or layers, each of them providing a more abstract representation of the data.

Worth mentioning are two major milestones enabling a successful and massive application of deep learning to data-driven problems, led by the dubbed "Godfather of deep learning", Geoffrey Hinton: in 2006, Hinton et al. [4] introduced a novel and effective way to train very deep neural networks by pre-training one hidden layer at a time, using the unsupervised learning procedure for restricted Boltzmann machines [22]. Later on, in 2012, one of his students, Alex Krizhevsky, designed a deep convolutional network called the *AlexNet*, which strongly helped to revolutionize the field of computer vision, by almost halving the error rate for object recognition at the 2012 ImageNet challenge [1]. This precipitated the rapid adoption and popularity of deep learning in computer vision problems, naturally extending later on to other domains. Also worth mentioning are the recent developments in Deep Reinforcement Learning [23], playing a critical role in today's success of deep learning, notably through the popular implementations of AlphaGo, AlphaZero, and more recently AlphaStar, all of them by research teams at Google DeepMind[2].

When introducing these concepts, one reference that comes to mind is the usage of artificial neural networks. Nowadays, the term deep learning appeals to a more general principle of *learning multiple levels of composition*, which can be applied in machine learning frameworks that are not necessarily neurally inspired. Nevertheless, many of the major breakthroughs of deep learning still fall within the scope of different kinds of deep neural networks-based models.

In the rest of this Chapter, a brief introduction to the different parts involved in commonly used deep learning models are introduced, bringing some theoretical

[1]Google. Cloud TPUs - ML accelerators for TensorFlow, Google Cloud Platform, `https://cloud.google.com/tpu/`

[2]DeepMind: the world leader in artificial intelligence research and its application for positive impact, `https://deepmind.com/`

background to the subject. This Chapter does not pretend to be an exhaustive guide to the field, but to introduce some basic concepts that can help the reader to follow up the ideas over the rest of the work. The following sections are heavily based on [3], [24] and [25]. The interested reader is encouraged to go through them in order to delve into more detail.

2.1 Mathematical model of a single unit

Artificial Neural Networks are a very powerful and flexible family of parametric, differentiable functions. Fig. 2.2 shows the mathematical model of a neural network unit. It has been primarily inspired by the goal of modeling how a neuron works, but this model was later found to be very coarse [26]. The output of the unit is the result of applying an activation function (or *non-linearity*) to a weighted sum of the inputs plus a vector called *bias*:

$$\textit{output} = f\left(\sum_i w_i x_i + b\right) \tag{2.1}$$

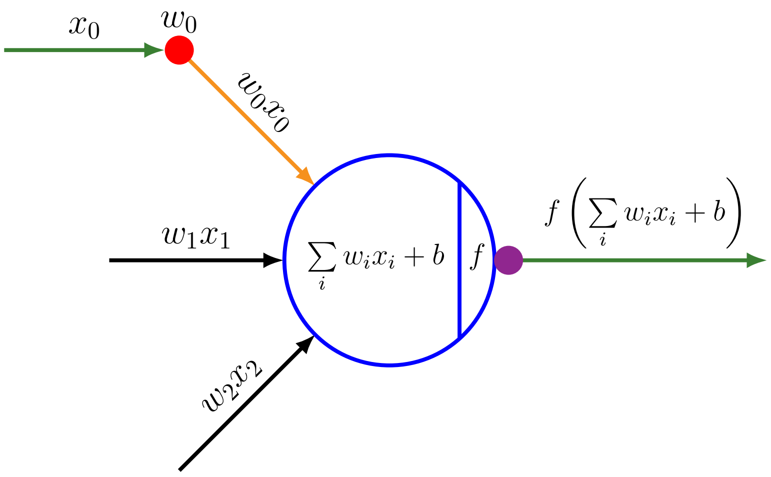


Figure 2.2: Mathematical model of a single neural network unit.

Every activation function takes a single number and performs a certain fixed mathematical operation on it. It is in charge of deciding if the neuron has to *fire* or not (using the brain analogy). Fig. 2.3 shows some commonly used activation functions for neural networks. The ReLU activation function is one of the most used in practice. It computes the formula $f(x) = \max(0, x)$, i.e., returns zero when $x < 0$ and is linear with slope 1 when $x > 0$. Compared to other non-linearities that involve expensive operations, the ReLU can be implemented by simply thresholding a matrix of activations at zero.

The learning process of neural networks involves finding the best set of weights w_i (eq. 2.1) that minimizes a certain loss function. The largest difference between

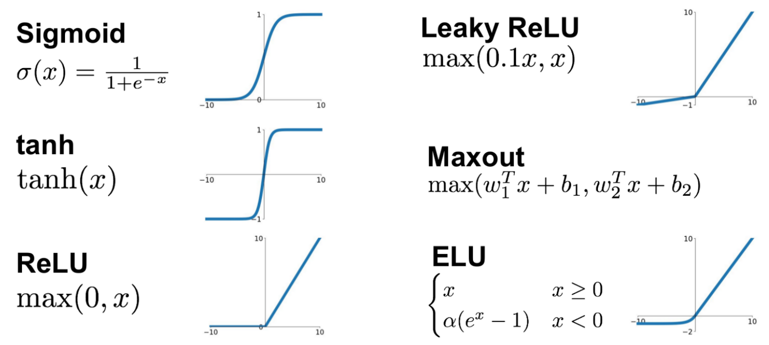


Figure 2.3: Commonly used activation functions for neural networks [24].

linear models and neural networks is that the non-linearity of a neural network causes most loss functions to become non-convex. For this reason, neural networks are usually trained by using iterative, gradient-based optimizers that merely drive the cost function to a very low value, being Stochastic Gradient Descent (SGD) and its variations the most used in practice. Unlike convex optimization, where convergence is guaranteed starting from any initial parameters, non-convex optimization has no such guarantee and its very sensitive to the parameter's initialization. For example, for fully-connected neural networks, it is important to initialize all weights to small random values, while the biases may be initialized to zero or to small positive values [3].

2.2 Fully-Connected

For regular neural networks, the most common layer type is the *Fully-Connected layer* (FC) in which units between two adjacent layers are fully pairwise connected, but units within a single layer share no connections. The output of each unit is connected to the input of every unit in the following layer, with no recursive connections present. Fig. 2.4 shows an example of a neural network topology using a stack of fully-connected layers.

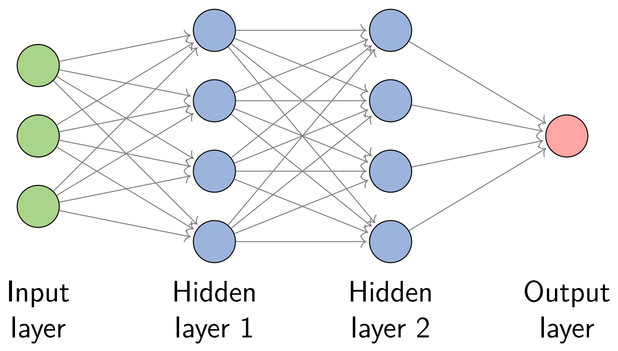


Figure 2.4: A 3-layer neural network with 3 inputs, 2 hidden layers of 4 units each and one output layer.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized kind of neural networks that use the mathematical operation called *convolution* in place of general matrix multiplication in at least one of their layers. CNNs are specially useful for processing data that has a known, grid-like topology. Examples include time-series data and image data, among others. Convolution leverages three important ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivariant representations.

A typical layer of a convolutional network consists of three stages (Fig. 2.5). In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations. In the second stage, each linear activation is run through a nonlinear activation function. In the third stage, a *pooling* function is used to modify the output of the layer further.

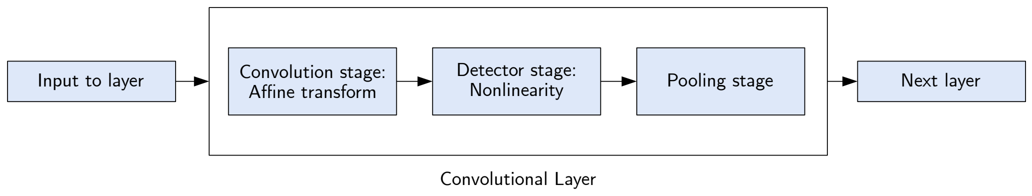

Input to layer
Convolution stage: Affine transform
Detector stage: Nonlinearity
Pooling stage
Next layer
Convolutional Layer

Figure 2.5: The components of a typical convolutional neural network layer.

2.3.1 Pooling

The main goal of the pooling function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. Overfitting occurs when the model starts to memorize patterns in the training samples and in turn loses the ability to generalize (the power of prediction over unseen data).

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, the *max pooling* operation reports the maximum output within a rectangular neighborhood. Fig. 2.6 shows an example of a max pooling operation applied to 1D data.

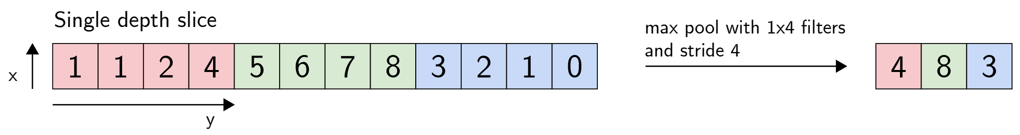


Figure 2.6: Max pooling operation of size 4 applied to one-dimensional data.

2.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a family of neural networks for processing sequential data. The term "recurrent" is associated with the fact that the output vector of a RNN is influenced not only by the actual input, but also on the entire history of inputs that have fed-in in the past, allowing information to persist.

Fig. 2.7 shows a diagram of a standard –*vanilla*– RNN block. The block A looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next. Unrolling this unit shows that the RNN consists of multiple copies of the same network, each passing a message to a successor. This chain-like nature reveals that RNNs are intimately related to sequences and lists. In the last few years, RNNs have been applied to a variety of problems, including: speech recognition, language modeling, translation and image captioning [25].

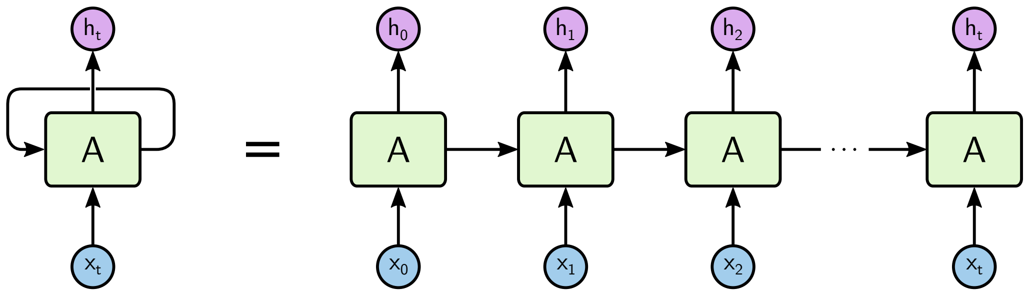
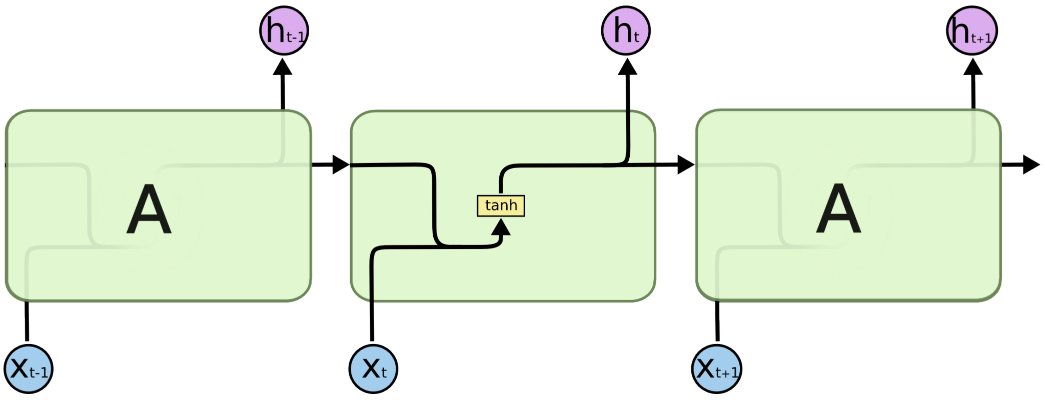


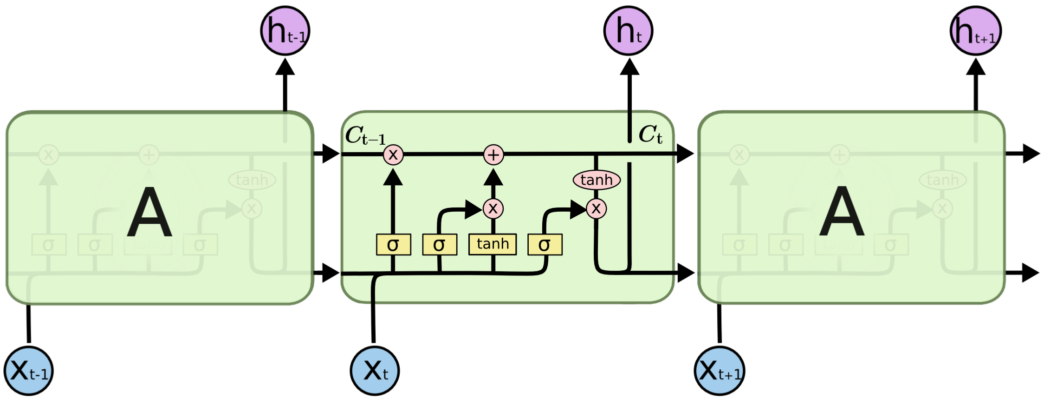
Figure 2.7: An unrolled recurrent neural network.

2.4.1 Long Short-Term Memory

Standard RNNs can have some problems when leading with long-term dependencies. When the existing gap between the relevant information to be used and the point where it is needed is small, RNNs can learn to use the past information. However, as this gap grows, RNNs become unable to learn to connect the information. The main reason behind this behaviour is the vanishing and exploding gradients caused by repeated matrix multiplications in the learning process of the RNNs. Long-Short Term Memory (LSTM) networks solve this problem by replacing the simple update rule of the standard RNN (usually, applying the tanh function) with a gating mechanism. The repeating module in an LSTM contains four interacting layers. The key to LSTMs is the *cell state*. The cell state works similarly to a conveyor belt: it runs straight down the entire chain, with only some minor linear interactions. The LSTM have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. LSTMs has three of these gates, to protect and control the cell state. Fig. 2.8 pictures this process, where the differences between the simple update rule of the standard RNN and the gating mechanism used in the update rule of LSTMs can be seen.



(a) Update rule in a standard *–vanilla–* RNN.



(b) Gating mechanism in the update rule of a LSTM network.

Figure 2.8: Differences between the internal modules of standard RNNs and LSTMs.

2.5 Easing the training process

The learning process of deep learning models can be tricky. However, there are some state-of-the-art techniques that can help to ease the training process. Specifically, focus will be made on batch normalization and dropout.

2.5.1 Batch Normalization

In the training process of deep neural networks, the distribution of each layer's inputs changes as the parameters of the previous layers change. This behaviour slows down the training by requiring lower learning rates and careful parameter initialization. Batch Normalization [27] addresses this problem. The idea is as follows: while training, the layer inputs are normalized for each mini-batch and this is included as a part of the model architecture. As a result, much higher learning rates can be used and the model becomes less sensitive to initialization.

2.5.2 Dropout

Dropout [28] is a technique for regularizing neural networks (i.e., controlling overfitting). In a nutshell, the key idea is to randomly drop units (along with their connections) from the neural network during training. Under this scenario, each unit learns to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of contexts in which it operates. Overfitting can be reduced by using dropout because it prevents complex co-adaptations on the training data. On each presentation of each training case, each hidden unit is randomly omitted from the network with a probability p, so a hidden unit cannot rely on other hidden units being present.

Another way to view the dropout procedure is as a *very efficient way of performing model averaging* with neural networks. A good way to reduce the error on the test set is to average the predictions produced by a very large number of different networks. This is called as bootstrap aggregating or *bagging.* The standard way to do this is to train many separate networks and then have all of the models vote on the output for test examples, but this is computationally expensive during both training and testing. Dropout can be thought of as a method of making *bagging* practical for ensembles of many large neural networks in a reasonable time. Fig. 2.9, taken from the original paper, shows an ensemble of different sub-networks trained as a consequence of using Dropout.

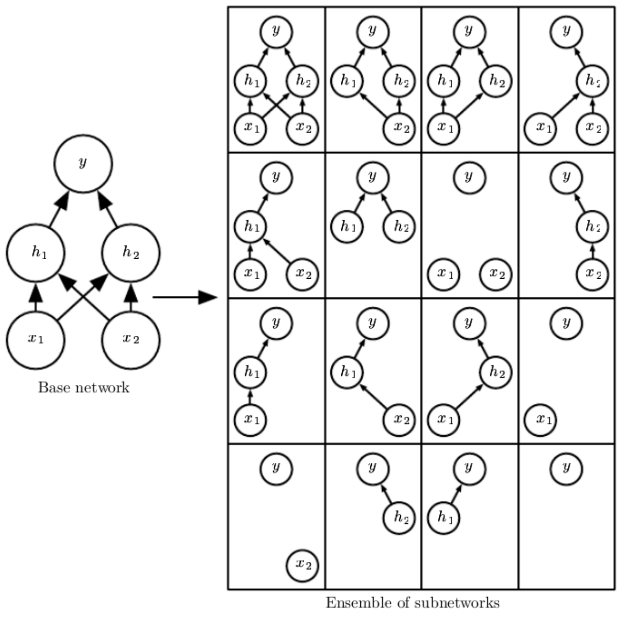


Figure 2.9: Ensemble of sub-networks trained as a consequence of using Dropout. Figure taken from the original paper [28].

Chapter 3

Related Work

In this Chapter, a selection of papers that represent the state-of-the-art regarding machine learning –and particularly deep learning– applied to the analysis of network traffic measurements is presented.

3.1 Shallow Machine Learning

The application of machine learning models to general network measurement problems is largely extended in the literature. Traffic prediction and classification [11] are two of the earliest machine learning applications in the networking field. In [5], authors present a recent and very comprehensive survey on machine learning for networking, discussing different applications and associated challenges. Similarly, [29] describe basic machine learning concepts, and discuss their application in communication networks. Other diverse surveys and papers overview the vast literature on the application of machine learning to multiple networking problems, including traffic prediction, traffic classification [11], routing and congestion control, network resources management, network security [8], anomaly detection [30–32], QoS and QoE management [33,34], and more [6,7,9,10,35].

Besides direct application of machine learning, other papers elaborate on more holistic approaches to introduce learning into communication networks, ranging from the seminal work of Clark et al. [36] proposing the "Knowledge Plane" for the Internet – a pervasive, distributed system within the network that would provide the required levels of abstraction and end-to-end visibility to realize an AI-boosted Internet, to more recent proposals taking advantage of current network flexibility to enable machine-learning capable networking [37,38].

Some of the most popular machine learning models used in the networking domain include: rule-based models, such as decision trees and random forest, feed-forward neural networks, SVM, k-Nearest Neighbours (k-NN), K-Means and DBSCAN, but the list is as vast as the associated literature. In most cases, the core of the learning problem falls within the design and selection of a set of features carefully built by domain expert knowledge. In simple words, the challenge has been so far on the feature engineering process.

3.2 Deep Learning

At the moment of writing this thesis, most of the articles presented in the literature that mention the use of deep learning for the analysis of network traffic measurements are mainly based on using the models to perform feature selection/extraction and classification from a set of selected, carefully-built expert features. Next, the most relevant papers are discussed.

In the cybersecurity domain, there are several works that tackle problems such as anomaly-based network intrusion detection [12, 13, 39, 40] and impersonation attack detection [41]. In addition, network traffic classification is the subject of [42–45], while [46] particularly explores network protocol recognition. More recently, the Internet of Things (IoT) has gained importance in the network traffic measurement analysis field [47].

In [39], D. Kwon et al. reviewed seven papers about deep learning models employed for anomaly-based network intrusion detection, all of which mainly discuss methodologies in terms of data dimensionality reduction and classification. All the papers use either `KDD-Cup 1999` dataset [48] or its successor `NSL-KDD` [49], which solves some of the inherent problems of the first (e.g., huge amount of redundant records). These datasets are widely used in the literature, they consist of 41 pre-calculated features and several instances. However, both datasets are heavily criticized for not being representative enough of existing real networks [50]. U. Fiore et al. [40] also tackle a similar problem, but with a semi-supervised approach using a Discriminative Restricted Boltzmann Machine (DRBM) [51], i.e., a RBM [22] that is trained more specifically to be a good classification model. DRBMs couple the ability to express much of the variability of the data –given by the generative model– with the good classification accuracy derived from the discriminative classifier. The dataset used for training the DRBM was the `KDD-Cup 1999`, where the authors manually selected 28 features out of 41. A different approach is carried out by B. J. Radford et al. in [13]. Instead of using network flows/sessions data, network traffic logs for cybersecurity monitoring are used. This is quite common when the goal is to detect attacks that are generated not only from inside the network, but also from commands directly inserted into a critical unit. The authors used a LSTM RNN to learn ordered sequences of *normal* network traffic and then evaluate the ability of this model to detect malicious activity on the same network. They do this in an unsupervised way, without the assistance of labeled training data. The dataset used for this research is a public dataset for intrusion detection from the University of New Brunswick's Canadian Institute for Cybersecurity (CIC) and the Information Security Centre of Excellence (ISCX), referred as `ISCXIDS2012` [52]. Also in the cybersecurity field, M. E. Aminanto et al. [41] focused on optimizing the impersonation attack detection over Wi-Fi networks. The dataset used in this case is `AWID` (Aegean Wi-Fi Intrusion Dataset, [53]), which consists of 154 pre-calculated features. The authors used a two-layer FC artificial neural network for feature selection and then a Stacked Auto Encoder (SAE) as a classifier, where the feature extraction stage is deployed in the SAE implicitly. SAE is a neural network with multiple layers of auto encoders, i.e., an unsupervised learning model that is trained to learn a representation of the inputs, so as to output a function $\hat{x}$ that is similar to input x. Different kinds of tests are performed using either all the classes or just two (whether the instance is an attack or normal traffic), using both balanced and unbalanced splits for performing the training.

Concerning network traffic classification, Z. Wang et al. presented in [46] a deep neural network for feature learning using FC networks and SAE. Their main goal is to perform network protocol recognition and anomalous protocol detection using a dataset made up of TCP flows collected from an internal network. Interestingly, they show that the distribution of byte-features discovered over the payload data is consistent to traditional methods used for feature engineering. Two different approaches are followed by W. Wang et al. in [42] and [43], using both 2D-CNN and 1D-CNN models to perform network traffic classification. Working with raw network traffic captures, they transform network flows and sessions (bi-direction flows) into images to fit as an input for the CNN models, using either the information of all the layers, or only from the application layer. In [42] the authors also presented a new dataset, named `USTC-TFC2016`, which they used to benchmark the different models. Moreover, in reference [44], M. Lotfollahi et al. presented a framework for encrypted traffic classification using CNN and SAE at the packet level. The dataset used for this work is the VPN-nonVPN dataset from CIC and ISCX, referred as `ISCXVPN2016` [54], which consists of both encrypted and non-encrypted traffic captures of different applications in `pcap` format. Before training, some pre-processing of the `pcap` files is performed in order to prepare the network traffic data so that it can be fed into the models properly. Furthermore, M. Lopez-Martin et al. presented in [45] a flow statistics-based supervised method composed by different deep learning models in order to classify the service being used by an IP network flow. For each network flow –only TCP and UDP are considered– a time-series of feature vectors is built, where each element contains the features of a packet in the flow. For each packet, the authors extract six features: source port, destination port, the number of bytes in packet payload, TCP window size, inter-arrival time and direction of the packet. The authors considered the information contained in the header packets but not the payloads. All captures were held over the `RedIRIS`, the Spanish academic and research network and the service labeling of the dataset was made using the well-known `nDPI-ntop` tool [55]. Three different models were trained: a pure CNN model, a pure LSTM model and a combination of both, being the latter the one that achieved the best performance. In the case of the CNN model, the authors considered the matrix formed by the time-series of feature vectors as an input image. As for the LSTM, the model is trained with a matrix of values with two dimensions: the temporal one and the vector of features. Finally, in the combined model, the final tensor of several CNNs chained is reshaped into a matrix that can act as the input to the LSTM network.

Lastly, a survey of deep learning models applied to IoT is presented by M. Mohammadi et al. in [47]. Even though this survey does not specifically address network traffic measurement and analysis problems, it is a wide summary of the different architectures and methods generally used in deep learning, specially those in which fast/real-time data streams processing is needed.

Most of these papers use deep learning models after some pre-processing of the data, or after building some set of handcrafted features to be used as meaningful input for the deep learning models. References [42–44] and [46] are the only ones that mention the usage of raw traffic, approaching the specific problem of network traffic classification. This thesis uses some of these concepts, but explores the power of different input representations and different architectures for the deep learning models (including hyperparameter optimization). In addition, three different datasets are created from raw network traffic captures in order to train the different models and test their performance.

Chapter 4

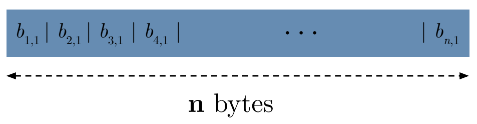
Input Representations and Datasets Construction

There is no silver bullet to select the input representations in a networking problem. Several valid choices can be made, such as: full-packets, packet's headers, packet's payloads, uni-directional flows and bi-directional flows, among others. It is also possible to select only the information present in some specific layers, and then choose one of the representations mentioned before. Similarly, the lack of a consensual publicly available, labeled raw traffic, full-packet capture dataset to train the models is a real issue when facing networking problems with machine learning. There is no MNIST [56], ImageNet [57] or CIFAR [58] –mainstream datasets used for image recognition– for networking. While one of the main reasons for this lack clearly arises from the data's sensitive nature – including end-user privacy – other limitations come from the efforts required to build proper and representative datasets in networking. For example, many datasets are forced to not include the payloads or just anonymize them. On the other hand, datasets that do include the payloads are usually built under controlled environments.

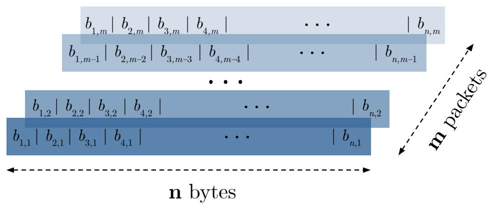
In this Chapter, two different input representations chosen for feeding the deep learning models are presented: *Raw Packets* and *Raw Flows*. Also, a specific dataset to fit each one of the input representations is created, using raw network traffic captures. Furthermore, the process involved for building the datasets, that includes a statistical analysis of the packets and flows of each class, is also described.

4.1 Input Representations

The input representation of the data, as well as the network architecture, are both key facts when building a deep learning model. To evaluate the feature representation power of the model from non-processed data, two types of raw representations are considered: packets and flows. In both cases, decimal normalized representation of every byte of every packet as a different feature is taken into account. In the packet approach, each packet is considered as a different instance, while in the flow approach a group of packets –that make up the flow– is considered as an input for the network, i.e., a tensor made of packets that represent a flow as an input instance is built. Both representations are depicted in Fig. 4.1. For the *Raw Packets* representation, it is necessary to set the number of bytes from the packet to consider (n), while in the *Raw Flows* representation it is also needed to set the number of packets per flow to consider (m). This is because, naturally, different packets and flows can have different sizes. In the following Section, the process involved for choosing the value for both n and m parameters is described.


$b_{1,1}$ | $b_{2,1}$ | $b_{3,1}$ | $b_{4,1}$ | ... | $b_{n,1}$
n bytes

(a) Packet representation for the input data. The shape of the input data is (n), i.e., the number of steps –bytes–.


$b_{1,m}$ | $b_{2,m}$ | $b_{3,m}$ | $b_{4,m}$ | ... | $b_{n,m}$
$b_{1,m-1}$ | $b_{2,m-2}$ | $b_{3,m-3}$ | $b_{4,m-4}$ | ... | $b_{n,m-1}$
...
$b_{1,2}$ | $b_{2,2}$ | $b_{3,2}$ | $b_{4,2}$ | ... | $b_{n,2}$
$b_{1,1}$ | $b_{2,1}$ | $b_{3,1}$ | $b_{4,1}$ | ... | $b_{n,1}$
m packets
n bytes

(b) Flow representation for the input data. A tensor of size (m, n) where m stands for the number of channels –packets– and n for the number of steps –bytes– represents the input data.

Figure 4.1: Different input representations for the deep learning models.

4.2 Building the Datasets

For building the datasets, *malware* and *normal* captures performed by the *Stratosphere IPS Project* [59] of the CTU University of Prague in Czech Republic are considered [60]. Since both malware and normal captures are gathered under controlled conditions, there are some biases in the IP and transport protocol headers that are not representative of *in the wild* traffic. This is the case, for example, of fixed values for IP addresses and ports and even some of the transport protocol flags. For this reason, the *payload* of every packet is considered as the key information to analyze and to build the datasets. Afterwards, a fixed threshold for the parameter n is set in order to trim each incoming packet to the first n bytes of payload. All packets with size larger than n bytes are trimmed, and packets with smaller size are zero-padded at the end. For the number of packets per flow, a number m is fixed, taking the first m packets of the flow, discarding the rest.

Note that the problem shouldn't be dependent on the chosen values, but are needed in order to build the datasets that will be further used to train the different deep learning models. To select the parameters n and m, focus will be made on a statistical analysis performed over the network captures. One important consideration to have in mind is that *small* values for both parameters will be preferred, specifically for the number of packets per flow. This is important because real-work applications of network security require early-classification, i.e., to be able to detect and mitigate the malicious flows early in time (e.g. after 1 to 4 packets have been captured). This is known as *early traffic classification* or *traffic classification on the fly* [61]. To select the parameters, the following three distributions will be analyzed:

1. Distribution of the payload size in the packets.
2. Distribution of the number of packets in the flows.
3. Distribution of the mean payload size of the packets in the flows.

To perform the flow analysis, the `pkt2flow` tool [62] was used. This tool divides each capture file in f different files, where f is the amount of flows detected. It can split up the following types of flows:

1. TCP flows with the SYN flag (three-way handshake completed).
2. TCP flows without the SYN flag, or "invalid" TCP flows.
3. UDP flows.
4. Other kinds of flows (non TCP/UDP), usually ICMP.

The concept of flow is defined as *every group of packets that share the same 5-tuple*: transport layer protocol, source IP, destination IP, source port and destination port. The timeout for the TCP flows is fixed to thirty minutes –default value in Cisco NetFlow– but this value can be modified inside the script code. For this analysis, the default timeout value was used and only TCP and UDP flows were considered.

4.2.1 Malware Captures

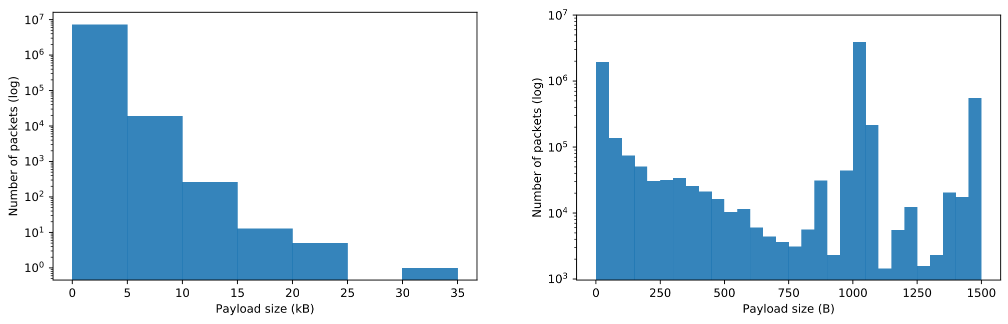
Before digging into the details let's start defining the concepts of *malware* and *botnet*. *Malware* is an abbreviated form of "malicious software". This is software that is specifically designed to gain access to or damage a digital device, usually without the knowledge of the owner. On the other hand, a *botnet* is nothing more than a string of connected computers coordinated together to perform a task[1]. Malicious botnets are connected devices designed to perform a malicious activity.

The malware captures are taken from a subset of captures referred by their authors as `CTU-13`. It consists of thirteen captures (named *scenarios*) of different botnet samples. On each scenario, a specific malware is executed, using several protocols (e.g., IRC, HTTP, P2P) and performing different actions (e.g., DDoS, port scan, click fraud, spam).

[1]Symantec: global leader in next-generation cybersecurity, https://symantec.com/

Distribution of the payload size in the packets

The distribution of the payload size in the packets for the malware captures is shown in Fig. 4.2, where a log scale was used to enhance the view. As can be seen, there is a group of packets that contain a payload size larger than the standard Ethernet MTU size ($\sim$ 1500 bytes). This is a common behaviour when the traffic is captured in the same equipment where it was generated, using the *Large Send Offload* (LSO) option. Under this scenario, the packet fragmentation is performed by the NIC card reducing CPU overhead. These packets conform the 9.3% of the total. Limiting the histograms to those packets with payload size less and equal than 1500 bytes (Fig. 4.2b), it can be seen that most of the packets have a payload size between 1000 and 1100 bytes, being the median equal to 1024 bytes. It is also noted that the 28.3% of the packets have a payload size less than 100 bytes.



(a) Whole payload size range. (b) Payload size up to 1500 bytes.

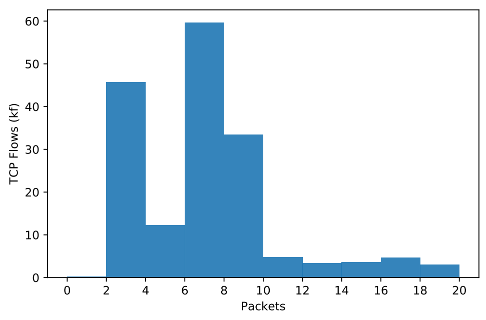
Figure 4.2: Distribution of the payload size in the packets for malware captures (log scale).

TCP flows

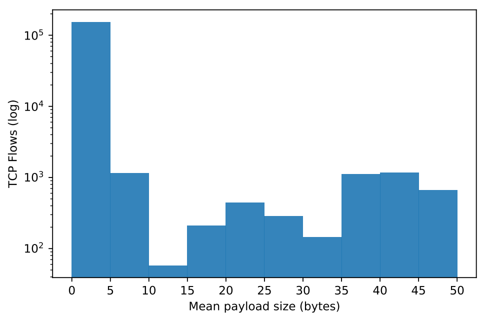
The distribution of the number of packets in TCP flows is shown in Fig. 4.3a. The 90% of the flows are composed of between 2 and 20 packets. In Fig. 4.3b the distribution of the mean payload size of the packets per TCP flow ($\overline{p_f}$) is shown. The value $\overline{p_f}$ for TCP flows in the malware captures varies between 0 and 1355 bytes and most of them gather in the range 0–50 bytes (84% of the flows). It can also be seen that the 60% of the TCP flows are composed of packets with no payload. This means that most of the TCP attacks within these captures are built by packets that, at first glance, contain no information (e.g., DDoS).

UDP flows

The distribution of the number of packets in UDP flows, for the range 0–20 packets is shown in Fig. 4.4a. The 93% of the UDP flows yield into this range, sharing the same behaviour as in the case of TCP flows. Moreover, most UDP flows have a value of $\overline{p_f}$ between 40 and 200 bytes (88.2%), as can be seen in Fig. 4.4b.

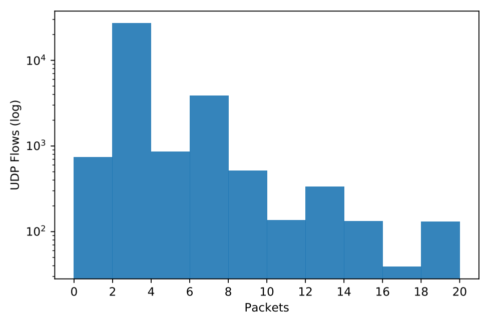


(a) Distribution of the number of packets (range 0–20 packets).

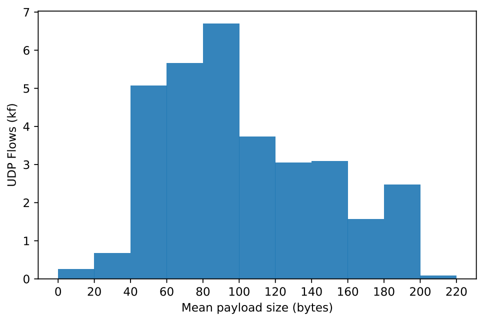


(b) Distribution of the mean payload size (range 0–50 bytes, log scale).

Figure 4.3: Packets and mean payload size distributions of TCP flows for malware captures.



(a) Distribution of the number of packets (range 0–20 packets, log scale).



(b) Distribution of the mean payload size (range 0–200 bytes).

Figure 4.4: Packets and mean payload size distributions of UDP flows for malware captures.

4.2.2 Normal Captures

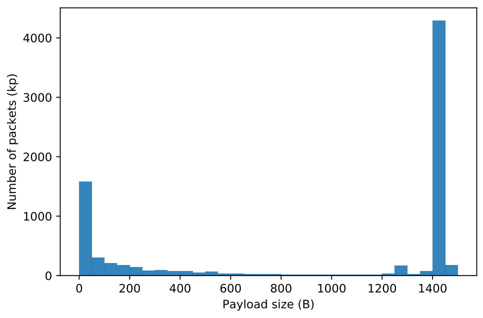
The normal traffic consists of twenty captures in which different actions that represent normal behaviour are performed, such as browsing the internet and file sharing over P2P.

Distribution of the payload size in the packets

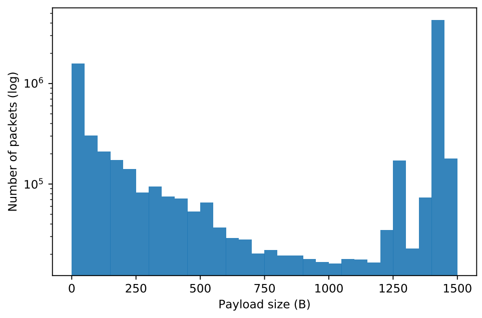
For the normal captures, as can be observed in Fig. 4.5, there are no packets with size greater than the Ethernet MTU size. Most of the packets have a payload size between 1400 and 1500 bytes (with median equal to 1420 bytes and mean equal to 929 bytes), while 23.8% of the packets have a payload size less than 100 bytes.

TCP flows

The packets and mean payload size distributions of TCP flows for normal captures are shown in Fig. 4.6. The histograms view is limited to the ranges in which most



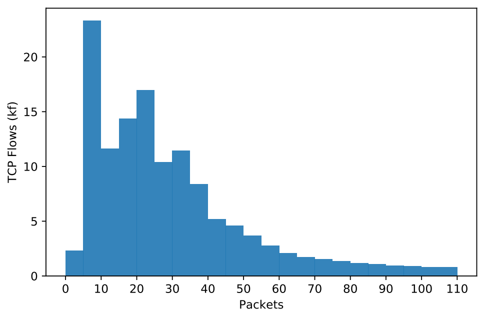
(a) Whole payload size range.



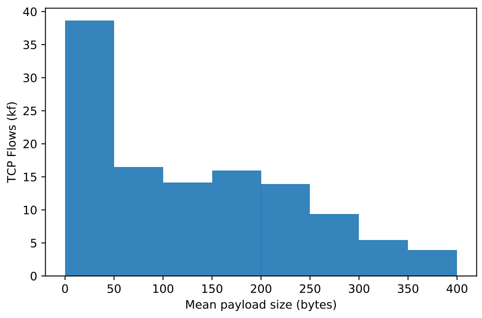
(b) Whole payload size range, log scale.

Figure 4.5: Distribution of the payload size in the packets for normal captures.

part of the flows are gathered. The 88.7% of the TCP flows in normal captures are made up of between 5 and 110 packets. The range of variation of $\overline{p_f}$ for most part of the TCP flows is 0–400 bytes (83.4%). Within this range, the 27.4% of the TCP flows have a value of $\overline{p_f}$ between 0 and 50 bytes on average, while the rest is uniformly distributed, being slightly smaller in the range 250–400 bytes. For normal captures, the 17.4% of the TCP flows have no payload.



(a) Distribution of the number of packets (range 0–110 packets).

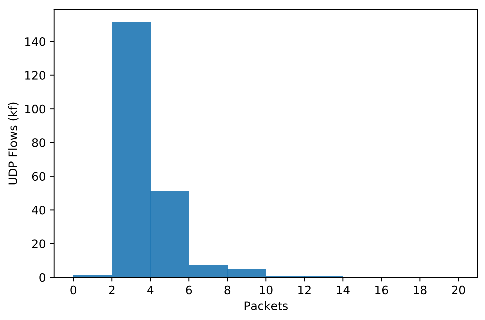


(b) Distribution of the mean payload size (range 0–400 bytes).

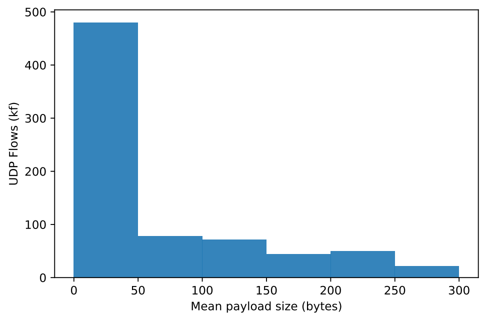
Figure 4.6: Packets and mean payload size distributions of TCP flows for normal captures.

UDP flows

The packets and mean payload size distributions of UDP flows for normal captures are shown in Fig. 4.7. The 98.9% of the flows are composed of between 2 and 10 packets. In this case, most flows fall within the range 0–300 of bytes of payload (99.6%), within which, the 83.71% fall within the range 50–200 bytes.



(a) Distribution of the number of packets (range 0–20 packets).



(b) Distribution of the mean payload size (range 0–300 bytes).

Figure 4.7: Packets and mean payload size distributions of UDP flows for normal captures.

4.2.3 Design Criteria

Raw Packets

In this input representation, the value of the parameter n is needed to be set. Given the previous analysis, note that setting a wrong value for n can lead to a strong bias in the learning model. In order to see this, let's consider the following example. Let's say that the fixed payload size is set to 1450 bytes. Under this scenario, larger payloads will be trimmed and the smaller ones zero-padded. Within this configuration, any naive classifier will be able to correctly classify any sample, just by looking for zeros beyond the byte number 1100. This is because the greatest part of the malware examples have a payload size below 1100 bytes (i.e., will be zero-padded to fill 1450 bytes), while the payload size of normal examples are mostly between 1400 and 1450 bytes.

A summary of the payload size distribution of packets is shown in Table 4.1.

Capture type	$\mathbf{n < 50\,B}$	$\mathbf{n \in [50, 1024]\ B}$	$\mathbf{n > 1024\,B}$
Malware	26.4%	60.5%	13.1%
Normal	20.0%	18.8%	61.2%

Table 4.1: Summary of the distribution of payload size of the packets (n) in bytes, for malware and normal captures.

With the information given by the different distributions, the following considerations were taken in order to build the *Raw Packets* version of the dataset:

1. Only the payloads of TCP, UDP and ICMP packets are considered.

2. Only the packets that have at least 50 bytes of payload are taken into account, the rest is discarded. This represents most part of the packets for each group of captures:
   - 73.6% of malware packets.
   - 80.0% of normal packets.
3. Since not all capture files contain the same number of packets, nor meet the conditions of the item number (2), and to be able to get a representative sample of each scenario, only those captures that at least have $20,000$ valid packets (malware) and $12,500$ valid packets (normal) were chosen. This leads to consider:
   - 10 valid malware captures (out of 13)
   - 16 valid normal captures (out of 20)

   conforming a balanced dataset of $200,000$ samples each.
4. The value of the parameter n is set in 1024 bytes: all packets with payloads larger than 1024 bytes are trimmed and the smaller ones are zero-padded.

Raw Flows

A summary of the statistics of the network traffic flows for the malware and normal captures is shown in Tables 4.2 and 4.3. For the *Raw Flows* representation, parameters n and m are needed to be set. As was stated at the beginning of this Section, only TCP and UDP flows are considered (other kind of flows represent only a small proportion of the total). TCP and UDP scenarios are quite different. For TCP, a high percentage of the flows are composed by packets with no payload, which is not the case for UDP flows. For this reason, TCP flows will not be considered, i.e., the scope of the *Raw Flows* approach will be only for UDP flows; hence the chosen values for the parameters n and m will be based on the median value of UDP flows.

Flow type	**Capture type**	**Qty. (pkt)**	**Med. (pkt)**	**Mean (pkt)**	**Std. (pkt)**	$\rho_{\mathbf{p}}$ **(pkt)**	**%**
TCP	Malware	188,548	6	17.3	129.7	[2–20]	90
UDP	Malware	35,572	3	29.5	976.1	[2–20]	93
TCP	Normal	140,931	25	78.3	1,498.9	[5–110]	88.7
UDP	Normal	217,008	2	4.58	560.2	[2–10]	98.9

Table 4.2: Summary of the distribution of the number of packets in the flows. Most flows are composed by a number of packets in the range ρ_p. The proportion of flows that fall within this range is represented by the percentage (%).

Flow type	**Capture type**	**Med. (bytes)**	**Mean (bytes)**	**Std. (bytes)**	$\rho_{\mathbf{b}}$ **(bytes)**	**%**
TCP	Malware	0	50.9	145.86	[0–50]	84.2
UDP	Malware	98	121.0	98.88	[40–200]	88.2
TCP	Normal	154.5	209.9	217.8	[0–400]	83.4
UDP	Normal	108	114.4	55.4	[50–200]	83.7

Table 4.3: Summary of the distribution of the mean payload size of the packets in the flows. Most flows have a $\overline{p_f}$ in the range ρ_b. The proportion of flows that fall within this range is represented by the percentage (%).

To summarize, two different datasets to fit each one of the considered input representations are built. The dataset for the *Raw Packets* representation, after removing duplicate instances, consists of $248,850$ instances. For the *Raw Flows* representation, the dataset consists of $67,494$ instances. For the learning process, both datasets are split according to the following schema: 80% of the samples for training, 10% for validation and 10% for testing. In Table 4.4 the parameters selection for each input representation is presented. Note that the chosen values obey the initial constraint about selecting *small* values for both n and m, and hence to be able to perform network traffic classification *on the fly*.

Representation	**Dataset size**	**n (bytes)**	**m (packets)**
Raw Packets	$248,850$	1024	N/A
Raw Flows	$67,494$	100	2

Table 4.4: Parameters selection for building the input representation for training the deep learning models.

Chapter 5

Deep Learning Architectures for Malware Detection

In this Chapter, the different deep learning architectures designed for both *Raw Packets* and *Raw Flows* input representations are presented. Since hyperparameter optimization is an important step in the training stage of deep learning models, a description of this process is also detailed.

5.1 Raw Packets deep learning architecture

As was explained in Chapter 4, the key information to use in both *Raw Packets* and *Raw Flows* input representations is included in the payload. The main goal here is to choose core layers for the deep learning model that can exploit the underlying characteristics of each class found in the payload. For this problem, one-dimensional data is to be processed in a single grid-like fashion. For this reason, and to build the feature representation of the spatial data inside the packets, a 1D-convolutional neural network (1D-CNN) layer will be considered as one of the core layers of the deep learning model architecture. This is the cornerstone in which it is expected the model to improve over shallow-like traditional methods.

In the early stages of training, only two 1D-CNN layers were used but this model was quite unstable and the performance was far low than expected. Afterwards, and to improve the model's performance, an LSTM recurrent layer was added to be used together with both 1D-CNNs. This allows the model to keep track of temporal information inside each packet supporting sequence prediction, for example, to be able to catch the application's protocol initialization. To deal with the different combinations of discovered features and bring a final estimation, two fully-connected layers are added at the end of the model's architecture.

The final architecture of the deep learning model for the *Raw Packets* input representation is shown in Fig. 5.1. It consists of:

1. Two 1D-CNN layers of 32 and 64 filters of size 5, respectively
2. A max-pooling layer of size 8

3. A LSTM layer consisting of 200 units per LSTM cell, i.e., at every moment the hidden state is a vector of size 200 and all hidden state outputs are returned

4. Two fully-connected layers of 200 units each

Spatial and normal batch normalization layers are added after each 1D-CNN and fully-connected layers to ease the training process. Dropout layers are also used to add regularization to the model.

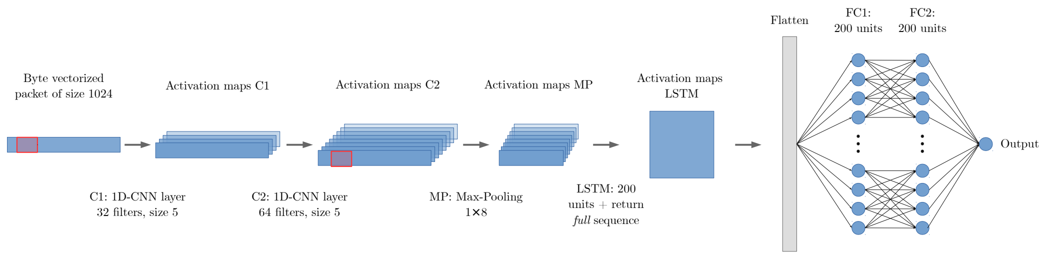

Byte vectorized packet of size 1024
Activation maps C1
Activation maps C2
Activation maps MP
Activation maps LSTM
Flatten
FC1: 200 units
FC2: 200 units
Output
C1: 1D-CNN layer 32 filters, size 5
C2: 1D-CNN layer 64 filters, size 5
MP: Max-Pooling 1×8
LSTM: 200 units + return *full* sequence

Figure 5.1: Deep learning architecture for *Raw Packets* representation.

As usual, a binary cross-entropy is used as the loss function. The reason behind choosing a cross-entropy loss function over others is strictly about the optimization process performance and the problem of vanishing and exploding gradients that can occur during the learning process of gradient-based optimization. Vanishing gradients make it difficult to know which direction the parameters should move to improve the cost function, while exploding gradients can make learning unstable. Historically, mean squared error and mean absolute error were popular in the 1980s and 1990s, but were gradually replaced by cross-entropy losses by the statistics and machine learning community. The use of cross-entropy losses greatly improve the performance of models with sigmoid and softmax outputs, which had previously suffered from saturation and slow learning when using the mean squared and mean absolute error losses. This is one reason that the cross-entropy cost function is more popular and is usually used in practice over the mean squared error or mean absolute error, even when it is not necessary to estimate an entire distribution $p(y|x)$ [3].

Table 5.1 shows the model summary, describing each one of the layers and the number of parameters. The total number of parameters of this model is $5,345,081$, where $5,344,089$ are trainable and 992 non-trainable. For the sake of completeness, the final list of hyperparameter is shown in Table 5.2.

Layer type	**Output shape**	**Nr. of params.**
1D-CNN	(None, 1020, 32)	192
Batch normalization	(None, 1020, 32)	128
Activation (ReLU)	(None, 1020, 32)	0
1D-CNN	(None, 1016, 64)	$10,304$
Batch normalization	(None, 1016, 64)	256
Activation (ReLU)	(None, 1016, 64)	0
1D-Max-Pooling	(None, 127, 64)	0
Dropout	(None, 127, 64)	0
LSTM	(None, 127, 200)	$212,000$
Flatten	(None, 25400)	0
FC	(None, 200)	$5,080,200$
Batch normalization	(None, 200)	800
Activation (ReLU)	(None, 200)	0
Dropout	(None, 200)	0
FC	(None, 200)	$40,200$
Batch normalization	(None, 200)	800
Activation (ReLU)	(None, 200)	0
FC	(None, 1)	201

Table 5.1: *Raw Packets* deep learning model summary.

5.2 Raw Flows deep learning architecture

For the *Raw Flows* approach, since the dataset is smaller than the *Raw Packets* one (not only in number of instances, but also in the number of features), the capacity of the model (i.e., representation power) does not have to be as high as in the *Raw Packets* case. Different architectures were tested before arriving to the final one, but this time, the knowledge gathered during the *Raw Packets* design was used to build this architecture. The starting point consisted of using the same architecture as in the *Raw Packets* model but removing the LSTM layer.

Hyperparameter	**Value**
Nr. filters (1D-CNN1)	32
Nr. filters (1D-CNN2)	64
Filter size	5
Max-Pooling size	8
Dropout p (1D-CNN2)	0.1
Dropout p (FC1)	0.1
Nr. LSTM/FC1/FC2 units	200
Optimizer	Adam
Learning rate	0.008
Learning rate decay	0.995
Batch size	256
Nr. of epochs	100

Table 5.2: *Raw Packets* deep learning hyperparameters summary.

In this initial setup, the model started to overfit quickly. Then, the following step consisted of removing one of the 1D-CNN layers and some of the hidden units in both the fully-connected layers. Furthermore, the same number of filters and filter size used for the *Raw Packets* architecture were tested for the 1D-CNN layer as a first choice. These settings proved to give the model enough capacity as well as a good performance during the learning process. The final *Raw Flows* architecture consists of:

1. One 1D-CNN layer of 32 filters of size 5
2. Two fully-connected layers of 50 and 100 units each

Also, binary cross-entropy is used as the loss function. The architecture is shown in Fig. 5.2. The layer summary of the model is shown in Table 5.3 and the final hyperparameters to train the model are shown in 5.4. The total number of parameters of this model is $86,331$, where $85,967$ are trainable and 364 non-trainable.

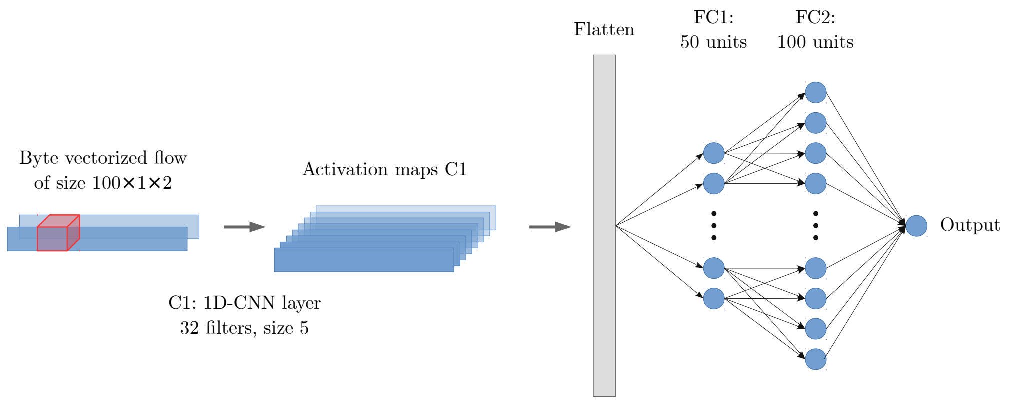


Figure 5.2: Deep learning architecture for *Raw Flows* representation.

Layer type	Output shape	Nr. of params.
1D-CNN	(None, 100, 32)	352
Batch normalization	(None, 100, 32)	128
Activation (ReLU)	(None, 100, 32)	0
1D-Max-Pooling	(None, 50, 32)	0
Flatten	(None, 1600)	0
FC	(None, 50)	80050
Batch normalization	(None, 50)	200
Activation (ReLU)	(None, 50)	0
FC	(None, 100)	5100
Batch normalization	(None, 100)	400
Activation (ReLU)	(None, 100)	0
FC	(None, 1)	101

Table 5.3: *Raw Flows* deep learning model summary.

5.3 Remarks on the number of parameters

When training deep learning models, it is nearly always the case to have far more parameters than training samples. This is not unrelated to this particular problem (cf., Tables 5.1 and 5.3). In [63], Zhang *et al.*, showed that a simple two-layer FC

Hyperparameter	**Value**
Nr. filters (1D-CNN)	32
Filter size	5
Max-Pooling size	2
Nr. FC1 units	50
Nr. FC2 units	100
Optimizer	Adam
Learning rate	0.001
Batch size	1024
Nr. of epochs	10

Table 5.4: *Raw Flows* deep learning hyperparameters summary.

network with $2n + d$ parameters can perfectly fit any dataset of n samples and d features. Nevertheless, deep neural networks can generalize very well. This is mainly because of the role of implicit regularization presented in the optimization process involved in the training and the usage of explicit regularization methods, such as dropout and weight-decay. More details and discussion about this topic can be consulted in [63].

5.4 Hyperparameter optimization

Hyperparameters are settings that are not learned by the learning algorithm itself, and so must be determined externally. Deep learning models have several of these, and many of them have a great impact over the performance. The hyperparameter optimization process can be tedious and needs to be strictly neat, i.e., modifying one parameter at a time is a must, in order to be able to discover which is the real impact over the learning process. Think about trying to solve a Rubik cube. Sometimes it may happen that when you finish solving one of the cube sides, and move on to solve the next one, some or several of the small cube units of the solved side were changed as a result of trying to fix the new one. This Rubik's cube-solving analogy can be used to understand the hyperparameter optimization process of deep learning models, where you have to be meticulous and organized when modifying the different settings. Next, some of the most significant considerations for setting the different hyperparameters are described.

5.4.1 Number of filters and filter size of CNNs layers

The number of filters and filter size are both essential to the convolutional layers. For these, different values were tested to get a set that finally performed well. For example, at first, a filter size of 200 was used. In this case, the model was found to be very difficult to train. It is possible that given the huge number of parameters to learn with a filter of such size caused the model to perform poorly. After several tests the filter size was reduced and the model started to perform better. The final filter size chosen for both architectures was 5.

The increasing number of filters in each layer is one of the many rules of thumb when designing CNN-based deep learning models. The idea is that successive layers containing two or four times the number of filters can help the network to learn hierarchical features. For the *Raw Packets* architecture, 32 and 64 filters were chosen for both 1D-CNN respectively; while in the case of the *Raw Flows* architecture, the final number of filters was 32.

5.4.2 Learning rate

The learning rate is one of the hyperparameters that is the most difficult to set because it has a significant impact on the model's performance, thus choosing a good learning rate candidate is very important for the learning process of the model. This may be chosen by trial and error, but it is usually best to choose it by monitoring learning curves that plot the objective function as a function of time. This is more of an art than a science, and most guidance on this subject should be regarded with some skepticism. The main question is how to set the *initial* learning rate. If it is too large, the learning curve will show violent oscillations, with the cost function often increasing significantly. Gentle oscillations are fine, especially if training with a stochastic cost function such as the cost function arising from the use of dropout. If the learning rate is too low, learning proceeds slowly, and if the initial learning rate is too low, learning may become stuck with a high cost value. Typically, the optimal initial learning rate, in terms of total training time and the final cost value, is higher than the learning rate that yields the best performance after the first 100 iterations or so. Therefore, it is usually best to monitor the first several iterations and use a learning rate that is higher than the best-performing learning rate at this time, but not so high that it causes severe instability [3].

The heuristic method for choosing a good candidate for the learning rate is as follows. As was mentioned, the main goal is that the loss decreases within the first iterations, so firstly, all regularization and learning rate decay is turned off. Then, an upper bound for the learning rate is searched, increasing the learning rate heavily until the loss explodes in the first couple of iterations. After that, the learning rate is logarithmically dropped until finding a value that causes the loss to go down. If using this learning rate candidate the loss function tends to decrease within the first thousands iterations and does not plateau, a good learning rate candidate was found and will be used for the training process.

For the *Raw Packets* and *Raw Flows* architectures, the chosen learning rates were 0.008 and 0.001, respectively. Also, in the *Raw Packets* architecture, a learning rate decay was added since it was noted during the learning process that the loss function was starting to plateau.

5.4.3 Optimizer

The optimizer is the function used to perform the parameter's update during the learning process. Given that the learning rate is one of the most important hyperparameters, recently a number of incremental (or mini-batch-based) methods that adapt the learning rates over time have been introduced. AdaGrad [64], RMSprop [65] and Adam [66] are examples of these algorithms with adaptive learning rates that are widely used in practice.

AdaGrad

The AdaGrad algorithm individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values. AdaGrad is designed to converge rapidly when applied to a convex function. Empirically, AdaGrad has been found that when applied to a non-convex function –such as deep neural network models– the accumulation of squared gradients from the beginning of training can result in a premature and excessive decrease in the effective learning rate. The AdaGrad updating rule is shown in Eq. 5.1, where g represents the gradient, ϵ the global learning rate, δ is a small constant used to stabilize division by small numbers, and $\Delta\theta$ is the parameter update.

$$\begin{aligned} \text{Accumulate squared gradient: } & r \leftarrow r + g \odot g \\ \text{Compute update: } & \Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g \end{aligned} \tag{5.1}$$

RMSProp

The RMSProp algorithm modifies AdaGrad to perform better in the non-convex setting by changing the gradient accumulation into an exponentially weighted moving average. RMSProp uses an exponentially decaying average to discard history from the extreme past so that it can converge rapidly after finding a convex bowl, as if it were an instance of the AdaGrad algorithm initialized within that bowl.

Compared to AdaGrad, the use of the moving average introduces a new hyperparameter, ρ, that controls the length scale of the moving average. The RMSProp updating rule is shown in Eq. 5.2.

$$\begin{aligned} \text{Accumulate squared gradient: } & r \leftarrow \rho r + (1 - \rho)\, g \odot g \\ \text{Compute update: } & \Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g \end{aligned} \tag{5.2}$$

Adam

Adam –short for adaptive moments– can be seen as a combination of RMSProp and momentum, with some distinctions. To begin with, a smooth version of the gradient is used instead of the raw (and perhaps noisy) gradient vector used by RMSProp. Secondly, Adam includes bias corrections to the estimates of both the first-order moments (the momentum term) and the (uncentered) second-order moments to account for their initialization at the origin. Eq. 5.3 shows the update rule of Adam's algorithm, where s and r represent the first and second order moment variables, ρ_1 and ρ_2 the exponential decay rates for moment estimates, and t the time step.

$$\begin{aligned}
\text{Update biased first moment estimate: } & s \leftarrow \rho_1 s + (1-\rho_1)g \qquad (5.3)\\
\text{Update biased second moment estimate: } & r \leftarrow \rho_2 r + (1-\rho_2)g \odot g\\
\text{Correct bias in first moment: } & \hat{s} \leftarrow \frac{s}{1-\rho_1^t}\\
\text{Correct bias in second moment: } & \hat{r} \leftarrow \frac{r}{1-\rho_2^t}\\
\text{Compute update: } & \Delta\theta = -\epsilon\frac{\hat{s}}{\sqrt{\hat{r}}+\delta}
\end{aligned}$$

Empirically, RMSProp and Adam have been shown to be effective and practical optimization algorithms for deep neural networks. They are currently two of the go-to optimization methods being employed routinely by deep learning practitioners. For this work, RMSprop and Adam were tested, being the latter the one that showed best performance and stability.

5.4.4 Regularization

Controlling overfitting was definitely an issue, specially in the *Raw Packets* architecture. L2 regularization and dropout were tested. Adding L2 regularization showed no improvement in the performance. In the case of Dropout, the model was found to be very sensitive to the parameter p. Choosing a high value for p showed a really unstable model, with a bouncing validation loss. Since dropout is turned off at validation, this suggested to try lower values for p. Also, different positions for the dropout layers over the sequential models were tested. Finally, dropout was used within the *Raw Packets* architecture after the convolutional layers and after the first FC layer, using in both cases a value of $p = 0.1$.

5.4.5 Checking the model capacity

As a sanity check, after choosing a network architecture, the capacity of the chosen model can be tested overfitting a small subset of the data, setting all regularization to zero. In a nutshell, if it's not possible to achieve zero cost over a small subset of the data, it is not worth to proceed over the full dataset, since at first glance the capacity of the model is not big enough. In Fig. 5.3 an example of the model loss and accuracy evolution after each epoch (being an epoch a single pass-through over the complete training dataset) using 20 samples for the *Raw Packets* configuration is shown.

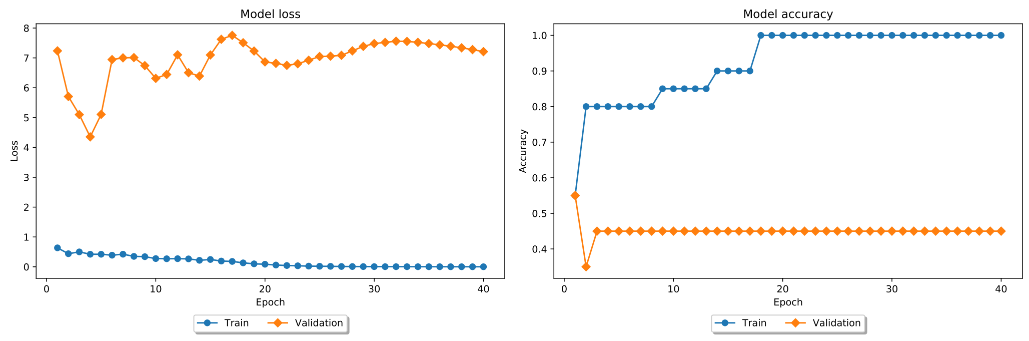


Figure 5.3: Overfit small subset of data to check model capacity. Example for *Raw Packets* configuration.

Chapter 6

Experiments and Results

In this Chapter, the experimental evaluations and results achieved for the network traffic malware detection problem, for both input representations are presented. An extension of the malware detection problem to a multiclass malware classification problem is also introduced at the end of this Chapter.

In all the experimental evaluations, some common data pre-processing is performed over the datasets before training [24]. Firstly, the data is zero-centered subtracting the mean across every individual feature. This operation has the geometric interpretation of centering the cloud of data around the origin along every dimension. Secondly, the data dimensions are normalized so that they are of approximately the same scale. In this case, standardization was performed, dividing each dimension by its standard deviation after it has been zero-centered.

All deep learning models were built using the Keras framework [67] running on top of TensorFlow [68], using the Big-DAMA platform [69], a big-data cluster for analyzing network traffic data with machine learning models.

6.1 Malware Detection: A First Approach Using Deep Learning

The focus of this section is exclusively on the problem of malware detection, posing it as a binary classification problem: either normal or malware. To show the main advantages of the proposed approaches, three evaluation questions are posed:

1. Is it possible to achieve high detection accuracy with low false positive rates using the raw-input, deep learning-based models?

2. Are the proposed deep learning-based models better than the commonly used shallow models for malware detection, when feeding them all with raw inputs (e.g., bytestreams)?

3. How good are the raw-input, deep learning-based models as compared to a traditional approach for malware detection, where shallow models take as input specific handcrafted features based on domain expert knowledge?

The following sections of this Chapter aim to answer these three questions.

6.1.1 Deep learning vs. shallow models with raw inputs

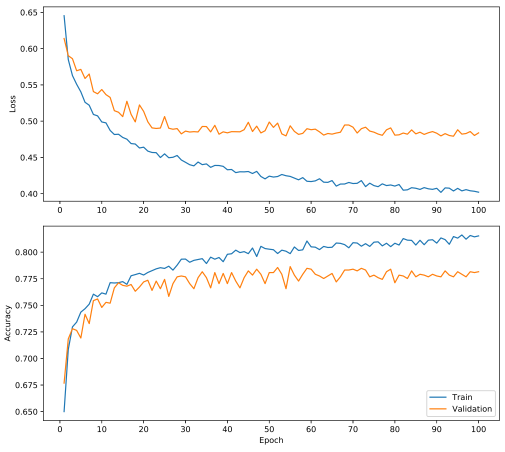
In order to answer the first question, a simple evaluation scenario is presented, detecting malware at the packet level. For this problem, *Raw Packets* deep learning architecture was trained using the respective dataset version consisting of roughly $250,000$ samples (cf., Chapter 4). The dataset was split using a 80/10/10 schema, i.e., 80% of the samples for training, 10% for validation and the remaining 10% for testing. Fig. 6.1a shows the learning process for the *Raw Packets* approach, where training over mini-batches of data was used for the parameters update and the model was trained over 100 epochs. Adam was used as the optimizer function, annealing the learning rate over time.

The performance metric chosen was the accuracy, given that the dataset is perfectly balanced. For the *Raw Packets* representation, an accuracy of 77.6% over the test set was achieved. Fig. 6.1b presents the initial results obtained by the *Raw Packets* deep learning model in the detection of malware packets, in the form of a ROC curve. The model is compared to a random forest one, using exactly the same input features and an internal architecture of 100 trees applying different pruning techniques to prevent overfitting, such as maximum depth and maximum number of instances per leaf. The random forest model was chosen based on the generally outstanding detection performance shown by the model in [70], using domain expert input features.

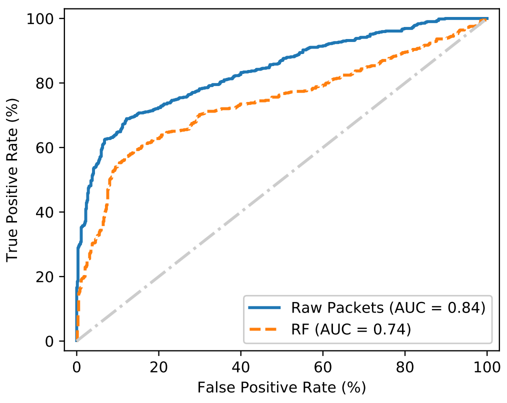
The deep learning model can detect about 60% of the malware traffic packets with a false positive rate of 6%, with an overall out-performance of nearly 25% as compared to the random forest. These results are highly encouraging, since they point to the ability of the deep learning-based model to better capture the underlying statistics of the malware, without requiring any specific handcrafted feature set. However, the absolute detection performance results are still not good enough so as to rely on such a deep learning model with raw packet inputs for malware detection in practice. Also, it can be noted in the learning curve that there is some potential overfitting for this scenario using the packet-level representation.

6.1.2 Packet vs. flow representation performance

In this Section, a step further is taken facing a similar comparison as before, but considering the *Raw Flows* representations as input. The dataset for the *Raw Flows* representation, after removing duplicate instances, consists of about $68,000$ instances. This dataset is split according to the same schema as before: 80% of the samples for training, 10% for validation and 10% for testing. Also, the training was performed using mini-batches and Adam was used as optimizer. The learning process is described in Fig. 6.2a. The learning process was held over 10 epochs. For the *Raw Flows* representation, an accuracy of 98.6% was achieved. As it was done before, Fig. 6.2b compares the detection performance of *Raw Flows* model against a random forest model using exactly the same raw input features. In this case, the data was flattened in order to fit the input to the random forest. Once again, a clear out-performance of the deep learning architecture can be observed as compared to the random forest model. Using this representation the deep learning



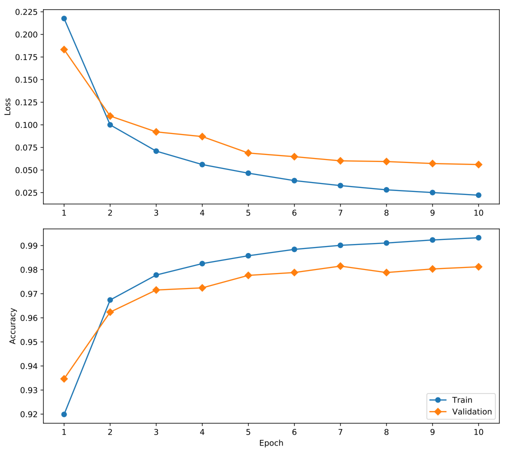
(a) Learning process.



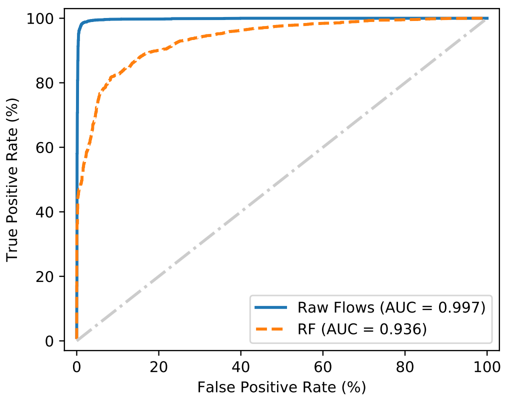
(b) ROC curve.

Figure 6.1: Learning process (loss and accuracy evolution after each epoch) and ROC curve for the *Raw Packets* representation.

model can detect as much as 98% of all malware flows with a false positive rate as low as 0.2%. This suggests that, when operating at the flow level, such raw input representation and associated deep learning architecture can actually provide highly accurate results, applicable in practice.



(a) Learning process.



(b) ROC curve.

Figure 6.2: Learning process (loss and accuracy evolution after each epoch) and ROC curve for the *Raw Flows* representation.

6.1.3 Domain knowledge vs. raw inputs

The last step of the evaluations tries to answer the third question regarding the goodness and advantages of the proposed approach with respect to the standard approach for machine learning-based malware detection. In particular, studying how good is the *Raw Flows*, deep learning-based model as compared to a random forest-based one, the latter using as input specific handcrafted features based on domain expert knowledge. The standard approach for detection of malware and network attacks in networking traffic is to rely on flow-level features, using traditional in-flow packet measurements such as traffic throughput, packet sizes, inter-arrival times, frequency of IP addresses and ports, transport protocols and share of specific flags (e.g., SYN packets). Therefore, a set of almost 200 of these features were built to feed a random forest model. Note that besides using traditional features such as min/avg/max values of some of the input measurements, their empirical distribution is also considered, sampling the empirical distribution at many different percentiles. This provides as input much richer information, as the complete distribution is taken into account.

First results on this subject published in [71] suggest that not surprisingly, the random forest model using expert domain features achieves highly accurate detection performance, detecting about 97% of all the malware instances with less than 1% of false positives. However, also under this scenario, the deep learning-based model, using the *Raw Flows* representations as input, slightly outperforms this domain expert knowledge based detector. As such, it can be concluded that the deep learning model can perform as good as a more traditional shallow-model based detector for detection of malware flows, without requiring any sort of expert handcrafted inputs.

Based on the three sets of evaluations, and recalling that the random forest model serves as performance benchmark, it can be concluded that the proposed deep learning model, in particular using the *Raw Flows* representation as input, can: (i) provide highly accurate and applicable-in-practice malware detection results, (ii) capture the underlying malware and normal traffic models better than a shallow-like, random forest-based model, and (iii) provide results as good as those obtained through a domain expert knowledge-based detector, without requiring any sort of handcrafted features.

6.2 One Step Further: from Malware Detection to Malware Classification

To complement the previous malware detection results, a variation of the binary classification problem is presented, considering now different sorts of malware attacks traffic as different classes, together with a "normal" class representing benign traffic.

To build the dataset, three different malware traffic classes are considered, corresponding to three different types of botnets, named: *Neris*, *Rbot* and *Virut*. Thus, the multiclass classification problem has four different classes: three that represent malware attacks and one that represents normal activity. The activity and protocols used for the scenarios chosen for building this dataset are depicted in Table 6.1. The dataset consists of $160,000$ samples, selected in a stratified way, i.e., the dataset is balanced with $40,000$ samples per class.

The deep learning model used for this task is quite similar to the one used for the *Raw Packets* representation. The difference is that now the activation function

Botnet	Protocol	Activity
Neris	IRC	spam, click fraud
Rbot	IRC	DDoS
Virut	HTTP	spam, port scan

Table 6.1: Protocols and attacks performed by different kinds of botnets.

used in the last fully connected layer is a Softmax function (a generalization of the binary logistic regression classifier for multiple classes), instead of the sigmoid used for the binary classification. The resulting architecture is depicted in Fig. 6.3. The corresponding loss function is also different for the multiclass classification problem. In this case a categorical cross-entropy was used, and the learning process was held over 50 epochs. The evolution of the learning process, including loss and accuracy after each epoch, is presented in Fig. 6.4. Not surprisingly, the evolution of the learning process is very similar to the previous raw packets-based model (cf. Fig. 6.1a), as both operate using a similar input representation.

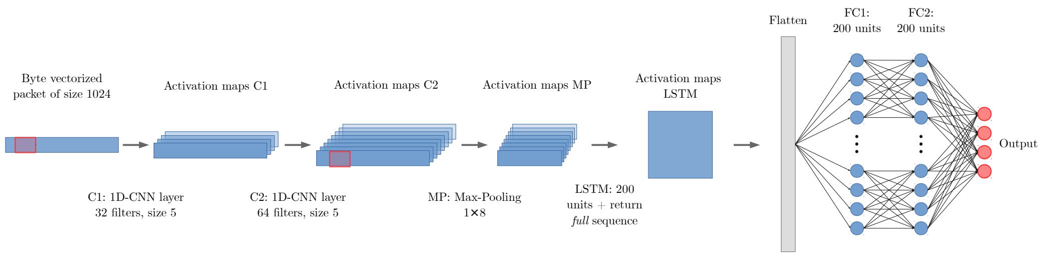

Byte vectorized packet of size 1024
Activation maps C1
Activation maps C2
Activation maps MP
Activation maps LSTM
Flatten
FC1: 200 units
FC2: 200 units
Output
C1: 1D-CNN layer 32 filters, size 5
C2: 1D-CNN layer 64 filters, size 5
MP: Max-Pooling 1×8
LSTM: 200 units + return *full* sequence

Figure 6.3: Deep learning architecture for multiclass classification of malware traffic, using *Raw Packets* representation.

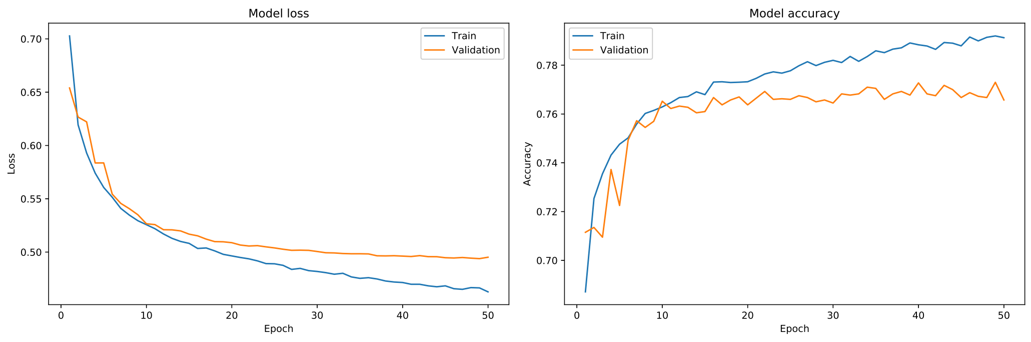
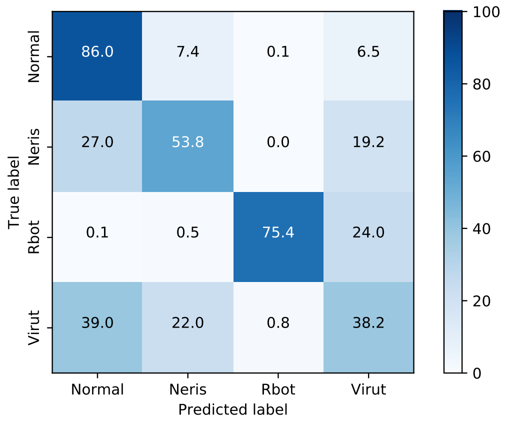


Figure 6.4: Evolution of the learning process (loss and accuracy after each epoch) for the multiclass malware classification problem.

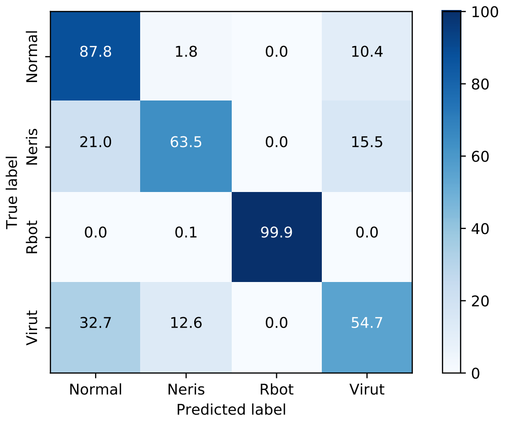
For this problem, the performance of the deep learning model against a random forest one is also compared, using the same input features for both. The Fig. 6.5 shows the normalized confusion matrices for both models (values are shown in percentages); the deep learning model outperforms the random forest for all classes. As a sanity check, note that the accuracy of the multiclass problem considering a binary approach (malware vs. normal) holds similar results as the *Raw Packets* approach presented in Sec. 6.1 (77.6% vs. 76.5%). To complement, the Table 6.2 shows the performance metrics of the deep learning model for the malware classification task. The chosen metrics are accuracy (AC), precision (PR), recall (RC) and F_1 score for each class:

$$\begin{aligned} AC &= \frac{TP+TN}{TP+TN+FP+FN} & PR &= \frac{TP}{TP+FP} \\ RC &= \frac{TP}{TP+FN} & F_1 &= 2 \times \frac{PR \times RC}{PR+RC} \end{aligned} \tag{6.1}$$

Values are computed in a *one-vs-all* schema, in which each class is evaluated against the rest, as if it was a binary problem. It is interesting to note that Rbot botnet is detected with an accuracy of 99.9% while Neris and Virut achieve 63.5% and 54.7%, respectively. The fact that both Neris and Virut share spam as an activity attack, could be a possible reason which makes them more difficult to distinguish one from each other – see Table 6.1.



(a) Random forest.



(b) Deep learning Multiclass.

Figure 6.5: Normalized confusion matrices (showing percentage values) for both random forest and deep learning models.

Class	**Accuracy**	**Precision**	**Recall**	**F_1 score**
Normal	0.878	0.621	0.878	0.727
Neris	0.635	0.814	0.635	0.714
Rbot	0.999	1.000	0.999	1.000
Virut	0.547	0.679	0.547	0.606

Table 6.2: Performance metrics for the deep learning model in the malware classification task.

Chapter 7

Conclusions and Further Work

In this thesis, a first study of the power of deep learning models to the analysis of network traffic measurements is presented. The focus is on the particular topic of malware network traffic detection and classification, considering *raw* representations of the input network data. Three different datasets were conformed from malware and normal captures in order to be able to train three different deep learning models, i.e., *Raw Packets* and *Raw Flows* for malware detection and *Raw Packets* for malware classification. The results presented in Chapter 6 show how using *Raw Flows* as input for the deep learning models achieves better results than using *Raw Packets*. Finally, a variation of the binary classification model using a multiclass approach to discriminate between different types of malware is also introduced. In all cases, the deep learning models outperform a strong random forest model, using exactly the same input features. Moreover, the *Raw Flows* architecture slightly outperforms a random forest model trained using expert domain knowledge features. This points to the power of deep learning models to better capture the underlying statistics of malicious traffic, as compared to more traditional, shallow-like machine learning models.

This thesis also leaves some open doors for future work. To begin with, the exploration of the *Raw Flows* input representation for the multiclass problem is of particular interest, and can possibly shed some light on the problems depicted in Section 6.2 with respect to some of the problematic classes. Besides, since only the payload data is being considered, there is some potentially useful information included in the header's packets that is not being taken into account for training the deep learning models. Therefore, a model for detecting attacks presented in TCP flows can take advantage of this information. Last but not least, the usage of other datasets could also be explored in order to improve the presented models.

As a result of the research work for this thesis, the following academic papers were published:

- Gonzalo Marín, Pedro Casas, and Germán Capdehourat. RawPower: Deep Learning Based Anomaly Detection from Raw Network Traffic Measurements. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters, Demos and Student Research Competition*, ACM SIGCOMM 2018, pages

75–77, New York, NY, USA, 2018. ACM. **2nd Place SRC Undergraduate Competition.**

- Gonzalo Marín, Pedro Casas, and Germán Capdehourat. DeepSec Meets RawPower – Deep Learning for Detection of Network Attacks Using Raw Representations. *ACM SIGMETRICS Performance Evaluation Review*, 46(3):147–150, January 2019

- Gonzalo Marín, Pedro Casas, and Germán Capdehourat. Deep in the Dark – Deep Learning-based Malware Traffic Detection Without Expert Knowledge. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy Workshops (SPW), 2nd Workshop on Deep Learning and Security, San Francisco, CA, USA*, IEEE Security and Privacy, 2019

- Pedro Casas, Gonzalo Marín, Germán Capdehourat, and Maciej Korczynski. MLSEC – Benchmarking Shallow and Deep Machine Learning Models for Network Security. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy Workshops (SPW), 2019 International Workshop on Traffic Measurements for Cybersecurity (WTMC), San Francisco, CA, USA*, IEEE Security and Privacy, 2019

Bibliography

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[2] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[5] Raouf Boutaba, Mohammad A. Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M. Caicedo. A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities. *Journal of Internet Services and Applications*, 9(1):16, Jun 2018.

[6] M. A. Alsheikh, S. Lin, D. Niyato, and H. Tan. Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications. *IEEE Communications Surveys Tutorials*, 16(4):1996–2018, Fourthquarter 2014.

[7] M. Bkassiny, Y. Li, and S. K. Jayaweera. A Survey on Machine-Learning Techniques in Cognitive Radios. *IEEE Communications Surveys Tutorials*, 15(3):1136–1159, Third 2013.

[8] Anna L. Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys and Tutorials*, 18(2):1153–1176, 2016.

[9] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani. State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems. *IEEE Communications Surveys Tutorials*, 19(4):2432–2455, Fourthquarter 2017.

[10] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza. A Survey of Machine Learning Techniques Applied to Self-Organizing Cellular Networks. *IEEE Communications Surveys Tutorials*, 19(4):2392–2431, Fourthquarter 2017.

[11] T. T.T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *Commun. Surveys Tuts.*, 10(4):56–76, October 2008.

[12] J. Saxe, R. Harang, C. Wild, and H. Sanders. A Deep Learning Approach to Fast, Format-Agnostic Detection of Malicious Web Content. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 8–14, 2018.

[13] B. J. Radford, L. M. Apolonio, A. J. Trias, and J. A. Simpson. Network Traffic Anomaly Detection Using Recurrent Neural Networks. *ArXiv e-prints*, March 2018.

[14] J. Zhao, S. Shetty, and J. W. Pan. Feature-based Transfer Learning for Network Security. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pages 17–22, 2017.

[15] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. LEMNA: Explaining Deep Learning Based Security Applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 364–379, New York, NY, USA, 2018. ACM.

[16] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. Applications of Deep Reinforcement Learning in Communications and Networking: a Survey. *CoRR*, abs/1810.07862, 2018.

[17] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, pages 50–56, New York, NY, USA, 2016. ACM.

[18] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 197–210, New York, NY, USA, 2017. ACM.

[19] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2015.

[20] Jürgen Schmidhuber. Deep Learning in Neural Networks: an Overview. *Neural Networks*, 61:85–117, 2015.

[21] Y. Bengio, A. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug 2013.

[22] Asja Fischer and Christian Igel. Training Restricted Boltzmann Machines: an Introduction. *Pattern Recognition*, 47(1):25–39, 2014.

[23] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An Introduction to Deep Reinforcement Learning. *CoRR*, abs/1811.12560, 2018.

[24] Stanford University. CS231n: Convolutional Neural Networks for Visual Recognition. `http://cs231n.stanford.edu/`, 2018.

[25] Christopher Olah. Understanding LSTM networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2018.

[26] Nicolas Brunel, Vincent Hakim, and Magnus JE Richardson. Single neuron dynamics and computation. *Current Opinion in Neurobiology*, 25:149 – 155, 2014. Theoretical and computational neuroscience.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[28] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[29] Alberto Castro, Matías Richart, Javier Baliosian, and Eduardo Grampín. Opportunities for ai/ml in telecommunications networks. In *Proceedings of the 10th Latin America Networking Conference*, LANC '18, pages 89–95, New York, NY, USA, 2018. ACM.

[30] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques. *J. Netw. Comput. Appl.*, 60(C):19–31, January 2016.

[31] Monowar H. Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K. Kalita. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys & Tutorials*, 16:303–336, 2014.

[32] Tarem Ahmed, Boris Oreshkin, and Mark Coates. Machine learning approaches to network anomaly detection. In *Proceedings of the 2Nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, SYSML'07, pages 7:1–7:6, Berkeley, CA, USA, 2007. USENIX Association.

[33] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. *SIGCOMM Comput. Commun. Rev.*, 43(4):339–350, August 2013.

[34] P. Casas, A. D'Alconzo, F. Wamser, M. Seufert, B. Gardlo, A. Schwind, P. Tran-Gia, and R. Schatz. Predicting qoe in cellular networks using machine learning and in-smartphone measurements. In *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6, May 2017.

[35] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang. Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 32(2):92–99, March 2018.

[36] David D. Clark, Craig Partridge, J. Christopher Ramming, and John T. Wroclawski. A Knowledge Plane for the Internet. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 3–10, New York, NY, USA, 2003. ACM.

[37] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J. Hibbett, Giovani Estrada, Khaldun Ma'ruf, Florin Coras, Vina Ermagan, Hugo Latapie, Chris Cassar, John Evans, Fabio Maino, Jean Walrand, and Albert Cabellos. Knowledge-Defined Networking. *SIGCOMM Comput. Commun. Rev.*, 47(3):2–10, September 2017.

[38] Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. Unleashing the Potential of Data-Driven Networking. In *Communication Systems and Networks - 9th International Conference, COMSNETS 2017, Bengaluru, India, January 4-8, 2017, Revised Selected Papers and Invited Papers*, pages 110–126, 2017.

[39] Donghwoon Kwon, Hyunjoo Kim, Jinoh Kim, Sang C. Suh, Ikkyun Kim, and Kuinam J. Kim. A survey of deep learning-based network anomaly detection. *Cluster Computing*, pages 1–13, 2017.

[40] Ugo Fiore, Francesco Palmieri, Aniello Castiglione, and Alfredo De Santis. Network anomaly detection with the restricted boltzmann machine. *Neurocomput.*, 122:13–23, December 2013.

[41] Muhamad Erza Aminanto and Kwangjo Kim. Detecting impersonation attack in wifi networks using deep learning approach. In Dooho Choi and Sylvain Guilley, editors, *Information Security Applications*, pages 136–147, Cham, 2017. Springer International Publishing.

[42] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. Malware traffic classification using convolutional neural network for representation learning. In *2017 International Conference on Information Networking (ICOIN)*, pages 712–717, Jan 2017.

[43] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In

2017 IEEE International Conference on Intelligence and Security Informatics (ISI), pages 43–48, July 2017.

[44] Mohammad Lotfollahi, Ramin Shirali Hossein Zade, Mahdi Jafari Siavoshani, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *CoRR*, abs/1709.02656, 2017.

[45] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access*, 5:18042–18050, 2017.

[46] Zhanyi Wang. The applications of deep learning on traffic identification. In *Black Hat USA, Las Vegas*, 2015.

[47] Mehdi Mohammadi, Ala I. Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. Deep learning for iot big data and streaming analytics: A survey. *CoRR*, abs/1712.04301, 2017.

[48] S. Hettich and S. D. Bay. The UCI KDD archive, 1999.

[49] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, CISDA'09, pages 53–58, Piscataway, NJ, USA, 2009. IEEE Press.

[50] John McHugh. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, November 2000.

[51] Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 536–543, New York, NY, USA, 2008. ACM.

[52] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Comput. Secur.*, 31(3):357–374, May 2012.

[53] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Stefanos Gritzalis. Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset. *IEEE Communications Surveys & Tutorials*, 18:184–208, 2016.

[54] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of encrypted and vpn traffic using time-related features. In *ICISSP*, 2016.

[55] Luca Deri et al. ntop – high performance network monitoring solutions based on open source and commodity hardware. `https://www.ntop.org/`. Accessed: 2018-11.

[56] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[57] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[58] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[59] Sebastian García et al. Stratosphere ips. `https://www.stratosphereips.org/`.

[60] S. García, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *Comput. Secur.*, 45:100–123, September 2014.

[61] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *SIGCOMM Comput. Commun. Rev.*, 36(2):23–26, April 2006.

[62] Xiaming Chen. pkt2flow: A simple utility to classify packets into flows. `https://github.com/caesar0301/pkt2flow`.

[63] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. 2017.

[64] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[65] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6b: A bag of tricks for mini-batch gradient descent. Coursera: Neural Networks for Machine Learning, 2012.

[66] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[67] François Chollet et al. Keras. `https://keras.io`, 2015.

[68] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda

Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[69] Pedro Casas, Alessandro D'Alconzo, Tanja Zseby, and Marco Mellia. Big-dama: Big data analytics for network traffic monitoring and analysis. In *Proceedings of the 2016 Workshop on Fostering Latin-American Research in Data Communication Networks*, LANCOMM '16, pages 1–3, New York, NY, USA, 2016. ACM.

[70] Pedro Casas, Alessandro D'Alconzo, Giuseppe Settanni, Pierdomenico Fiadino, and Florian Skopik. Poster: (semi)-supervised machine learning approaches for network security in high-dimensional network data. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1805–1807, New York, NY, USA, 2016. ACM.

[71] Gonzalo Marín, Pedro Casas, and Germán Capdehourat. Deep in the Dark – Deep Learning-based Malware Traffic Detection Without Expert Knowledge. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy Workshops (SPW), 2nd Workshop on Deep Learning and Security, San Francisco, CA, USA*, IEEE Security and Privacy, 2019.

[72] Gonzalo Marín, Pedro Casas, and Germán Capdehourat. RawPower: Deep Learning Based Anomaly Detection from Raw Network Traffic Measurements. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters, Demos and Student Research Competition*, ACM SIGCOMM 2018, pages 75–77, New York, NY, USA, 2018. ACM. **2nd Place SRC Undergraduate Competition.**

[73] Gonzalo Marín, Pedro Casas, and Germán Capdehourat. DeepSec Meets RawPower – Deep Learning for Detection of Network Attacks Using Raw Representations. *ACM SIGMETRICS Performance Evaluation Review*, 46(3):147–150, January 2019.

[74] Pedro Casas, Gonzalo Marín, Germán Capdehourat, and Maciej Korczynski. MLSEC – Benchmarking Shallow and Deep Machine Learning Models for Network Security. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy Workshops (SPW), 2019 International Workshop on Traffic Measurements for Cybersecurity (WTMC), San Francisco, CA, USA*, IEEE Security and Privacy, 2019.

List of Tables

4.1 Summary of the distribution of payload size of the packets (n) in bytes, for malware and normal captures. 23
4.2 Summary of the distribution of the number of packets in the flows. Most flows are composed by a number of packets in the range ρ_p. The proportion of flows that fall within this range is represented by the percentage (%). 24
4.3 Summary of the distribution of the mean payload size of the packets in the flows. Most flows have a $\overline{p_f}$ in the range ρ_b. The proportion of flows that fall within this range is represented by the percentage (%). 25
4.4 Parameters selection for building the input representation for training the deep learning models. 25

5.1 *Raw Packets* deep learning model summary. 29
5.2 *Raw Packets* deep learning hyperparameters summary. 30
5.3 *Raw Flows* deep learning model summary. 31
5.4 *Raw Flows* deep learning hyperparameters summary. 32

6.1 Protocols and attacks performed by different kinds of botnets. . . . 41
6.2 Performance metrics for the deep learning model in the malware classification task. 42

List of Figures

2.1 A Venn diagram showing the relationship between AI, machine learning, representation learning and deep learning [3]. 5

2.2 Mathematical model of a single neural network unit. 7

2.3 Commonly used activation functions for neural networks [24]. . . . 8

2.4 A 3-layer neural network with 3 inputs, 2 hidden layers of 4 units each and one output layer. 8

2.5 The components of a typical convolutional neural network layer. . 9

2.6 Max pooling operation of size 4 applied to one-dimensional data. . 9

2.7 An unrolled recurrent neural network. 10

2.8 Differences between the internal modules of standard RNNs and LSTMs. 11

2.9 Ensemble of sub-networks trained as a consequence of using Dropout. Figure taken from the original paper [28]. 12

4.1 Different input representations for the deep learning models. 18

4.2 Distribution of the payload size in the packets for malware captures (log scale). 20

4.3 Packets and mean payload size distributions of TCP flows for malware captures. 21

4.4 Packets and mean payload size distributions of UDP flows for malware captures. 21

4.5 Distribution of the payload size in the packets for normal captures. 22

4.6 Packets and mean payload size distributions of TCP flows for normal captures. 22

4.7 Packets and mean payload size distributions of UDP flows for normal captures. 23

5.1 Deep learning architecture for *Raw Packets* representation. 28

5.2 Deep learning architecture for *Raw Flows* representation. 31

5.3 Overfit small subset of data to check model capacity. Example for *Raw Packets* configuration. 36

6.1 Learning process (loss and accuracy evolution after each epoch) and ROC curve for the *Raw Packets* representation. 39

6.2 Learning process (loss and accuracy evolution after each epoch) and ROC curve for the *Raw Flows* representation. 39
6.3 Deep learning architecture for multiclass classification of malware traffic, using *Raw Packets* representation. 41
6.4 Evolution of the learning process (loss and accuracy after each epoch) for the multiclass malware classification problem. 41
6.5 Normalized confusion matrices (showing percentage values) for both random forest and deep learning models. 42