Mobile Encrypted Traffic Classification Using Deep Learning

Giuseppe Aceto[1,2], Domenico Ciuonzo[2], Antonio Montieri[1], Antonio Pescapé[1,2]
[1]University of Napoli "Federico II" (Italy) and [2]NM2 s.r.l. (Italy)
{giuseppe.aceto, antonio.montieri, pescape}@unina.it, ciuonzo@nm-2.com

***Abstract*—The massive adoption of hand-held devices has led to the explosion of mobile traffic volumes traversing home and enterprise networks, as well as the Internet. Procedures for inferring (mobile) applications generating such traffic, known as Traffic Classification (TC), are the enabler for highly-valuable profiling information while certainly raise important privacy issues. The design of accurate classifiers is however exacerbated by the increasing adoption of encrypted protocols (such as TLS), hindering the applicability of highly-accurate approaches, such as deep packet inspection. Additionally, the (daily) expanding set of apps and the moving-target nature of mobile traffic makes design solutions with usual machine learning, based on manually- and expert-originated features, outdated. For these reasons, we suggest Deep Learning (DL) as a viable strategy to design traffic classifiers based on automatically-extracted features, reflecting the complex mobile-traffic patterns. To this end, different state-of-the-art DL techniques from TC are here reproduced, dissected, and set into a systematic framework for comparison, including also a performance evaluation workbench. Based on three datasets of real human users' activity, performance of these DL classifiers is critically investigated, highlighting pitfalls, design guidelines, and open issues of DL in mobile encrypted TC.**

***Index Terms*—traffic classification; mobile apps; Android apps; iOS apps; encrypted traffic; deep learning; automatic feature extraction.**

I. Introduction

Several tools, such as security/quality-of-service enforcement devices and network monitors, operate assuming the knowledge of the application generating the traffic and thus are limited (or impaired) when this requirement is not fully satisfied. The process of associating network traffic with specific applications is known as Traffic Classification (TC) and has a long-established application in several fields [1]. TC is increasingly challenged by the massive diffusion of handheld devices (as supported by recent evaluations in Internet usage [2]), which is revolutionizing the nature of traffic traveling over home and enterprise networks and the Internet. Thereupon, both the necessity and the difficulty of mobile TC have become high nowadays, fueled (other than common drivers for TC) by the potential for valuable profiling information (e.g., to advertisers, insurance companies, and security agencies), while also implying privacy downsides (e.g., recognition of context-sensitive apps, such as health and dating ones, and in case of bring-your-own-device policies from companies).

TC comes with its own challenges and requirements that are even exacerbated in a mobile-traffic context, usually characterized by a large number of apps to discriminate from and an inadequate number of training samples per app, hindering the achievement of satisfactory performance. Moreover, the increasing adoption of encrypted protocols (TLS) [3] as well as NAT and dynamic ports, poses new challenges to accurate classification, defeating established approaches such as Deep Packet Inspection (DPI) [4] and port-based methods. Indeed, the presence of Encrypted Traffic (ET) is a severe limitation that can be bypassed only in closed-world enterprise scenarios by employing workarounds as man-in-the-middle proxies [5]. Hence, classifiers based on Machine Learning (ML) are deemed the most appropriate, especially in this context, since they suit also ET while not necessarily relying on port information [6, 7].

However, the successful use of standard ML classifiers relies on obtaining handcrafted (domain-expert driven) features, which in TC context correspond to statistics extracted from the sequence of packets [6, 7] or message sizes [8, 9]. Such process is time-consuming, unsuited to automation, and it is becoming rapidly outdated when compared to the evolution and mix of mobile traffic, being a constantly *moving target*, and precluding the design of *accurate* and *up-to-date* mobile-traffic classifiers [7, 10] with "traditional" ML approaches. Accordingly, we believe that Deep Learning (DL), which allows to train classifiers directly from input data by *automatically* learning structured feature representations [11], may be the stepping stone toward the achievement of high performance in the dynamic and challenging mobile TC context.

This paper aims at providing a systematic framework for the comparison of DL techniques declined in the mobile TC scenario. This originates from a critical analysis of several DL classifiers recently appeared in TC literature [12, 13, 14, 15, 16] and here reproduced. In detail, the proposed framework dissects the problem from *different viewpoints*, such as: (A) the TC object adopted, (B) the type of input data fed to the DL classifier, (C) the DL architecture employed, and (D) the required set of performance measures. As an application example of our framework, we report an illustrative comparison, based on three datasets of real human users' activity, to assess the most appealing techniques and highlight open issues for real-time and accurate mobile TC via DL. To the best of our knowledge, no similar systematic approach and experimental investigation have been performed in the mobile scenario to date. The outcomes of this work underline the *deficiencies of current DL-based traffic classifiers* and the need for: (1) unbiased, informative, and heterogeneous inputs extrapolated from traffic data, (2) more sophisticated DL architectures, and

(3) a rigorous performance evaluation workbench.

The rest of the paper is organized as follows. Sec. II discusses current state-of-the-art in ML-based mobile TC and recent works applying DL architectures to standard TC, whereas Sec. III describes a general DL framework for mobile TC, focusing on key aspects to be addressed; the performance evaluation workbench is then described in Sec. IV, while experimental results are discussed in Sec. V; finally, Sec. VI provides take-home messages and highlights open issues.

II. Background

TC of mobile apps has seen huge interest by several recent works, mainly based on ET assumption. This section first describes the literature on TC applied to mobile context. Then, it focuses on several efforts for DL applied to Internet TC.

Mobile Encrypted TC

Stöber et al. [17] propose a device fingerprinting scheme by learning their traffic patterns through background activities. Statistical features of 3G traffic are extracted from *bursts* of data and fed to a Support Vector Classifier (SVC) and K-Nearest Neighbors. Results show that using ≈ 15 minutes (6 hours) of traffic testing (training) leads to an accuracy $\geq 90\%$. Wang et al. [18] design a system for classifying app usage (13 iOS apps from 8 distinct categories) over 802.11 ET. A Random Forest (RF) classifier is trained/tested with the set of features obtained by running the apps for 5 minutes. The need for an accurate ground-truth labeling is also raised, highlighted by some counterintuitive per-app performance, varying the training time. *AppScanner* is proposed in [7] as a framework for fingerprinting and identification of mobile apps. Network traces are collected by *automatically* running 110 most popular apps from Google Play Store and are pre-processed to remove background traffic and extract features from sets of packets (defined through timing criteria and destination IP address/port), and then used to train an SVC and an RF. Experimental results report 99% average accuracy in app identification, and up to 86.9% in classifying them, *outperforming* state-of-the-art alternatives devised for the (conceptually-)similar website fingerprinting task [19]. These latter methods are also employed by Alan and Kaur [20] to investigate if Android apps can be identified from their launch-time traffic using only the first 64 payload sizes. Results show that apps can be identified with 88% accuracy when training and testing is performed on the same device, while a significant drop up to 26% for the best classifier is observed when the OS/vendor is different. Recently, in [10] a novel classifier fusion approach is devised for capitalizing state-of-the-art classifiers for mobile TC. Based on a dataset of real users' activity collected by a mobile solutions provider, it is shown that classification performance can be improved, by means of combination techniques, up to $+9.5\%$ of recall with respect to the best state-of-the-art classifier.

DL Applied to Standard TC

A first approach for DL-based TC is introduced in [12], focusing on *clear traffic* and considering artificial neural networks and Stacked Autoencoders (SAE). The dataset employed is made of 300k pre-processed records (after removal of HTTP traffic) and numerical results (referred to best-performing SAE) pertain to either protocol identification or anomalous protocol detection, with 58 different typologies considered. In the former case, both precision and recall achieve $\geq 90\%$ on the top 25 popular protocols. In the latter case, the SAE is tested on flows unrecognizable via DPI ($\geq 17\%$ of the whole dataset), reporting that in 6716 out of 10k cases its class prediction probability is $\geq 80\%$.

Wang et al. [13] propose a novel malware TC method based on 2D Convolutional Neural Networks (CNNs), using two different choices of raw "traffic images" (named "ALL" and "L7"). Performance is evaluated on a self-generated dataset (of $\approx$ 752k instances) and organized into two parts: ten types of malware traffic from public websites and ten types of normal traffic. The 2D-CNN described is employed for two different scenarios: malware/normal (binary) classification and traffic-type classification (20 classes). Results show that biflow-based TC is *more accurate* than its flow-based counterpart, with "ALL" input being more informative than "L7". Then, employing "Biflow + ALL" combination, the authors achieve $\geq 89\%$ per-class precision, recall, and F-measure. The work in [14] builds upon [13] but adopts a 1D-CNN (due to its appeal in handling time-series) for TC and compares it with the 2D-CNN earlier designed in [13]. Experimental results pertain to a selection from the "ISCX VPN-nonVPN" dataset [21], including 6 types of regular ET and 6 types of (VPN-)encapsulated traffic. The authors consider four setups: (i) VPN/nonVPN (binary) classification, (ii) encrypted TC (6 classes), (iii) TC of VPN-encapsulated data (6 classes), and (iv) encrypted TC (12 classes). The proposed 1D-CNN model is shown to perform best in the "Biflow + ALL" form, consistently with [13]. The proposed configuration-optimized 1D-CNN *always* achieves higher accuracy than the 2D-CNN in [13], up to $+2.51\%$ (being both however $\geq 80\%$) for the four considered setups, and outperforms the C4.5 classifier of [21] for 11 out of 12 metrics.

Lotfollahi et al. [15] propose *Deep Packet*, a DL-based traffic classifier, able to work at packet-level with *encrypted payload* as input data and employing either SAE or 1D-CNN. These DL models are evaluated on the "ISCX VPN-nonVPN" dataset [21] focusing on two different tasks: application identification (17 classes, disregarding VPN/nonVPN condition), and traffic characterization (12 activities). Both 1D-CNN and SAE achieve an average F-measure of 95% and 97%, respectively, in the two tasks. Also, Deep Packet is shown to *outperform* state-of-the-art classifiers, based on handcrafted features, known to achieve the highest performance on the considered dataset.

Finally, in [16], a number of DL architectures for encrypted TC, based on hybrid combinations of Long Short-Term Memory (LSTM) and 2D-CNN layers and traffic input in the form of traffic time-series/images, are proposed and compared to standard CNN and LSTM. Performance is evaluated on the "RedIRIS" dataset, captured on the Spanish academic

backbone network, made of $\approx$ 266k (TCP/UDP) biflows and collectively belonging to 108 *distinct services*. The best-performing model is a combination of 2D-CNN and LSTM layers (named "CNN+RNN-2A"), attaining an accuracy (resp. F-measure) of 96.32% (resp. 95.74%). In this case, it is shown that inter-arrival time information *slightly degrades performance*, but also that $5 \div 15$ packets are typically sufficient for excellent detection results (for other DL architectures too).

III. Traffic Classification

This section dissects the state-of-the-art of DL in TC, by focusing on the following *viewpoints*: (A) the traffic view (i.e. the type of traffic aggregate), (B) the type of input data fed to the DL architecture, and (C) the DL architecture employed.

It is worth pointing out that all the DL classifiers proposed for TC have been carefully analyzed and reproduced, e.g. by setting the hyper-parameter values suggested in their respective works or performing a basic tuning procedure when the latter are not reported. Specifically, we leveraged DL models provided by *Keras* (Python) API running on top of *TensorFlow* to implement and test the approaches described in the following.

A. Traffic View

Different traffic objects have been considered in the TC literature. The definition of a specific traffic object determines how raw traffic is segmented into multiple discrete traffic units [1]. It is worth noticing that all the works approaching the TC using DL [12, 13, 14, 16] considered either *flows* or *biflows* as the relevant objects of classification, with the sole exception of [15]. More specifically, a *flow* is defined as all the packets having the same 5-tuple (i.e. source IP, source port, destination IP, destination port, and transport-level protocol) taking into account their directions. Differently, a *biflow* includes both directions of traffic sharing a given tuple (i.e. the source and the destination are *interchangeable*).

Finally, in [15] the relevant object of classification is the *single packet* (i.e. the classification procedure is performed at packet level), corresponding to the *finest* granularity for a TC problem (and virtually representing the *hardest setup* for the corresponding classification task).

B. Types of Input Data

The type of data being fed to the surveyed DL architectures may be roughly categorized within *three* types:

I the first N bytes of payload of TC object [13, 14, 18];
II the first N bytes of raw data pertaining to the PCAP file related to the TC object [13, 14];
III informative data fields of first N_p packets [16].

In the *first* case, the data being fed to the DL architecture is represented by *payload only*, with input data in *binary format*. In all these works, the payload is arranged in a *byte-wise* fashion and normalized (by 255) so as to constrain it within $[0, 1]$. The choice is always justified as a means to *reduce the input size* for the DL architecture. On the other hand, the *layer* and *size* of the payload being chosen depend on the specific work. For example, in [12] these correspond to the first 1000 bytes of TCP payload. A similar choice is made in [13, 14] for the input labeled as "L7", where 784 bytes from the *application layer* in TCP/IP model are considered. Differently, in [15] the authors consider the first 1500 payload bytes at *layer 2*, i.e. the IP header and the first 1480 bytes of each IP payload which results in a 1500 bytes input vector.[1]

The *second* type of input data attempts to gather information from *all protocol layers* (denoted with "ALL" layers in [13, 14]) as in some relevant cases the data from levels lower than layer 7 also contain some useful traffic information (such as transport-layer ports or flags), as pointed out in [13, 14]. Then, since the considered data are typically captured at *data-link layer*, the payload from frames of *layer* 2 is extracted. However, the traffic provided in this case is always in the form of PCAP files, containing information that could introduce a bias in the classification results.[2] Specifically, in [13, 14] only the first 784 bytes of each TC object are employed.

Finally, the *third* type of input data is represented by selected protocol fields (not pertaining to the explicit inspection of encrypted payload) of the first N_p packets. For example, in [16] the authors consider only the *first* 20 *packets* exchanged into a TC object (a biflow), and, for each packet, the following 6 fields are extracted (thus a 20×6 matrix is obtained for each TC object): source and destination ports, number of bytes in transport layer payload, TCP window size[3], inter-arrival time, and packet direction ($\in \{0, 1\}$). We highlight that the sequence of packets/messages directions has been also recently employed in DL-based website fingerprinting [22].

Finally, we conclude the discussion mentioning that in all the above cases, there may be instances *longer* or *shorter* than the considered *fixed-length* data inputs. In such cases, *longer* instances are truncated to the designed length of bytes or packets, in the case of first/second or third type of data, respectively, whereas in the case of *shorter* instances, padding with zeros is always applied in all the discussed works.

C. DL Classification Algorithms

Here we review the DL architectures employed for TC. To this end, we define the i^{th} input of the training set (made of M samples) as $\boldsymbol{x}_{(i)}$ while the corresponding label with $\ell_{(i)}$. All the considered DL classifiers are trained to minimize the *categorical cross-entropy* [11], achieved by standard local optimizers (e.g. SGD, Adam, etc.) via back-propagation.

SAE: The SAE relies on the basic AutoEncoder (AE) concept, employed for (unsupervised) feature learning, and whose objective is to (ideally) set the output $\boldsymbol{y}_{(i)} \approx \boldsymbol{x}_{(i)}$, $\forall i = 1, \ldots, M$, by learning a *compressed* data representation. Specifically, the first AE block (i.e. the *encoder* $\boldsymbol{h} = \boldsymbol{\sigma}(\boldsymbol{W}\boldsymbol{x} +$

[1] Additionally, the author apply also a pre-processing step to cope with unequal transport-layer header lengths (e.g. TCP and UDP), by injecting zeros at the end of the UDP-datagram headers to make them equal with TCP-segment headers in length.

[2] Note that, by extracting "ALL Layers", input includes PCAP metadata besides raw packet data (from MAC layer, included). In detail, PCAP global header is of 24 bytes and each packet is also prepended with a header of 16 bytes, including a timestamp at μs granularity and info on the packet size.

[3] The TCP window size is set to *zero* for UDP packets.

$\boldsymbol{b}$)) reduces the dimension of data by providing a compressed representation (via the hidden layer $\boldsymbol{h}$, made of a number of neurons), whereas the second block tries to reconstruct the data from the low-dimensional representation obtained by the encoder (i.e. the *decoder* $\boldsymbol{y} = \overline{\boldsymbol{\sigma}}(\overline{\boldsymbol{W}}\boldsymbol{h} + \overline{\boldsymbol{b}})$).

In practice, to obtain improved performance, a more complex (hierarchical) architecture, named Stacked AE (SAE), is proposed [11]. This scheme employs *unsupervised greedy layer-wise pre-training* which stacks up several AEs so that the lower-dimensional representation obtained from j^{th} AE is used as the input of $(j+1)^{th}$ AE, that is $\boldsymbol{x}^{(j+1)} = \boldsymbol{h}^{(j)}$ (i.e. each layer of network is trained by keeping the weights of lower layers *frozen*). After greedy training of all AE layers, the classification task requires a final softmax layer to be added. Then, supervised *fine-tuning* (i.e. a *refinement* of all layers' weights) of the whole network is performed (i.e. exploiting $\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(M)}$ along with $\ell_{(1)}, \ldots, \ell_{(M)}$).

A relevant application of SAE to TC is given in [15], consisting of *five* stacked layers (with $\{400, 300, 200, 100, 50\}$ neurons, respectively, all employing *Rectified Linear Units* (ReLUs)). Also, to mitigate over-fitting, after each layer the *dropout* technique with 25% drop-probability is applied [11].

CNN: The Convolutional Neural Networks (CNNs) are widely-used DL models, inspired by visual mechanism of living organisms, achieving feature learning via several convolutional layers. Each of them comprises a set of translation-invariant *filters* with a limited extent (the "receptive field") which are *convolved* with the input with the aim of extracting features of a certain input region. CNN layers can be conceived in either 1D or 2D form, depending on the specific input nature. The key idea in CNN is based on chaining several convolutional layers, to extract increasingly complex and abstract features *automatically*. Another important CNN component are the *pooling* layers, typically found in-between successive convolutional layers and whose function is to perform down-sampling (*max-* and *average-pooling* are the most common) of intermediate representations, aiming at complexity reduction and *overfitting* mitigation. The higher CNN layers are usually a few final fully-connected (compressing, similar to AE encoding stages) layers, with the first taking as input the (flattened) set of features associated to all regions and the last having an essential *softmax* activation for the classification task.

For example, the architecture in [14] is made of *two* 1D convolutional layers (with 32 and 64 filters, respectively), each followed by a 1D max-pooling, and terminated with *two* fully-connected layers. A similar CNN architecture is presented in [13], i.e. by replacing 1D with 2D (pooling/convolutional) layers and interpreting the input as an equivalent "traffic image". A 2D-CNN is also considered in [16], where *batch normalization* [11] is also applied after each max-pooling layer. On the other hand, in [15] a 1D-CNN consisting of *two* 1D convolutional layers (with 200 and 80 filters, respectively), followed by a 1D average-pooling layer, is considered. The CNN is terminated with *seven* fully-connected layers (with $\{600, 500, 400, 300, 200, 100, 50\}$ neurons) having ReLUs. Also, to avoid over-fitting, 25% dropout after pooling layer and *early stopping* technique are adopted [11].

LSTM: A Long Short-Term Memory (LSTM) represents a popular variant of Recurrent Neural Networks (RNNs, having connections between units which form a directed cycle, thus exhibiting a *dynamic* temporal behavior), capable of modeling "long-term dependencies" and being easier to train [11]. An architecture made of LSTM units is called an LSTM network. An LSTM unit is responsible for "remembering" values (in the form of a state vector $\boldsymbol{h}[t]$) over arbitrary time intervals and is composed of a *cell* ($\boldsymbol{c}[t]$), an *input* gate ($\boldsymbol{i}[t]$), an *output* gate ($\boldsymbol{o}[t]$), and a *forget* gate ($\boldsymbol{f}[t]$), while having as input a time-series of vectors of length T (*for each instance*), here denoted as $\boldsymbol{x}[1], \ldots, \boldsymbol{x}[T]$ (i.e. each training instance of an LSTM is a matrix representing the *temporal evolution* of a *vector of inputs*). The key equations of an LSTM unit are those governing the evolution of the three gates, having the same form $\boldsymbol{\sigma}_g(\boldsymbol{W}\,\boldsymbol{x}[t] + \boldsymbol{U}\,\boldsymbol{h}[t-1] + \boldsymbol{b})$, i.e. they are updated based on both the current input $\boldsymbol{x}[t]$ and the previous state $\boldsymbol{h}[t-1]$. Once the gates are updated, the cell and state update equations are $\boldsymbol{c}[t] = \boldsymbol{f}[t] \circ \boldsymbol{c}[t-1] + \boldsymbol{i}[t] \circ \boldsymbol{\sigma}_c(\boldsymbol{W}_c\,\boldsymbol{x}[t] + \boldsymbol{U}_c\,\boldsymbol{h}[t-1] + \boldsymbol{b}_c)$ and $\boldsymbol{h}[t] = \boldsymbol{o}[t] \circ \boldsymbol{\sigma}_h(\boldsymbol{c}[t])$, where "$\circ$" denotes the element-wise product.[4] The final hidden state $\boldsymbol{h}[T]$ corresponds to the output of LSTM unit. A simple LSTM network for classification is usually terminated with a few fully-connected layers, with last having a softmax activation. On the other hand, when several LSTM layers are stacked, their outputs (except for the last one) are finer-grained and correspond to the state time-evolution $\boldsymbol{h}[1], \ldots, \boldsymbol{h}[T]$, forming the input to the higher LSTM layer.[5]

For example in [16] a simple LSTM ending with *two fully-connected* layers of 100 and 108 nodes (the latter being the number of services to discriminate from) is considered. Interestingly, a stack of LSTM layers is also proposed in [16] in the context of hybrid architectures, as described henceforth.

Hybrid DL Architectures: The discussed elementary learning layers can be jointly *employed* within a single DL architecture. For example, architectures based on the combination of 2D convolutional and LSTM layers may be conceived [16], where the output tensor of the convolutional layer is reshaped into a matrix that can act as the input of an LSTM unit.

IV. Performance Evaluation of DL classifiers

First, our comparison include the following common performance measures [1]: overall accuracy, precision, and recall. Since the latter two are defined on a per-app (per-class) basis, we employ their arithmetically averaged (viz. macro) versions and consider the *F-measure* $\mathrm{F} \triangleq (2 \cdot \mathrm{prec} \cdot \mathrm{rec})/(\mathrm{prec} + \mathrm{rec})$, so as to account for both of them *concisely*. Moreover, the concept of *Top-K accuracy* (recently used in website fingerprinting, see e.g. [23]) is employed, defining a correct classification event if the true app is within the top K predicted labels (K is a free parameter).[6] Furthermore, we also consider

[4] Both the activations $\boldsymbol{\sigma}_{\boldsymbol{c}}(\cdot)$ and $\boldsymbol{\sigma}_h(\cdot)$ are usually hyperbolic tangents, whereas $\boldsymbol{\sigma}_g(\cdot)$ is usually a sigmoid.

[5] We highlight that for successive LSTM layers, the temporal-dimension of data input does not change, whereas the vector-size of the successive inputs does, being function of the size of the hidden state.

[6] Of course $K = 1$ coincides with the standard accuracy.

the confusion matrices of DL classifiers with the aim of identifying the most frequent misclassification patterns.

To provide a complete performance picture, classifiers are also tested when they are enriched with a "reject option" (i.e. the classification is performed only if the highest class prediction probability exceeds a threshold γ and "unsure" classifications are then *censored*), whose adoption has been justified in the mobile context [7], as there is no inherent requirement to label all unknown flows, since there remains high chance to identify apps from their more distinctive ones, as they typically send multiple flows when used. Hence, tuning γ can be effective to improve classification performance while incurring negligible drawback, i.e. a *decreased* ratio of classified instances. For completeness, as a preliminary investigation of the computational complexity of DL-architectures training phase, we report their per-epoch run-time [11].

Finally, for each considered analysis, our evaluation is based on a (stratified) ten-fold cross-validation, representing a stable performance evaluation setup. For completeness, we report both the mean and the variance of each performance measure as a result of the evaluation on the ten different folds.

V. Experimental Results

The present section investigates and compares performance of considered DL classifiers, according to Sec. IV, based on the three mobile traffic datasets described next.

A. Datasets Description

The three datasets considered in this work have been all collected by *human users*. Also, the ground truth has been obtained by labeling each trace with the generating app (since they have been run *separately*, thus limiting the presence of background traffic) and, for the sake of a consistent comparison among all DL-based TC works published so far (except for [15]), we have chosen to operate at the *biflow level*.

The first two datasets are from an international mobile solutions provider and are generated from a total of 49 apps (resp. 45) on Android (resp. iOS) devices (i.e. multi-class datasets).[7] These traces have been collected within Sept. 2014 - Jan. 2017 and provided already anonymized and cleaned from background traffic. Mobile traffic has been generated by different human-users employing various devices, without specific constraints on the operating system / app version, being the latter a worst case for mobile TC [20]. Further details can be found in [10]. Globally, after *biflow* segmentation, we obtained about 77.5k (resp. 44.0k) labeled biflows.

The third dataset has been collected in our laboratory (ARCLAB) at the University of Naples "Federico II", during the time frame May-Nov. 2017. More specifically, the captures pertain to either Facebook (FB) or Facebook Messenger (FBM) traffic data (i.e. a binary dataset), and run on a Xiaomi Mi5 with Android OS 6.0.1 (CyanogenMod 13.0 distribution). Such dataset has been collected to analyze the capabilities of DL classifiers to discriminate from almost "overlapped" apps' fingerprints, e.g. with the objective of billing differentiation. *More than* 100 *users* have been involved in its construction and required to perform different activities for both the apps (to explore their *diversity*), also in conjunction with login/registration/logged-use cases.[8] Each traffic-capture session had a duration of $5 \div 10$ minutes, and more than 1000 traffic traces have been collected. As a whole, the dataset is made up of ≈ 27.5k biflows, with 10.5k (resp. 17.0k) biflows generated by FBM (resp. FB) app, with a $38\%/62\%$ share.

It is worth noting that depending on the particular classification approach and input data considered, preprocessing operations could have been carried out on the datasets, varying the actual number of biflows.

B. Classification Results

In this section, we provide a systematic comparison of the considered DL architectures so as to draw out important guidelines. For the sake of completeness, two baseline approaches are included in our analysis: (i) the flow-based RF developed in [7], taking as input 40 carefully handcrafted features, representing the current state-of-the-art mobile-traffic classifier, but applicable only in the case of "post-mortem" TC, and (ii) a Multi-Layer Perceptron with only one hidden layer (with 100 nodes), here denoted as MLP-1, trained on the same inputs as DL architectures, so as to underline the performance achievable by *shallow learning* in the considered scenario.

In the following, we will refer to the input data corresponding to the first N bytes of payload (resp. raw) data as "L7-N" (resp. "ALL-N") [12, 13, 14] and to the 20×6 matrix extracted from each biflow following [16] as "MAT" (see Sec. III-B).[9]

First, in Tab. I we report the results of state-of-the art DL-based (and baseline) approaches fed with inputs (and features) extracted from multi-class Android and iOS datasets, and binary FM/FBM dataset. We highlight that performance with diamond ($\Diamond$) markers represent results for *biased inputs* (cf. Sec. III-B), therefore they should not be considered as meaningful elements of comparison. From the inspection of results it is apparent that, referring to multi-class dataset, DL approaches are able to provide improved performance with respect to shallow classifiers with analogous unbiased inputs, i.e. MLP-1 (L7-1000/L7-784/MAT), and even outperform flow-based state-of-the-art RF. Indeed, in Android setup, 86.01% accuracy and 78.97% F-measure are achieved by 1D-CNN (L7-784), as opposed to 83.79% and 74.12% by the RF. A similar reasoning applies to iOS case, where the 1D-CNN (L7-784) (resp. LSTM) performs the best in terms of accuracy (resp. F-measure). Finally, referring to the binary dataset FB/FBM, only a 2D-CNN (L7-784) is able to outperform the shallow classifiers MLP-1 (L7-1000/L7-784) in terms of both accuracy and F-measure. Nonetheless, neither the best DL classifier in the binary dataset is able to achieve performance

[7] Due to NDA with the provider we can not report its name, details of its network, detailed information on the data set, nor release the data set.

[8] Precisely, FB use comprises friends adding/deleting, dashboard messages posting, likes/reactions adding to posts/comments, etc., whereas FBM use includes private messages sending/receiving, file sharing, (video-)calls, etc.

[9] We will not consider, for brevity, the input data corresponding to payload at layer 2 [15], as similar conclusions can be drawn from the case "ALL-N".

Table I: Accuracy and F-measure [%] comparison of DL-based and baseline traffic classifiers. Results refer both to the multi-class and binary datasets and are in the format *avg. (± std.)* obtained over 10-folds. Results with diamond (◊) are from *biased* inputs. Starred (*) results refer to inputs including TCP/UDP ports. **Best-performing** DL-based classifiers are highlighted.

Architecture	Android		iOS		FB/FBM	
	Accuracy	F-Measure	Accuracy	F-Measure	Accuracy	F-Measure
SAE [15] (L7-1000)	39.19 (± 13.71)	27.08 (± 12.30)	33.74 (± 7.55)	20.81 (± 6.63)	68.80 (± 0.81)	66.13 (± 1.24)
2D-CNN [13] (L7-784)	85.89 (± 0.50)	**79.19 (± 1.36)**	83.09 (± 1.36)	74.76 (± 1.65)	**75.41 (± 1.79)**	**72.10 (± 1.07)**
2D-CNN [13] (ALL-784)◊	95.68 (± 0.40)	92.14 (± 0.81)	96.13 (± 0.33)	93.38 (± 0.54)	76.19 (± 1.47)	73.55 (± 1.47)
1D-CNN [14] (L7-784)	**86.01 (± 0.55)**	78.97 (± 1.21)	83.74 (± 0.57)	75.32 (± 1.09)	73.91 (± 2.02)	70.72 (± 4.62)
1D-CNN [14] (ALL-784)◊	96.50 (± 0.23)	93.26 (± 0.86)	96.10 (± 0.27)	92.67 (± 0.57)	76.79 (± 1.53)	74.24 (± 2.36)
2D-CNN [16] (MAT)*	82.68 (± 0.62)	71.40 (± 1.23)	81.27 (± 1.08)	74.04 (± 1.69)	71.60 (± 0.95)	69.14 (± 1.20)
LSTM [16] (MAT)*	81.64 (± 0.60)	69.80 (± 1.33)	**83.69 (± 1.16)**	**77.04 (± 1.70)**	71.85 (± 1.04)	69.11 (± 1.20)
LSTM + 2D-CNN [16] (MAT)*	84.25 (± 0.49)	72.75 (± 0.94)	82.51 (± 0.60)	74.74 (± 0.97)	72.42 (± 0.70)	69.22 (± 0.99)
MLP-1 (L7-1000)	77.87 (± 0.56)	68.27 (± 1.57)	80.11 (± 1.08)	72.20 (± 1.31)	74.05 (± 1.05)	71.42 (± 0.91)
MLP-1 (L7-784)	78.41 (± 0.48)	69.73 (± 1.10)	80.84 (± 1.01)	72.16 (± 1.84)	73.60 (± 1.14)	71.72 (± 0.78)
MLP-1 (ALL-784)◊	96.35 (± 0.43)	93.93 (± 0.76)	97.26 (± 0.44)	95.21 (± 0.81)	76.16 (± 0.85)	74.09 (± 0.89)
MLP-1 (MAT)*	72.13 (± 0.54)	57.22 (± 1.09)	66.58 (± 0.52)	55.95 (± 1.02)	69.94 (± 0.58)	65.72 (± 1.04)
RF [7] (flow-based)	83.79 (± 0.44)	74.12 (± 1.63)	81.11 (± 0.72)	70.86 (± 0.99)	81.64 (± 0.94)	79.41 (± 1.05)

Table II: Top-K accuracy [%] comparison of DL-based and baseline traffic classifiers. Results refer to the multi-class datasets and are in the format *avg. (± std.)* obtained over 10-folds. Starred results refer to inputs including TCP/UDP ports.

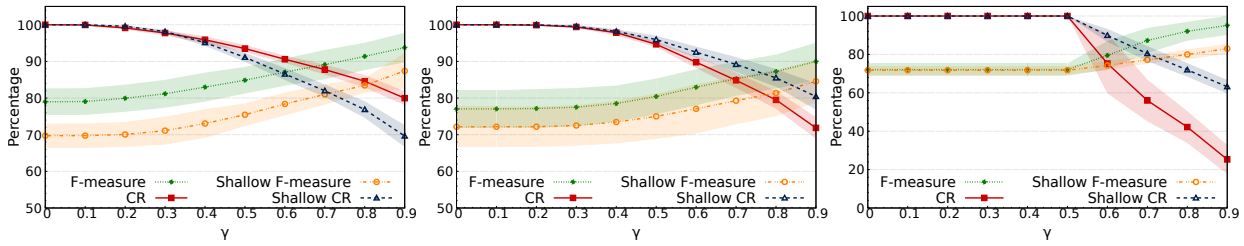
Architecture	Android			iOS		
	$K=1$	$K=3$	$K=5$	$K=1$	$K=3$	$K=5$
2D-CNN [13] (L7-784)	85.89 (± 0.50)	91.68 (± 0.35)	93.64 (± 0.36)	83.09 (± 1.36)	91.04 (± 0.65)	93.43 (± 0.63)
1D-CNN [14] (L7-784)	86.01 (± 0.55)	91.78 (± 0.30)	93.73 (± 0.20)	83.74 (± 0.57)	91.47 (± 0.67)	93.64 (± 0.58)
2D-CNN [16] (MAT)*	82.68 (± 0.62)	91.50 (± 0.32)	94.28 (± 0.24)	81.27 (± 1.08)	92.50 (± 0.52)	95.41 (± 0.36)
LSTM [16] (MAT)*	81.64 (± 0.60)	92.01 (± 0.44)	95.00 (± 0.28)	83.69 (± 1.16)	94.77 (± 0.37)	97.04 (± 0.25)
LSTM + 2D-CNN [16] (MAT)*	84.25 (± 0.49)	91.80 (± 0.27)	94.31 (± 0.21)	82.51 (± 0.60)	92.57 (± 0.37)	95.44 (± 0.25)
MLP-1 (L7-1000)	77.87 (± 0.56)	86.20 (± 0.41)	89.41 (± 0.39)	80.11 (± 1.08)	88.44 (± 0.69)	91.27 (± 0.49)
MLP-1 (L7-784)	78.41 (± 0.48)	86.81 (± 0.35)	89.87 (± 0.31)	80.84 (± 1.01)	89.29 (± 0.61)	92.01 (± 0.52)
MLP-1 (MAT)*	72.13 (± 0.54)	85.46 (± 0.35)	90.10 (± 0.43)	66.58 (± 0.52)	83.95 (± 0.60)	89.75 (± 0.47)
RF [7] (flow-based)	83.79 (± 0.44)	91.25 (± 0.24)	93.74 (± 0.29)	81.11 (± 0.72)	90.91 (± 0.56)	93.85 (± 0.37)

comparable with flow-based RF. This may attributed to the need of a more informative type of input, having a higher discriminative power in the case of very similar apps, like FB and FBM. Focusing on the DL approaches with "MAT" input, it should be noted that the performance refers to an input comprising *TCP/UDP source and destination ports* (see [16] for details). Further investigations, led *without* considering these latter fields, have revealed different trends between multi-class and binary datasets. In details, a drop in performance up to -16.28% (resp. -19.46%) in accuracy (resp. F-measure) is shown for multi-class datasets (the worst drop affects LSTM in the iOS case) when TCP/UDP ports are not considered as inputs. On the other hand, FB/FBM classification task turns out to be *port-independent* showing also an accuracy (resp. F-measure) gain of $+1.29\%$ (resp. $+1.40\%$).

Further investigating the performance of DL-based classifiers, in Tab. II we report their Top-K accuracy ($K \in \{1, 3, 5\}$) on the multi-class dataset. From now on we exclude, for brevity, the results achieved by DL classifiers based on biased inputs. By looking at these fine-grained results, it is apparent that although 1D-CNN (L7-784) reports the highest accuracy, LSTM presents a better "global behavior". Indeed, in Android and iOS scenarios, the latter classifier is able to reach 92.01% and 94.77% (resp. 95.00% and 97.04%) accuracy when the Top-3 (resp. Top-5) predicted apps are considered.

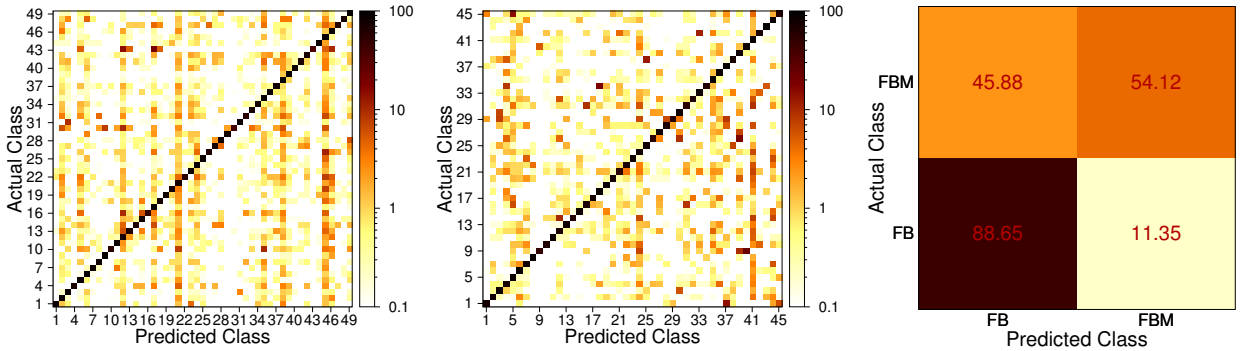
As a complementary analysis, Fig. 1 shows the F-measure and ratio of classified samples of both the best DL approach and shallow classifier on each of the three datasets vs. the censoring threshold γ. By looking at the results, all the methods globally benefit from increasing γ at the price of a decreasing ratio of classified instances. Note that the accuracy–not shown for brevity–presents analogous trends with γ. However, only in the multi-class dataset scenario, it is evident a relevant performance improvement with a negligible ratio of unclassified samples, whereas in the binary dataset this trend appears to be *sharper* and less advantageous (although the best DL classifier tends to be "less wrong" than its shallow counterpart). Specifically, by rejecting the classification of only 10% of instances, in the case of Android and iOS datasets, 1D-CNN (L7-784) and LSTM are able to achieve $\geq 84\%$ F-measure. Sadly, in the FB/FBM case, the same target F-measure requires $\geq 40\%$ *biflows to be censored*. This result underlines the inability of DL framework to tackle an "overlapped-apps" classification task with the present input/architecture choices.

Then, Fig. 2 shows the confusion matrices of best-performing DL-approaches in the three datasets, so as to



(a) 1D-CNN [14] (L7-784). (b) LSTM [16] (MAT). (c) 2D-CNN [13] (L7-784).

Figure 1: F-Measure and ratio of classified samples (CR) [%] vs. censoring threshold γ of the best DL-based classifier for the (a) Android, (b) iOS, and (c) FB/FBM datasets. Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown.



(a) 1D-CNN [14] (L7-784). (b) LSTM [16] (MAT). (c) 2D-CNN [13] (L7-784).

Figure 2: Confusion matrices of the best DL-based classifier for the (a) Android, (b) iOS, and (c) FB/FBM datasets. Log scale is used to evidence small errors (except for FB/FBM dataset). Class labels–not shown for brevity–are the same as in [10].

investigate possible relevant error-patterns. While the 1D-CNN (L7-784) and LSTM achieve almost-uniform error patterns, in the Android and iOS cases, respectively, it is apparent that 2D-CNN (L7-784) operating on the binary dataset entails a prediction imbalance toward FB app, as a consequence of the higher number of samples in the dataset. This aspect contributes to the unsatisfactory performance of DL-based classifiers when compared to shallow classifiers and RF, as reported in Tab. I.

Finally, we mention that the average run-time per epoch of the three best DL classifiers equals 136.2 $(\pm 0.8)s$, 22.7 $(\pm 0.6)s$ and 65.2 $(\pm 0.9)s$ for 1D-CNN (L7-784), LSTM, and 2D-CNN (L7-784), respectively, with the corresponding overall training times equal to 5310.9 $(\pm 184.4)s$, 2043.8 $(\pm 53.1)s$ and 3909.3 $(\pm 52.2)s$.

VI. Take-home Messages and Open Issues

We tackled classification of mobile traffic via DL architectures, providing a framework for comprehensive evaluation and comparison, by dissecting existing DL works in standard TC. This thorough analysis has allowed to emerge a list of guidelines and sparks, and highlight all the caveats which pertain to network traffic analysis domain, so as to avoid pitfalls in the design and evaluation of (mobile) traffic classifiers and provide a step forward toward real-world implementations [24]. These are summarized hereinafter as *spotlight messages*.

Choice of TC object: The present analysis, for brevity and consistency with surveyed DL-based traffic classifiers, only considered biflow-based TC. However, recent mobile literature has shown the appeal of TC objects accounting for the bursty nature of traffic, see e.g. [7]. Although appealing, a definition of reasonable (and effective) input data in the latter case is not straightforward and deserves further attention in our opinion.

Unbiased and informative input: Mobile TC presents its own peculiarities, which hinder the straightforward application of DL classifiers originated from other domains (e.g., image/speech processing), as clearly shown in this work. Indeed, a DL classifier fed with all the data contained in a packet likely leads to *misleading performance results*. One relevant case is [13, 14], adopting the "ALL layers" input, and thus overlooking the presence of PCAP metadata. Similarly, the input proposed in [16] includes port numbers, yielding DL statistical port-based architectures.[10] A key outcome of this study was to *skim informative and unbiased information from traffic data* to be used as DL classifiers' input.

Choice of DL traffic classifier: Results in Sec. V-B, based on SAE, CNN, LSTM, and hybrid architectures, highlighted that there is no "killer" DL architecture for mobile TC. Indeed, the most the DL model fits the nature of the input data, the better it is expected to perform. One relevant example is the comparison of 1D- and 2D-CNN with payload data which is, by definition, *one-dimensional*. Similar reasoning applies to DL classifiers based on "MAT" input [16], where the presence of LSTM layers reflects the time-series nature

[10]Also, whether destination port may be useful in some "static" contexts, this is never the case for the source port.

of the packet fields considered. Thus, given the heterogeneous information available from traffic data, the need for *advanced hybrid DL architectures* arises. Moreover, although a key issue of DL is the high requirement on training data (to allow the "surfacing" of deep representations), in the supervised context of mobile TC, the aspect of the *purity* of labeled samples employed for training (i.e. the ground-truth quality) is equally important, with (coarse) trace-level labeling probably not representing the "purest" strategy (i.e. including some non-app instances). Equally important, although DL architectures relieve the designer from the feature design issue, they come with many hyper-parameters to be tuned (such as the optimizer, the number of layers/hidden nodes, the regularizers, etc.). Hence, a grid search on these parameters (although prone to automation) may be as tedious as manually extracting discriminative features. Nonetheless, it is common experience that high performance can be achieved only if (sufficiently) "informative" data is fed to the DL classifier.

Comprehensive performance evaluation framework: The presence of several DL architectures highlights the need for a rigorous performance evaluation framework in (mobile) TC. This work *provided a first attempt of its formalization*. Recent literature has ascertained that a naïve accuracy comparison is not sufficient, and measures reflecting a per-app behavior (F-measure, confusion matrices, etc.) are increasingly considered [7, 10]. Going further, we investigated DL architectures output at a finer detail by means of Top-K accuracy and performance analysis with a reject option, being essential in highly multi-instance and multi-class classification tasks, respectively, such as the mobile one. Finally, for completeness, the framework also included a baseline "shallow" network to clearly assess DL performance gain, and analyzed the computational complexity of classifiers' training phase.[11]

REFERENCES

[1] A. Dainotti, A. Pescapè, and K. C. Claffy, "Issues and future directions in traffic classification," *IEEE Network*, vol. 26, no. 1, pp. 35–40, 2012.

[2] N. Heuveldop *et al.*, "Ericsson mobility report," *Ericsson AB, Technol. Emerg. Business, Stockholm, Sweden, Tech. Rep. EAB-17*, vol. 5964, 2017.

[3] A. Razaghpanah, A. A. Niaki, N. Vallina-Rodriguez, S. Sundaresan, J. Amann, and P. Gill, "Studying TLS usage in Android apps," in *ACM CoNEXT'17*.

[4] G. Aceto, A. Dainotti, W. De Donato, and A. Pescapè, "PortLoad: taking the best of two worlds in traffic classification," in *IEEE INFOCOM'10*.

[5] H. Yao, G. Ranjan, A. Tongaonkar, Y. Liao, and Z. M. Mao, "Samples: Self adaptive mining of persistent lexical snippets for classifying mobile application traffic," in *ACM MobiCom'15*.

[6] B. Saltaformaggio, H. Choi, K. Johnson, Y. Kwon, Q. Zhang, X. Zhang, D. Xu, and J. Qian, "Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic," in *USENIX WOOT'16*.

[7] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *IEEE EuroS&P'16*.

[8] A. Hajjar, J. Khalife, and J. Díaz-Verdejo, "Network traffic application identification based on message size analysis," *Elsevier JNCA*, vol. 58, pp. 130–143, 2015.

[9] C.-N. Lu, C.-Y. Huang, Y.-D. Lin, and Y.-C. Lai, "High performance traffic classification based on message size sequence and distribution," *Elsevier JNCA*, vol. 76, 2016.

[10] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "Multi-classification approaches for classifying mobile app traffic," *Elsevier JNCA*, vol. 103, pp. 131–145, 2018.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[12] Z. Wang, "The Applications of Deep Learning on Traffic Identification." Black Hat USA, Las Vegas, 2010.

[13] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using CNN for representation learning," in *IEEE ICOIN'17*.

[14] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional CNNs," in *IEEE ISI'17*.

[15] M. Lotfollahi, R. Shirali, M. J. Siavoshani, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using DL," *arXiv*, 2017.

[16] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for Internet of Things," *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017.

[17] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, "Who do you sync you are? smartphone fingerprinting via application behaviour," in *ACM WISEC'13*.

[18] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone: Inferring app usage over encrypted data traffic," in *IEEE CNS'15*.

[19] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial Naïve-Bayes classifier," in *ACM CCSW'09*.

[20] H. F. Alan and J. Kaur, "Can Android applications be identified using only TCP/IP headers of their launch time traffic?" in *ACM WiSec'16*.

[21] G. D. Gil, A. H. Lashkari, M. Mamun, and A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related features," in *SciTePress ICISSP'16*.

[22] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated feature extraction for website fingerprinting through Deep Learning," *arXiv*, 2017.

[23] S. E. Oh, S. Sunkam, and N. Hopper, "Traffic analysis with deep learning," *arXiv*, 2017.

[24] W. De Donato, A. Pescapè, and A. Dainotti, "TIE: an open platform for traffic classification," *IEEE Network*, vol. 28, no. 2, pp. 56–64, 2014.

[11] Indeed, while test complexity is directly associated to the classifier at run-time, training complexity equally represents a key aspect in mobile TC, where periodical re-training is required due to apps and/or OS updates.