Network traffic classifier with convolutional and recurrent neural networks for Internet of Things

Manuel Lopez-Martin (Senior Member, IEEE), Belen Carro, Antonio Sanchez-Esguevillas (Senior Member, IEEE) and Jaime Lloret (Senior Member, IEEE)

Abstract— **A Network Traffic Classifier (NTC) is an important part of current network monitoring systems, being its task to infer the network service that is currently used by a communication flow (e.g. HTTP, SIP…). The detection is based on a number of features associated with the communication flow, for example, source and destination ports and bytes transmitted per packet. NTC is important because much information about a current network flow can be learned and anticipated just by knowing its network service (required latency, traffic volume, possible duration…). This is of particular interest for the management and monitoring of Internet of Things (IoT) networks, where NTC will help to segregate traffic and behavior of heterogeneous devices and services. In this paper, we present a new technique for NTC based on a combination of deep learning models that can be used for IoT traffic. We show that a Recurrent Neural Network (RNN) combined with a Convolutional Neural Network (CNN) provides best detection results. The natural domain for a CNN, which is image processing, has been extended to NTC in an easy and natural way. We show that the proposed method provides better detection results than alternative algorithms without requiring any feature engineering, which is usual when applying other models. A complete study is presented on several architectures that integrate a CNN and an RNN, including the impact of the features chosen and the length of the network flows used for training.**

Index Terms—Convolutional Neural Network; Deep Learning; Network traffic classification; Recurrent Neural Network

This work has been partially funded by the Ministerio de Economía y Competitividad del Gobierno de España and the Fondo de Desarrollo Regional (FEDER) within the project "Inteligencia distribuida para el control y adaptación de redes dinámicas definidas por software, Ref: TIN2014-57991-C3-2-P", and the Project "Distribucion inteligente de servicios multimedia utilizando redes cognitivas adaptativas definidas por software", Ref: TIN2014-57991-C3-1-P, in the Programa Estatal de Fomento de la Investigación Científica y Técnica de Excelencia, Subprograma Estatal de Generación de Conocimiento.

M. Lopez, B. Carro and A. Sanchez are with Dpto. TSyCeIT, ETSIT, Universidad de Valladolid, Paseo de Belén 15, Valladolid 47011, Spain; (manuel.lopezm@uva.es; belcar@tel.uva.es; antoniojavier.sanchez@uva.es).

J. Lloret is with Instituto de Investigación para la Gestión Integrada de Zonas Costeras, Universitat Politècnica de València, Camino Vera s/n, Valencia 46022, Spain; (jlloret@dcom.upv.es).

I. INTRODUCTION

A Network Traffic Classifier (NTC) is an important part of current network management and administration systems. An NTC infers the service/application (e.g. HTTP, SIP...) being used by a network flow. This information is important for network management and Quality of Service (QoS), as the service used has a direct relationship with QoS requirements and user contracts/expectations.

It is clear that Internet of Things (IoT) traffic will pose a challenge to current network management and monitoring systems, due to the large number and heterogeneity of the connected devices. NTC is a critical component in this new scenario [1][2], allowing to detect the service used by dissimilar devices with very different user-profiles. Network traffic identification is crucial for implementing effective management of network policy and resources in IoT networks, as the network needs to react differently depending on traffic profile information.

There are several approaches to NTC: port-based, payload-based, and flow statistics-based [3][4]. Port-based methods make use of port information for service identification. These methods are not reliable as many services do not use well-known ports or even use the ports used by other applications.

Payload-based approaches the problem by Deep Packet Inspection (DPI) of the payload carried out by the communication flow. These methods look for well-known patterns inside the packets. They currently provide the best possible detection rates but with some associated costs and difficulties: the cost of relying on an up-to-date database of patterns (which has to be maintained) and the difficulty to be able to access the raw payload. Currently, an increasing proportion of transmitted data is being encrypted or needs to assure user privacy policies, which is a real problem to payload-based methods.

Finally, flow statistics-based methods rely on information that can be obtained from packets header (e.g. bytes transmitted, packets interarrival times, TCP window size…). They rely on packet header high-level information which makes them a better option to deal with non-available payloads or dynamic ports. These methods usually rely on machine learning techniques to perform service prediction [3]. Two machine learning alternatives are available in this case: supervised and unsupervised methods. Supervised methods learn an association between a set of features and the desired labeled output by training an algorithm with samples containing ground-truth labeled outputs. In unsupervised methods, we do not have data with their associated ground-truth labeled outputs; therefore, they can only try to separate the samples in groups (clusters) according to some intrinsic similarities.

In this paper, we propose a new flow statistics-based supervised method to detect the service being used by an IP network flow. The proposed method employs several features extracted from the headers of packets exchanged during the flow lifetime. For each flow, we build a time-series of feature vectors. Each element of the time-series will contain the features of a packet in the flow. Likewise, each flow will have an associated service/application (a labeled value) which is required to train the algorithm.

To ensure data confidentiality our method only makes use of features from the packet's header, not including the IP addresses.

In order to train the method, we have used more than 250,000 network flows which contained more than 100 distinct services. As an additional challenge, the frequency distribution of these services was highly unbalanced.

The proposed method is a classifier based on a deep learning model formed by the combination of a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN).

One of the main drivers of this work was to assess the applicability of deep learning advances to the NTC problem. Therefore, we have studied the adequacy of different deep learning architectures and the influence of several design decisions, such as the features selected, or the number of packets per flow included in the analysis.

In the paper, we present a comparison of performance results for different architectures; in particular, we have considered RNNs alone, CNNs alone and different combinations of CNN and RNN.

In order to apply a CNN to a time-series of feature vectors, we propose an approach that renders the data as an associated pseudo-image, to which CNN can be applied.

When assessing the suitability of a new method it is important to apply it to real data. We have made use of data from RedIRIS, which is the Spanish academic and research network.

The paper is organized as follows: Section II presents the related works. Section III describes the work performed. Section IV describes the results obtained and finally, Section V provides discussion and conclusions.

II. Related works

Comparison of work results is difficult in NTC because the datasets being studied and the performance metrics applied are very different. NTC is intrinsically a multi-class classification problem. There is no single universally agreed metric to present results in a multi-class problem, as will be shown in Section IV. Considering these facts, we now present several related works.

There are many works that apply neural networks to NTC, but the network models employed are very different in nature to the ones presented here.

In [5] they propose a multi-layer perceptron (MLP) with zero or one hidden layer, but it is actually adopted as the internal architecture to apply a fully Bayesian analysis. The best one vs. rest accuracy, using 246 features, for 10 grouped labels is 99.8%, and a macro averaged accuracy of 99.3% (10 labels).

An ensemble of MLP classifiers with error-correcting output codes is applied in [6], achieving an average overall accuracy (for 5 labels) of 93.8%. Meanwhile, in [7] an MLP with a particle swarm optimization algorithm is employed to classify 6 labels with a best one vs. rest accuracy of 96.95%. Somehow related, the purpose of [8] is to investigate neural projection techniques for network traffic visualization. Towards that end, they propose several dimensionality reduction methods using neural networks. No classification is performed. Another work [9] explores the applicability of rough neural networks to deal with uncertainty but does not give any performance results for NTC.

Zhou et al. [10] apply an MLP with 3 hidden layers and different numbers of hidden neurons to the Moore dataset [11]. They give an overall accuracy greater than 96%, for a grouping of labels in 10 classes, resulting in a final class distribution very unbalanced (a frequency of almost 90% for highest frequency class), no F1 score is provided. A Parallel Neural Network Classifier Architecture is used in [12]. It is made up of parallel blocks of radial basis function neural networks. To train the network is employed a negative reinforcement learning algorithm. They classify 6 labels reporting a realistic overall accuracy of 95%, no F1 score is provided.

Another set of papers applies general machine learning techniques, not related with neural networks, to the NTC problem.

Kim et al. [13] propose an entropy-based minimum description length discretization of features as a preprocessing step to several algorithms: C4.5, Naïve Bayes, SVM and kNN. Claiming an enhanced performance of the algorithms, achieving a one vs. rest accuracy of 93.2%- 98% for 11 grouped labels. In [14] authors apply different machine learning techniques to NTC (C4.5, Support Vector Machine, Naïve Bayes) reporting an average accuracy of less than 80% using 23 features and detecting only five services (www, dns, ftp, p2p, and telnet)

Wang et al. [15] employ an enhanced random forest with 29 selected features. They group the services in 12 classes, providing only one vs. rest metrics (not aggregated). Having F1 scores in the interval 0.3-0.95, with only 3 classes higher than 0.96. They use their own dataset. Authors of [16] include flows correlation in a semi-supervised model providing overall accuracy of less than 85% and a one vs. rest F1 score, for 10 labels, of less than 0.9 (except two labels with 0.95 and 1). They use the WIDE backbone dataset [16]. They report having better results than other works using C4.5, kNN, Naïve Bayes, Bayesian Networks and Erman´s semi-supervised methods [17][18][19][20][21].

A Directed Acyclic Graph-Support Vector Machine is proposed in [22], attaining an average accuracy of 95.5%. The method is applied to a one-to-one combination of classes with a dataset provided by the University of Cambridge (Moore dataset) [11]. Yamansavascilar et al. [23] study the application of several algorithms: J48, Random Forest, Bayes Net, and kNN to UNB ISCX Network Traffic dataset, with 14 classes and 12 features, reporting the best accuracy of 93.94%.

Yuan et al. [24] present a variant of decision tree algorithm C4.5 working on the Hadoop platform. They classify 12 labels giving a one vs. rest accuracy in the interval 60-90% with only two labels higher than 90%. The dataset is the Moore set from Cambridge University [11].

In this paper, we present the first application of the RNN and CNN models to an NTC problem. The combination of both models provides automatic feature representation of network flows without requiring costly feature engineering.

III. WORK DESCRIPTION

Following sections present the dataset used for this work and a description of the different deep learning models that were applied.

A. *Selected dataset*

For this work, we have made use of real data from RedIRIS. RedIRIS is the Spanish academic and research backbone network that provides advanced communication services to the scientific community and national universities. RedIRIS has over 500 affiliated institutions, mainly universities and public research centers.

We have extracted 266,160 network flows from RedIRIS. These flows contained 108 distinct labeled services, with a highly unbalanced frequency distribution. Fig. 1 shows the names and frequency distribution for the 15 most frequent services. The frequency distribution is based on the proportion of flows with a specific service.

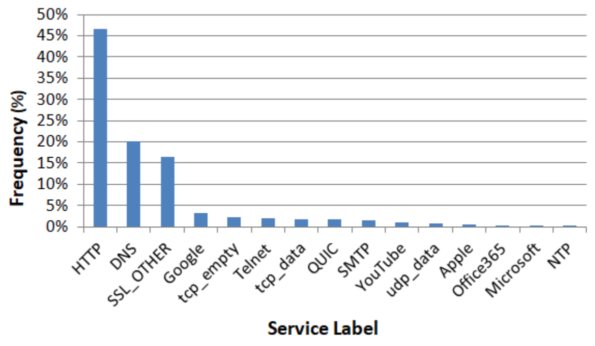


Fig. 1. Frequency distribution of the 15 most frequent services.

A network flow consists of all packets sharing a unique bi-directional combination of source and destination IP addresses and port numbers, and transport protocol: TCP or UDP. We include encrypted packets, as algorithms considered do not rely on payload content.

Each flow is associated with a particular service. In order to train and evaluate the models, we need to assign a ground-truth service to each flow. This assignment is not initially available and has been made possible by applying the nDPI tool [25] to the packets exchanged during the flow lifetime. nDPI applies a DPI technique to perform service detection. DPI provides the best available classification results by inspecting both the header and payload of the packet. With this in mind, we assume the output of a DPI tool as our best approximation to the ground-truth service. nDPI handles encrypted traffic and it is the most accurate open source DPI application [26]. The flows which nDPI was not able to label were discarded. For this work, we have considered UDP and TCP flows.

Each flow is formed by a sequence of up to 20 packets. For each packet, we have extracted the following six features: source port, destination port, the number of bytes in packet payload, TCP window size, interarrival time and direction of the packet. The TCP window size (TCP flow control) is set to zero for UDP packets. The packet address may have a value of 0-1 indicating whether the packet goes from source to destination or in the opposite direction.

We have considered only the first 20 packets exchanged in a flow lifetime. In the case of flows with more than 20 packets, we have discarded any packet after packet number 20. As we will see, 20 packets are more than enough to obtain a good detection rate, and even a much smaller number still provides excellent performance.

Finally, from these flows, we have built our dataset. Therefore, the dataset consists of 266,160 flows, each flow containing a sequence of 20 vectors, and each vector is made up of 6 features (the six features extracted from the packets' header). The final result is a time-series of feature vectors associated with each flow.

To evaluate the models, we set apart a 15% of flows as a validation set. All the performance metrics given in this paper correspond to this validation set. In order to build the validation set, we sampled the original flows, keeping the same labels frequency between the validation set and the remaining flows (training set).

Fig. 2 presents the final arrangement of a network flow inside the dataset.

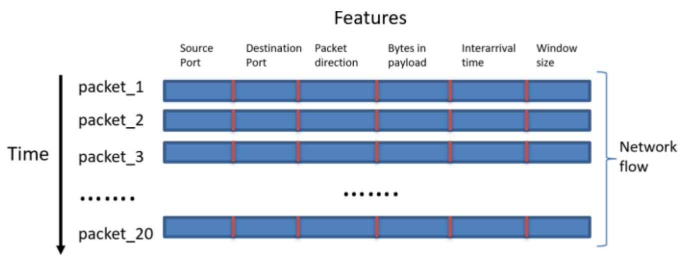


Fig. 2. The composition of a network flow.

B. Models description

Different deep learning models have been studied. The model with best detection performance has been a combination of CNN [27] and RNN [28]. In this section, we will show all the models considered for this work.

The first model analyzed (Fig. 3) was a simple RNN. In particular, we used a variant of an RNN called LSTM [29], which is easier to train (it solves the vanishing gradient problem). An LSTM is trained with a matrix of values with two dimensions: the temporal dimension and a vector of features. LSTM iterates a neural network (cell) with the time sequential feature vectors and two additional vectors associated with its internal hidden and cell states. The final hidden state of the cell corresponds to the output value. Therefore, the output dimension of an LSTM layer is the same as the size of its internal hidden state (LSTM units). In the model of Fig. 3 we add at the end several fully connected layers. Two layers are fully connected when each node of the previous layer is fully forward connected to every node of the consecutive layer. The fully connected layers have been added to all models.

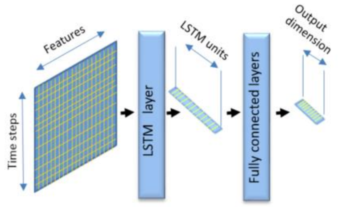


Fig. 3. Deep learning RNN model

In Fig. 4 a pure CNN network is shown. CNNs were initially applied to image processing, as a biologically inspired model to perform image classification, where feature engineering was done automatically by the network thanks to the action of a kernel (filter) which extracts location invariant patterns from the image. Chaining several CNNs allows extracting complex features automatically.

In our case, we have used this image-processing metaphor to apply the technique to a very different dataset. In order to do that, we consider the matrix formed by the time-series of feature vectors as an image. Image pixels are locally correlated; similarly, feature vectors associated with consecutive time slots present a correlated local behavior, which allows us to adopt this analogy.

Each CNN layer generates a multidimensional array (tensor) where the dimensions of the image get reduced but, at the same time, a new dimension is generated, having this new dimension a size equal to the number of filters applied to the image. Consecutive CNN layers will further decrease the image dimensions and increase the new generated dimension size. To top off the model it is necessary to transform the tensor to a vector that can be the input to the final fully connected layers. To accomplish this transformation a simple tensor flattening can be done (Fig. 4).

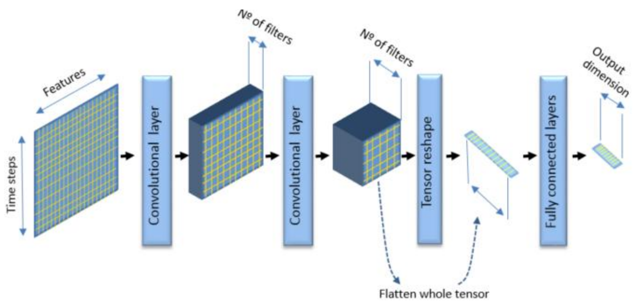


Fig. 4. Deep learning CNN model

The previous models can be combined in a single model as presented in Fig. 5. In this combined model, the final tensor of several chained CNNs is reshaped into a matrix that can act as the input to an RNN (LSTM network). To reshape the tensor as a matrix we keep the dimension associated with the filter's action unchanged, performing a flattening on the other two dimensions, to finally reach a matrix shape. The values produced by the filters of the last CNN will be the equivalent of feature vectors, and the flattened vector produced by the reshaping operation will act as the time dimension needed by the LSTM layer.

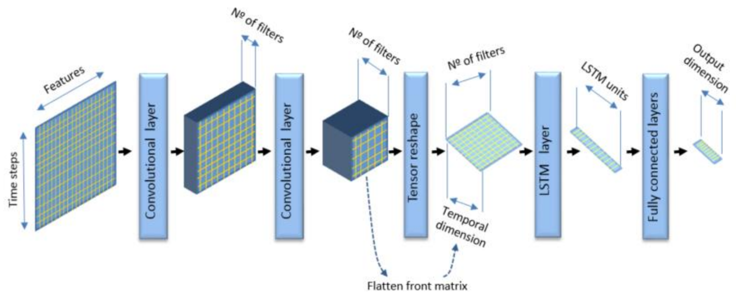


Fig. 5. Combination of CNN and single-layer RNN

Finally, the model introduced in Fig. 6 is similar to the previous model with the inclusion of an additional LSTM layer. When several LSTM layers are concatenated, the LSTM behavior is different to the one explained previously (Fig. 3). In this case, all LSTM layers (except the last one) adopt a 'return-sequences' mode that produces a sequence of vectors corresponding to the successive iteration of the recurrent network. This sequence of vectors can be grouped in a time sequence, forming the entry point to the next LSTM layer. It is important to note that, for successive LSTM layers, the temporal-dimension of data input does not change (Fig. 6), but the vector-dimension of the successive inputs does.

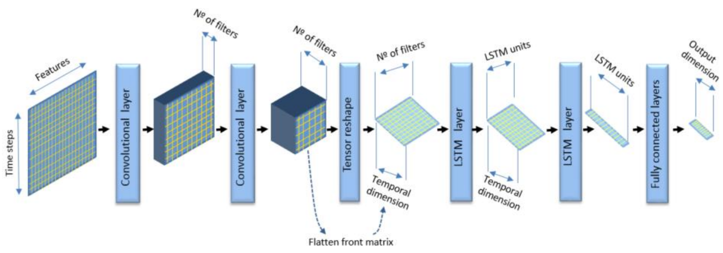


Fig. 6. Combination of CNN and two-layer RNN

Additionally, to the different types of layers presented previously, we have made use of some additional layers: batch normalization, max pooling, and dropout layers.

A dropout layer [30] provides regularization (a generalization of results for unseen data) by dropping out (setting to zero) a percentage of outputs from the previous layer. This apparently nonsensical action forces the network to not over-rely on any particular input, fighting over-fitting and improving generalization.

Max pooling [31] is a kind of convolutional layer. The difference is the filter used. In max pooling, it is used a max-filter, that selects the maximum value of the image region to which the filter is applied. It reduces the spatial size of the output, decreasing the number of features and the computational complexity of the network. The result is a down-sampled output. Similar to a dropout layer, a max pooling layer provides regularization.

Batch normalization [32] makes training convergence faster and can improve performance results. It is done by normalizing, at training time, every feature at batch level (scaling inputs to zero mean and unit variance) and re-scaling again later considering the whole training dataset. The newly learned mean and variance replace the ones obtained at batch-level.

IV. RESULTS

This section presents the results obtained when applying several deep learning models to NTC. The influence of several important hyper-parameters and design decisions is analyzed, in particular: the model architecture, the features selected and the number of packets extracted from the network flows.

In order to appreciate the detection quality of the different options, and considering the highly unbalanced distribution of service labels, we provide the following performance metrics for each option: accuracy, precision, recall, and F1. Considering all metrics, F1 can be considered the most important metric in this scenario. F1 is the harmonic mean of precision and recall and provides a better indication of detection performance for unbalanced datasets. F1 gets its best value at 1 and worst at 0.

We base our definition of accuracy, F1, precision, and recall in the following four previous definitions: (1) false positive (FP) that happens when there is actually no detection but we conclude there is one; (2) false negative (FN) when we indicate no detection but there is one; (3) true positive (TP) when we indicate a detection and it is real and (4) true negative (TN) when we indicate there is no detection and we are correct. Considering these previous definitions:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

$$\text{F1} = 2\,\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

We have used Tensorflow to implement all the models, and the python package scikit-learn to calculate performance metrics. All computations have been performed in a commercial PC (i7-4720-HQ, 16GB RAM).

A. *Impact of network architecture*

We have tried different deep learning architecture models to see their suitability for the NTC problem. In order to build the different architectures, we have considered different combinations of RNN and CNN: RNN only, CNN only, and various arrangements of a CNN followed by an RNN. In all cases, we have added at the end two additional fully connected layers.

In Table I, we provide a description of the different architectures and in Fig. 7 we present their performance metrics. From Fig. 7, we can see that the model CNN+RNN-2a gives the best results for both accuracy and F1.

The architecture description provided in Table I is as follows: Conv(z,x,y,n,m) stands for a convolutional layer with z filters where x and y are the width and height of the 2D filter window, with a stride of n and SAME padding if m is equal to S or VALID padding if m is equal to V (VALID implies no padding and SAME implies padding that preserves output dimensions). MaxPool(x,y,n,m) stands for a Max Pooling layer where x and y are the pool sizes, with a stride of n and SAME padding if m is equal to S or VALID padding if m is equal to V (VALID implies no padding and SAME implies padding that preserves output dimensions). BN stands for a batch normalization layer. FC(x) stands for a fully connected layer with x nodes. LSTM(x) stands for an LSTM layer where x is the dimensionality of the output space; in the case of several LSTM in sequence, each LSTM, except the last one, will return the successive recurrent values which will be the entry values to the following LSTM. DR(x) stands for a dropout layer with a dropout coefficient equal to x.

In all cases, the training was done with a number of epochs between 60-90 epochs, with early stopping if the last 10 epochs did not improve the loss function. We consider an epoch as a single pass of the complete training dataset through the training process.

All the activation functions were Rectified Linear Units (ReLU) with the exception of the last layer with Softmax activation. The loss function was Softmax Cross Entropy and the optimization was done with batch Stochastic Gradient Descent (SGD) with Adam.

In Table I, an added suffix ‘a’ to a model name, implies that the model has only changed the dropout percentage at the dropout layers.

Model	Architecture details
RNN-1	LSTM(100)-FC(100)-FC(108)
CNN-1	Conv(32,4,2,1,V)-MaxPool(3,2,1,V)-BN-Conv(64,4,2,1,V)-MaxPool(3,2,1,V)-BN-FC(200)-FC(108)
CNN+RNN-1	Conv(32,4,2,1,V)-MaxPool(3,2,1,V)-BN-Conv(64,4,2,1,V)-MaxPool(3,2,1,V)-BN-LSTM(100)-FC(100)-FC(108)
CNN+RNN-2	Conv(32,4,2,1,V)-BN-Conv(64,4,2,1,V)-BN-LSTM(100)-FC(100)-FC(108)
CNN+RNN-2a	Conv(32,4,2,1,V)-BN-Conv(64,4,2,1,V)-BN-LSTM(100)-DR(0.2)-FC(100)-DR(0.4)-FC(108)
CNN+RNN-3	Conv(16,4,2,1,V)-BN-Conv(32,4,2,1,V)-BN-Conv(64,4,2,1,V)-BN-Conv(128,4,2,1,V)-BN-LSTM(100)-DR(0.1)-LSTM(200)-DR(0.3)-FC(200)-DR(0.5)-FC(108)
CNN+RNN-3a	Conv(16,4,2,1,V)-BN-Conv(32,4,2,1,V)-BN-Conv(64,4,2,1,V)-BN-Conv(128,4,2,1,V)-BN-LSTM(100)-DR(0.2)-LSTM(200)-DR(0.4)-FC(200)-DR(0.5)-FC(108)
CNN+RNN-4	Conv(32,3,2,1,S)-BN-Conv(32,3,2,1,S)-MaxPool(2,2,2,S)-BN-Conv(64,3,2,1,S)-BN-Conv(64,3,2,1,S)-MaxPool(2,2,2,S)-BN-Conv(128,3,2,1,S)-Conv(64,1,1,1,S)-LSTM(100)-DR(0.1)-LSTM(200)-DR(0.3)-FC(200)-DR(0.5)-FC(108)
CNN+RNN-5	Conv(32,3,2,1,S)-BN-Conv(32,3,2,1,S)-MaxPool(2,2,2,S)-BN-Conv(64,3,2,1,S)-BN-Conv(64,3,2,1,S)-MaxPool(2,2,2,S)-BN-Conv(128,3,2,1,S)-Conv(64,1,1,1,S)-LSTM(100)-DR(0.1)-LSTM(150)-DR(0.3)-FC(150)-DR(0.5)-FC(108)

Table I. Details of deep learning network models applied to NTC problem

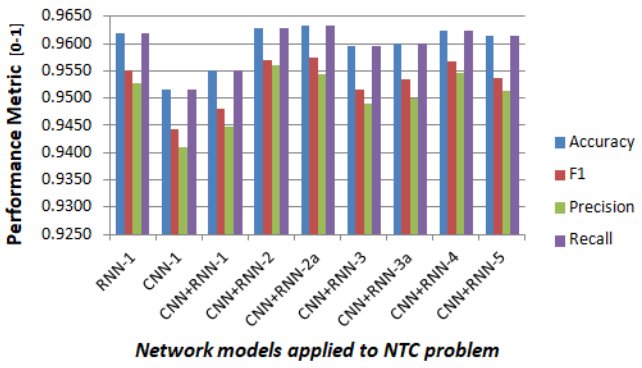


Fig. 7. Classification performance metrics (aggregated) vs. network models

The best model attains an accuracy of 0.9632, an F1 score of 0.9574, a precision of 0.9543 and a recall of 0.9632 (model CNN+RNN-2a).

Analyzing the results, we can see that a simple model, of two CNN layers followed by one LSTM layer with two fully connected layers at the end, provides best detection results, both in terms of accuracy and F1. The inclusion of MaxPooling or additional CNN or LSTM layers does not improve results. Batch normalization between CNN layers and the inclusion of some dropout layers at the end of the network do improve results.

For this problem, we have 108 distinct service labels to be detected. This is a multi-class classification problem. There are two possible ways to give results in this case: aggregated and One-vs.-Rest results.

For One-vs.-Rest, we focus in a particular class (label) and consider the other classes as a single alternative class, simplifying the problem to a binary classification task for each particular class (one by one). In the case of aggregated results, we try to give a summary result for all classes. There are different alternatives to perform the aggregation (micro, macro, samples, weighted), varying in the way the averaging process is done.

The performance metrics in Fig. 7 are aggregated metrics using a weighted average. We have used the weighted average provided by scikit-learn [33], to calculate the aggregated F1, precision, and recall scores.

In Fig. 8 we provide the One-vs.-Rest metrics for the classification of the first 15 more frequent labels (results obtained with model CNN+RNN-2a). An important observation in Fig. 8 is that for all labels with a frequency higher than 1% (Fig. 1) we achieve accuracy always higher than 98%, and many cases higher than 99%, and an F1 score higher than 0.96. The macro averaged accuracy for these 15 labels is 99.59% (best value in literature).

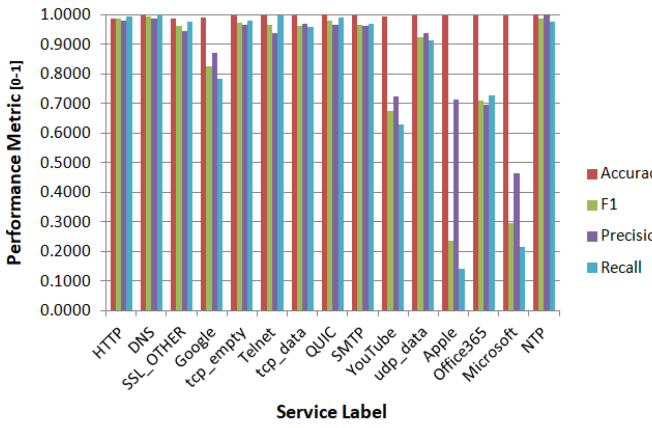


Fig. 8. Performance metrics (one vs. rest) for the classification of several service labels (15 more frequent)

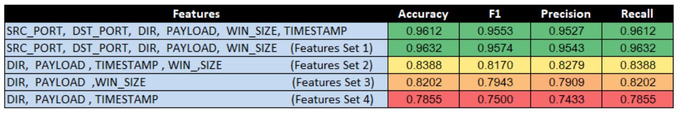
B. *Impact of features*

In Table II we can see the influence of the features employed in the learning process. Fig. 2 gives the full set of possible features, but it is important to appreciate which features have a higher importance in the detection process.

Table II shows the importance of features by analyzing the detection metrics as we remove different features. The first column in Table II gives the features that are used to train the model (grouped in feature sets) and the right columns the usual aggregate performance metrics for the detection process. The chart in Fig. 9 presents the same results in a different format, to make it easier to compare different feature sets.

As expected, in general, the more features render better results. But, interestingly the packets inter-arrival time (TIMESTAMP) gives slightly worse results when added to the full features set. It seems it provides some not well-aligned information with the source and destination ports, because, as soon as we take away the source and destination port, it is clear it becomes again an important feature. It is also interesting to appreciate the importance of the feature TCP window size (WIN SIZE), being more important than TIMESTAMP when operating with a reduced set of features.

Table II provides metric values with a color code to make it easier to rank the results. In this color code, darker green colors mean better results whereas darker red colors mean worse results.



Features	Accuracy	F1	Precision	Recall
SRC_PORT, DST_PORT, DIR, PAYLOAD, WIN_SIZE, TIMESTAMP	0.9612	0.9553	0.9527	0.9612
SRC_PORT, DST_PORT, DIR, PAYLOAD, WIN_SIZE (Features Set 1)	0.9632	0.9574	0.9543	0.9632
DIR, PAYLOAD , TIMESTAMP , WIN_,SIZE (Features Set 2)	0.8388	0.8170	0.8279	0.8388
DIR, PAYLOAD ,WIN_SIZE (Features Set 3)	0.8202	0.7943	0.7909	0.8202
DIR, PAYLOAD , TIMESTAMP (Features Set 4)	0.7855	0.7500	0.7433	0.7855

Table II. Classification performance metrics (aggregated) vs. features employed (model CNN+RNN-2a)(Table)

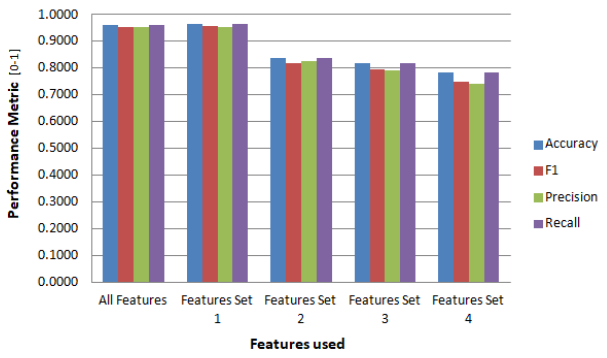


Fig. 9. Classification performance metrics (aggregated) vs. features employed (model CNN+RNN-2a)(Chart)

Model CNN+RNN-2a was used to obtain the results presented in Table II, but the same relationship between results and feature sets was maintained when repeating this same study with the other models.

C. *Impact of time-series length*

An important parameter to be studied is the influence of the number of packets to be considered when we analyze the network flows. There are flows with hundreds of packets whereas others have only a single packet.

An important doubt at the beginning of the study was the possible influence of this parameter, since increasing the number of included packets could improve detection but at the cost of much higher computing time and resources. As a balanced decision, we opted for a maximum of 20 packets. We have considered only the first 20 packets exchanged in a flow lifetime. In the case of flows with more than 20 packets, we have disregarded any packet after packet number 20. Flows with less than 20 packets were padded with zeros.

Then it was important to know the impact of the number of packets in the overall detection problem and to confirm whether 20 packets was a sensible number. To this aim, we analyzed the performance considering a different number of packets and different architectures. We present here the results for two representative architectures (RNN-1 and CNN+RNN-2a)

We can see the results for the RNN-1 architecture in Fig. 10, and it is important to note that overall detection quality is not significantly changed by using fewer packets until the number of packets is less than five per flow. Therefore, it is enough to consider the very first packets of a flow to have most of the information that allows us to infer their service. In Fig. 10 we can easily appreciate that the detection quality starts to degrade when the number of packets per flow is less than five.

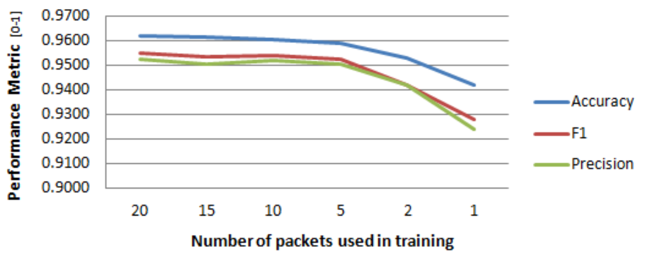


Fig. 10. Classification performance metrics (aggregated) vs. time-series length for architecture RNN-1

Fig. 11 shows the results on the impact of the number of packets for the architecture CNN-RNN-2a. This model supports a smaller reduction in the number of packets (the model needs a minimum length of 7 in the temporal-dimension), but we can still appreciate that regardless of an initial performance decrease, this reduction is not monotonic with the decrease in the number of packets, in fact, it keeps approximately constant for packets lengths in the interval from 7 to 15 (disregarding some intermediate noisy values).

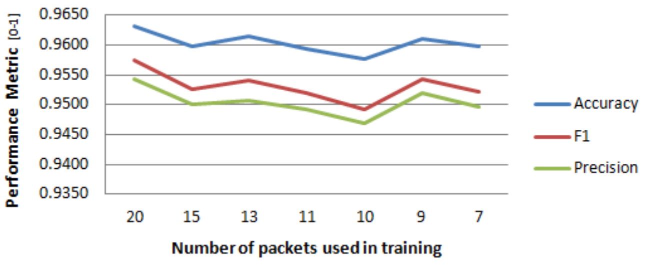


Fig. 11. Classification performance metrics (aggregated) vs. time-series length for architecture CNN+RNN-2a

Therefore, it seems clear that although a minimum number of packets is important, a large number of packets (that is architecture-dependent) is not necessary. In general, between 5 and 15 packets are enough to achieve excellent detection results.

V. Conclusion

This work is a contribution to improve the available alternatives and capabilities of NTC in current network monitoring systems; being specially targeted to IoT networks, where traffic classification is highly required [1][2].

As far as we know, there is no previous application of the RNN and CNN deep learning models to an NTC problem. Therefore, the work presented in this paper is original in essence.

This work provides a thorough analysis of the possibilities provided by deep learning models to NTC. It shows the performance of RNN and CNN models and a combination of them. It demonstrates that a CNN can be successfully applied to NTC classification, giving an easy way to extend the image-processing paradigm of CNN to a vector time-series data (in a similar way to previous extensions to text and audio processing [34][35]).

A model based on a particular combination of CNN plus RNN gives the best detection results, being these results better than other published works with alternative techniques.

The impact of selected features is demonstrated, and also that it is not necessary to process a large number of packets per flow to have excellent results: any number of packets higher than 5-15 (a number which is architecture-dependent) gives similar results.

The proposed method is robust and gives excellent F1 detection scores under a highly unbalanced dataset with over 100 different classification labels. It works with a very small number of features and does not require feature engineering.

To train the models we have made use of high-level header-based data extracted from the packets. It is not required to rely on IP addresses or payload data, which are probably confidential or encrypted.

A simple RNN model provides already very good results, but it is interesting to appreciate that these results improve when the RNN model is combined with a previous CNN model.

Being it possible to improve results with the inclusion of a CNN shows how the initial intuition that allowed us to assimilate the vector time-series extracted from network packets' features as an image is correct, and therefore CNNs are valid candidates for dealing with vector time-series of similar nature

Being the deep learning architectures such a fruitful source of new models, we consider, as future work, to experiment with new applications and variants of the CNN and LSTM models.

This work can be especially applicable for new IoT networks in which NTC can be used to differentiate or segregate different classes of traffic, e.g. device identification [36], target detection in Wireless Sensor Networks (WSN) [37] or user priority based [38].

REFERENCES

[1] B. Ng, M. Hayes and W. K. G. Seah, "Developing a traffic classification platform for enterprise networks with SDN: Experiences & lessons learned," 2015 IFIP Networking Conference (IFIP Networking), Toulouse, 2015, pp. 1-9.

[2] A. Sivanathan, D. Sherrat, H. Habibi Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman , "Characterizing and Classifying IoT Traffic in Smart Cities and Campuses", IEEE INFOCOM Workshop on SmartCity: Smart Cities and Urban Computing, USA, May 2017.

[3] T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," IEEE Commun. Surv. Tutorials, vol. 10, no. 4, pp. 56–76, 2008.

[4] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust Network Traffic Classification," IEEE/ACM Trans. Netw., vol. 23, no. 4, pp. 1257–1270, 2015.

[5] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian Neural Networks for Internet Traffic Classification," IEEE Trans. Neural Networks, vol. 18, no. 1, pp. 223–239, Jan. 2007.

[6] X. Xie, B. Yang, Y. Chen, L. Wang and Z. Chen, "Network Traffic Classification Based on Error-Correcting Output Codes and NN Ensemble," *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, Tianjin, 2009, pp. 475-479.

[7] Chen, Z., Wang, H., Abraham, A., Grosan, C., Yang, B., Chen, Y., Wang, L., "Improving Neural Network Classification Using Further Division of Recognition Space". International Journal of Innovative, Computing, Information and Control 5(2) (2009)

[8] Herrero, A., Corchado, E., Gastaldo, P., Zunino, R., "Neural projection techniques for the visual inspection of network traffic". Neurocomputing, Volume 72, Issue 16, Pages 3649-3658, 2009.

[9] Kothari, A., Keskar, A., "Rough Set Approaches to Unsupervised Neural Network Based Pattern Classifier", Advances in Machine Learning and Data Analysis, Springer Netherlands, Dordrecht, pp. 151-163, 2010.

[10] W. Zhou, L. Dong, L. Bic, M. Zhou and L. Chen, "Internet traffic classification using feed-forward neural network," *2011 International Conference on Computational Problem-Solving (ICCP)*, Chengdu, 2011, pp. 641-646.

[11] A Moore D Zuev L. Crogan "Discriminators for use in flow-based classification", Technical Report RR-05-13. Department of Computer Science Queen's Mary University, 2005.

[12] Bereket Mathewos, Marco Carvalho, and Fredric Ham. 2011. "Network traffic classification using a parallel neural network classifier architecture". In *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research* (CSIIRW '11), Frederick T. Sheldon, Robert Abercrombie, and Axel Krings (Eds.). ACM, New York, NY, USA, , Article 33 , 1 pages.

[13] Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., Lee, K.: "Internet traffic classification demystified: myths, caveats, and the best practices", In *Proceedings of the 2008 ACM CoNEXT Conference* (CoNEXT '08). ACM, New York, NY, USA, pp. 11:1–11:12, 2008.

[14] M. Shafiq, Xiangzhan Yu, A. A. Laghari, Lu Yao, N. K. Karn and F. Abdessamia, "Network Traffic Classification techniques and comparative analysis using Machine Learning algorithms," 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, China, 2016, pp. 2451-2455.

[15] C. Wang, T. Xu and X. Qin, "Network Traffic Classification with Improved Random Forest," 2015 11th International Conference on Computational Intelligence and Security (CIS), Shenzhen, 2015, pp. 78-81.
doi: 10.1109/CIS.2015.27

[16] Jun Zhang, Chao Chen, Yang Xiang, Wanlei Zhou, and Athanasios V. Vasilakos, "An Effective Network Traffic Classification Method with Unknown Flow Detection", IEEE Transaction on Network and ServiceManagement, Vol 12, Dec 2013

[17] Y.-s. Lim, H.-c. Kim, J. Jeong, C.-k. Kim, T. T. Kwon, and Y. Choi, "Internet traffic classification demystified: on the sources of the discriminative power," In Proceedings of the 6th International COnference (Co-NEXT '10). ACM, New York, NY, USA, pp. 9:1–9:12. 2010.

[18] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Offline/realtime traffic classification using semi-supervised learning," Performance Evaluation, vol. 64, no. 9-12, pp. 1194–1213, Oct. 2007.

[19] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification," SIGCOMM Comput. Commun. Rev., vol. 36, pp. 5–16, Oct. 2006.

[20] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification," in Proc. 2004 ACM SIGCOMM Conference on Internet Measurement, pp. 135–148.

[21] A. W. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," SIGMETRICS Perform. Eval. Rev., vol. 33, pp. 50–60, June 2005.

[22] S. Hao, J. Hu, S. Liu, T. Song, J. Guo and S. Liu, "Network traffic classification based on improved DAG-SVM," 2015 International Conference on Communications, Management and Telecommunications (ComManTel), DaNang, 2015, pp.256-26

[23] B. Yamansavascilar, M. A. Guvensan, A. G. Yavuz and M. E. Karsligil, "Application identification via network traffic classification," 2017 International Conference on Computing, Networking and Communications (ICNC), Santa Clara, CA, 2017, pp. 843-848.

[24] Z. Yuan and C. Wang, "An improved network traffic classification algorithm based on Hadoop decision tree," *2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS)*, Chongqing, 2016, pp. 53-56.
doi: 10.1109/ICOACS.2016.7563047

[25] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, “nDPI: Open-source high-speed deep packet inspection,” in 2014 International Wireless Communications and Mobile Computing Conference (IWCMC), 2014, pp. 617–622.

[26] T. Bujlow, V. Carela-Español, and P. Barlet-Ros, “Independent comparison of popular DPI tools for traffic classification,” Comput. Networks, vol. 76, pp. 75–89, 2015.

[27] Behnke, Sven, “Hierarchical Neural Networks for Image Interpretation”, Volume 2766 of Lecture Notes in Computer Science, Springer-Verlag, 2003

[28] Zachary C. Lipton, John Berkowitz, Charles Elkan (2015), “A Critical Review of Recurrent Neural Networks for Sequence Learning”, arXiv:1506.00019 [cs.LG]

[29] Klaus Greff; Rupesh Kumar Srivastava; Jan Koutník; Bas R. Steunebrink; Jürgen Schmidhuber (2015). "LSTM: A Search Space Odyssey". arXiv:1503.04069

[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. “Dropout: a simple way to prevent neural networks from overfitting”. J. Mach. Learn. Res. 15, 1 (January 2014), 1929-1958.

[31] Chen-Yu Lee, Patrick W. Gallagher, Zhuowen Tu (2015), “Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree”, arXiv:1509.08985 [stat.ML]

[32] Sergey Ioffe, Christian Szegedy (2015), “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, arXiv:1502.03167 [cs.LG]

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, “Scikit-learn: Machine Learning in Python”, Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[34] T. N. Sainath, O. Vinyals, A. Senior and H. Sak, "Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks," 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, QLD, 2015, pp. 4580-4584.

[35] Y. Xiao, K. Cho, “Efficient Character-level Document Classification by Combining Convolution and Recurrent Layers”, arXiv:1602.00367v1 [cs.CL] 1 Feb 2016

[36] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici. 2017. “ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis”. In Proceedings of the Symposium on Applied Computing (SAC '17). ACM, New York, NY, USA, 506-509

[37] S. Althunibat, A. Antonopoulos, E. Kartsakli, F. Granelli and C. Verikoukis, "Countering Intelligent-Dependent Malicious Nodes in Target Detection Wireless Sensor Networks," in IEEE Sensors Journal, vol. 16, no. 23, pp. 8627-8639, Dec.1, 2016.

[38] M. Grajzer, M. Koziuk, P. Szczechowiak and A. Pescape, "A Multi-Classification Approach for the Detection and Identification of eHealth Applications," 2012 21st International Conference on Computer Communications and Networks (ICCCN), Munich, 2012, pp. 1-6.