Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Fingerprinting Mobile Application with Deep Learning

Bachelor's Thesis

In partial fulfilment of the requirements for the degree in Telecomunications Systems Engineering

Author: Albert Pou Marti
Advisor: Marc Juarez
Advisor: Verónica Vilaplana

Universitat Politècnica de Catalunya (UPC)
Katholic University of Leuven (KUL)
May 2019

Abstract

Nowadays, Traffic Network Classification is having a high impact on various studies such as firewalls, intrusion detection systems or status reports and Quality of Service systems. The success of the results is due to the evolution of Deep Learning in this area of study during the past years.

In this thesis, a 1D-CNN&LSTM model is designed that, through the encrypted flows coming from a smart-phone, can discover from which applications come from and thus showing is possible to attack the privacy of the people.

Based on a study with 73 different apps, 83% accuracy is achieved in the classification of traces, indicating which are the most vulnerable. To complete the study, applications that provide more sensitive information of the user are selected and which have more samples, and an individual examination is carried out.

Resum

Avui en dia, la Classificació de Trànsit de dades està tenint un gran impacte en diversos estudis com ara tallafocs, sistemes de detecció d'intrusos o informes d'estat i sistemes de Qualitat del Servei. L'èxit dels resultats es deu a l'evolució de l'Aprenentatge Profund en aquesta àrea d'estudi durant els darrers anys.

En aquest tesi, es dissenya un model 1D-CNN&LSTM que, mitjançant els fluxos encriptats procedents d'un telèfon intel·ligent, pot descubrir de quina aplicació provenen i així mostrar que és possible atacar la privacitat de la gent.

Basant-se en un estudi amb 73 aplicacions diferents, s'aconsegueix un 83% d'encert en la classificació de traces, indicant quines són les més vulnarables. Per completar l'estudi, es seleccionen les aplicacions que aporten informació més sensible de l'usuari i també de les que es tenen més mostres, i es realiza un estudi individual.

Resumen

Hoy en día, la Clasificación de Tráfico de datos está teniendo un gran impacto en diversos estudios como cortafuegos, sistemas de detección de intrusos o informes de estado y sistemas de Calidad del Servicio. El éxito de los resultados se debe a la evolución del Aprendizaje Profundo en esta área de estudio durante los últimos años.

En este tesis, se diseña un modelo 1D-CNN&LSTM que, mediante los flujos encriptados procedentes de un teléfono inteligente, puede descubrir de qué aplicació provienen y así mostrar que es possible atacar la privacidad de la gente.

Basándose en un estudio con 73 aplicaciones diferentes, se consigue un 83% de acierto en la clasificación de trazas, indicando cuáles son las más vulnarables. Para completar el estudio, se seleccionan las aplicaciones que aportan información más sensible del usuario y también de las que se tiene más muestras, y se realiza un estudio individual.

Acknowledgements

First of all, I would like to thank my supervisor from KU Leuven, Marc Juarez, to accept me as an Erasmus student and offer me the project. I also appreciate the information taught to me as well as all the proposals and the collection of data that he has given to me. Without him, it would not have been possible to carry out the project.

Secondly, I would like to acknowledge my UPC supervisor, Verónica Vilaplana, for both the attention and the advice given.

I would also like to appreciate the institutions of the UPC and the KU Leuven to offering me the experience of going to carry out the project abroad.

Last but not least, I would like to thank all the Erasmus colleagues I have met and have made me live an unforgettable experience, as well as encouraging me to work. And of course thanks to my family for the support given.

Revision History and Approval Record

Revision	Date	Purpose
0	03/02/2019	Document creation
1	10/04/2019	Document revision
2	08/05/2019	Document revision
3	11/05/2019	Document approval

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Albert Pou Martí	albertpou20@gmail.com
Marc Juarez	marc.juarezmiro@esat.kuleuven.be
Verónica Vilaplana	veronica.vilaplana@upc.edu

Written by:		Reviewed and approved by:	
Date	05/05/2019	**Date**	11/05/2019
Name	Albert Pou	**Name**	Verónica Vilaplana
Position	Project Author	**Position**	Project Supervisor

Contents

List of Figures

List of Tables

List of Abbreviations

ADAM	Adaptative Moment Estimation
ANN	Artificial Neural Network
BP	Back-Propagation
CNN	Convolutional Deep Neural Networks
DL	Deep Learning
DNN	Deep Neural Network
DPI	Deep Packet Inspection
ET	Encrypted Traffic
FN	False Negative
FP	False Positive
LTSM	Long Short-Term Memory
ML	Machine Learning
MSE	Mean Square Error
NN	Neural Network
pdf	Probability Distribution Function
SVC	Support Vector Machines
TC	Traffic Classification
TN	True Positive
TP	True Positive

Chapter 1

Introduction

In this chapter, is presented the statement of the purpose of this project (1.1 which provides a comprehensive overview of the project. Then, its requirements and specifications (1.2) are detailed. Finally, we mention the work planning (1.4), showing the general project's organization and deadlines and the incidents we have encountered (1.5) and how they have modified the initial plan.

1.1 Statement of Purpose

Nowadays, the vast majority of mobile applications are encrypted in order to protect user information and prevent any attack on privacy. If someone knows the applications that a user uses, it could reveal sensitive information such as their sexual orientation, political leaning or religion.

However, several articles have shown that encryption protects trivial information but is not enough. Simply interacting with the interface of an application produces a unique package sequence respect other applications. Consequently, it has been related that classifiers based on Machine Learning are very appropriate to classify Traffic Encryption (ET). On the other hand, the exit of the results depends on getting handcrafted (domain-expert driven) features, and this process requires much time while mobile traffic is constantly evolving.

Accordingly, because of the success of Neural Networks in fields such as image and speech recognition, is believed that the Deep Neural Network (DNN) can achieve high performance in the dynamic and challenging mobile TC context.

Along this project, is presented a DNN able to identify whether an app is installed or not in a mobile device by only looking at the encrypted network traffic the machine generates.

It ought to mention that during the work we use various abbreviations to entertain the explanations. A list with all the used ones can be found at the beginning of the document. The project has been carried out at the Department of Electrical Engineering (ESAT) in The Katholic University of Leuven (KUL).

1.2 Requirements and Specifications

The main requirement of the thesis is classifying network encrypted flows from different apps with the development of a DNN.

For this, we must learn the fundamentals and practice of deep learning in order to propose and train a Deep Learning Architecture.

The specifications have been decided during the project, taking into account the needs of the system and the resources available, and they will be explained throughout the work.

1.3 Methods and Procedures

As will be explained throughout the work, the system is based on two significant parts (see Figure 1.1). The first stage, once the dataset is collected, performs the pre-processing of the raw input data and convert them into features. The next block is the DNN, which classifies

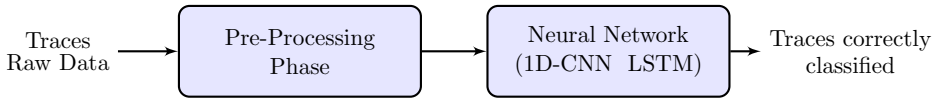


Figure 1.1: Block diagram of the system proposed

all the traces. Some code of the baseline model is open source thanks to the contribution of Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem & Wouter Joosen who provided the Website fingerprinting through Deep Learning implementation.

Additionally, the Dataset acquisition was made by Marc Juarez from imec-COSIC, ESAT, KU Leuven.

This project has been developed using Python 3 as the programming language. Also, Keras is used which is a deep learning framework that provides from TensorFlow with a high-level API to build and train deep learning models.

All developed models have been trained under the operating system Mac OS X.

1.4 Work Plan

1.4.1 Gantt Diagram

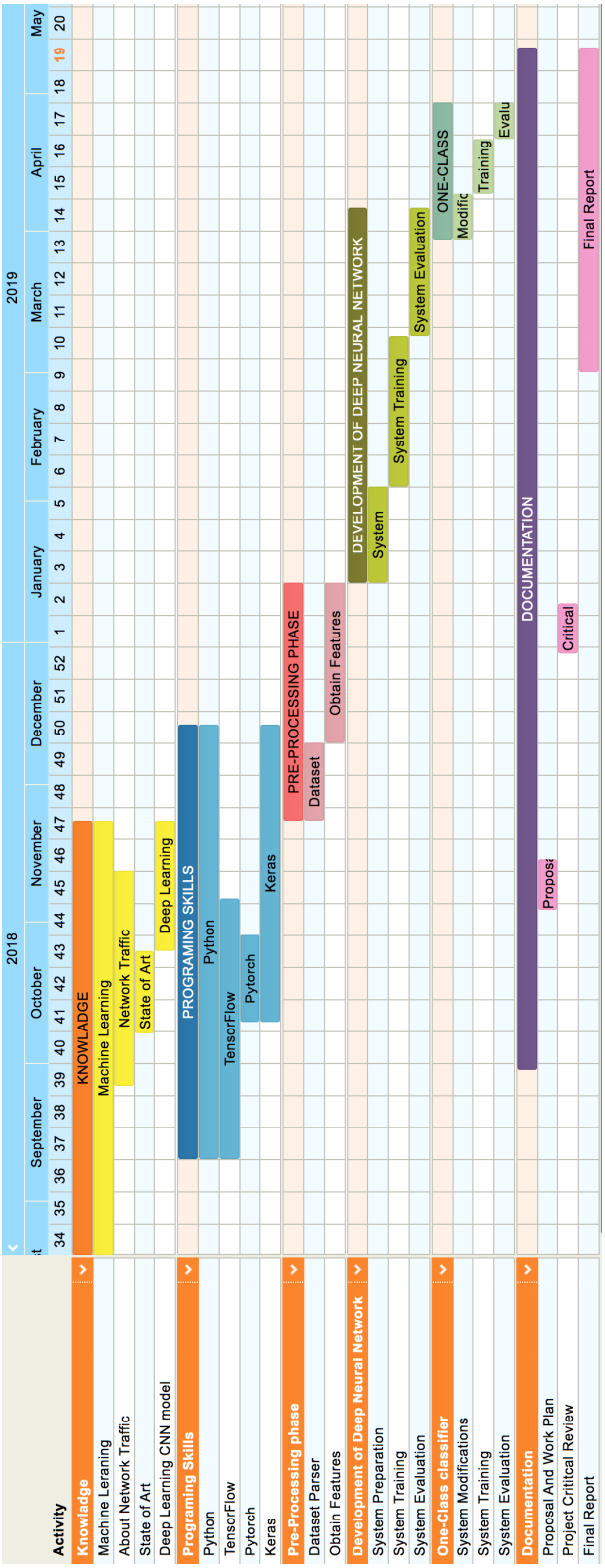


Figure 1.2: Gantt Diagram

1.4.2 Work Packages

Project: Knowledge	**WP ref: WP1**	
Major constituent: theoretical learning	Sheet 1 of 6	
Short description: Study of the generation of Neural Networks and flow-based features.	Planned start date: 20/08/2018 Planned end date: 22/11/2018	
	Start event: Start Machine Learning Crash Course End event: Convolution Neural Network model learned	
Internal task T1: Machine Learning Internal task T2: Network Traffic Internal task T3: State of the art Internal task T4: Deep Learning CNN model	Deliveries:	Dates:

Project: Programming skills	**WP ref: WP2**	
Major constituent: programming learning	Sheet 2 of 6	
Short description: Learn to work with Machine Learning Python libraries.	Planned start date: 10/09/2018 Planned end date: 13/12/2018	
	Start event: Machine Learning Crash Course chapter 4 End event: Keras knowledge acquired	
Internal task T1: Python Internal task T2: TensorFlow Internal task T3: Pytorch Internal task T4: Kerash	Deliveries:	Dates:

Project: Pre-Processing phase	**WP ref: WP3**	
Major constituent: convert the data raw into features	Sheet 3 of 6	
Short description: Pre-processing of the raw input data and convert them into features.	Planned start date: 23/11/2018 Planned end date: 13/01/2019	
	Start event: Dataset parser End event: Obtain Features	
Baseline task B1: Parser the dataset Baseline task B2: Obtain features	Deliveries: Model Flow-based features	Dates: 13/01/2019

Project: Development of Deep Neural Network	**WP ref: WP**	
Major constituent: development of the Neural Network of the system	Sheet 4 of 6	
Short description: Apply the learned programming skills and the analysis of the traffic network to develop a NN.	Planned start date: 14/01/2019 Planned end date:05/04/2019	
	Start event: Database preparation End event: System evaluation	
Baseline task B1: System preparation Baseline task B2: System training Baseline task B3: System evaluation	Deliveries: Neural Network implementation System evaluation report	Dates: 03/02/2019 05/04/2019

Project: One-Class classifier	**WP ref: WP5**	
Major constituent: one-class study simulating an open-world	Sheet 5 of 6	
Short description: Use the model to study the accuracy of a concrete app in open-world assumption.	Planned start date: 30/03/2019 Planned end date:28/04/2019	
	Start event: System modifications End event: System evaluation	
Baseline task B1: System modification Baseline task B2: System training Baseline task B3: System evaluation	Deliveries: System modifications System finally evaluation	Dates: 08/04/2019 28/04/2019

Project: Documentation	**WP ref: WP6**	
Major constituent: project reports deliveries	Sheet 6 of 6	
Short description: Write the different project reports and interesting information.	Planned start date: 03/09/2018 Planned end date: 08/03/2019	
	Start event: Machine Learning Crash Course End event: Final Report.	
Documentation task D1: Proposal and Workplan Documentation task D2: Project Critical Review Documentation task D3: Final Report	Deliveries: PW PCR FR	Dates 05/10/2018 30/11/2018 10/05/2019

1.5 Incidents and Modifications

In the middle of the project, it was proposed to use the same designed model but changing the data set. Instead of using captured data in a controlled environment (as it has been done and explained in section 3.1.1), catch them in the air.

In this case, the packages that are relayed, the acks and those that come from other applications cannot be filtered because there is no known where the data come from and, therefore, they realistically simulate the fact that a sniffer capture someone's traffic.

The problem that appeared a bug in the capture filter and the data were not captured correctly. To solve this, we tried to transform the data from the first data set by entering the relay packets and the acks. The results have not been useful, and this part of the thesis is annulled. This reason prompted us to change the objective and focus on the One-Class study (3.2.3).

Chapter 2

State of the Art

Traffic Classification (TC) is the process of associating (labelling) network traffic with specific applications. Starting from earlier port-based methods, to those based on payload inspection, approaches based on Machine Learning classifiers are deemed to be the most appropriate, especially in the context of Encrypted Traffic (ET) analysis. This chapter provides a brief introduction to the background of the project, discussing the classical methods of TC and its literature applied on the mobile context (2.1), as well as a brief review of the Deep Learning topic, its elements, techniques and terminology (2.2). Finally, it is made a brief mention of the state of the art techniques of deep learning applied in this particular task (2.3).

2.1 Traffic Classification

TC has a long-established relevance in numerous disciplines, backed by extensive scientific research. However, both the need and the difficulty of TC of mobile traffic have become very high nowadays.

State of the art in traffic classification has encountered a significant advance in the past few years, included in the number of papers and research groups centred on the topic.

The evolution of traffic classification technology (see fig.2.1) has created a complex panorama.

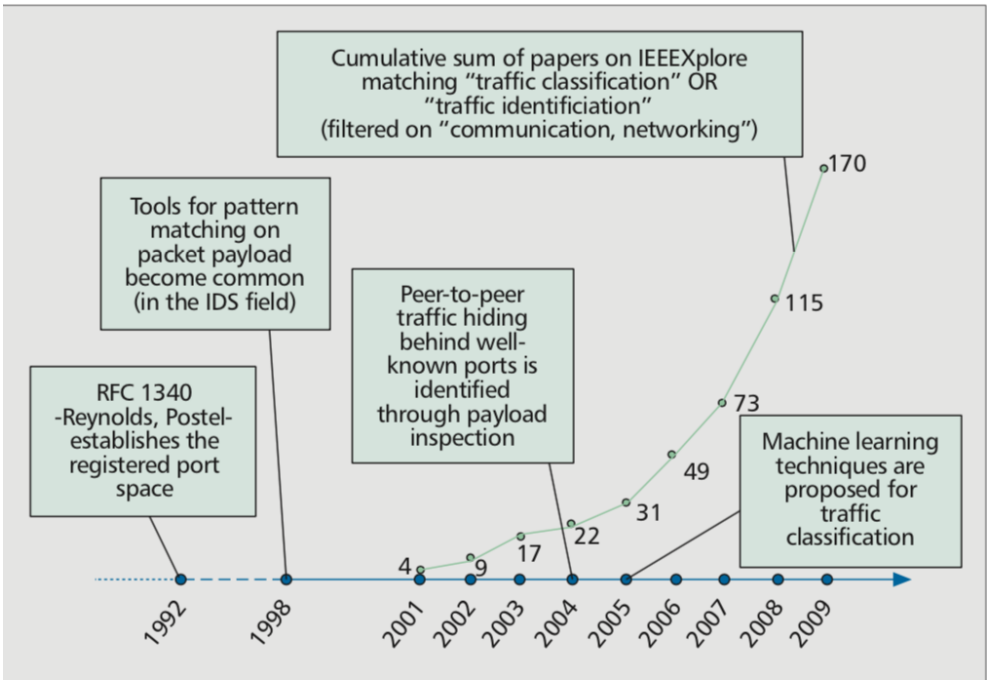


Figure 2.1: Evolution of approaches and literature in traffic classification (Xue et al., 2013)

In the next sections, is reviewed the possible techniques and is explained the decreasing security of the port-based approach by the strength of machine-learning strategies to DPI techniques (figure 2.2).

Although earlier results have been published on this topic, the traffic of mobile apps is a moving target for classifiers due to its dynamic evolution. Thus mobile TC constitutes an open and evolving research field.

2.1.1 Port-based Method

The port-based approach is the oldest, quickest and the easiest method for classifying network traffic packets and, because of this has been extremely used.

Several applications had fixed port numbers and, then, traffic belonging to these apps could be easily identified. Nevertheless, apps are increasingly complex, and the growth of reasons to hide traffic so that filters or blockages can be avoided have made traditional techniques antiquated (Xue et al., 2013).

2.1.2 Payload-based Method

The Deep Packet Inspection (DPI) techniques are based on inspecting the part of the packet even beyond the header, including the valuable part of the information, payload, and, finally, to obtain statistical data. It is based on the establishment of signatures created from said content, and that allow the characterization of the traffic by checking for each package or flow of whether or not it complies with the different signatures defined (Gil Delgado, 2015).

The problem is that with traffic encryption this information can easily be hidden and other classifications techniques have been sought that do not require payload examination.

2.1.3 Host Behavior-based Method

The host techniques are algorithms that classify traffic through non-rigorous methods. They are based on the knowledge and own experience of the investigator on the data traffic or even of determined knowledge of the network.

As explains in (Xue et al., 2013) One advantage of these methods is that they manage to classify traffic even when is encrypted, but the problem is that the host behaviour-based methods require that the number of specific streams for a particular type of traffic reach a certain threshold before it can be classified.

2.1.4 Flow Feature-based Method

Using the ML techniques, it has been shown that with the algorithms in the pattern recognition field gets good results. These systems learn from empirical data to automatically associate objects with their corresponding classes. Within this technique, there are two branches:

If the researcher defines the classes, and the samples are introduced to the system previously labelled by categories, this is called supervised algorithm. On the other hand, if it is the same model the one that identifies and labels the samples is about unsupervised algorithm.

While supervising ML approaches have achieved comparable results to DPI, unsupervised ML techniques are a promising way to cope with the constant changes in network traffic, as new applications emerge faster than it might be possible to identify unique signatures and train machine-learning classifiers.

The performance of the classifiers does not only depend on the type of algorithm used (i.e. decision tree, Bayesian techniques, neural networks). An important part for the proper functioning of the model is to choose the classification features, which are the data used to describe each object to the ML system. Features include common flow properties (i.e. per-flow duration and volume, mean packet size) as well as more specific properties, such as sizes and inter-packet times of the first n packets of a flow, or entropy of byte distribution in packet headers or payload.

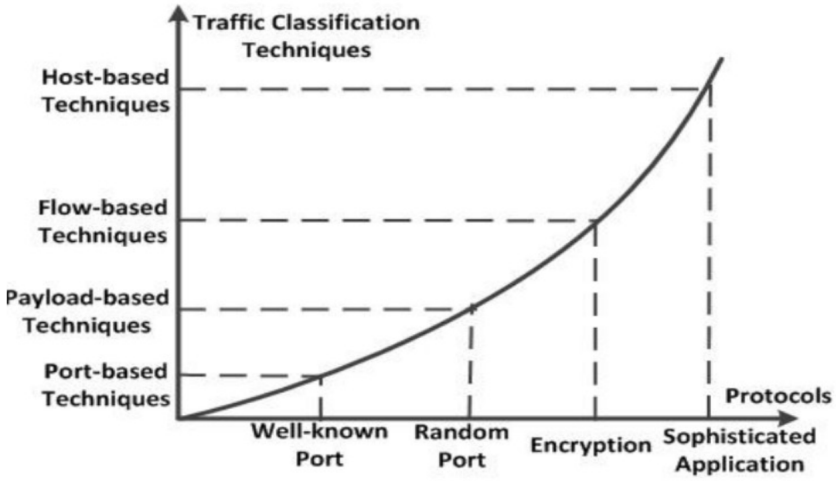


Figure 2.2: TC Techniques evolution vs Protocols (Santos Lebrato, 2018)

2.2 Deep Learning

Deep learning is based on machine learning and is nothing more than a family of algorithms whose purpose is to simulate the behaviour that our brain performs to recognize images, words or sounds. The significant advance of this algorithm with respect to Machine Learning is based on the fact that the system can train itself to find coherence in random data. From this, it improves the performance with high levels of precision, achieving systems that surpass the humans themselves in tasks of classification and detection in images.

In the human brain, with a stimulus, certain neurons are activated, evaluate the information they have received, react and communicate with other neurons. In subsequent stimuli, they add new data to those they already know, evaluate the result of the previous actions and correct their operation to have the best possible reaction The algorithms cited are used by artificial neural networks to act similar to the brain, but simplifying the set of neurons used in different layers. Thus, each group of neurons analyzes the input data, processes them and delivers them to another group of neurons in the form of data. The good results of this method are possible due to the number of connections between the different layers.

From a mathematical point of view, deep learning can be seen as a model that processes the information it receives in the input, coded as numbers, to pass it through a series of mathematical operations (layers of neurons) and obtain information as output encoded that can be adapted to carry out a specific desired function. In this way, Deep Learning works thanks to the architecture formed by the different layers and the routine that it uses to optimize said architecture.

2.2.1 Neural Network

A Network of Neurons (NN), commonly called Artificial Neural Network (ANN), is a mathematical model formed by a series of operations that, for an input vector x, offers a different output vector $f(x)$. Another way of viewing them, as already indicated in the previous section, is like a processor that receives incoming data encoded as numbers and makes a series of operations and produces outgoing information that is also coded as numbers.

There are three types of layers in which the neurons are grouped: (a) The input layers are the neurons that enter the input values in the network, so there are as many neurons as input data. No processing occurs in these neurons. (b) Hidden layers are those that do intermediate calculations of the network. Each hidden neuron contains a weight and a

parameter that are the elements that transform the input data. The number of hidden layers is variable. Finally (c) the output layers are neurons whose values correspond to the output states of the neural network. In this layer, each neuron also has associated weights and parameters.

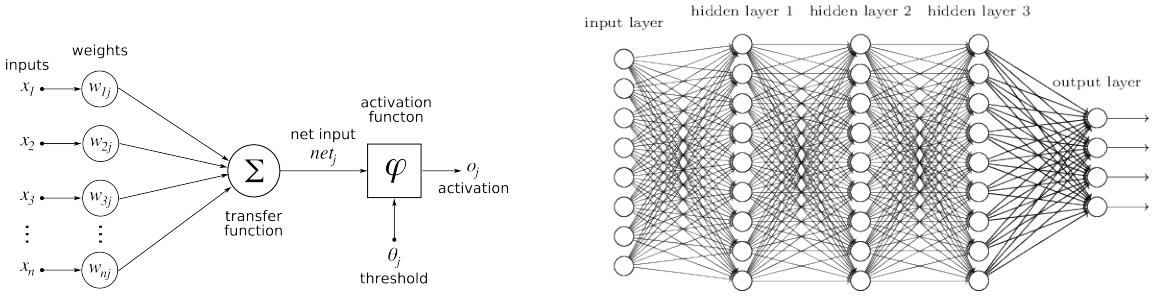


Figure 2.3: On the left, the basic unit of the NN, a neuron (a) (Wikipedia contributors, 2019) and on the right, a DNN (b). (Nielsen, 2015)

Figure 2.3 helps to understand the procedure. Neurons are organized in layers, where each one connects with all of the next layers. Each connection has a weight associated, so the main operation is a multiplication between the value of the neuron and its outgoing link. In this way, the neurons of the successive layers receive the previous results and apply non-linear functions to produce a new result.

At this point, depending on the NN used, the functions will be of one type or another. In this thesis, the Softmax function is applied. The first it does is to squash the outputs of each unit between 0 and 1 and divide each output such that the total sum of the outputs is equal to 1. Then the output of the Softmax function is equivalent to a categorical probability distribution, i.e. tells the probability that any of the classes are true. Thus we want the desired category to have the highest score of all classes, which means it is the most likely category.

In the described architecture, the learning process of a NN consists, based on a series of examples in which the output value for the corresponding input is known, in induce the relation between inputs and outputs for unknown data. This process is called network training, and it can be seen as the process of adjusting the parameters of the network. Starting from a set of random weights, the learning process seeks to adapt the value of those that allow the system to achieve the desired behaviour. The learning process is an iterative process in which the solution is refined until reaching a satisfactory level of operation. Most of the training methods used in NN consist of defining an error function that measures the performance of the network based on the weights (see section 2.2.4). From it, the objective is to find a set of weights that minimize or maximize it, determining the optimal point of the NN.

After the training, the results are tested in a different data set. In this way, the ability of the model is proved on samples that the system have not seen during training.

Once it has been explained what a NN is, it can be said that the most common architecture is the DNN that refers to having multiple hidden layers in the network. This hierarchy increases the complexity and abstraction at each level and is the key that makes deep learning networks capable of handling large data sets.

Depending on the type of entry, different DNN architectures are used. For systems where the input is sequential, such as audio, images or traffic data, the CNN model is very used, which is explained in the following section.

2.2.2 Convolutional Neural Network

Convolutional Neural Networks (CNN) is a particular type of NN. In a classic NN, it is assumed that all inputs and outputs are independent of each other. For tasks that involve sequential inputs, such as text data, it is often better to use CNN. CNN has shown excellent learning ability in image understanding thanks to its method to extract critical features. Comparing with other Deep Learning architectures, two characteristics make CNN unique: (a) Locally connected layers, which means the neurons in the output of the layers are connected only with their local nearby input neurons instead of the entire input neurons in fully connected layers. (b) Pooling mechanism, which significantly reduces the number of parameters required to train the model while at the same time ensuring that the essential features are preserved.

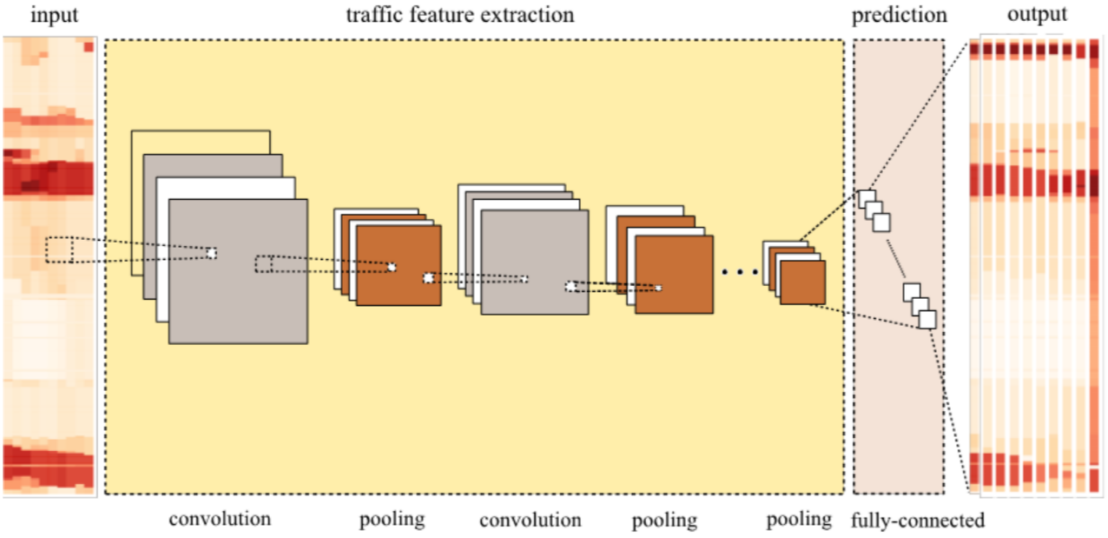


Figure 2.4: DL architecture of the global CNN model(Ma et al., 2017)

Figure 2.4 deals with the structure of the CNN model with its four main sections, which are the inputs, the extraction of features, the prediction and the outputs. Each part is detailed below.

To begin with, the input model is a matrix $x^i = \{[m_0, m_1, x_2, ..., m_N - 1]\}$ where i is the index of the sample, N the length and m_i columns of data, in the case of TC, representing the flows.

Second, feature extraction is the combination of the convolution and pooling layers. It is the core part of the CNN model.

Being ${x_l}^j$ the input, ${o_l}^j$ the output, $({W_l}^j, {b_l}^j)$ the parameters of the lth layer (where L is the depth of CNN) and j the index channel considering the multiple convolution filters in the convolution layer. cl is the number of convolutions filters in the lth layer. Then, the output after the first convolution and pooling layers is

$$ {o_l}^j = pool(\sigma({W_l}^j \cdot {x_l}^j + {b_l}^j)), j \in [1, c_l] \tag{2.1} $$

where σ is the activation function, which is discussed below. The output in the $lth(l1, l = 1..L)$ convolution and pooling layers can be written as

$$ {o_l}^j = pool(\sigma(\sum_{k=1}^{c_{l-1}} {W_l}^j \cdot {x_l}^k + {b_l}^j)), j \in [1, c_{l-1}] \tag{2.2} $$

The number of layers in the cnn are hundreds, which means hundreds of features can be learned. The CNN algorithm transforms the input model into deep features through these layers.

In the prediction, the features that have been learned are concatenated in the dense vector which only contains the most valuable features. The dense vector is known as

$$o_L{}^{flatten} = flatten([o_L^1, o_L^2, ..., o_L^j]), j = c_L, \tag{2.3}$$

where flatten is the concatenating procedure explained before.

Finally, the vector transforms as model outputs through a fully connected layer. Then, the model output is written as:

$$\hat{a} = W_f \cdot o_L^{flatten} + b_f = W_f(flatten(pool(\sigma(\sum_{k=1}^{c_{l-1}} W_l{}^j \cdot x_l{}^k + b_L{}^j)))) + b_f \tag{2.4}$$

where W_f and b_f are parameters that came from the fully connected layer and $\hat{a}$ are the predicted output.

Before present the explicit layers, it should be noted that an activation function activates each layer. It transforms the output into a manageable and scaled data range, which is beneficial for modelling training. Another advantage of using the activation function is that the combination of the activation function through the layers can simulate very complex non-linear functions that make the CNN powerful enough to handle highly complex data. Relu function can be formulated as is defined as

$$g_1(x) = \begin{cases} x, & if x > 0 \\ 0, & otherwise \end{cases} \tag{2.5}$$

Convolution layers differ from traditional NN because each input neuron is connected to each output neuron, and the network is fully connected (fully connected layer). CNN uses convolution filters on the layer of its input and obtains local connections where only the local input neurons are connected to the output neuron (convolution layer). Many of the filters are applied to the input, and the results are combined in each of the layers. A filter can take a function out of the input layer and, therefore, many filters can extract many functions. Those obtained functions are combined even more to extract a higher level and more abstract functions. This process forms the composition of the CNN model, which means that each filter composes a local route from the lower level to higher grade characteristics. After a convolution filter W_l^r is applied to the input, the output results are

$$a_{conv} = \sum_{e=1}^{m} \sum_{f=1}^{n} ((W_l{}^r)_{ef} \cdot d_e f), \tag{2.6}$$

where both m and n are the two dimensions of the filter, d_{ef} and $(W_l{}^r)_{ef}$ the data value of the input matrix and the coefficient of the convolution filter respectively at e and f positions and finally a_{conv} is the output.

Pooling layers are designed to reduce the sample and add data since they only extract outgoing numbers from the specific area. These layers ensure that CNN is locally invariant, which means that CNN can always extract the same feature from the input, regardless of the changes, rotations or scales of the features ((LeCun et al., 1995)). Based on the above facts, the grouping layers can not only reduce the CNN network scale but also identify the most outstanding features of the input layers. Taking the maximum operation as an example, the pooling layer can be formulated as

$$a_{pool} = max(d_{ef}), e \in [1..p], f \in [1..q], \tag{2.7}$$

where d_ef is already defined, p and q are two dimensions of pooling window size and a_{pool} the pooling output.

Finally, the fully-connected layer is connected to the output of the last convolutional layer of the network. It is the core of the classification stage. It requires an intermediate adaptation that converts the data to a vector, being that the Flatten layer. Finally, the Softmax layer is responsible for associating a label with the input image by applying a probability of likeness.

2.2.3 Long Short Term Memory

As the project model will use a Long Short Term Memory (LSTM) layer, its operation will be introduced.

LSTMs are a particular type of Recurrent Networks. The main characteristic of RN is that information can persist by introducing loops in the network diagram, so that, basically, they can "remember" previous states and use this information to decide which one will be next. This feature makes them very suitable for handling time series. LSTMs can learn long dependencies, so it could be said that they have a longer-term "memory".

2.2.4 Optimization

As explained above, the NN learns from examples where each vector of features x represents a target t. To training the system, every time an input is passed through the network, the w weights vector is best adapted to the category is wanted to predict t. This estimate is done with a descending gradient.

This algorithm minimizes the error function $J(w)$ so that the difference between the output of the network and the prediction is minimal (which is the ultimate goal of the system). This error or cost or function must be defined by the MSE difference, the cross-entropy between two probability distributions (Golik et al., 2013), Negative Logarithmic Likelihood, and so on.

In order to adjust the vector of weights, it is the gradient vector which calculates how modifies the error in case it distributes one way or another. Once fixed in the best way, the vector of pesos is modified. This process is called a step. To the speed at which the weight adjusts and goes to the minimum is called learning rate η. In the next level, the weight vector is updated in the opposite direction to redirect it to the minimum of the error function and is performed as this way

$$\omega = \omega - \eta \bigtriangledown_\omega J(\omega) \tag{2.8}$$

The loss function averaged over all the training examples, can be seen as a landscape in the high-dimensional space of weight values, as illustrated in the Figure 2.5. The negative gradient vector indicates the direction of most descendent step in the landscape, taking it closer to a minimum, where the output error is low on average.

Before explaining the distressing gradient descent methods, some theoretical concepts are explained. Primary, is known as an epoch the pass of the whole training data set through the network. However, not all the complete data set is passed into the neural net at once, since it is divided into batches or sets. Consequently, we define the batch size as the number of training examples in a pass through the network. Then the number of iterations is the number of passes needed to complete one epoch, each pass using batch size instances. For example, if you have 60 training examples, and your batch size is 20, then it will take three iterations to complete one epoch.

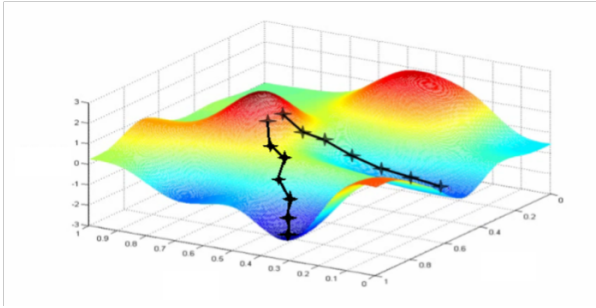


Figure 2.5: Gradient descent path (each black vector represent the movement in one step) over the loss function landscape, averaged over all the training examples. (Andrew Ng, 2018)

There are two principal alternatives to measure the gradient, and it depends on the data used: (a) the mini-batch gradient descent provides an update of the loss function for mini-batch of n-examples. With them is managed to decrease the variance in each update controlling the speed of the algorithm. As a consequence, the dimension of the mini-batch is one more hyper-parameter to keep in mind. This method is the one used in the design. On the other hand, (b) the stochastic gradient descent makes the estimation of the gradient updating the loss function for each example; in other words, a batch size of one. It is a fast algorithm, but it has much variance in the updates (see Eq.(2.8)).

There are several gradient descent algorithms; the most famous ones are Momentum, Nesterov, Adagrad, Adadelta, RMSprop, and Adam. RMSprop (Vitaly Bushaev, 2018), is the one is used in the thesis. Gradients of very complex functions like NN tend to either disappear or collapse as the energy is propagated through the task. Furthermore, the effect has a cumulative nature; the more complicated is the function, the problem becomes critical. Rmsprop has an ingenious way to deal with the problem. It uses a moving average of squared gradients to normalize the gradient itself. That has an effect of adjusting the step size; decrease the step for the large gradient to avoid exploding, and increase the step for the small gradient to prevent fading.

2.3 Deep Neural Network-based Traffic Classification

DL has shown its smooth functioning in the area of Pattern Recognition (PR). Especially, the fields where it is more used are the characterization of text, image classification and speech recognition. From here, and because the methods of TC have become obsolete by the security measures taken, it was beginning to investigate the similarity between the traffic data and the examples of PR. Since images and reports can be learned feature and classified very well by DL, it's reasonable to expect excellent performance in traffic classification and it began to treat the bytes of the data flows with DL.

Several researchers have started to classify the traffic of web pages due to the extensive network traffic that exists today. (Wang et al., 2017b) proposes a system based on the 2D-CNN model to classify malware traffic with an accuracy of 89%. Based on this study, (Wang et al., 2017a) adopts a 1D-CNN model for TC and compares them. Testing four different types of data sets, this second model always gets better results.

It has been later when it has been seen the need to explore the terrain of smart-phones, and it started the use of DL to classify mobile apps. Many papers have been created to find which is the best model and algorithm for categorizing the ET from mobile applications. The ones that have the most relevance are below.

In (Stöber et al., 2013) using the Support Vector Clasifier (SVC) and K-Nearest Neighbors model, and considering only statistical features that learn traffic through background

activities, obtain more than 90% accuracy.

Later, (Taylor et al., 2016) made a breakthrough in the results of the identification of mobile applications. The data is captured through an Android device that automatically runs 110 applications. Later the traces are pre-processed to eliminate the noise that could exist and uses the models SVC and Random Forest (RF) to evaluate the system. The results obtained are that it identifies 99% of applications and classifies 86.9% of them. This research has created a before and after in State of the art of the classification of mobile ET and several studies and reports have been done using the same methods.

Finally, the [Network traffic classifier with convolutional and recurrent neural networks for Internet of Things] introduces different combinations between CNN and LSTM models according to the number of dimensions. The evaluation is made on a data set with 266,160 network flows and 108 services different, and the best result is obtained with the combination of 2D-CNN and LSTM achieving an accuracy of 96.32%.

Chapter 3

Methodology

As a reminder, the main objective of the project is to identify the different applications that have generated traffic through its captured flows. From here, some studies are carried out as the effect of varying the number of packages and the number of traces to train to see how they modify the results.

This section begins by analyzing the entire process of collecting (3.1.1), preparing and pre-processing the data to achieve data traffic in a format that is understandable for the algorithm (3.1.2). Although how to structure the data seems a trivial process, when it comes to working with DL, it is the most critical and challenging part to implement. One step of this process consists in extract the features so is explained which and why are chosen to do the training part (3.1.3). Then, the paper continues with the explanation of the architecture's system (3.2.1) and the parameters that have been chosen to achieve the best performance (3.2.2). Finally, a second study is introduced which wants to verify, in the most realistic way, the performance of the same model with a single and concrete app, simulating then, an open world learning(3.2.3).

3.1 Data Preparation

3.1.1 Data Collection

An indispensable requirement when using DL is to have a representative data set. This one must contain numerous and abundant samples of each class. In (Aceto et al., 2017), is shown that the accuracy decreases by 26% when the server is different between the training set and test set. In a real environment, it would not be necessary to obtain traffic captures because it would come from an already obtained network traffic. In the development of this project, it has been necessary to generate the traffic of each app separately to be able to associate each flow to a specific app and subsequently check the reliability of the classification obtained.

However, in the case of traffic data, the large number of existing apps makes impossible to work with all of them and a closed world learning must be simulated. This means that this paper studies the hypothetical case that there exist only the apps that are shown in the project. 117 Android apps have been captured (see A) to perform the analysis and to represent the whole range of applications that exist(i.e., games, meetings, social, shops). Some of them are sensitive and reveal information about the person who uses it (i.e. if he/she is single).

As indicated in the section 1.3 The data collection has been done by my supervisor Marc Juarez from the Katholic University of Leuven. However, the process that has followed is detailed because it is an important part to understand the system well.

The first step is to create traffic captures corresponding to applications through the tcpdump traffic analyzer and the NoSmoke2.0 crawler. The explanation of both tools is out of the reach of this study.

Using the tcpdump tool, computer captures the network traffic (only IPs that coincides with that smart-phone) and export details of the packets as time, source and destination address, ports, packet size, and protocol. Simultaneously, NoSmoke2.0 crawler is used to produce traffic. It reads a script previously prepared with all the apps that must be installed and, randomly, it choose one to install and interact with the interface simulating human behaviour.

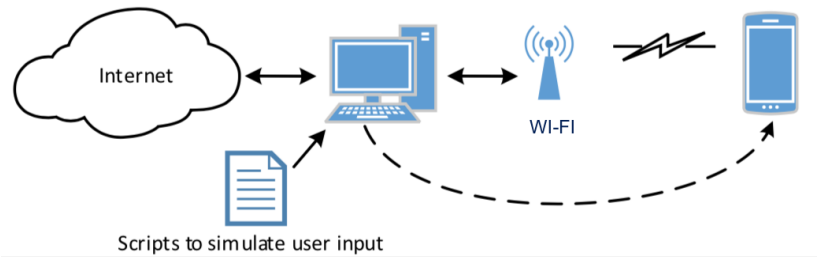


Figure 3.1: Data set acquisition process (Taylor et al., 2016)

For each service, a particular flow is associated. A network flow consists of a sequence of packages that are sent from the source and are directed to the same destination. For each flow, an independent capture file is created and contains exclusively traffic which belongs to the corresponding app. To achieve this, security measures have been taken such as using a private router, filtering the IP of the smart-phone and working in a controlled environment, since the capture point was made from the network access point. All these measures ensure no other source negatively affects.

This process performed by the crawler is repeated many times achieving a total of 117 apps and a set of 8197 traces (see table A)

Despite working in the right environment, it is not easy to distinguish and eliminate background traffic. The study [19] show that 70% of the mobile traffic is noise and only 30% is directly related to the interactions of the traffic user. Therefore, later on, some traces must be filtered.

As mentioned above, the acquisition of the files serves the model to learn a distinctive pattern. However, these data are saved with the extension ".pcap" (since tcpdump does by default) and encrypted language so they should be parsed to get traffic flow.

3.1.2 Data Pre-Processing

The pre-Processing phase is the procedure to clean and convert the raw traffic data (in this case .pcap extension as mentioned before) in the required input format for the model. This process affects the result significantly. Duplicate acks and messy or relay packets may change the traffic pattern of the apps. Some studies demonstrate improvements once they have eliminated these packages (Dubin et al., 2017) while others inform that there are no difference (Aceto et al., 2017). This is caused by the difference in the data set, the features used for the classification and the number of packages taken.

Once the flows of each application are available, the next step is to extract the parameters that interest us. Section 3.1.3 explains why the selected characteristics chosen to train the model are the length of the package and the direction (income or outgoing network).

After this, the representation of the traffic flow is a sequence of values (length of the package) positive (outgoing) or negative (incoming) forming a vector as shows the figure 3.2.

Not from all captured traces are extracted useful information; In order to have representative instances, the flows that have few packets (or directly empty), the relayed flows and the acks packets are removed. To finish the filtering and to avoid an imbalance in the number of instances of each app which would produce an oversampling.

On the other hand, to associate each flow with its application, there's a vector full of 0 except that position which corresponds on that app.

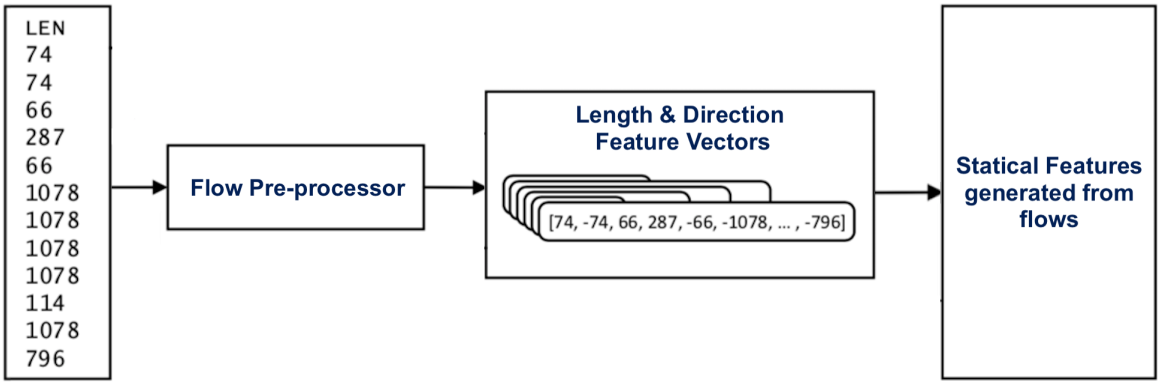


Figure 3.2: Data pre-processing process

3.1.3 Features

As mentioned in the State of the art (2.1, time series features can work both with encrypted and unencrypted traffic. On the other hand, the methods that use characteristics of the payload or information of the header use data of upper layers that are not available when working with ET. That is why, in this study, the only possibility of achieving a model with real accuracy is using time series features.

Each flow is associated with a particular service, and from each packet of the flow, six features can be extracted: source and destination port, packet length, transport protocol, inter-arrival time and direction of the packet.

In this project, only TCP packets are analyzed because, as (A. L. Narasimha Reddy) says, this protocol uses 90% of all internet traffic since it has an integrated mechanism to avoid congestion.

In the paper (Lopez-Martin et al., 2017) assess the influence of the set of features used in the detection process. They show how inter-arrival time dos not always give good results. The reason is that it can be distorted because it has a strong dependence on network conditions. For this reason, it has been decided to ignore this feature and, at the same time, saves computing memory and time.

Finally, the model will be trained extracting the length of TCP packet in bytes and the direction of each packet of each flow. When it refers to the direction, it only differs if the data is outgoing from the source or incoming. In the first case, the positive sign (+1) is added and the second case the negative sign (-1).

Besides, in case that an online classification is made (in real time), choosing a few features facilitates the great behaviour of the model. On the contrary, it would provoke a high latency, a high computational cost, and memory (Li and Moore, 2007).

3.2 CNN-LSTM Model

3.2.1 System Architecture

The choice of features and the size of input data set are very correlated with the model of the DL that must be chosen. CNN consists of a set of layers with learning parameters that, unlike other techniques, works fine when it comes to working with a high-dimension input which involves a large number of parameters in the hidden layer.

This project has been inspired by several studies dealing with encrypted traffic classification to build the most appropriate model. Specifically, the 1D-CNN model is used because the

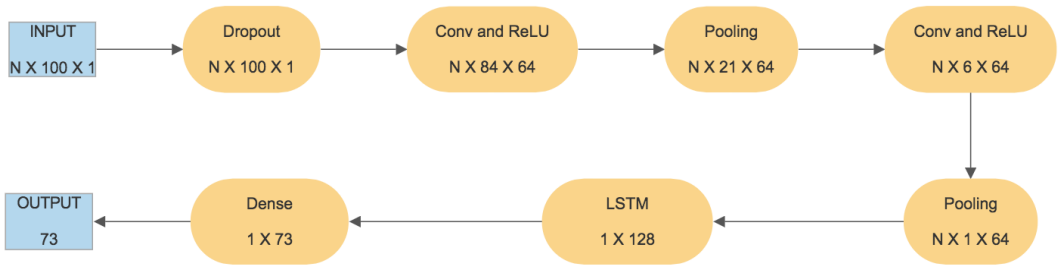


Figure 3.3: Architecture of the model's project

raw packet's data are of one dimension and the authors of (Vinayakumar et al., 2017) and (Wang et al., 2017a) show that in these cases it is better to work with 1D-CNN than with other CNN dimensions.

After studying and testing various possibilities, a model similar to the study (Lopez-Martin et al., 2017) has been chosen which combines the output of the convolutional layer with the layer LSTM. Here, the combination works as the CNN model extracts features while the LSTM layer is responsible for interpreting them (Brownlee). Finally, the architecture selected is a combination of 1D-CNN and LSTM as can be seen in figure 3.3 where N is the number of instances and the input 100 is the length of the flow. In the State of the art (2.2.2 and 2.2.3), the management of the CNN and LSTM algorithms has been explained to have a general idea. This section describes the architecture of the actual proposed model.

The first layer is the Dropout layer. The idea of the dropout layer is to "remove" a random set of neurons in that layer by setting them to zero. This fact, in some way, forces the network to be redundant. It means that the model must be able to provide the correct classification or output for a specific example even if some of the neurons are eliminated. An important note is that this layer is only used during training, and not during the test time.

Let x_i be each packet from a flow (represented by its length and direction). The input of the model is $X_{1:n}$ where the full flow with length n:

$$X_{1:n} = \left\{ x_1 \bigoplus x_2 \bigoplus x_3, ..., x_n \right\} \quad (3.1)$$

and $\bigoplus$ the symbol to concatenate vectors. The convolution operation introduces a random filter hR of length L which is applied to flow X_k and generates a new feature defined as:

$$C_k = f(h \cdot x_{k:k+L-1} + b) \quad (3.2)$$

In this case, bR is the bias and f is the activation function ReLU where its function is $f(x) = max(0, x)$. This filter is applied to all possible windows $\{x_{1:L}, x_{2:L+1}, ..., x_{n-L+1:n}\}$ generating the vector of feature c:

$$c = [c_1, c_2, ..., c_{n-L+1}] \quad (3.3)$$

with cR and where $N = n - L + 1$ is the number of characteristics that is extracted.

Is then when the operation max-pooling is applied in the feature vector C to select the maximum value $C = max\{c\}$ as a feature for the entry to the next layer.

In our model, there are two convolution layers and two max-pooling layers to extract the best characteristics. As said, this is the role of the CNN model. These features are passed to LSTM which transforms the vector sequence into a single vector of less size, containing information about the entire sequence. This vector is finally passed to the fully connected

Softmax layer with cross-entropy minimization as a cost function (explained in section 4.1) and RMSprop as optimizer (detailed in section 2.2.4)and its output is the probability of distribution of the input flow. The output and input parameters of each layer are shown in the table 3.1.

Layer	Operation	Activation shape	Parameters
1	dropout	100 x 1	0
2	conv + ReLU	85 x 64	1088
3	1D max pool	21x64	0
4	conv + ReLU	6x64	65600
5	1D max pool	1 x 64	0
6	LSTM	128	98816
7	softmax	73	9417

Table 3.1: Parameters per layer of the model

Note that both convolution filter and max-pooling are operations of one dimension. This is the essential feature of the 1D-CNN models.

The model also applies the criterion of Early Stopping, which prevents overfitting. This phenomenon appears when the model does not improve the prediction accuracy on validation data, although it improves the prediction accuracy on training data. When this happens, the model must stop training (Sarle, 1996). The Early Stopping method memorizes the data set validation losses. After each epoch, checks if the losses increased or remained unchanged. Finally, if correct and no sign of improvements is observed within a specific number of epochs, early stopping stops further model training.

3.2.2 CNN Hyper-parameters Tuning

When implementing a CNN model, two key factors need to be taken into account to achieve the best classification performance and, at the same time, enhance the capabilities to classify the unknown traffic: choose the hyper-parameters of the Deep Neural Network and the depth of CNN. The selection of hyper-parameters corresponds to the experience of experts. There are no general rules that can be applied directly. Therefore, the strategy followed has been to choose a representative sub-sample of all the set date and, by testing different parameters oriented to other similar studies, such as (LeCun et al., 1989) or (Krizhevsky et al., 2012), the best results were chosen:

To reduce the search space, the model uses the same parameters in each layer. Using the same kernel in all the entries helps the model to capture the most invariant features more quickly. Finally, the chosen settings are those shown in table 3.2.

Hyper-parameters	Space	Value
optimizer	RMSProp, SGD, Adam	RMSProp
learning rate	0.0005 ... 0.0025	0.008
batch size	8 .. 256	128
number of epochs	25 .. 200	124
number of layers	4 .. 9	7
input units	50 ... 300	100
dropout	0 .. 0.3	0.1
lstm	100..300	150
activation	tanh, ReLU	ReLU
kernels	16 .. 128	64
kernel size	2 .. 20	16
pool size	2.. 16	4

Table 3.2: Hyper-parameters to train the 1D-CNN-LSTM model

3.2.3 One-vs-Rest Study

The original study that has been performed makes the closed world assumption; The classes that the model has seen in the training are the same that have been seen when testing. A more realistic scenario is to expect unknown classes during the test phase. In this second study, it is wanted to check the accuracy of a single app against all other apps simulating the case that only interests if the user is using or not a specific app. The system uses the same model with some modifications that are explained in section (4.1). Binary classification is performed according to whether the test data corresponds or not to the selected app.

The main difference between the previous model is the labelling of the classes since they have grouped according to whether they belong to the interest app or not forming a model with only one output instead of the 73 that previously were.

Chapter 4

Evaluation and results

The present section shows the results obtained in the different studies that have been carried out during the project and the influence that had particular design decisions such as the number of packages selected and the number of flows per application. First, the experimental methodology is detailed as well as the used metrics (4.1). Then, a review of results obtained as well as an objective evaluation comparing the outcomes according to design changes (4.2) as well as a comparison of all the investigated classes (4.3). All these steps are subsequently repeated later in the study of One-vs-Rest (4.4) to finish the section.

4.1 Experimental Methodology and Evaluation Metrics

There are several approaches to train and test the accuracy of the model in supervised learning algorithms. The data set is divided into three groups: training, validation and test set. The first is the set of data with which the model trains and learns while the second is used to provide an unbiased evaluation of a fit model on the training data set while tuning model hyper-parameters. The test set is used after the training to evaluate the model with data that it has not seen before.

In some cases, the cost of setting aside a significant portion of the data set, like the holdout method requires is too high. As a solution, in these circumstances, a resampling-based technique such as cross-validation may be used instead. In our project, the k-fold validation method is used. This process consists in dividing the whole data set into k subsets. The model is trained from k times and each time, one of the subsets k is used as a test, and the others k-1 as training sets.

The global evaluation is calculated throughout all the trials, thus achieving a stable performance evaluation setup. For completeness, we report both the mean and the variance of each performance measurement as a result of the evaluation on the different folds. In figure 4.1) the operation of k-fold validation is clearly shown.

To perform the measurements of the project model is used k = 10 and four standard performance metrics are determined: Accuracy, Precision, Recall and Cross-Entropy loss. Assuming a sample X (in the case this study, X is a flow), True Positive (TP) is the number of samples classified correctly, but if it is typed incorrectly, it counts as False Positive (FP). On

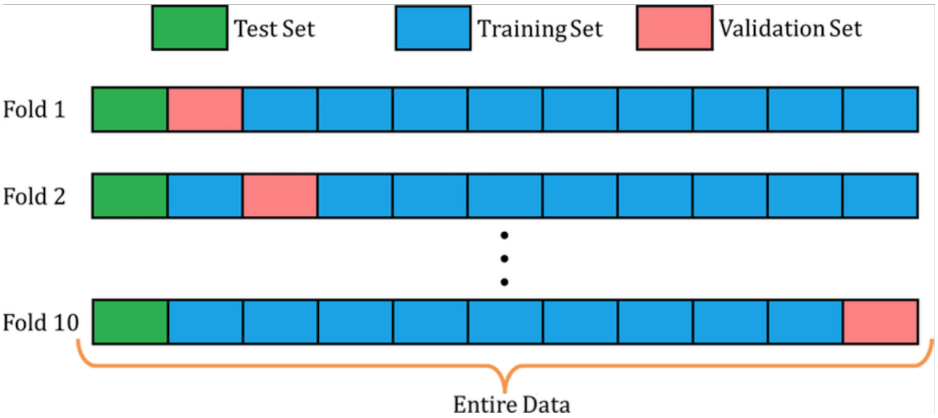


Figure 4.1: Data division and behaviour of the 10-fold process

the other hand, True Negative (TN) is the number of packets classified correctly as Not-X, and if the packet is classified as Not-X incorrectly, we count False Negative (FN). Then, metrics are calculated as follows

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP * FN} \tag{4.3}$$

As said above, the Softmax layer evaluates the loss of the model with cross-entropy minimization as a cost function (Rob DiPietro, 2018). It indicates the difference between what the model believes the output distribution should be, and what the original distribution really is. It is defined as

$$CE(y, p) = iyilog(pi) \tag{4.4}$$

where i is the class, y the prediction and p the real output.

Note that in order to carry out the training, only classes with at least 40 traces are used. It is done in this way because, as is checked later, the results below this quantity of instances have too much deviation. Then, the experiment is carried out with a total of 73 apps and 2920 traces.

4.2 Experimental Results and Analysis

In order to appreciate the detection quality of the different options to train the model, this section compares the results of the classification according to the number of instances and the number of packages per example to see the impact that causes over the model.

4.2.1 Effect of the Number of Packets per Flow

In this section, five data sets are created where the apps and traces are the same, but the number of packets for each evidence varies. This study shows the importance of this factor in the functioning of the model.

Note that the traffic was captured in randomly times. Then, it provoked traces longer than others, and the length does not depend on the app. This fact can be a drawback if the system takes it as a feature. However, it is observed that a large percentage of traces have a minimum of 100 packages and that is why the behaviour of the model is evaluated up to 100 packets.

The results obtained are shown in table 4.1 and in a graphical comparison 4.2 with the number of packages compares their accuracy and loss.

Number of packets	Test Accuracy	Test Loss
25	mean 0.6216, std 0.0265	mean 1.5057, std 0.1219
50	mean 0.7592, std 0.0436	mean 1.1233, std 0.1767
75	mean 0.7733, std 0.0456	mean 1.1251, std 0.1704
100	mean 0.7818, std 0.0282	mean 1.0848, std 0.1494

Table 4.1: Performance of the model depending of the number of packets per flow

In both table and graph can see that working with more packages results in better and more reliable classification. Regarding accuracy, it is clear that 25 packets are few to train the

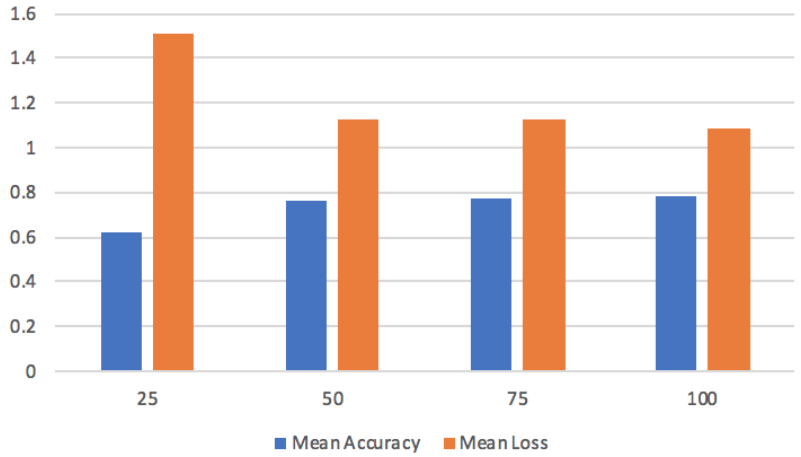


Figure 4.2: Comparison graphic of the results depending on the number of packets per flow

algorithm, but between 50 and 100 packets the difference is not significant, and the value stabilizes. It indicates that for many packages that would use, the accuracy of the model would be around the value of 80% However, what is observed is that as the system trains with more packages, it becomes more reliable. Both Loss and Standard deviation are diminishing, and it is likely that with more packages it continues to decrease.

The results of the best option (100 packets per trace) are shown in the figures B.

4.2.2 Effect of the Number of Flows

The interest in this section is to see from how many samples the model begins to learn a reasonably good pattern to classify flows. To study this effect, the applications are filtered getting only the ones with a minimum of 40 flows (73 apps). Then, there are three different sets of data, each one with a different number of samples. The first has 20 per application (1460 instances), the second 30 (2190 instances), and the last has 40 flows per application (2920). Both the table 4.2 and the graph 4.3 can see the results obtained.

As expected, the model with more samples is the one that gets better results. However, it should be noted that with 30 or 20 there is a reasonably close result to that of 40. On the other hand, the deviation and the loss augment too much.

Number of flows per app	Test Accuracy	Test Loss
20	mean 0.7473, std 0.0408	mean 1.2884, std 0.1997
30	mean 0.7534, std 0.0331	mean 1.2445, std 0.1669
40	mean 0.7818, std 0.0282	mean 1.0848, std 0.1494

Table 4.2: Performance of the model depending of the number of flows per app

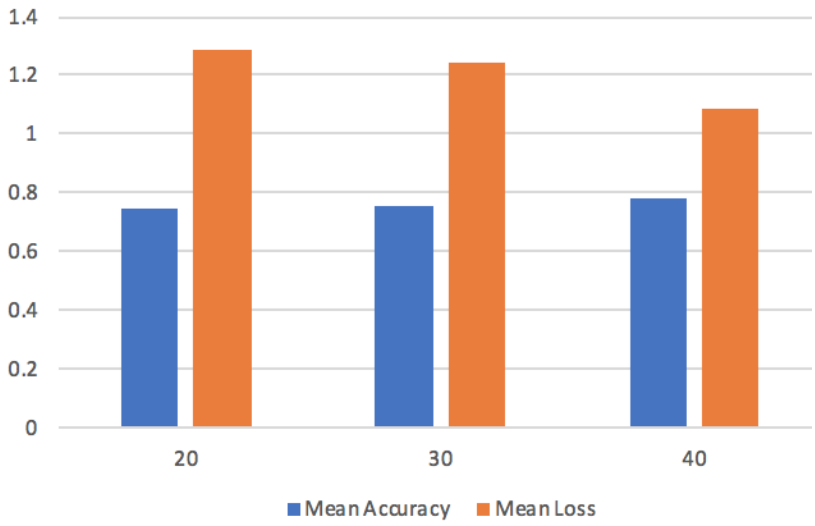


Figure 4.3: Comparison graphic of the results depending on the number of flows per app

4.3 Per-Class Study

To end the main study of the project is evaluated the results obtained by the model for each one of the classes separately. In this way, it can identify which applications most adapt to our model. The C table shows the average of the precision and recall calculated in all k-folds (10-folds) for each of the apps in the project.

The parameters are the same as the previous section where 40 traces of 100 packages are used for each app. The objective of the project is to classify each flow with its corresponding application. This study envisages which classes are most vulnerable when it comes to preserving privacy by using the designed model. Finally, the results are that 37 applications (51% of the total) have accuracy above 85%, 31 apps (42%) have a recall over 85%, and 20 applications (28%) have this two measures above 85%. Therefore, a large number of apps get an accurate classification. On the other hand, there are 9 applications (12%) that have both measures below 60% and thus ensure proper security.

It should be noted that in the set of training data two applications can be sensitive and show if a person is single or not and their sexual orientation. These are *wildec.dating.meetu* and *dating.app.datingapps* (in bold in table C). As the results show, the first one gets shallow measurements, while the second gets one of the best results of classification.

4.4 One-Class Accuracy

In this last study, introduced in the ref sec: section, it begins by explaining the design changes and how the data set is divided (4.4.1). To finish the results are shown with the help of graphs (4.4.2).

4.4.1 Experimental Methodology

As indicated, is wanted to make a model with only one output that indicates whether the sample is from the app or not. With the same data set that the whole project, the data is Pre-Processed extracting the same features (length and direction). The difference with respect to the rest of the project is that, when it comes to tagging the instances, there are only *App* and *Noapp* classes.

Actually, in *App* class, there is one application while *Noapp* class contain 116 applications. As it can see, the difference in samples in both classes is enormous. When this happens, it is called Imbalanced data set (Gil Delgado, 2015). If this is not deal as a special way, it can cause overfitting and detect all classes as *Noapp* due to the simple fact of the large number of samples it has. To solve this, the data set has been treated in such a way that in the training set there is the same number of "App" samples as *Noapp*.

Be A the total samples of "App" and Ta, Va, and ta the samples of its Training, Validation and Test set, N the total samples of *Noapp* and Tn, Vn and tn he samples of its Training, Validation and Test set respectively. Then, the data set of the *Noapp* class is divided as follows:

$$Ta = A \cdot 0.6; Va = A \cdot 0.15; ta = A \cdot 0.15 \tag{4.5}$$

$$Tn = Ta; Vn = \frac{N - Tn}{2}; tn = \frac{N - Tn}{2} \tag{4.6}$$

In this case, more samples than normally in the test and the validation set are used to have more samples to classify. This imbalance causes that, at the time of testing, the model finds many more negative samples than positive instances. Also, from these negatives, the most likely is that many applications have not been trained and are unknown for the model. That is why it is said that in this section is simulating a case of an open world.

To represent the results in a binary classification, the true positive rate (TPR) and false positive rate (FPR) are used, but as indicated by (Juarez et al., 2016), the size of the sets are heavily unbalanced, so using only TPR and FPR can lead to incorrect interpretation due to the base-rate fallacy. Therefore, the measurements that are used to calculate the results are the same metrics (Precision Recall) explained. The ideal scenario that we all want is that the model should give 0 FP and 0 FN, but it is not possible as any model is 100% accurate most of the times.

As mentioned in the section 2.2.1, the last layer of the model returns a probability. Because it is a binary model, if the probability of the output is above the threshold indicates "App" but if its a smaller value indicates *Noapp*. The value of this threshold is variable since it depends on the interests. If it is wanted to focus more on minimizing False Negatives, the Recall needs to be as close to 100% as possible without Precision being too bad, and if the objective is to focus on minimizing false positives, then it is necessary to make Precision as close to 100% as possible.

Another aspect to take into account is that learning from the model depends on the flows of applications that are randomly inside the Training set of *Noapp*. For this reason, the training is repeated 3 times and each time it mixes the samples.

Finally, once the procedure has been explained, the following section shows the results of the two applications that reveal sensitive information about the user: (a) *wildec.dating.meetu* and (b) *dating.app.datingapps*, and represent the Precision Recall graphics of the two application with more instances collected:(a) *com.tomtom.speedcams.android.map.dump* and (b) *com.king.candycrushsaga.dump*.

4.4.2 Experimental Results and Analysis

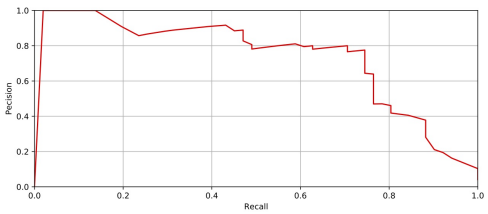
In all the experiments of this section, the whole data set is used. In cases of both applications (a) *wildec.dating.meetu* and (b) *dating.app.datingapps*, doing this study with only 40 samples for the class "App" versus 8157 samples for the class *Noapp* means that the results are very scattered and the curve Precision-Recall cannot be represented since it is almost impossible that an unknown application does not classify as an "App." Therefore, they are unreliable results. The best results for each CV are shown in the table 4.3:

App	CV	TP	FP	FN	Precision	Recall
com.wildec.dating.meetu.dump	1	2	264	4	0.0075	0.3333
com.wildec.dating.meetu.dump	2	3	177	3	0.0167	0.5
com.wildec.dating.meetu.dump	3	3	95	3	0.0306	0.5
dating.app.datingapps.dump	1	6	29	0	0.1714	1
dating.app.datingapps.dump	2	5	12	1	0.2941	0.8333
dating.app.datingapps.dump	3	5	30	1	0.1429	0.8333

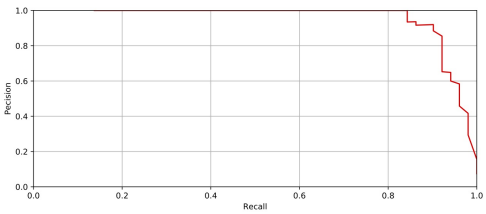
Table 4.3: Performance of the model depending of the number of flows per app

However, there is a significant difference in the comparison of the results. Although they are bad in both cases, it is evident that app (b) achieves better results in both measures. Then the results correspond with the Per-class study 4.3. Therefore, it can be stated that the traces of the app (a) are more difficult to classify and personal privacy is preserved.

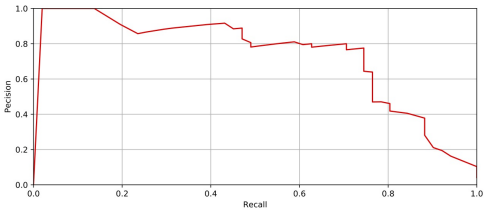
On the other hand, to assess if the model works, the two applications with more instances have been chosen (a) *com.tomtom.speedcams.android.map.dump* (330 flows) and (b) *com .king.candycrushsaga.dump* (337 flows) and the evolution of the Accuracy-Recall curve is represented in figures 4.4 and 4.5.



(a) tomtom.speedcams.android.map - CV 1



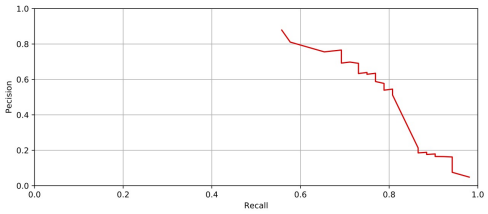
(b) tomtom.speedcams.android.map - CV 2



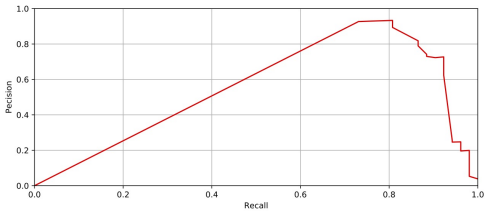
(c) tomtom.speedcams.android.map - CV 3

Figure 4.4: Accuracy - Recall curve of tomtom.speedcams.android.map

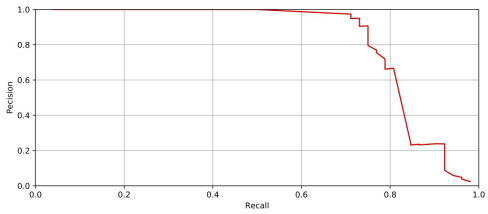
According to the previous study 4.3, they are two applications that are classified correctly, and In the graphs, the result corresponds. Both apps can get both 100% Accuracy and Recall, as well as a good result for both measurements at the same time.



(a) king.candycrushsaga - CV 1



(b) king.candycrushsaga - CV 2



(c) king.candycrushsaga - CV 3

Figure 4.5: Accuracy - Recall curve of king.candycrushsaga

Chapter 5

Budget

The cost of the project consists of the junior engineer's salary since it has been possible to work using free software such as python. It is considered a salary of 10€/h, so the cost depends on the hours invested in the project.

At the beginning, it was estimated that in order to complete the project, a total of 540h would be necessary. However, the expected hours were extended to a total of 600h. These extra hours are paid at 14€/h. Therefore, the cost of the project has been as follows:

	Type	**Wage/hour**	**Dedication**	**Total**
Junior engineer	Normal work hours	10.00 €/h	540h	5 400 €
Junior engineer	Extra work hours	14.00€/h	60h	840 €
			Total	6 240 €

Table 5.1: Estimated budget of the project

Chapter 6

Conclusions

This project develops a system to classify encrypted traces from mobile applications. First, the encrypted flows are processed to transform them into numerical vectors and subsequently are trained in a 1D-CNN designed model that can detect patterns of each trace by only extracting the lengths of the packages and their direction.

Through a data set with 73 applications, each one with 40 traces, a closed-world is simulated. It achieves 78% of accuracy where 51% of the applications have an Accuracy of more than 85%. Additionally, an open-world experiment is done. It consists in classify an application versus the rest. This last study only works with apps with a lot of instances, and even not good results had achieved for the apps with sensitive information, the good performance of the model has been demonstrated.

However, there are three limitations and related future work about the thesis. First, more training data would help the model to distinguish different applications better, and the Standard deviation would diminish. Second, our system only uses spatial features of traffic and ignores temporary elements. In future researches should investigate how to add these features as well as new scenarios with more convolutional layers. Finally, and as it has been tried to do in the current project without results (1.5), in order to complete the study, the model should be tested with traffic data captured from an uncontrolled scenario. This would allow getting results very close to reality.

Overall, the study reveals the need for certain applications to improve their safety measures. Most apps encrypt traffic to hide what is being retransmitted but, as it has been proven, using deep learning, is possible to know which application has been used on the smart-phone and with this simple information, attack the privacy of the user.

Bibliography

A. L. Narasimha Reddy. Network elements based on partial state. Dept. of Electrical Engineering, Texas A & M University, [Online; accessed 12-November-2018].

Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Traffic classification of mobile apps through multi-classification. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

Andrew Ng. Machine learning, 2018. [Online; accessed 17-February-2019].

Jason Brownlee.

Ran Dubin, Amit Dvir, Ofir Pele, and Ofer Hadar. I know what you saw last minute—encrypted http adaptive video streaming title classification. *IEEE Transactions on Information Forensics and Security*, 12(12):3039–3049, 2017.

Luis Gil Delgado. Clasificación de tráfico de internet mediante análisis de datos. 2015.

Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Interspeech*, volume 13, pages 1756–1760, 2013.

Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security*, pages 27–46. Springer, 2016.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Wei Li and Andrew W Moore. A machine learning approach for efficient traffic classification. In *2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 310–317. IEEE, 2007.

Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access*, 5:18042–18050, 2017.

Xiaolei Ma, Zhuang Dai, Zhengbing He, Jihui Ma, Yong Wang, and Yunpeng Wang. Learning traffic as images: a deep convolutional neural network for large-scale transportation network speed prediction. *Sensors*, 17(4):818, 2017.

Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.

Rob DiPietro. A friendly introduction to cross-entropy loss, 2018. [Online; accessed 24-January-2019].

Raúl Santos Lebrato. Desarrollo de una solución de clasificación de tráfico de red utilizando técnicas de aprendizaje automático. B.S. thesis, 2018.

Warren S Sarle. Stopped training and other remedies for overfitting. *Computing science and statistics*, pages 352–360, 1996.

Tim Stöber, Mario Frank, Jens Schmitt, and Ivan Martinovic. Who do you sync you are?: smartphone fingerprinting via application behaviour. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pages 7–12. ACM, 2013.

Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 439–454. IEEE, 2016.

R Vinayakumar, KP Soman, and Prabaharan Poornachandran. Applying convolutional neural network for network intrusion detection. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1222–1228. IEEE, 2017.

Vitaly Bushaev. Understanding rmsprop - faster neural network learning, 2018.

Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen 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. IEEE, 2017a.

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. IEEE, 2017b.

Wikipedia contributors. Artificial neuron — Wikipedia, the free encyclopedia, 2019. URL https://en.wikipedia.org/w/index.php?title=Artificial_neuron&oldid=885171186. [Online; accessed 10-May-2019].

Yibo Xue, Dawei Wang, and Luoshi Zhang. Traffic classification: Issues and challenges. In *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 545–549. IEEE, 2013.

Appendix A

Apps and Traces in the whole data set

List of the applications with the number of traces. In total there is a data set with 8197 traces for 117 apps.

App	Number of traces
com.valvesoftware.android.steam.community.dump	204
com.sinyee.babybus.taxi.dump	194
dating.app.datingapps.dump	40
com.appswiz.kodistreamings.dump	100
com.tomtom.speedcams.android.map.dump	330
com.havit.android.dump	40
com.gomo.calculator.dump	183
com.igg.android.lordsmobile.dump	100
com.contextlogic.wish.dump	160
homeworkout.homeworkouts.noequipment.dump	50
com.alibaba.intl.android.apps.poseidon.dump	100
ecom.psafe.msuite.dump	127
com.leftover.CoinDozer.dump	20
com.microsoft.office.outlook.dump	286
com.instagram.android.dump	232
com.roblox.client.dump	231
shayari.funnyshayari.dump	20
com.streema.simpleradio.dump	184
com.spotify.music.dump	171
com.pnn.obdcardoctor.dump	40
com.snapchat.filters.lenses.stickers.forSnapchat.dump	194
net.malingo.wortsuchesprachen.android.wordsearch.dump	20
com.gamingsauce.redballiv.dump	20
com.sirma.mobile.bible.android.dump	20
jp.co.nestle.chocollabo.dump	100
com.jiubang.goclockex.dump	100
com.soundcloud.android.dump	126
com.king.candycrushsaga.dump	337
com.beprod.cofidis.dump	91
com.playrix.homescapes.dump	50
com.alien.shooter.galaxy.attack.dump	132
com.ss.android.article.topbuzzvideo.en.dump	131
com.ea.gp.fifamobile.dump	100
com.waze.dump	100
com.wallapop.dump	99
com.crazylabs.sausage.run.dump	200

Table A.1: Total apps with the number of traces (1)

App	Number of traces
com.inapp.cross.stitch.dump	20
com.domobile.applock.dump	26
com.pl.premierleague.dump	20
com.lajeuneandassociatesllc.barberchopdev.dump	40
com.igg.castleclash.dump	79
com.kiloo.subwaysurf.dump	250
com.jiubang.fastestflashlight.dump	100
music.christmas.dump	190
de.zalando.mobile.dump	55
com.wunderkinder.wunderlistandroid.dump	40
com.supercell.clashroyale.dump	50
MyING.be.dump	40
com.plonkgames.apps.iqtest.dump	50
com.onetengames.architect.craft.build.construction.dump	40
io.walletpasses.android.dump	40
ru.artemnaumov.soulbough.dump	20
com.pinkapp.dump	4
com.zhiliaoapp.musically.dump	71
com.notdoppler.earntodie.dump	13
net.peakgames.toonblast.dump	50
com.wildec.dating.meetu.dump	40
com.gismart.realpianofree.dump	100
com.wordgame.words.connect.dump	29
com.ubercab.dump	107
com.bitmango.go.wordcookies.dump	100
com.whatsapp.dump	50
jp.nagoyastudio.shakocho.dump	26
de.stocard.stocard.dump	50
bubbleshooter.orig.dump	20
com.netease.chiji.dump	55
com.pinterest.dump	238
com.apnax.wordsnack.dump	100
com.gamefirst.Jurassic.Survival.Island.ARK.dump	40
com.mxtech.ffmpeg.vneon.dump	8
com.tricore.face.projector.dump	40
com.robtopx.geometrydashsubzero.dump	183
com.ehawk.proxy.freevpn.dump	20
com.midasapps.roomcreator.dump	40
com.epicactiononline.ffxv.ane.dump	135
com.calea.echo.dump	40
com.movga.fleetcommand.gp.dump	20
com.facebook.katana.dump	20
com.tnature.Mahjong.dump	47
air.com.pixelstudio.venom.angry.crash.rush.io.online.dump	40
com.playrix.gardenscapes.dump	45
air.com.pixelstudio.venom.angry.crash.rush.io.online.dump	40
com.disney.WMWLite.dump	44
com.fillword.cross.wordmind.en.dump	9

Table A.2: Total apps with the number of traces (2)

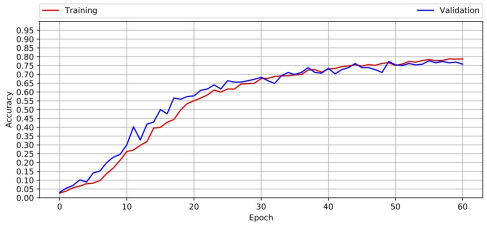
App	Number of traces
com.wallapop.dump	99
com.crazylabs.sausage.run.dump	200
com.inapp.cross.stitch.dump	20
com.domobile.applock.dump	26
com.pl.premierleague.dump	20
com.lajeuneandassociatesllc.barberchopdev.dump	40
net.joyplustech.androidmermaid.dump	40
com.ketchapp.fingerdriver.dump	50
com.lafourchette.lafourchette.dump	40
com.pixel.art.coloring.color.number.dump	20
us.zoom.videomeetings.dump	16
com.mxtech.ffmpeg.v.dump	4
be.scarlet.mobile.dump	21
com.sunofcode.slimedoh.dump	40
com.tiles.piano.animemusic.dump	40
be.wandelknooppunt.dump	20
com.italia.autovelox.autoveloxfissiemoibli.dump	40
com.escapefun.theroomiv.dump	20
com.aminota.tupreferes.dump	4
com.bitstrips.imoji.dump	5
air.com.flaxbin.chat.dump	20
appinventor.aiatomantapps.rmeeps.dump	10
com.shas.kshethram.thayinerimuchilot.dump	20
be.appsolution.lotterienationale.dump	2
com.cc.citycasino.dump	20
com.ketchapp.rider.dump	77
com.amazon.avod.thirdpartyclient.dump	24
eu.optimile.mobiflow.MobilityApp.dump	12
com.visiotalent.app.dump	20
com.js.offroad.car.driving.simulator.hill.climb.driving.dump	4
com.androidringtones.sonneriespopulaires.melodies.dump	40
com.abichus.worchypicturewordsearch.dump	24
com.plexapp.android.dump	20
com.jamesots.android.contractiontimer.dump	17
sandbox.art.sandbox.dump	4
roid.spikesroid.tvremoteforlg.dump	5
be.parcapp.dump	2
fixedcom.microsoft.office.outlook.dump	1
com.outfit.mytalkingtomfree.dump	1
com.mason.wooplus.dump	2

Table A.3: Total apps with the number of traces (3)

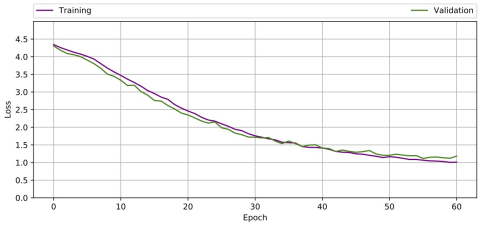
Appendix B

Accuracy and Loss graphs

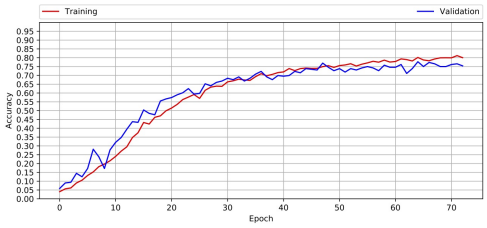
Graphs of the results of the training with 40 flows and 100 packets.



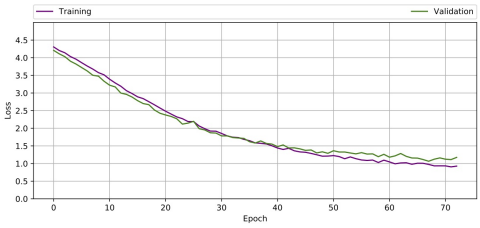
(a) Accuracy - Fold 1



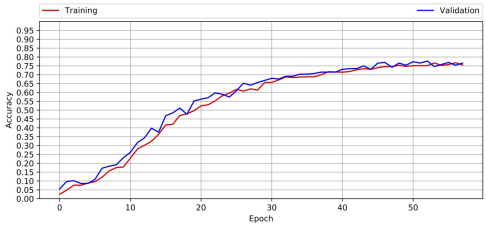
(b) Loss - Fold 1



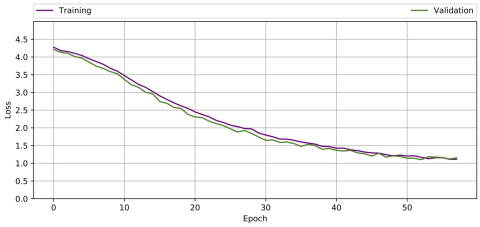
(c) Accuracy - Fold 2



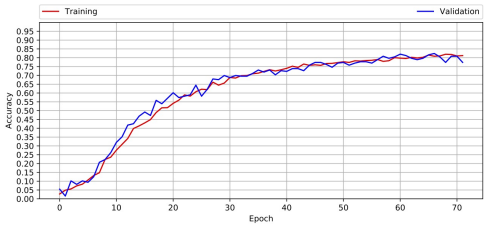
(d) Loss - Fold 2



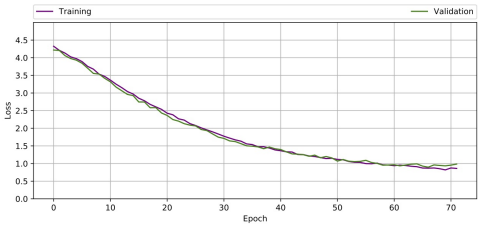
(e) Accuracy - Fold 3



(f) Loss - Fold 3

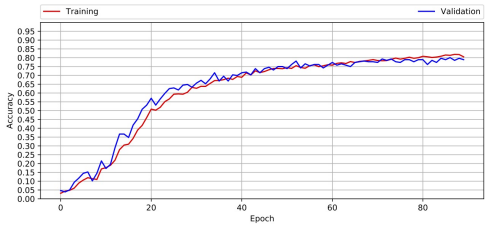


(g) Accuracy - Fold 4

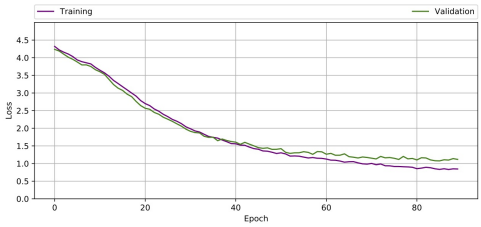


(h) Loss - Fold 4

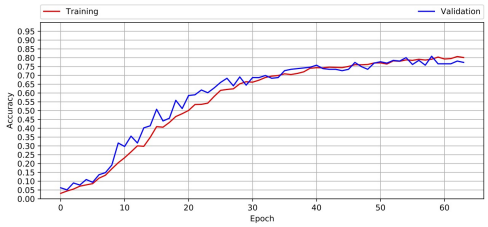
Figure B.1: Graphs of the results of the training with 40 flows and 100 packets (1)



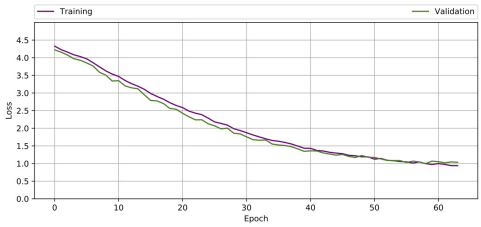
(a) Accuracy - Fold 5



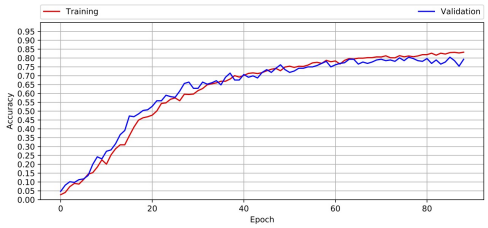
(b) Loss - Fold 5



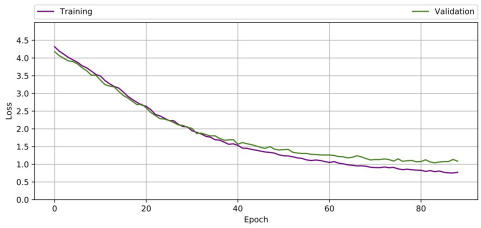
(c) Accuracy - Fold 6



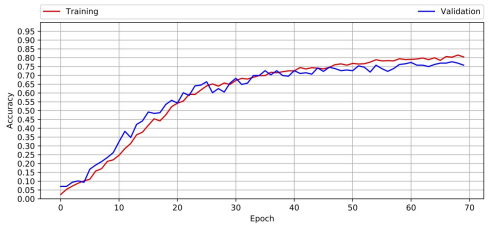
(d) Loss - Fold 6



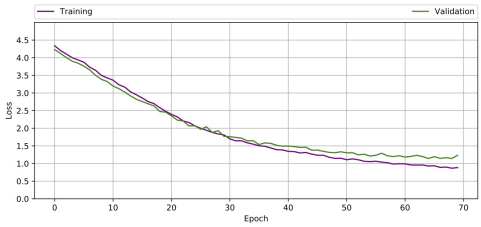
(e) Accuracy - Fold 7



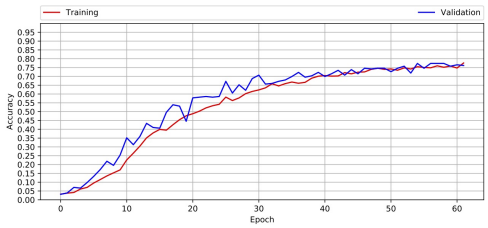
(f) Loss - Fold 7



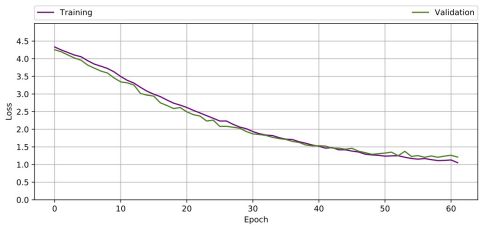
(g) Accuracy - Fold 8



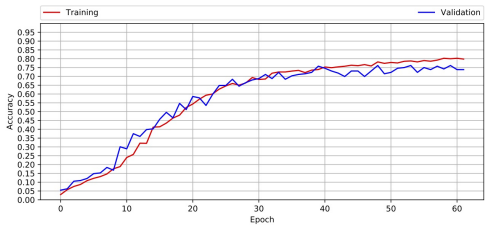
(h) Loss - Fold 8



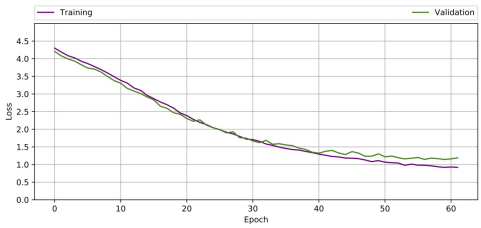
(i) Accuracy - Fold 9



(j) Loss - Fold 9



(k) Accuracy - Fold 10



(l) Loss - Fold 10

Figure B.2: Graphs of the results of the training with 40 flows and 100 packets (2)

Appendix C

Results of the Per-Class study

List of the performance of the model for each class.

App	Precision	Recall
com.jiubang.goclockex.dump	0.705	0.750
com.waze.dump	0.799	0.925
com.jiubang.fastestflashlight.dump	0.582	0.425
com.microsoft.office.outlook.dump	0.893	0.850
com.crazylabs.sausage.run.dump	0.927	0.900
com.ketchapp.rider.dump	0.932	0.850
com.alibaba.intl.android.apps.poseidon.dump	0.668	0.500
com.wallapop.dump	0.293	0.225
com.bitmango.go.wordcookies.dump	0.892	0.825
com.ea.gp.fifamobile.dump	0.855	0.725
com.wildec.dating.meetu.dump	0.555	0.475
net.joyplustech.androidmermaid.dump	0.784	0.850
com.sunofcode.slimedoh.dump	0.897	0.750
com.tricore.face.projector.dump	0.852	0.775
com.italia.autovelox.autoveloxfissiemoibli.dump	0.705	0.725
MyING.be.dump	0.884	0.925
com.androidringtones.sonneriespopulaires.melodies.dump	0.516	0.475
com.midasapps.roomcreator.dump	0.658	0.525
com.playrix.gardenscapes.dump	0.858	0.900
com.lajeuneandassociatesllc.barberchopdev.dump	0.908	0.800
com.lafourchette.lafourchette.dump	0.783	0.825
com.pinkapp.dump	0.756	0.900
dating.app.datingapps.dump	0.960	0.900
air.com.pixelstudio.venom.angry.crash.rush.io.online.dump	0.902	0.900
com.wunderkinder.wunderlistandroid.dump	0.880	0.875
com.streema.simpleradio.dump	0.547	0.550
io.walletpasses.android.dump	0.862	0.850
music.christmas.dump	0.471	0.550
com.gamefirst.Jurassic.Survival.Island.ARK.dump	0.888	0.900
com.calea.echo.dump	0.805	0.775
com.gomo.calculator.dump	0.519	0.575
com.tiles.piano.animemusic.dump	0.868	0.925
com.pnn.obdcardoctor.dump	0.905	0.825
com.tomtom.speedcams.android.map.dump	0.762	0.700

Table C.1: .
Performance of the model for each class(1)

App	Precision	Recall
com.king.candycrushsaga.dump	0.942	0.750
com.appswiz.kodistreamings.dump	1.000	0.850
com.gismart.realpianofree.dump	0.833	0.575
com.apnax.wordsnack.dump	0.877	0.925
com.igg.android.lordsmobile.dump	0.892	0.900
jp.co.nestle.chocollabo.dump	0.895	0.725
com.pinterest.dump	0.653	0.775
com.onetengames.architect.craft.build.construction.dump	0.800	0.950
com.valvesoftware.android.steam.community.dump	0.880	0.775
com.havit.android.dump	0.453	0.300
com.snapchat.filters.lenses.stickers.forSnapchat.dump	0.823	0.875
com.sinyee.babybus.taxi.dump	0.884	0.725
com.disney.WMWLite.dump	0.935	0.925
com.beprod.cofidis.dump	0.789	0.825
com.netease.chiji.dump	0.613	0.625
de.zalando.mobile.dump	0.733	0.575
com.whatsapp.dump	1.000	0.900
com.spotify.music.dump	0.885	0.925
com.instagram.android.dump	0.917	0.950
com.zhiliaoapp.musically.dump	0.479	0.575
com.supercell.clashroyale.dump	0.764	0.925
homeworkout.homeworkouts.noequipment.dump	0.662	0.600
net.peakgames.toonblast.dump	0.872	0.925
com.plonkgames.apps.iqtest.dump	0.795	0.900
com.robtopx.geometrydashsubzero.dump	0.800	0.925
com.kiloo.subwaysurf.dump	0.784	0.800
de.stocard.stocard.dump	0.858	0.650
com.contextlogic.wish.dump	0.980	0.950
com.ketchapp.fingerdriver.dump	0.772	0.650
com.tnature.Mahjong.dump	0.865	0.825
com.playrix.homescapes.dump	0.875	0.925
com.roblox.client.dump	0.947	0.925
com.psafe.msuite.dump	0.712	0.800
com.igg.castleclash.dump	0.813	0.850
com.soundcloud.android.dump	0.947	0.975
com.ubercab.dump	0.913	0.850
com.epicactiononline.ffxv.ane.dump	0.960	0.925
com.ss.android.article.topbuzzvideo.en.dump	0.657	0.475
com.alien.shooter.galaxy.attack.dump	0.867	0.900

Table C.2: Performance of the model for each class (2)